

Trabajo Fin de Grado

Grado en Ingeniería de las Tecnologías Industriales

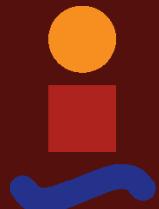
Prototipo de sistema electrónico portátil de
vigilancia y supervisión

Autor: Gonzalo Jiménez Alonso

Tutor: Federico Barrero García

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025



Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Prototipo de sistema electrónico portátil de vigilancia y supervisión

Autor:
Gonzalo Jiménez Alonso

Tutor:
Federico Barrero García
Catedrático de Universidad

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2025

Trabajo Fin de Grado: Prototipo de sistema electrónico portátil de vigilancia y supervisión

Autor: Gonzalo Jiménez Alonso
Tutor: Federico Barrero García

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Quiero agradecer a todas las personas que me han ayudado a poder completar este TFG, tanto de forma directa como indirectamente, y a todos los que han estado conmigo en estos años del grado.

En primer lugar, gracias a mis compañeros y amigos, con los que hemos compartido tanto, bueno y malo, risas y frustraciones, pero que han hecho que el tiempo en la escuela fuera más fácil, en especial a Pedro, por estar siempre ahí y apoyarnos cuando el otro lo necesitaba.

En segundo lugar gracias a mis profesores, a todos los que han conseguido transmitirme las ganas de conocimiento en la ingeniería y me han enseñado que es mucho más que simplemente, resolver problemas, sobre todo, a mis profesores de las asignaturas de electrónica, los que han conseguido descubrirme esta pasión gracias a su amor y dedicación a lo que enseñaban.

También dar las gracias a mi tutor, Federico, por haber estado disponible cuando lo necesitaba, fuese para hablar del TFG, o simplemente sentarme en el despacho y charlar, demostrar que crees en un alumno y motivarle es algo que pocos profesores consiguen hacer, y tú lo has conseguido. Quiero agradecer también a Esteban, que con su atención y su disposición para ayudarme se convirtió en un apoyo más durante todo este proceso.

No puedo olvidarme de mis padres y mi hermana, Pilar, los que me han aguantado hasta altas horas de la noche con mis quejas, mis frustraciones, mis preguntas, y me han apoyado siempre, pase lo que pase. Habéis estado ahí, cuando me caía, para ayudarme a levantarme, y celebrando mis logros conmigo como si fueran vuestros. Pilar, sin tu consejo y tu apoyo, no hubiera llegado donde estoy, gracias.

Por ultimo agradecer a Dida, me has dado fuerzas diariamente para poder continuar, buscando siempre lo mejor y confiando en mí. A pesar de no entender mucho, cada pequeño avance, preguntabas, te interesabas, e intentabas ayudar en lo que fuera posible. *Didocas, sem ti não tinha conseguido, obrigado de coração.*

Gonzalo Jiménez Alonso

Sevilla, 2025

Resumen

La seguridad ha sido una preocupación constante a lo largo de la historia, y para el hombre, proteger sus bienes, se convierte en una prioridad. Con la inclusión de la electrónica en nuestro día a día, surge la necesidad de desarrollar también sistemas de seguridad más eficientes y accesibles.

El siguiente proyecto propone el diseño y prototipado de un sistema electrónico de seguridad portátil, basado en la lectura de varios sensores y en la activación en caso de robo de ciertos actuadores. Nuestro sistema estará además conectado por Bluetooth a un dispositivo móvil, desde el que podremos controlarlo y observar cambios.

Se buscará implementar este sistema en un procesador de señales digitales de tamaño reducido con el objetivo de hacer un sistema "wearable" de tamaño reducido que pueda ser adaptado a cualquier objeto.

Por último, contará con un modo alternativo de confort en el que se monitorizará la temperatura de la sala y se buscará reducirla mediante la activación de un ventilador. Su objetivo será el de añadirle funcionalidad cuando no esté activo el modo "anti-robo".

Abstract

Security has been a constant concern throughout history, and for mankind, protecting his assets, becomes a priority. With the inclusion of electronics in our daily lives, emerges the need to develop more efficient and accessible security systems.

The following project proposes the design and prototyping of a portable electronic security system, based on the reading of several sensors and the activation of certain actuators in case of theft. Our system will also be connected by Bluetooth connection to a mobile device, from which we will be able to control it and observe changes.

We will seek to implement this system in a small digital signal processor with the aim of making a wearable system of reduced size that can be adapted to any object.

Finally, it will have an alternative comfort mode in which the temperature of the room will be monitored and reduced by activating a fan. Its aim will be to add functionality when the ‘anti-theft’ mode is not active.

Índice Abreviado

<i>Resumen</i>	III
<i>Abstract</i>	V
<i>Índice Abreviado</i>	VII
<i>Notación</i>	XIII
1 Introducción	1
1.1 Contexto y motivación	1
1.2 Ámbito	2
1.3 Objetivo	2
1.4 Alcance	3
1.5 Aplicaciones	3
1.6 Estructura del proyecto y metodología empleada	4
2 Fundamentos Teóricos	7
2.1 Conceptos básicos sobre seguridad electrónica	7
2.2 Sistemas embebidos	8
2.3 Sensores y actuadores	9
2.4 Conexión inalámbrica	10
2.5 Criterios en el diseño electrónico	11
2.6 IoT y dispositivos wearables	12
3 Estado actual de los sistemas de seguridad	15
3.1 Breve historia de los sistemas de seguridad	15
3.2 Sistemas de seguridad modernos	16
3.3 Seguridad aplicada a dispositivos electrónicos portátiles	18
4 Evaluación de necesidades del sistema	21
4.1 Elección de componentes	21
4.2 Esquema de conexiones	29
4.3 Funcionamiento del sistema	30
4.4 Implementación	31
4.5 Selección y evaluación de sensores alternativos	40
4.6 Conclusión de necesidades del sistema	46

5 Desarrollo del prototipo del sistema wearable	47
5.1 Selección de la placa de desarrollo	47
5.2 Elección e integración de nuevos componentes	51
5.3 Elección del software de desarrollo	52
5.4 Diagrama de pines ESP32-C3	52
5.5 Funcionamiento del sistema	53
5.6 Implementación	55
5.7 Diseño de la PCB	61
5.8 Montaje y diseño físico de la maqueta	63
5.9 Pruebas de funcionamiento e integración	64
6 Conclusiones y futuro trabajo	67
Apéndice A Tabla de componentes utilizados	69
Apéndice B Código en Code Composer Studio	71
Apéndice C Código en Arduino IDE	79
<i>Índice de Figuras</i>	87
<i>Índice de Tablas</i>	89
<i>Índice de Códigos</i>	91
<i>Bibliografía</i>	93

Índice

<i>Resumen</i>	III
<i>Abstract</i>	V
<i>Índice Abreviado</i>	VII
<i>Notación</i>	XIII
1 Introducción	1
1.1 Contexto y motivación	1
1.2 Ámbito	2
1.3 Objetivo	2
1.4 Alcance	3
1.5 Aplicaciones	3
1.5.1 Sistema de seguridad portátil	3
1.5.2 Sistema de seguridad avanzado	3
1.5.3 Dispositivo docente	3
1.5.4 Sistema de monitorización clínica	4
1.5.5 Sistema de geolocalización	4
1.6 Estructura del proyecto y metodología empleada	4
2 Fundamentos Teóricos	7
2.1 Conceptos básicos sobre seguridad electrónica	7
2.2 Sistemas embebidos	8
2.3 Sensores y actuadores	9
2.3.1 Sensores	9
Sensores analógicos	9
Sensores digitales	9
2.3.2 Actuadores	10
2.4 Conexión inalámbrica	10
2.5 Criterios en el diseño electrónico	11
2.6 IoT y dispositivos wearables	12
2.6.1 Accesorios	13
Dispositivos de muñeca	13
Dispositivos montados en la cabeza (head-mounted)	13
Otros accesorios	13
2.6.2 Ropa electrónica (e-textiles)	13
Prendas inteligentes	13

	Dispositivos para manos y pies (foot/hand-worns)	13
2.6.3	Parches electrónicos (e-patches)	13
3	Estado actual de los sistemas de seguridad	15
3.1	Breve historia de los sistemas de seguridad	15
3.1.1	Primeros sistemas mecánicos y aparición de la vigilancia estructurada	15
3.1.2	Transición hacia sistemas eléctricos y electrónicos (siglo XX)	16
3.2	Sistemas de seguridad modernos	16
3.2.1	Alarmas electrónicas convencionales	17
3.2.2	Inteligencia artificial y análisis predictivo en seguridad	17
3.2.3	Limitaciones y retos actuales	17
3.3	Seguridad aplicada a dispositivos electrónicos portátiles	18
3.3.1	Necesidad creciente de protección física de equipos personales	18
3.3.2	Soluciones electrónicas: sensores de movimiento, GPS, alarmas acústicas	18
3.3.3	Retos en la portabilidad, consumo y eficiencia	19
4	Evaluación de necesidades del sistema	21
4.1	Elección de componentes	21
4.1.1	Microcontrolador	21
4.1.2	LM35 (Sensor de temperatura)	22
4.1.3	Grayscale Sensor (Sensor de proximidad)	22
4.1.4	Shake Sensor (Sensor de vibración)	23
	Interrupción Externa (XINT)	24
	eCAP	25
	ADC	26
4.1.5	Motor	27
4.1.6	Altavoz	27
4.1.7	Vibrador	28
4.1.8	LED	28
4.1.9	Módulo de comunicación (SCI) - Terminal de Comunicación Serie	29
4.1.10	Alimentación	29
4.2	Esquema de conexiones	29
4.3	Funcionamiento del sistema	30
4.3.1	Modo Confort	30
4.3.2	Modo Antirrobo	30
4.3.3	Diagrama de flujo	31
4.4	Implementación	31
4.4.1	ADC	32
4.4.2	PWM	34
	Time Base	34
	Counter Compare	35
	Action Qualifier	35
4.4.3	SCI	36
	Bluetooth	37
	Programación del módulo	38
	Bluetooth Electronics	39
4.5	Selección y evaluación de sensores alternativos	40
4.5.1	Motivos de la búsqueda	40

4.5.2	Primera modificación: Acelerómetro	40
4.5.3	Segunda modificación: Sensor de infrarrojos	42
4.5.4	Última modificación: Sensor de ultrasonido	43
4.5.5	Diagrama de flujo	45
4.6	Conclusión de necesidades del sistema	46
5	Desarrollo del prototipo del sistema wearable	47
5.1	Selección de la placa de desarrollo	47
5.1.1	Raspberry Pi Pico W	47
5.1.2	Arduino LilyPad	48
5.1.3	Adafruit FLORA	49
5.1.4	SeeedStudio XIAO ESP32-C3	49
5.1.5	Comparación entre las diferentes opciones y elección final	50
5.2	Elección e integración de nuevos componentes	51
	Módulo Bluetooth HC-05	51
	Pantalla LCD	51
5.3	Elección del software de desarrollo	52
5.4	Diagrama de pines ESP32-C3	52
5.5	Funcionamiento del sistema	53
	Modo Confort	53
	Modo Antirrobo	54
5.6	Implementación	55
5.6.1	ADC	55
5.6.2	PWM	56
5.6.3	Bluetooth Low Energy	57
	Radio Bluetooth LE y capa física (PHY)	57
	Controlador de Enlace (Link layer controller)	57
	Procedimiento de comunicación	57
5.6.4	LCD	59
5.6.5	HTML	60
5.7	Diseño de la PCB	61
5.8	Montaje y diseño físico de la maqueta	63
5.9	Pruebas de funcionamiento e integración	64
6	Conclusiones y futuro trabajo	67
Apéndice A	Tabla de componentes utilizados	69
Apéndice B	Código en Code Composer Studio	71
Apéndice C	Código en Arduino IDE	79
<i>Índice de Figuras</i>		87
<i>Índice de Tablas</i>		89
<i>Índice de Códigos</i>		91
<i>Bibliografía</i>		93

Notación

mV	milivoltios
V	voltios
°C	grados Celsius
kB	kilobytes
MB	megabytes
MHz	megahercios
kHz	kilohercios
dB	decibelios
ms	milisegundos
GPIO	Entrada/Salida de propósito general (General Purpose Input/Output)
ADC	Convertidor Analógico-Digital (Analog-to-Digital Converter)
PWM	Modulación por Anchura de Pulso (Pulse Width Modulation)
SCI	Interfaz de Comunicación Serie (Serial Communication Interface)
SPI	Interfaz Periférica Serial (Serial Peripheral Interface)
I2C	Interfaz de Circuito Integrado (Inter-Integrated Circuit)
UART	Receptor-Transmisor Asíncrono Universal (Universal Asynchronous Receiver-Transmitter)
BLE	Bluetooth Low Energy
Wi-Fi	Conectividad inalámbrica (Wireless Fidelity)
PCB	Placa de circuito impreso (Printed Circuit Board)
LDO	Regulador de baja caída de tensión (Low Dropout Regulator)
ADC1, ADC2	Canales del convertidor analógico-digital del ESP32-C3
ESP32-C3	Microcontrolador con arquitectura RISC-V y conectividad Wi-Fi/Bluetooth
RP2040	Microcontrolador desarrollado por Raspberry Pi
HC-05	Módulo de comunicación Bluetooth clásico
HC-SR04	Sensor ultrasónico de distancia
LCD1602	Pantalla de cristal líquido de 16x2 caracteres
NeoPixel	LED RGB direccionable
SDA	Línea de datos serie para comunicación I2C
SCL	Línea de reloj para comunicación I2C
PCF8574T	Chip convertidor serie-paralelo para el módulo I2C del LCD
0x27, 0x3F	Direcciones I2C utilizadas para la pantalla LCD

Advertising	Proceso de emisión periódica de paquetes de información en BLE
Scanning	Proceso de exploración de dispositivos BLE cercanos
Link Layer Controller	Controlador de capa de enlace en BLE
PHY	Capa física de la comunicación inalámbrica
UUID	Identificador único universal para servicios y características BLE

1 Introducción

En un mundo cada vez más interconectado y dependiente de la tecnología, el Internet de las cosas o "IoT" ha adquirido un papel fundamental en numerosos ámbitos. El avance de la electrónica, reflejado en una mayor rapidez de respuesta de los sistemas y la reducción del tamaño, permite su inclusión en infinidad de campos, entre ellos, la monitorización y, más en concreto, la vigilancia y supervisión de bienes o personas. La idea principal de este proyecto consiste en el diseño y prototipado de un sistema electrónico portátil de vigilancia y supervisión.



Figura 1.1 Internet de las cosas o IoT.

1.1 Contexto y motivación

En este proyecto, se busca crear un sistema de vigilancia portátil, que integre tanto sensores de precisión, como diferentes actuadores, con la capacidad de detectar hurtos en caso de que el usuario active este modo. De la misma forma, se añade un modo adicional, que cuenta con un sensor de temperatura y un pequeño ventilador con el objetivo de tener un sistema de confort cuando el usuario está haciendo uso del objeto y no necesita ser vigilado. Todo esto podrá ser controlado por un dispositivo externo mediante una conexión inalámbrica del estilo Bluetooth.

El código y el prototipo inicial serán desarrollados en un procesador de señales digitales (de ahora en adelante, *DSP*), en concreto el TMS320F28335 de Texas Instruments, programado en lenguaje C, mediante el entorno Code Composer Studio. Una vez se encuentre programado, y comprobemos que el funcionamiento se corresponde con el esperado, obtendremos las especificaciones mínimas que queremos que tenga nuestro sistema, haciendo un estudio del modelo más eficiente, y al mismo tiempo óptimo en cuanto a tamaño, buscando incorporar este código en un dispositivo "*wearable*" (vestible en español).

La principal motivación para este proyecto radica en el interés por el estudio de los sistemas DSP, así como la incorporación de diversos sensores y actuadores, y el manejo y monitorización de nuestro sistema a través de una conexión inalámbrica. Las posibilidades y diferentes proyectos que se pueden realizar con ellos son ilimitados, y forman parte de la proyección futura de la tecnología y la ingeniería, por lo tanto, su manejo y entendimiento son considerados fundamentales.

Otra de las motivaciones está relacionada con el auge de los dispositivos inalámbricos, y en especial con conexión Bluetooth. Con ellos surge la necesidad de adaptación de los sistemas de seguridad a las nuevas tecnologías, con la intención de hacer su uso más accesible y simple para todos los usuarios, sin necesidad de conocimiento en programación, o incluso sistemas integrados.

Asimismo, la incorporación de la electrónica "wearable" al mercado y en los dispositivos de uso cotidiano, hace imprescindible su investigación, y estudio de un posible prototipado, ya que nos permite una amplia gama de posibilidades, desde la monitorización de parámetros sanitarios, hasta "artefactos" o, como son más conocidos en inglés, "gadgets" para multitud de usos.

Por último, la motivación más importante ha sido el interés personal de profundizar en una materia interesante y con gran proyección de futuro. Al mismo tiempo, representa un desafío que no solo busca aplicar conocimientos adquiridos en el grado, sino también fomentar el desarrollo de habilidades y pensamiento ingenieril mediante el planteamiento de nuevos retos y desafíos.

1.2 Ámbito

Este proyecto abarca los campos de la electrónica, análogica y digital, y la programación de sistemas integrados mediante software. Igualmente, se incluye dentro de la parte de ingeniería de control, con la toma de decisiones y monitorización del estado de un sistema en tiempo real.

1.3 Objetivo

El objetivo principal de este proyecto, es el diseño y programación de un prototipo de un sistema electrónico de vigilancia y supervisión portátil. El funcionamiento del sistema puede ser dividido en sus dos modos principales, *Anti-Robo* y *Confort*. El primero de ellos, monitoreará varios sensores, tanto lumínicos, como de vibración, con el objetivo de detectar movimientos indeseados. En caso de ser detectados, activará varios actuadores, tanto lumínicos como sonoros, para poder avisar al usuario de que su dispositivo está siendo manipulado y/o robado.

Por otro lado, con el objetivo de diseñar un sistema lo más completo posible, se añadirá un modo de confort, con el fin de dotar de una finalidad alternativa cuando el sistema de seguridad no se encuentre activado, siempre a elección del usuario. Este modo, contará con una medición de la temperatura ambiental en tiempo real, y un ventilador, para que, en caso de superarse la temperatura que el usuario considere, reducirla con el fin de mejorar la comodidad.

Al mismo tiempo, una vez programado en nuestro DSP, en concreto el TMS320F28335 de Texas Instruments, se diseñará y construirá un prototipo funcional, ahora sustentado en un dispositivo basado en CPU de un tamaño significativamente menor, considerado como "wearable".

La principal dificultad de nuestro proyecto, radica en la elección de este último dispositivo wearable. Tomando como referencia el número de pines de entrada y salida, y las diferentes características que queremos que nuestro sistema tenga, se buscará reducir el tamaño al máximo posible, intentando no reducir el rendimiento, contando con una frecuencia de funcionamiento relativamente elevada, y una memoria suficiente para nuestro programa.

Basado en la idea de un control de nuestro sistema de forma inalámbrica, se buscará también que nuestro sistema cuente con conectividad Bluetooth y/o WiFi, para poder ser monitorizado y controlado a través de un dispositivo móvil externo.

1.4 Alcance

El alcance de este proyecto se centra en el diseño y prototipado del ya mencionado sistema de seguridad. Para ello, nos centraremos en la programación de sistemas integrados, en el entorno Code Composer Studio, junto con el diseño del prototipo.

El enfoque principal es el de la construcción de la lógica y el funcionamiento de los sistemas de seguridad basados en el campo de la electrónica, así como su adaptación a la tecnología "wearable".

- Programación del DSP TMS320F28335 en el entorno Code Composer Studio.
- Lectura de sensores tanto digitales como analógicos y manejo de datos recibidos.
- Actuación en consecuencia de los datos obtenidos de los sensores en diferentes componentes, tanto para interactuar con el ambiente como con el usuario.
- Manejo de la conectividad Bluetooth, bien por un módulo independiente, o por la conectividad propia integrada en el dispositivo.
- Integración y adaptación del código en un wearable de tamaño reducido, modificando también el entorno de programación utilizado en caso de que sea necesario.

Dentro del alcance de nuestro proyecto, no se encuentra la inclusión de sensores de alta complejidad como reconocimiento facial o sensores capacitivos, ni tampoco se contemplarán las hipótesis de funcionamiento en condiciones fuera de las normales para una sala. Se buscará fijar las bases de un sistema de vigilancia en un dispositivo portátil, con la posibilidad de ampliación o profundización en el ámbito deseado en un futuro.

1.5 Aplicaciones

Las aplicaciones de nuestro sistema son innumerables, tanto del sistema de vigilancia como del dispositivo wearable; sin embargo, intentaremos describir las principales aplicaciones de ambos sistemas combinados.

1.5.1 Sistema de seguridad portátil

La principal aplicación, y para la cual está pensado este proyecto, es la de poder incluir este sistema en cualquier objeto con la finalidad de poder monitorizarlo y asegurar su seguridad. De esta forma, puede ser añadido a bienes personales, como una mochila, una maleta de viaje o incluso dispositivos electrónicos. Debido a su tamaño, incluso con el DSP "wearable", no se tendrá en cuenta su uso en dispositivos móviles, sino algunos de más tamaño, como ordenadores portátiles o de sobremesa, o incluso tablets.

1.5.2 Sistema de seguridad avanzado

Muy relacionado con el apartado anterior, puede ser adaptado para funcionar como un sistema de seguridad avanzado, pudiendo ser integrado con la domótica del hogar o de instalaciones en general. Ampliando el sistema con más sensores, la interfaz Bluetooth permite el manejo de este sistema para todo tipo de usuarios.

1.5.3 Dispositivo docente

Debido a la estructura del sistema, se combina la lectura de sensores, con la activación y desactivación de actuadores, así como módulos típicos de la electrónica de sistemas integrados, tales como PWM, SCI...etc, pudiendo ser una buena opción para el desarrollo de habilidades y comprensión de estos conceptos en estudiantes que se encuentren en proceso de aprendizaje.

1.5.4 Sistema de monitorización clínica

Con la debida adaptación del modo "confort" de nuestro sistema, así como la inclusión de sensores biométricos, una de las aplicaciones posibles, sería su utilización como un sistema de monitorización clínico, ya sea para pacientes, o usuarios saludables, emulando las funciones de los relojes inteligentes ahora presentes en el mercado.

1.5.5 Sistema de geolocalización

En este caso, bastaría con añadir un sensor de posición, para poder obtener un sistema de geolocalización en tiempo real, ya sea para personas o incluso animales, suponiendo la descripción de una zona a través de nuestro dispositivo móvil, y alertando en caso de sobrepasar esos límites.

1.6 Estructura del proyecto y metodología empleada

En la elaboración de nuestro proyecto, se ha empleado, un desarrollo lógico, implementando en primer lugar nuestro sistema en el DSP TMS320F28335, y posteriormente, en el dispositivo wearable, seguido de la construcción el prototipo funcional.

1. Distribución del trabajo y planteamiento inicial: En esta fase se distribuyeron las tareas a realizar y se delimitó el alcance de nuestro proyecto.
2. Búsqueda de componentes: Ya delimitado nuestro proyecto, y distribuido el trabajo, el primer paso, antes de empezar con la programación, consistió en la elección de los componentes, tanto sensores como actuadores, a través de portales web, tiendas de componentes electrónicos, y material disponible en el departamento de Ingeniería Electrónica.
3. Programación en el DSP TMS320F28335: Una vez sabemos los componentes con los que podemos contar, y el enfoque del proyecto, llegó la hora de programar. Para ello usamos el entorno Code Composer Studio (a partir de ahora CCS), en el cual se diseñó el funcionamiento del sistema con todos los sensores y actuadores.
4. Adición del módulo Bluetooth: Con el programa ya desarrollado y funcional, y pudiendo cambiar entre modos a través de UART, teniendo que estar conectado por cable al ordenador, se añadió un módulo Bluetooth, y se diseñó la interfaz del usuario en la aplicación móvil Bluetooth Electronics.
5. Elección del dispositivo: El siguiente paso, fue la elección de nuestro dispositivo basado en CPU portátil. A raíz del código elaborado en CCS, pudimos realizar una evaluación de las necesidades de nuestro sistema, para la búsqueda; se valoraron los aspectos de tamaño, precio y eficiencia sobre todo.
6. Adaptación del código: Cuando ya teníamos elegido nuestro DSP, la próxima tarea, fue la de adaptar nuestro código, del entorno CCS en lenguaje de programación C, a otro entorno de programación e incluso a otro lenguaje en caso de que fuera necesario respondiendo a las necesidades de nuestro sistema.
7. Diseño del prototipo: Para terminar, una vez comprobado el funcionamiento de nuestro sistema, en varias situaciones, se procedió al diseño y construcción de nuestro prototipo, buscando la simplicidad y manteniendo el tamaño reducido del sistema.
8. Evaluación del sistema completo: El último paso, fue el de evaluar nuestro prototipo, comprobando de forma individual todos los sensores y actuadores, el intercambio entre modos, y la interfaz del usuario.

Cabe destacar que, por comodidad y eficiencia de tiempo, algunas de las etapas han sido realizadas en paralelo, tales como la búsqueda de componentes y la del nuevo dispositivo, o la programación tanto en CCS, como la adaptación del mismo. Siguiendo siempre una metodología ordenada y lógica para comprobar el funcionamiento del sistema paso por paso.

2 Fundamentos Teóricos

Los científicos estudian el mundo como es, los ingenieros, crean el mundo que nunca ha existido

THEODORE VON KÁRMÁN

Antes de abordar el diseño y desarrollo de cualquier sistema, es imprescindible cimentar una base teórica sólida que fundamente las decisiones técnicas que serán tomadas a lo largo del proyecto.

En el caso de un sistema electrónico de seguridad portátil, deberemos comentar los fundamentos teóricos relacionados con los sistemas de seguridad electrónicos, el funcionamiento de los sistemas embebidos, la interacción entre sensores y actuadores, las tecnologías de comunicación, y los criterios a tener en cuenta a la hora del diseño.

Este capítulo tiene como objetivo presentar esos conceptos fundamentales que sustentan el diseño y funcionamiento del sistema que se propone. No únicamente una revisión técnica, sino también dotar de un marco conceptual que nos permite comprender las limitaciones, posibilidades, y futuras decisiones clave que se plantearán en fases posteriores.

El fin, por tanto, es justificar técnicamente las posibles elecciones de componentes, estructuras y tecnologías y asegurar que el sistema cumple su función y además lo hace de forma eficiente, fiable y escalable.

2.1 Conceptos básicos sobre seguridad electrónica

La seguridad electrónica, consiste en una rama dentro de la ingeniería que se encarga del diseño e implementación de sistemas, capaces de proteger bienes y/o personas. La definición propia de un sistema de seguridad electrónica es un medio o método por el cual se asegura algo a través de un sistema de componentes y dispositivos que funcionan y se comunican entre sí [1].

Los sistemas de seguridad, tradicionalmente fijos, han evolucionado hacia sistemas portátiles y adaptables, gracias al avance de las tecnologías y más en específico del campo de la electrónica, permitiendo una mayor versatilidad de aplicaciones.

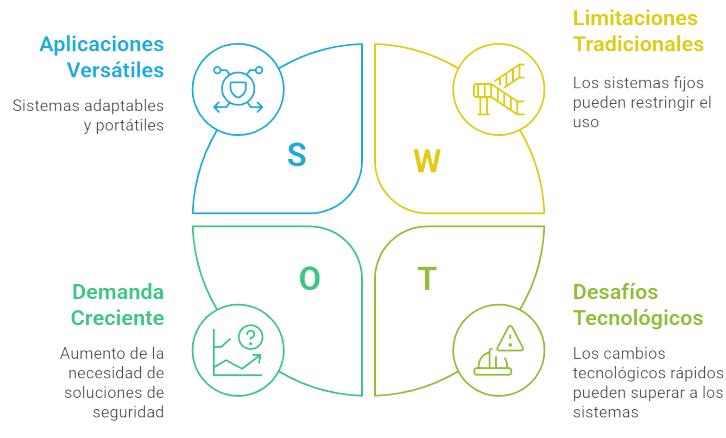
Existen diversos tipos de sistemas de seguridad desde sistemas de vigilancia CCTV, alarmas y servicios de intervención inmediata, hasta control de accesos.

También podemos clasificar estos sistemas según la tecnología usada:

- Sistemas perimetrales: orientados a la detección de intrusos en puntos de acceso.
- Sistemas volumétricos: monitorización de movimientos o presencia en un espacio determinado.

- Sistemas portátiles: permiten proteger objetos personales al estar integrados en dispositivos de un tamaño reducido.
- Sistemas domóticos: integrados en entornos de hogar o de la industria.

El desarrollo de estos sistemas electrónicos requiere del estudio de conceptos como el ambiente en el que se encontrarán presentes (lectura de temperatura, vibraciones, movimientos), la gestión de estos datos por un microcontrolador, y la posterior respuesta de sensores y actuadores



Made with Napkin

Figura 2.1 Análisis de sistemas de seguridad electrónica.

2.2 Sistemas embebidos

Un sistema embebido o integrado, es un dispositivo informático, integrado en un dispositivo que no es un ordenador, y destinado a realizar tareas informáticas específicas. Estas tareas pueden ir desde la adquisición o transferencia de datos sobre el trabajo realizado por el dispositivo madre, hasta la visualización de información o el control de dispositivos madre. [2]

Estos sistemas se encuentran compuestos de varios elementos, como un microprocesador, una memoria RAM y otra ROM, convertidores analógicos-digitales (y los inversos), y pines de propósito general (I/O Interface), entre muchos otros. [3]

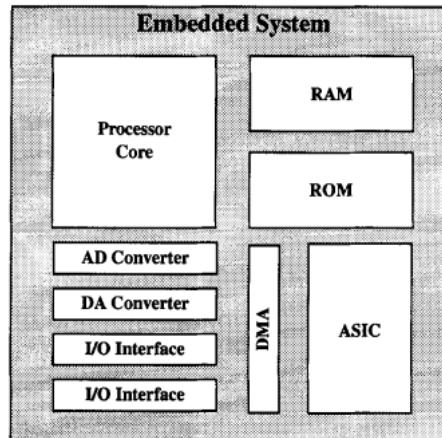


Figura 2.2 Esquema típico de un sistema integrado.

Los sistemas integrados, no son un componente nuevo, hay muchos ejemplos de este tipo de sistemas en nuestro uso diario desde hace varios años, por ejemplo los dispositivos MP3 (Reproductores de música), PDAs (Asistente Personal Digital), o incluso muchos juguetes inteligentes.

Sin embargo, con el avance de la electrónica, su uso se ha extendido sin precedentes, encontrándose actualmente en todos los electrodomésticos inteligentes, sistemas de control en los automóviles, equipos médicos e incluso sistemas de seguridad.

El uso de estos sistemas en soluciones de vigilancia permite:

- Reducir el tamaño del sistema físico.
- Minimizar el consumo energético y así aumentar la autonomía.
- Facilitar la integración con sensores y actuadores específicos.
- Permitir el procesamiento local de los datos sin necesidad de una conexión a la red.

2.3 Sensores y actuadores

Para comprender el uso de sensores y actuadores, y poder hacer una buena elección de ellos, será necesario entender que son y los diferentes tipos que existen de cada uno de ellos.

2.3.1 Sensores

Un sensor es un dispositivo que traduce una magnitud física en una señal electrónica. Su clasificación puede hacerse de diferentes formas, según su uso, su naturaleza o sus componentes, sin embargo todos pueden dividirse en dos tipos, analógicos y digitales.

Sensores analógicos

Un sensor analógico es un dispositivo que mide una magnitud o fenómeno físico y proporciona una señal de salida directamente proporcional a la magnitud medida [4]. Las principales características de estos sensores incluyen:

- Salida continua: proporcionan una salida que varía de forma continua en relación a los cambios en el parámetro medido.
- Precisión reducida: estos sensores tienen limitación en su precisión en comparación con los sensores digitales debido al efecto del ruido o las condiciones ambientales.
- Necesidad de conversión: el principal inconveniente de estos sensores es la necesidad de convertir estos valores debido a la incompatibilidad del manejo de datos analógicos por parte del microprocesador en cuestión.

Estos sensores son ampliamente utilizados, y son especialmente importantes en aplicaciones donde es necesario una monitorización continua y en tiempo real principalmente en los campos de la automatización industrial, la monitorización ambiental y el estudio científico.

Sensores digitales

Por otro lado, los sensores digitales, son dispositivos, que mide una cantidad física (temperatura, humedad, luz, presión) y cuantifica esa medida en una digital. Es decir, obtiene datos físicos analógicos y los transforma en valores discretos (0s y 1s). Sus principales características son las siguientes:

- Salida discreta: estos sensores devuelven datos discretos, representados de forma general en valores en código binario, lo que facilita su lectura y procesamiento por parte de sistemas digitales y microcontroladores.

- Alto nivel de precisión: son conocidos por su alta precisión en comparación con los sensores analógicos, por lo que suelen ser usados en sistemas donde es crucial una alta sensibilidad.
- Compatibilidad con microcontroladores: debido a la salida de los datos en valores digitales y binarios, facilita la lectura de los datos por parte del microcontrolador siendo ésta directa sin necesidad de convertidores.

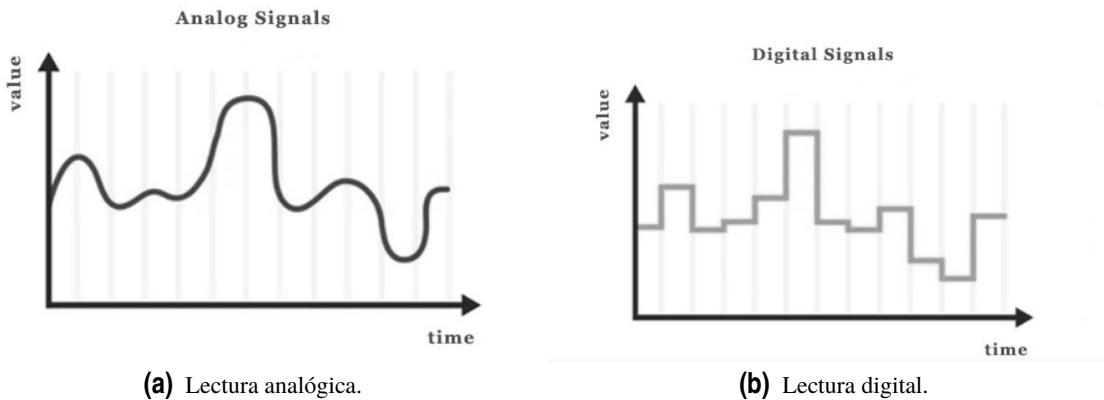


Figura 2.3 Sensor analógico VS digital.

Los sensores digitales cuentan con muchos beneficios lo que justifica la expansión e inclusión de estos sensores en muchos sistemas actuales.

2.3.2 Actuadores

Los actuadores tienen el efecto contrario a los sensores, son dispositivos que traduce una señal electrónica en una magnitud física modificando el ambiente. La forma de interactuar con estos actuadores varía dependiendo del tipo que sea, desde encender y apagar (1s y 0s), hasta el control por modulación de ancho de pulso.

2.4 Conexión inalámbrica

Una conexión inalámbrica se define como una red informática que utiliza conexiones de radiofrecuencia (RF) entre todos los puntos de la red. Son una solución popular para hogares, empresas y redes de telecomunicaciones [5].

Inicialmente, cuando hablamos de conexión inalámbrica pensamos simplemente en el tipo WI-FI, sin embargo hay muchos más en los que entraremos ahora en detalle.

Antes de ello, resulta interesante comentar la diferencia entre la conexión cableada y la inalámbrica. En términos simples, una red inalámbrica, mantiene a los dispositivos conectados a una red permitiendo desplazarse libremente sin la limitación física de los cables. Sin embargo existen diferencias tecnológicas entre estos dos tipos de redes.

La mayor parte de las redes cableadas actuales son "full duplex" [6], lo que significa que tienen la capacidad de enviar y recibir paquetes en ambas direcciones de forma simultánea, además gran parte de las redes cableadas tienen un cable específico que llega a cada dispositivo del usuario final.

Por otro lado, en las redes inalámbricas, como el WI-FI, el medio (la radiofrecuencia utilizada), es un recurso compartido, además de para otros usuarios, para otras tecnologías (p.ej. microondas). Por ello cuenta con varios inconvenientes:

1. Las redes inalámbricas no pueden enviar y recibir al mismo tiempo, lo llamado "half duplex"[6].
2. Todos los usuarios que comparten el mismo espacio, deben tomar turnos para hablar.

3. Todo el que sea parte de esta red puede recibir el tráfico de datos.

Dentro de estas redes inalámbricas existen 4 tipos dependiendo de su uso y a qué tipo de dispositivos van dirigidos, son los que siguen [7]:

- **LAN**(Red de Área Local): red que opera en un único espacio físico. Conecta dispositivos como ordenadores, impresoras y sistemas de almacenamiento utilizando componentes como switches, routers y cables Ethernet. El ejemplo más común es el WI-FI.
- **PAN**(Red de Área Personal): red centrada en los dispositivos de una única persona en un espacio físico concreto. Incluye ordenadores, dispositivos móviles, consolas de videojuegos y otros periféricos. El Bluetooth es el tipo más común de PAN inalámbrica.
- **MAN**(Red de Área Metropolitana): Se diferencia con la LAN en que cubre una zona más amplia como una ciudad o un campus y se utiliza para conectar edificios y gestionar sistemas como climatización y electricidad en ese área.
- **WAN**(Red de Área Amplia): red que abarca grandes extensiones como ciudades países o incluso continentes. Las redes móviles pueden ser consideradas como ejemplos comunes de estas conexiones.

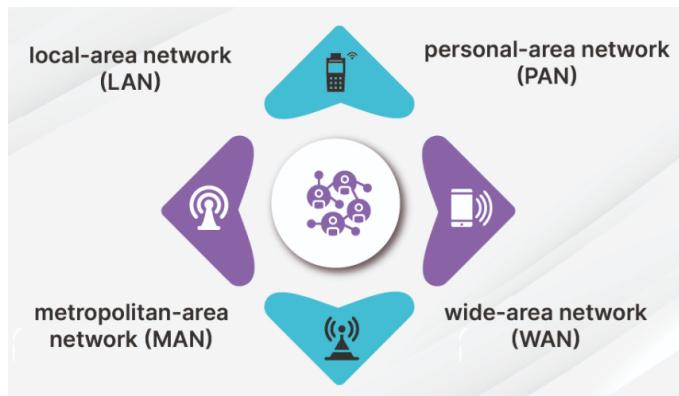


Figura 2.4 Tipos de redes inalámbricas.

2.5 Criterios en el diseño electrónico

De forma previa a la construcción del prototipo, deberemos destacar los elementos a tener en cuenta a la hora del diseño electrónico. El objetivo es buscar los criterios técnicos que van a influir en la viabilidad y al mismo tiempo eficiencia de nuestro diseño. Estos criterios serán los siguientes [8]:

- **Tamaño deseado**: el primer elemento que tendremos que definir será el tamaño de nuestro sistema. Dependiendo de nuestra elección, tendremos más o menos limitaciones, en nuestro caso, al ser un sistema portátil, este será el criterio principal a tener en cuenta a la hora del diseño.
- **Requisitos de voltaje/corriente/potencia**: el segundo punto importante a estudiar, serán los requisitos de nuestro sistema, contando con los componentes que tenemos, cual es la corriente y/o voltaje que necesitan. Además necesitaremos saber que potencia pueden dar, sobre todo para poder controlar el consumo energético, algo fundamental en la autonomía de un sistema portátil.
- **Pines de entrada/salida**: una vez definido los componentes que tendrá nuestro sistema, deberemos diseñar un sistema electrónico que cumpla con el número de pines tanto de entrada

como de salida que necesitamos, así como módulos especiales que necesitemos como ADC (Convertidor Analógico-Digital), o PWM (Modulación por anchura de pulso).

- **Escalabilidad:** sin perder de vista el objetivo de este proyecto, queremos construir un prototipo, por lo que será esencial un nivel de escalabilidad que nos permita futuras ampliaciones o mejoras del sistema.

2.6 IoT y dispositivos wearables

El concepto de Internet de las Cosas (IoT) se refiere a la conexión de varios dispositivos físicos y objetos a través del mundo mediante internet. Consiste en una red de objetos físicos como instrumentos, dispositivos, vehículos, edificios y otros objetos dotados de componentes electrónicos que les permiten recoger e intercambiar datos entre ellos [9].

La evolución de internet puede clasificarse en 5 etapas principales:

1. El Internet de los Documentos: librerías electrónicas y páginas web.
2. El Internet del Comercio: comercio electrónico, banca online, y sitios web de compraventa de acciones.
3. El Internet de las aplicaciones: web 2.0.
4. El Internet de las Personas: llegada de las redes sociales.
5. El Internet de las Cosas: Dispositivos y máquinas conectadas entre sí

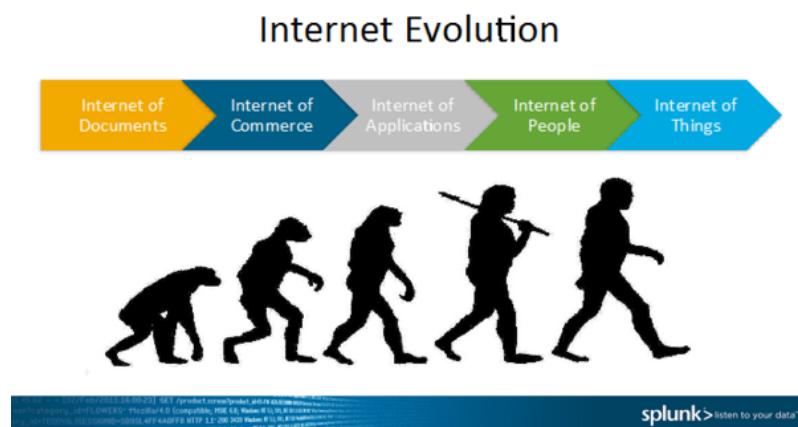


Figura 2.5 Evolución de Internet.

El término IoT, describe la próxima generación de Internet, en la que se podrá acceder a los objetos físicos e identificarlos a través de Internet. La definición, varía en función de las diferentes tecnologías de aplicación, sin embargo, lo fundamental del IoT, implica que los objetos, puedan identificarse de forma única en las representaciones virtuales. Dentro de una misma red, todas las cosas pueden intercambiar datos y si es necesario, procesarlos según esquemas predefinidos.

Por otro lado, los dispositivos *wearables* (dispositivos portátiles), representan las diferentes tecnologías que un usuario puede llevar en el cuerpo. A diferencia de los móviles, están diseñados para ser utilizados de forma constante durante todo el día. Su objetivo principal es el de mejorar la calidad de vida mediante la monitorización constante del usuario y facilitar la interacción del usuario con el entorno.

Estos dispositivos *wearables*, permiten ejecutar tareas como navegación, pagos electrónicos, seguimientos de salud física, análisis de la actividad deportiva... El crecimiento de esta tecnología,

viene acompañado del auge de la inversión en investigación y desarrollo así como en la producción y comercialización de productos de este tipo.

Según su funcionalidad y uso pueden clasificarse en tres grandes categorías principales.

2.6.1 Accesorios

Se incluyen en esta sección los *wearables* de uso externo y que no forman parte de la ropa del usuario. Actualmente son los mas habituales en el mercado y se dividen en varias subcategorías:

Dispositivos de muñeca

Principalmente se encuentran los relojes inteligentes (smartwatches) y pulseras inteligentes (smartbands). Los smartwatches funcionan como una extensión del teléfono móvil que permiten recibir alertas, gestionar diferentes aplicaciones, y recolectar información personal como el ritmo cardíaco, los pasos, o las calorías gastadas.

En cambio, las smartbands, se enfocan más en el ámbito del bienestar y la salud, priorizando funciones como el seguimiento del sueño, y la temperatura corporal o la actividad física, generalmente sin pantallas y con un diseño reducido.

Dispositivos montados en la cabeza (head-mounted)

Se refiere a las gafas inteligentes (smart eyewear) y los auriculares o hearables. Las gafas inteligentes, fusionan diferentes sensores, pantallas y conexión para dotar de funciones de realidad aumentada, navegación o alertas.

Por otra parte, los dispositivos hearables, han evolucionado desde simples manos libres, hasta aparatos capaces de detectar signos vitales, traducir simultáneamente, o funcionar como asistentes de voz.

Otros accesorios

Esta categoría abarca elementos menos comunes como joyas inteligentes (anillos, colgantes o clips), diseñados para notificaciones, pagos sin contacto o monitorización de la actividad, así como cinturones y bandas (cinturones inteligentes o brazaletes) que supervisan señales fisiológicas o reconocen gestos del usuario.

2.6.2 Ropa electrónica (e-textiles)

Estos dispositivos son prendas de ropa que incorporan sensores y componentes electrónicos, sin afectar la comodidad del usuario. Podemos dividirlo en:

Prendas inteligentes

Tales como camisetas, pantalones, o ropa interior, capaces de medir la actividad muscular, frecuencia cardíaca, respiración e incluso realizar funciones de rehabilitación. Algunos de ellos están dirigidos al sector deportivo, mientras que otros están concebidos para la protección en ambientes de trabajo.

Dispositivos para manos y pies (foot/hand-worns)

Como calcetines, guantes o zapatos inteligentes que ofrecen características como el monitoreo de la pisada, la postura, la ubicación, o el control por gestos. Estos productos están orientados a grupos vulnerables como bebés o personas con enfermedades crónicas.

2.6.3 Parches electrónicos (e-patches)

Los e-patches, son dispositivos electrónicos que se adhieren a la piel tales como parches o incluso tatuajes, destacando estos dispositivos por su capacidad de adaptación al cuerpo humano, su ligereza, y su capacidad para usos futuros en áreas como salud, seguridad y comunicación.

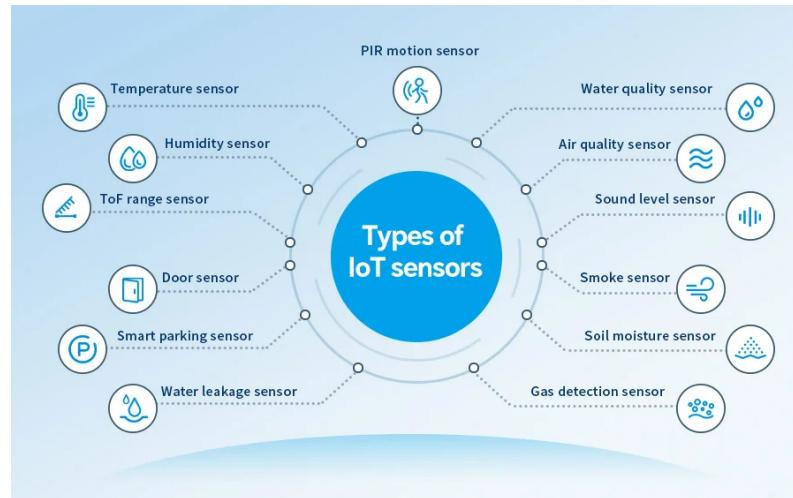


Figura 2.6 Tipos de sensores IoT.

En resumen, los dispositivos *wearables* presentan gran diversidad en relación a su forma, función e incluso aplicación. Con el análisis realizado, podemos comprender sus limitaciones, potencial desarrollo, y fundamentar el diseño de nuevas soluciones tecnológicas portátiles.

3 Estado actual de los sistemas de seguridad

La seguridad no es un producto sino un proceso

BRUCE SCHNEIER

Para poder entender el objetivo de este proyecto, y poder contextualizar sus finalidades, será necesario evaluar como han evolucionado los sistemas de seguridad desde los inicios de la civilización hasta nuestros días, siendo estos cambios guiados por las necesidades humanas y acompañados por el avance de la tecnología permitiendo así ser más precisos y sobre todo, instantáneos.

Se detallarán los métodos tradicionales usados en los dispositivos de seguridad, tanto en el ámbito del hogar, como en los dispositivos portátiles, así como un estudio sobre las nuevas tecnologías incorporadas en estos dispositivos.

3.1 Breve historia de los sistemas de seguridad

3.1.1 Primeros sistemas mecánicos y aparición de la vigilancia estructurada

La necesidad de la protección, es un instinto que el ser humano desarrolló desde tiempos antiguos. Desde la Europa medieval, los castillos contaban con fosos, puentes levadizos y muros para evitar el acceso de intrusos al interior. Los ejemplos son innumerables, desde la construcción con tablas de madera para poder oír pasos, hasta la ubicación de las mismas en colinas para poder ganar tiempo para defenderse.

Sin embargo, en cuanto a la protección de bienes personales, será necesario remontarse más atrás en el tiempo, aproximadamente al 4000 a.c donde, en Egipto y China, se desarrollaron primitivas cerraduras en puertas de madera. Un pasador horizontal de madera, se colocaba en la parte posterior de la puerta, siendo deslizado por una guía, para encajar luego en un orificio diseñado para ello. Para poder acceder a este mecanismo, fue necesario una barra de metal con acabado curvo, surgiendo así la llave primitiva.[10]

Ya por el año 200 a.C, el pueblo romano inventó el primer candado de la historia, pero no sería hasta el año 850 d.C en York (Inglaterra), en el asentamiento Jorvik Viking que se encuentran los primeros candados de púas de primavera, predecesores directos de los actuales candados. [11]

Con el desarrollo de las ciudades y las fortificaciones, surgieron sistemas de vigilancia pasiva más organizados que constituyen los inicios de los actuales sistemas de seguridad. Junto con la invención de las armas de fuego, estos sistemas, permitían tener un control de acceso más organizado y poder



Figura 3.1 Cerradura primitiva en madera.

prevenir la entrada de usuarios, así como algo que todavía no había sido explorado, el aviso al usuario. Desde torres de control con soldados y señales de humo, donde la presencia de una persona era imprescindible, hasta alertas sonoras y visuales como las campanas y contrapesos. Estos podían ser combinados con sistemas mecánicos, aumentando así su fiabilidad, pero todavía muy lejos de lo esperable de los sistemas de hoy en día.

3.1.2 Transición hacia sistemas eléctricos y electrónicos (siglo XX)

El siglo XX y la introducción de tecnologías basadas en la electricidad y la electrónica, permitieron cambiar la manera que las personas tenían para protegerse. A la par del avance tecnológico de la sociedad, crecía también las necesidades de seguridad, pudiéndose desarrollar soluciones más eficientes.

A principios del siglo XX, los sistemas eran relativamente simples, basados principalmente en elementos mecánicos como timbres y campanas, que no tenían fallas en su funcionamiento, pero si tenían un alto grado de dependencia en la intervención humana para su correcto funcionamiento.

El gran avance, y por lo tanto el paso de la tecnología mecánica a la eléctrica, fue la invención de la alarma conectada a la red eléctrica. De esta forma ya no dependían de elementos mecánicos, siendo estos sustituidos por circuitos eléctricos que al unirse a una red eléctrica centralizada, permitía el aviso de forma directa a estaciones de policía o empresas de seguridad privada.

La tecnología digital y los microprocesadores a finales del siglo XX acabó por revolucionar los sistemas de seguridad mejorando la precisión en los sensores y la integración en los dispositivos de uso diario, sentando así las bases para la evolución que llegaría en el siglo XXI con la llegada de la inteligencia artificial, los sistemas de seguridad avanzados, y la automatización de procesos y respuestas.

3.2 Sistemas de seguridad modernos

Dentro de los sistemas de seguridad, encontramos varios tipos, cada uno enfocado en diferentes aspectos y que varían tanto en las tecnologías empleadas, como en la finalidad y el tipo de usuario al que está dirigido.

3.2.1 Alarmas electrónicas convencionales

Cada vez, es una realidad más presente, la instalación de distintos sistemas de seguridad en los hogares. En España, aproximadamente el 84% de los hogares cuentan con alguna medida de seguridad, y un porcentaje significativo de estos hogares tienen una alarma conectada a una central de alarmas [12]. Estos sistemas, cuentan por lo general con instalaciones de seguridad como puertas blindadas, rejas en los accesos a la vivienda, timbres con grabación de vídeo, e incluso detectores de presencia y movimiento y gran parte de ellos se encuentran directamente conectados con el centro de control del proveedor de alarmas.

En cuanto a entornos más amplios, ya sean públicos o privados, una opción muy recurrida es la de usar cámaras de videovigilancia y sistemas de CCTV (Circuito Cerrado de TV), que permiten monitorear en tiempo real y grabar imágenes para su posterior análisis. Pueden incluir de forma adicional, funciones como la visión nocturna, o sensores de movimiento.

3.2.2 Inteligencia artificial y análisis predictivo en seguridad

Por otro lado, la llegada de la inteligencia artificial al usuario y sobre todo a la industria, ha permitido la evolución de los sistemas, hacia otros más avanzados que incorporan IA y *Machine Learning* para reconocer patrones, rostros, y comportamientos inusuales.

Debido a la gran cantidad de datos que la IA es capaz de analizar, puede ser utilizado para investigar y prevenir crímenes, desde el seguimiento de mensajes, hasta ubicación y cantidad en pagos online o con tarjeta. [13]

Sin embargo, estos no son sus únicos campos de aplicación, por el contrario, su uso se encuentra muy extendido, siendo el más conocido el reconocimiento facial, usado en aeropuertos y edificios de instituciones oficiales. Con el uso de estas tecnologías, es posible determinar la edad del usuario, si se encuentra en alguna base de datos, e incluso existen proyectos en los que es capaz de determinar el estado de ánimo de la persona a partir de las rasgos faciales. Estos proyectos pueden ser también integrados en pequeños ordenadores como los conocidos de la marca *Raspberry Pi*[14].

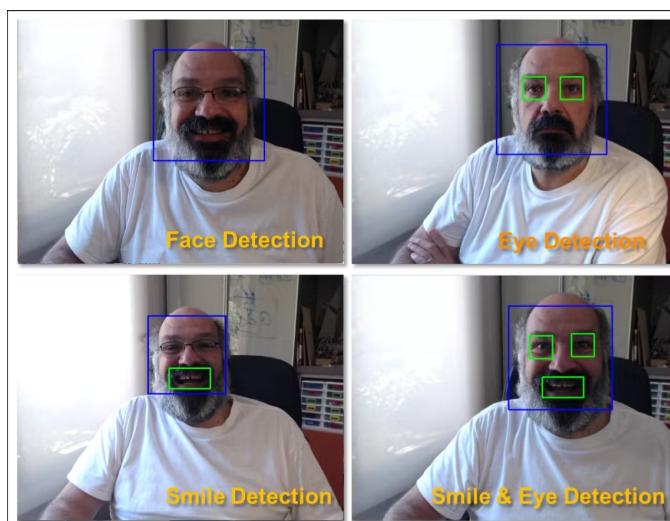


Figura 3.2 Reconocimiento facial con IA en *Raspberry Pi*.

3.2.3 Limitaciones y retos actuales

Aunque los sistemas de seguridad modernos han mejorado en precisión y funcionalidad, siguen existiendo ciertas limitaciones importantes. Una de las más comunes es la generación de falsos positivos, que no siempre son evitables, y pueden reducir la fiabilidad del sistema de forma considerable.

Otro aspecto a tener en cuenta es la dependencia de una conexión estable a internet, lo que puede dejar un dispositivo inoperativo en caso de que esta falle. Además de esta necesidad de conexión, el consumo energético limita su desarrollo, y es un factor considerable en los sistemas portátiles, especialmente si tienen que estar activos constantemente.

Además, el gran reto planteado es la gestión de la privacidad: cámaras, micrófonos y sensores conectados que pueden recopilar datos sensibles, con el riesgo de fugas o accesos no autorizados. A esto se suma la posibilidad de ciberataques, si los dispositivos no están bien protegidos o actualizados.

Por último, el coste y la complejidad de uso pueden dificultar su implantación generalizada, especialmente en entornos con pocos recursos o bajos conocimientos técnicos.

3.3 Seguridad aplicada a dispositivos electrónicos portátiles

3.3.1 Necesidad creciente de protección física de equipos personales

Con el avance de la electrónica y las tecnologías de fabricación, se ha consolidado una tendencia hacia la miniaturización de los dispositivos, buscando lograr productos cada vez más compactos y eficientes en el uso del espacio. Un claro ejemplo de ello son los teléfonos móviles; desde el primer terminal lanzado en el año 1973, con un peso de 1kg, y unas dimensiones de 33 x 4,5 x 8,9 cm, [15] hasta el último modelo de los teléfonos de la marca *Apple*, con unas dimensiones de 5,5 mm de grosor y un peso de 145g [16].

Sin embargo, en paralelo con la reducción de tamaño, se produce el aumento de información que estos dispositivos contienen, desde datos personales, como números de teléfono y nombres, hasta acceso a cuentas bancarias. Por ello, la necesidad de poder proteger estos dispositivos ha crecido hasta convertirse en un asunto de vital importancia.

3.3.2 Soluciones electrónicas: sensores de movimiento, GPS, alarmas acústicas

Una de las estrategias más eficaces para la protección de dispositivos electrónicos, y en la que centraremos nuestro trabajo, será la integración de soluciones electrónicas que detecten posibles robos. Las tecnologías que pueden ser empleadas, corresponden a un abanico muy amplio, sin embargo algunas de ellas son usadas en la mayoría de dispositivos portátiles del momento.

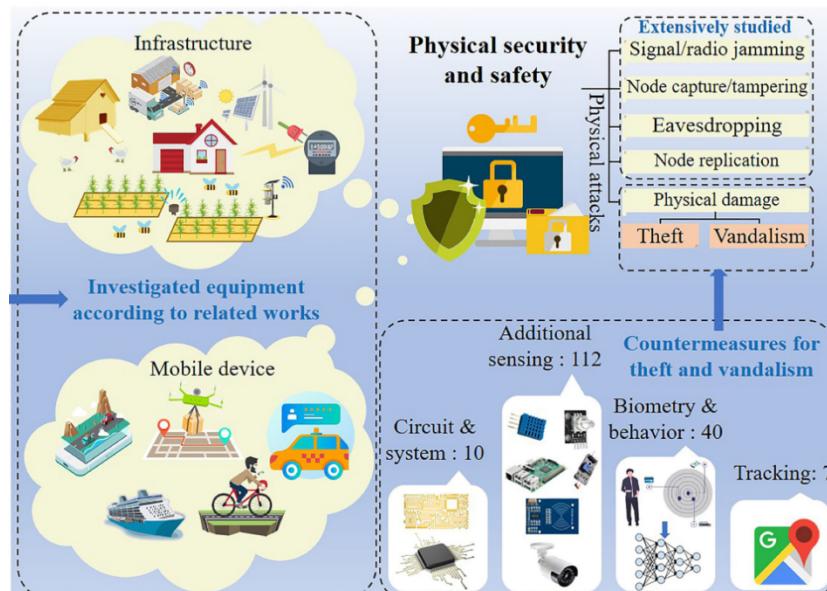


Figura 3.3 Mecanismos de seguridad física.

Entre las tecnologías más empleadas podemos encontrar los acelerómetros, dispositivos que detectan variaciones en cualquier de los 3 ejes del espacio, y que en el caso de ser movimientos bruscos, mandan un aviso informando de la perturbación. Por otro lado, los sensores de infrarrojos, ampliamente utilizados en sistemas de seguridad estáticos, también pueden ser adaptados a dispositivos portátiles con el fin de detectar intrusos.

Otros métodos ampliamente usados, son las alarmas acústicas, pudiendo estas ayudar al usuario a evitar un robo con la intención de asustar al intruso, o incluso para ayudar en la localización del dispositivo en caso de pérdida o robo.

Por último, el módulo de geolocalización o GPS, puede complementarse con las tecnologías anteriormente mencionadas obteniendo un menor tiempo de respuesta, y facilitando la recuperación del dispositivo [17]

Un ejemplo práctico de la implementación conjunta de estas tecnologías, puede ser el sistema de seguridad de los dispositivos *Apple*, que cuentan con detección de movimiento, notificación en caso de que el usuario se encuentre repentinamente lejos de uno de sus dispositivos, e incluso alarma sonora para ayudar en la localización de los mismos, así como bloqueo remoto de los terminales.[18]

3.3.3 Retos en la portabilidad, consumo y eficiencia

Sin embargo, estos avances de la tecnología, vienen acompañados por nuevos desafíos que superar, entre ellos, el límite de la portabilidad, el consumo y la eficiencia.

En cuanto al límite de la portabilidad, la ley de Moore, descrita por Gordon E. Moore, co-fundador de *Intel*, establece como el número de transistores en un mismo microprocesadores se duplica cada 2 años debido a la miniaturización de los transistores y otros componentes [19]. En teoría el límite físico para el campo de la electrónica sería el nivel atómico, sin embargo, hay un factor adicional, el límite tecnológico.

En 2020, se consiguió reducir el tamaño de los transistores a 5 nm, habiendo sufrido estos una bajada exponencial en el tamaño, pero parece ser que la industria se encuentra cercana al límite al que es capaz de elaborar estos elementos. Actualmente nos encontramos en el nivel sub-micrónico profundo (dimensiones por debajo de 1 μm) de miniaturización de los transistores, con una estructura conocida como FinFET [20]. En caso de que los transistores continúen disminuyendo su tamaño sin experimentar un cambio de tecnología, podrían enfrentarse a efectos cuánticos como el efecto túnel, lo que generaría un cambio en su comportamiento habitual.

Esto puede implicar la imposibilidad de integrar todos los sistemas de seguridad deseados, especialmente cuando se busca una reducción del tamaño del dispositivo.

Otro de los posibles retos, es el del consumo, a mayor cantidad de dispositivos y periféricos o componentes funcionando de forma simultánea, se aumenta el consumo energético llegando a ser una barrera física debido al reducido tamaño de las baterías incluidas en los dispositivos portátiles cuya única solución será la de buscar la disminución del consumo de esos componentes hasta su máximo punto.

La eficiencia a la hora de la medición y la notificación de estas medidas, también constituye un gran desafío para la obtención de sistemas fiables. Según ciertos autores [17], algunos sistemas modernos implementan modelos de detección en dispositivos móviles utilizando exclusivamente los sensores iniciales internos, alcanzando tasas de detección del 95 % con una media de solo 4,7 segundos de retardo, siendo valores muy aceptables, pero que indican que todavía queda un campo de mejora para llegar a una tasa de detección total.

4 Evaluación de necesidades del sistema

Si no sabes hacia dónde vas, cualquier camino te llevará allí.

LEWIS CARROLL, 1865

El primer paso para poder diseñar el sistema, será el de evaluar las necesidades del mismo: número de pines GPIOs, número de pines ADC y PWM, capacidad de la memoria RAM y Flash y, por supuesto, establecer el funcionamiento deseado.

4.1 Elección de componentes

Cabe destacar que tanto el microcontrolador, como los sensores y actuadores han sido provistos por el departamento de Ingeniería Electrónica de la Escuela, lo que ha facilitado mucho la búsqueda. También es importante destacar que tanto los sensores como los actuadores, son de la familia *Gravity* de la marca DFRobot. Estos cuentan con una ventaja considerable, y es que ya vienen con los circuitos integrados necesarios, tales como resistencias, condensadores y controladores, lo que nos ha permitido simplificar en gran escala el montaje del sistema, pudiendo prescindir del uso de una placa de pruebas o protoboard.

4.1.1 Microcontrolador

El microcontrolador que ha sido escogido para la evaluación de las necesidades, como ya ha sido comentado, es el **TMS320F28335** (A partir de ahora F28335). Un microcontrolador de la familia **C200** de Texas Instruments, diseñado especialmente para aplicaciones de control en tiempo real y con rendimiento elevado. A continuación se muestran las características que posee[21]:

- **CPU:** DSP de 32 bits basado en la CPU TMS320C28x, con una frecuencia de reloj de hasta 150 MHz, y con unidades de punto flotante.
- **Memoria:** Cuenta con una memoria Flash de 512 kB, y una memoria RAM de 68 kB.
- **Periféricos de comunicación:** Incorpora varios módulos de comunicación; sin embargo, se detallarán posteriormente únicamente los más habituales y planteados para ser usados en nuestro sistema. 3 módulos de UART (SCI), 1 módulo SPI y, por último, 1 módulo de I2C.
- **Periféricos de control:** En esta parte cuenta principalmente con dos periféricos principales, PWM y eCAP. Por parte de la modulación por ancho de pulso, (a partir de ahora PWM), cuenta con 6 módulos con 2 salidas independientes cada uno, lo que hace un total de 12

salidas independientes. Y para la medición de tiempo y frecuencia cuenta con 6 módulos de captura de eCAP.

- **Conversor Analógico-Digital (ADC):** Posee 16 canales de entrada analógica de 12 bits, y con una tasa de conversión de 12.5 MSPS (millones de muestras por segundo).
- **Pines de uso general (GPIO):** 88 pines que pueden ser configurados como entradas o salidas y se les puede añadir pull-up/pull-down internos.
- **Alimentación:** El sistema de desarrollo en torno al microcontrolador, es capaz de alimentar tanto a 5V como a 3.3V, más en particular, cuenta con 2 pines a 5v y 8 pines a 3.3V.

4.1.2 LM35 (Sensor de temperatura)

El sensor de temperatura escogido ha sido el LM35, más concretamente el *LM35 Linear Temperature Sensor v4* de *DFRobot*, un sensor de gran precisión con una salida analógica proporcional (10 mV/°C). Tiene un rango de medición entre los -55 °C y los 150 °C[22]. Después de convertir la temperatura de analógico a digital, será necesario convertir los mV a la escala propia de grados centígrados; para ello usaremos la siguiente ecuación:

$$\text{Temperatura}(\text{°C}) = \frac{V_{\text{analogico}}}{10} \quad (4.1)$$

Con esta ecuación conseguimos una precisión de dos decimales, pudiendo tener una medida fiable actualizada a la frecuencia de nuestro microcontrolador (100 ms).

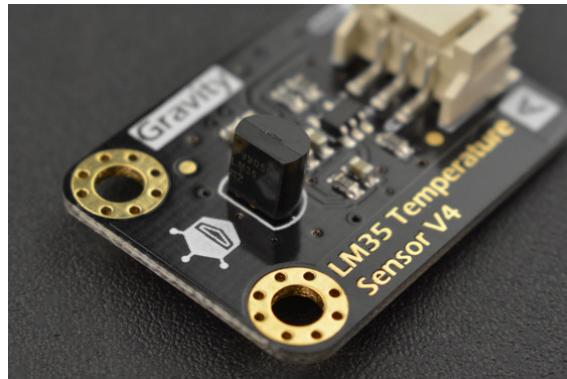


Figura 4.1 Sensor LM35.

El sensor será usado únicamente en el modo de confort, para medir la temperatura ambiente, y poder tomar decisiones de control respecto a la situación en tiempo real (encender el motor del ventilador en caso de que la temperatura supere un cierto valor). Se encontrará conectado al pin A0.

4.1.3 Grayscale Sensor (Sensor de proximidad)

El segundo de nuestros sensores será el *grayscale sensor*, compuesto por un emisor de luz (LED) y un fotoreceptor. El modelo *Analog Grayscale Sensor DFR0022*, a pesar de ser un sensor de escala de grises, será usado como sensor de proximidad. El LED blanco, tiene una forma cónica, y va acompañado de una fotocélula (resistencia variable controlada por la intensidad lumínica).

El objetivo es que, al detectar la presencia de personas u objetos, la luz reflejará en mayor cantidad, dando resultado a un valor analógico convertido mayor. En este caso no será posible la estandarización de la conversión en una escala propia (como sí ocurre con el caso de la temperatura).

La conversión a valores digitales, al igual que en el caso de la temperatura será usando los módulos ADC de nuestro DSP, que al tener una sensibilidad de 12 bits, dota al sistema de una precisión de 0,73 mV (puede variar entre 0 y 4095)[23].



Figura 4.2 Sensor Grayscale.

Sin embargo, el problema de este sensor, viene dado por la dificultad que encontramos en poder estandarizar las medidas, ya que al ser un detector de luz, dependiendo de la luz ambiente dará un valor u otro, por lo que dependiendo de la luminosidad del ambiente en el que sea diseñado, dará unos valores u otros.

Con el objetivo de poder normalizar las mediciones, a la hora del diseño del prototipo, se coloca el sensor de forma invertida sobre un folio blanco, para que el impacto de la luminosidad específica de la sala sea el menor posible. De ese modo, podemos tipificar que en el momento que el valor analógico supere los 2800 mV ($V_{analogico} > 2800mV$), el sensor ha sido levantado, y por lo tanto podremos activar la alarma.

Este sensor estará conectado al pin A2 del microcontrolador.

4.1.4 Shake Sensor (Sensor de vibración)

El último de los sensores utilizados será un detector de vibraciones (*shake sensor*). Es de tipo digital y al detectar cualquier tipo de vibración, en un eje (dibujado en el propio sensor), envía un flanco de bajada al microcontrolador avisando del movimiento.

Tiene dos posibles estados:

- Alta (HIGH o 1): Indica que no se ha detectado un evento (movimiento).
- Baja (LOW o 0): Indica que sí se ha detectado movimiento.

Cuenta con un pequeño muelle que permite detectar las vibraciones en el eje detectado, además de un pequeño circuito de filtro, permite evitar "falsas alarmas" del sensor.



Figura 4.3 Sensor de vibraciones.

El problema del sensor viene a la hora de la lectura del flanco de bajada. Se han probado 3 métodos basándonos en los módulos disponibles en nuestro DSP, algunos con mayor eficacia que otros.

El periférico utilizado, como se ha comentado previamente, ha sido el *Gravity Digital Shake Sensor SEN0289*, de *DFRobot*[24]. Este sensor, se anuncia como digital, sin embargo el funcionamiento no es realmente el esperado.

En la página del producto, se describe que, cuando el sensor recibe un movimiento o vibración en el eje, emite un flanco de bajada al microcontrolador. Tras realizar ciertas pruebas en el laboratorio con un osciloscopio, vimos como al conectarlo a la tensión de 3.3V, el flanco de bajada es prácticamente inapreciable, solo cuando se conecta a una alimentación de 5V el flanco es notable.

En ese momento, exploramos las diferentes formas de leer este sensor:

Interrupción Externa (XINT)

Nuestra primera elección para leer el sensor, fue como una interrupción externa. Conectándolo al pin GPIO30 (Definido por el fabricante)[21], seríamos capaces de generar una interrupción externa cuando el sistema leyera un flanco de bajada. Para ello se elaboró el siguiente código:

Código 4.1 Función *init_ex_int*.

```
void init_ex_int(void)
{
    EALLOW;
    // Configura GPIO30 como entrada con pull-up interno
    GpioCtrlRegs.GPAPUD.bit.GPIO30 = 0; // Habilita la resistencia pull-up
    GpioCtrlRegs.GPAMUX2.bit.GPIO30 = 0; // Configura GPIO30 como GPIO
    GpioCtrlRegs.GPADIR.bit.GPIO30 = 0; // Configura GPIO30 como entrada
    GpioCtrlRegs.GPACTRL.bit.QUALPRD3 = 8; // Periodo de calificación
    GpioCtrlRegs.GPAQSEL2.bit.GPIO30 = 1; // Calificación con 3 muestras

    // Asigna GPIO30 como fuente de XINT1
    GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 30;

    // Configuración de XINT1 para flanco de bajada
    XIntruptRegs.XINT1CR.bit.POLARITY = 0; // Detecta flanco de bajada
    XIntruptRegs.XINT1CR.bit.ENABLE = 1; // Habilita XINT1

    // Habilita el bloque PIE y la interrupción en el grupo 1
    PieCtrlRegs.PIECTRL.bit.ENPIE = 1; // Habilita el bloque PIE
    PieCtrlRegs.PIEIER1.bit.INTx4 = 1; // Habilita la interrupción XINT1 en grupo 1
    IER |= M_INT1; // Habilita interrupciones del grupo 1 en el CPU
    EDIS;
```

Una vez hecho eso, necesitaremos leer si se ha activado la bandera de la interrupción; en ese caso, activar una variable auxiliar y borrar el flag:

Código 4.2 Limpieza de la variable *mov_det*.

```
// Verifica si la interrupción externa se ha activado
if (PieCtrlRegs.PIEIFR1.bit.INTx4 == 1) // Revisa si el flag de la interrupción está
    activo
{
    mov_det = 1; // Activa la variable mov_det
    PieCtrlRegs.PIEIFR1.bit.INTx4 = 1; // Limpiar flag de interrupción
}
```

Una vez hecho eso, necesitaremos leer si se ha activado la bandera de la interrupción; en ese caso, activar una variable auxiliar y borrar el flag: la variable *mov_det* tendrá que ser borrada de forma manual, ayudándonos del *cpu_timer0*, con un contador que, tras estar activa durante un segundo, la vuelve a desactivar.

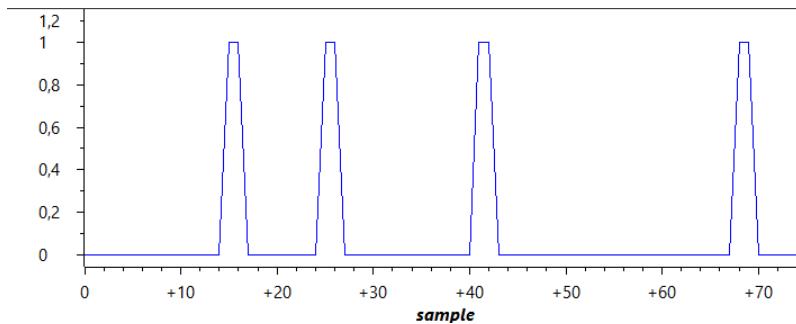


Figura 4.4 Variable *mov_det* con Int_Ext.

El funcionamiento es el esperado; sin embargo, la sensibilidad es considerablemente baja. Solo con grandes vibraciones conseguimos que se active la interrupción, por lo que optamos por otros métodos.

eCAP

El siguiente de los métodos con el que intentamos leer el sensor fue con el módulo eCAP, ya incluido en nuestro microcontrolador[21]. Este módulo se utiliza para medir y capturar eventos relacionados con el tiempo y las señales PWM. Permite medir velocidades de máquinas rotativas, tiempos transcurridos entre pulsos de sensores de posición, y analizar el período y el ciclo de trabajo de señales de tren de pulsos.

También es útil para decodificar amplitudes de corriente o voltaje codificadas en ciclo de trabajo. Incluye registros para capturar eventos con marcas de tiempo, selección de polaridad de los bordes, generación de interrupciones y diferentes modos de captura, como único, continuo, absoluto o diferencial. Además, puede configurarse como una salida PWM de un canal si no se usa en modo de captura.

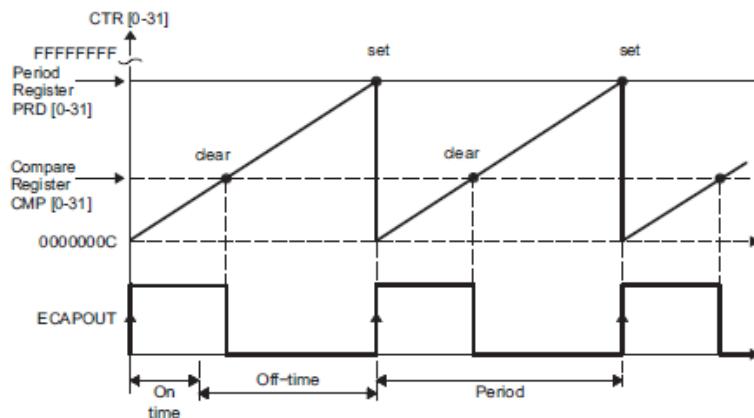


Figura 4.5 Funcionamiento del módulo eCAP.

En resumen, puede ser usado, para transformar una señal, que puede ser triangular o irregular, en una función cuadrada, y medir el periodo de estos pulsos. Sin embargo nosotros usamos el reloj propio del módulo, para poder transformar nuestra señal, que no alcanzaba el valor suficiente para que el microcontrolador lo detectase como un 0, en una señal cuadrada, de forma que cuando recibiera un pulso de bajada, se activase la función del eCAP.

Para ello, elaboramos el siguiente código:

Código 4.3 Función *ConfigurarECAP1*.

```

void ConfigurarECAP1(void) {
    EALLOW;
    SysCtrlRegs.PCLKCR1.bit.ECAP1ENCLK = 1; // Habilitar el reloj para eCAP1
    ECap1Regs.ECEINT.all = 0x0000;           // Deshabilitar todas las interrupciones del eCAP
    ECap1Regs.ECCLR.all = 0xFFFF;            // Borrar todas las banderas de interrupcion
    ECap1Regs.ECEINT.bit.CEVT1 = 1;          // Habilitar interrupcion en el evento 1 de captura

    ECap1Regs.ECCTL1.bit.CAP1POL = 1;        // Configurar polaridad de flanco de bajada en evento 1
    ECap1Regs.ECCTL1.bit.CTRRST1 = 1;         // Reiniciar contador despues de evento 1
    ECap1Regs.ECCTL2.bit.CONT_ONESHOT = 0;    // Modo continuo
    ECap1Regs.ECCTL2.bit.TSCTRSTOP = 1;       // Iniciar el contador
    ECap1Regs.ECCTL2.bit.CAP_APWM = 0;         // Modo de captura
    EDIS;

}

// Interrupcion del eCAP1
interrupt void eCAP1_ISR(void) {
    Mov_detectado = 1;                      // Flanco de bajada detectado, establecer flag
    ECap1Regs.ECCLR.bit.CEVT1 = 1;           // Borrar bandera de evento de captura 1

    ECap1Regs.ECCLR.bit.INT = 1;              // clear the global interrupt flag;
    ECap1Regs.ECCTL2.bit.REARM = 1;           // acknowledge next edge detect
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP4; // Reconocer la interrupcion en el grupo 4 del PIE
}

```

Primero configuramos el módulo, configurando la polaridad del flanco de bajada, habilitando el reloj para el eCAP1 y por último seleccionando el pin GPIO5 como entrada para el eCAP, además de otros parámetros que no serán importantes. Una vez detectada la interrupción por el eCAP, se activará la función de interrupción que pone a 1, nuestra variable *mov_detectado*, que al igual que en el caso anterior, tendrá que ser desactivada manualmente, con un contador en el *cpu_timer0* o con una función *delay*.

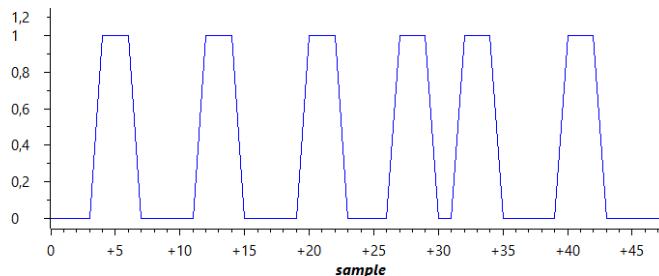


Figura 4.6 Variable *mov_detectado* con módulo eCAP.

Con esta solución, la sensibilidad aumenta de forma considerable, por lo que podría ser una buena opción; sin embargo, parece relativamente desproporcionado usar el módulo eCAP si no estamos aprovechando la función por la que se caracteriza de medición de tiempo. Por ello experimentamos con una última opción.

ADC

La última de las soluciones que se ha valorado ha sido el contemplar este sensor como analógico. Al enviar una señal que no hace de forma correcta el flanco de bajada, puede considerarse como una señal analógica que se encuentra a un cierto valor (veremos que será en torno a los 3V, cumpliendo con los 3V3 máximos del convertidor A-D) que será considerado como '1', y cuando baje de otro cierto valor (aproximadamente 2.8V), se considerará como un flanco de bajada.

En este caso no será necesario reiniciar la variable auxiliar, ya que interactuaremos directamente con la magnitud en analógico, y después de emitir el flanco de bajada, vuelve a los valores normales.

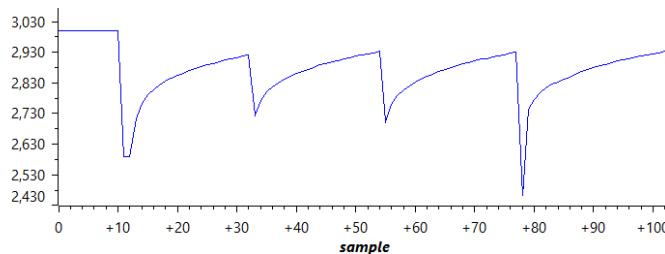


Figura 4.7 Variable *mov_detect* con módulo ADC.

Podemos observar como, aún haciendo vibraciones con diferentes intensidades, podemos detectar el valor analógico de forma clara. Con esta solución, aprovechamos el módulo ADC que ya está siendo utilizado previamente para los otros dos sensores analógicos, y además obtenemos una precisión de 0,73 mV. La conversión la hacemos de la siguiente forma:

Código 4.4 Conversión binario a mV.

```
Vana_Shk = 3000.0*(float)Vbin_Shk/4095.0; // Conversion de binario a analogico en mV
```

4.1.5 Motor

El primero de los actuadores a comentar será el ventilador, en concreto el *DFR0411 Gravity 130DC Motor*[25]. Está compuesto por un pequeño motor DC, que al igual que el resto de componentes ya tiene el circuito integrado, y que tras ser conectado a la alimentación de 5V, podremos controlar su velocidad por PWM a través del pin GPIO2. Este pequeño ventilador será empleado en el modo de confort, para que en caso de que la temperatura supere un cierto valor, se active, o a velocidad media, o a velocidad máxima, con el objetivo de bajar la temperatura ambiente.



Figura 4.8 Motor.

4.1.6 Altavoz

El siguiente de los actuadores que ha sido utilizado ha sido un módulo de altavoz o *speaker*. Este modelo es el *FIT0449 DFRobot Speaker v1.0* y puede ser usado como buzzer e incluso es capaz de reproducir sonido de alta calidad[26]. En nuestro sistema será usado como buzzer indicador de robo, a modo de alarma. Necesita de una alimentación entre los 2-5V, por lo que haremos la conexión con el pin a 3.3V. El control del sonido será por PWM, alternando la frecuencia entre dos valores (25 % y 90 %) para conseguir un aviso acústico en caso de robo.

De modo adicional, en el circuito integrado, cuenta con un pequeño potenciómetro con el que podremos aumentar o reducir el volumen con el que reproducirá el sonido.



Figura 4.9 Altavoz.

4.1.7 Vibrador

Con el objetivo de acentuar el aviso en caso de robo, se ha optado por añadir un módulo vibrador, más en concreto el *DFR0440 Gravity Vibration Module*[27]. Este actuador, al igual que ocurre con el ventilador y el altavoz puede ser controlado por modulación PWM, en este caso desde el pin GPIO0.

Con este módulo se busca incrementar la señal de alarma, sin dotar de una dificultad extra a la hora de la programación. Para simplificar el control de este actuador, se ha mantenido a un valor estable siempre que la alarma está activa (en concreto un 90 % de su intensidad).



Figura 4.10 Vibrador.

4.1.8 LED

El último de los actuadores será un diodo led, el *DFR0021 Digital Red LED module v2*[28]. Al igual que previamente comentado, ya incluye el circuito integrado con la resistencia para limitar la corriente que circula en conducción por él. Será activado de forma manual a través del pin GPIO17, y se usa de forma consonante junto con el altavoz, encendiendo y apagando de forma intermitente para reforzar esta señal acústica.



Figura 4.11 LED.

4.1.9 Módulo de comunicación (SCI) - Terminal de Comunicación Serie

Para controlar la comunicación entre el usuario y el DSP, se ha optado por el módulo SCI (Interfaz Serial de comunicación - *Serial Communication Interface*). De este modo, se permite al usuario poder navegar entre ambos modos de operación, con el envío de un simple comando ('a' para Modo Confort y 'a' para Modo Antirrobo).

Deberemos además reservar dos pines en nuestro DSP, uno para transmisión y otro para recepción, en nuestro caso, la entrada digital al pin GPIO29 y salida digital al pin GPIO28 [21].

A pesar de que queremos dotar al sistema de conectividad "Bluetooth", inicialmente, usaremos la UART, a través de una aplicación terminal, en concreto "Hércules", siendo esta conexión por cable para poder comprobar en primer lugar el funcionamiento del sistema, y posteriormente, añadir la conectividad inalámbrica.

4.1.10 Alimentación

El sistema es capaz de alimentar a diferentes tensiones según las necesidades de los actuadores y los sensores.

- **5V:** Máxima alimentación ofrecida principalmente para el motor y el Shake Sensor (Al tener que detectar el flanco de bajada, es solo apreciable a esta tensión).
- **3.3V:** Regulada para prácticamente todos los demás sensores y actuadores.

4.2 Esquema de conexiones

El sistema está configurado según el siguiente esquema de pines del microcontrolador:

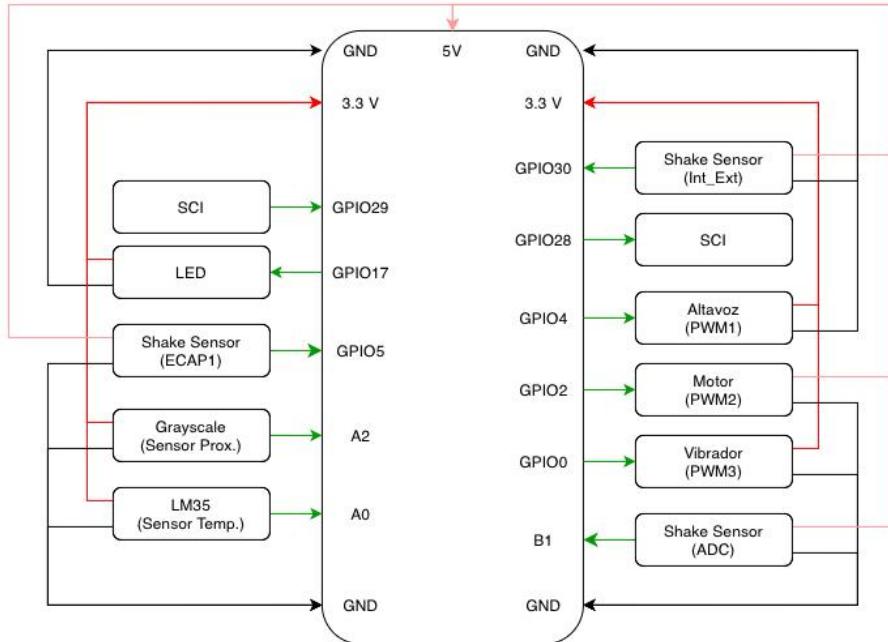


Figura 4.12 Esquema de conexiones.

Tabla 4.1 Esquema de conexiones.

Componente	Pin del Microcontrolador	Tipo de Entrada/Salida
SCI (Comunicación)	GPIO29/GPIO28	Entrada/Salida digital
LED	GPIO17	Salida digital
Shake Sensor (ECAP1)	GPIO5	Entrada digital
GrayScale Sensor	A2	Entrada analógica
LM35	A0	Entrada analógica
Shake Sensor (Int_Ext)	GPIO30	Entrada digital
Altavoz (PWM1)	GPIO4	Salida PWM
Motor (PWM2)	GPIO2	Salida PWM
Vibrador (PWM3)	GPIO0	Salida PWM
Shake Sensor (ADC)	B1	Entrada analógica

4.3 Funcionamiento del sistema

El sistema diseñado opera en dos modos principales: el **Modo Confort** y el **Modo Antirrobo**, controlados mediante comandos recibidos a través del puerto SCI. Cada modo responde a condiciones específicas detectadas por sensores, y su operación se describe a continuación:

4.3.1 Modo Confort

El *modo Confort* es el que inicialmente se encuentra establecido. Está diseñado para el control ambiental del entorno en el que se encuentra nuestro sistema, utilizando el sensor de temperatura **LM35** para establecer la máxima sensación de bienestar al usuario. Este modo se activa al recibir el comando ‘a’ a través de la interfaz serial (SCI). El modo de operación es el siguiente:

- **Inicio del modo:** El sistema desactiva todos los actuadores inicialmente. El motor permanece apagado con una potencia de 0%, asegurando un estado inactivo hasta que se evalúe la temperatura ambiente.
- **Lectura del sensor de temperatura LM35:** El microcontrolador realiza lecturas periódicas del sensor de temperatura LM35, conectado a una entrada analógica. La señal analógica es convertida a digital mediante el ADC interno del microcontrolador para procesar el valor de la temperatura. Posteriormente, ese valor en digital es convertido a escala de temperatura (0-100 °C).
- **Control:** El sistema ajusta la potencia del motor según la temperatura:
 - $T < 25^{\circ}\text{C}$: Motor apagado (0%).
 - $25^{\circ}\text{C} \leq T < 28^{\circ}\text{C}$: Motor al 50%.
 - $T \geq 28^{\circ}\text{C}$: Motor al 75%.
- **Transición:** Mientras el sistema se encuentra en *modo confort*, monitorea constantemente el puerto SCI. Si recibe el comando ‘b’, el sistema pasa a *Modo Antirrobo*. Una vez en modo antirrobo, aunque la temperatura cambie, el motor permanecerá parado.

4.3.2 Modo Antirrobo

El Modo Antirrobo se activa al recibir el comando ‘b’ por el puerto SCI. Este modo está diseñado para monitorear movimientos sospechosos y proximidad de objetos utilizando los sensores **Grayscale** y **Shake Sensor**. El modo de operación es el siguiente:

- **Inicio:** Al activar este modo, el sistema desactiva inicialmente todos los actuadores (altavoz, motor y vibrador) y realiza lecturas de los sensores para determinar el estado inicial.

• **Condiciones:** La alarma se activa si:

- El sensor Grayscale evalúa la intensidad de la luz reflejada. Si el valor leído supera un umbral crítico ($Vana_{Gsc} > 2000$), se considera que el sistema se ha levantado.
- El Shake Sensor detecta movimientos unidireccionales y envía una señal digital (flanco de bajada) al microcontrolador. Si se detecta movimiento ($Mov_detect = 1$), se activa una alerta.

• **Respuesta:**

- *Altavoz:* Señal PWM intermitente (25 % y 90 %).
- *Vibrador:* Activación al 90 %.
- *LED:* Iluminación intermitente

- **Transición:** Si el valor del Grayscale Sensor cae por debajo del umbral ($Vana_{Gsc} < 2000$), el sistema asume que ya no hay riesgo y desactiva las alarmas, permaneciendo aún en *Modo Antirrobo*. Únicamente hasta no recibir el comando ‘a’ de forma manual a través del SCI, el sistema permanece en este modo. Cuando esto ocurre, el sistema pasa a *Modo Confort* de nuevo, quedando desactivadas todas las alarmas (tanto visuales como sonoras).

4.3.3 Diagrama de flujo

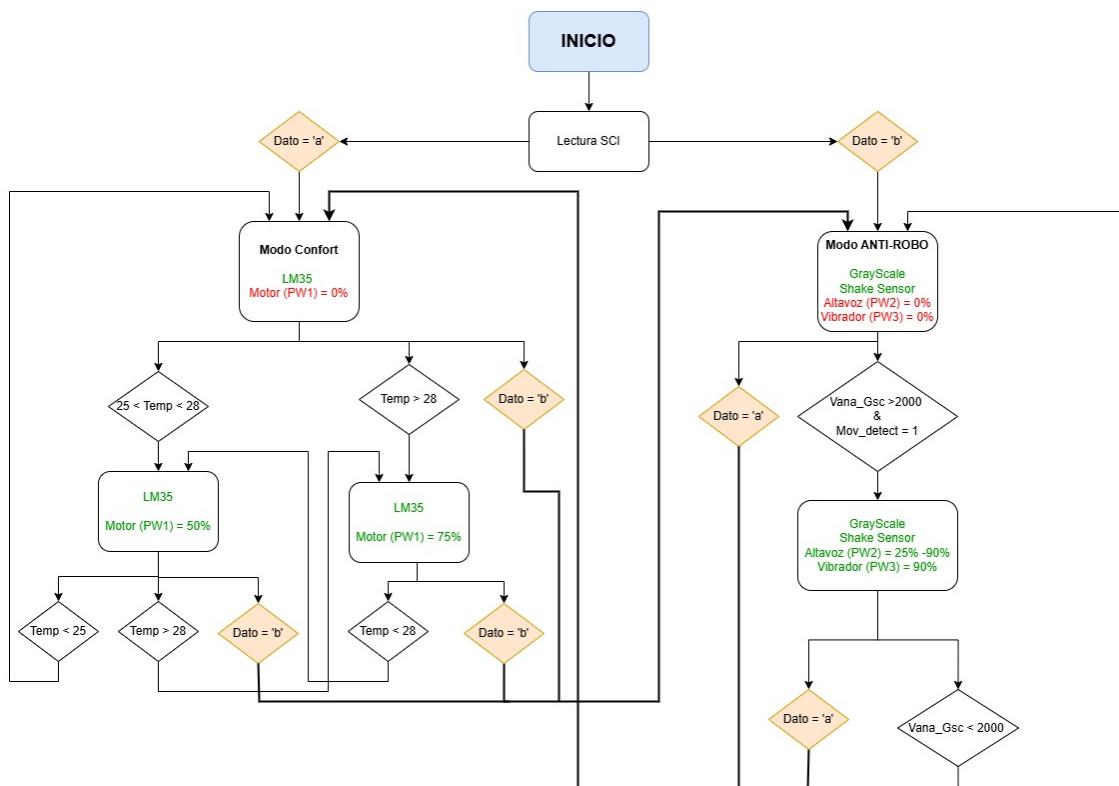


Figura 4.13 Diagrama de flujo.

4.4 Implementación

En este apartado se explicarán los principales módulos usados, todos ellos disponibles en nuestro DSP F28335. Simplemente explicaremos los parámetros que han tenido que ser configurados en

el código para su correcto funcionamiento. Estos módulos, configurados y optimizados según las necesidades de nuestro sistema, permiten la interacción entre los sensores, actuadores y la lógica de control.

La documentación para su uso ha sido encontrada en el *datasheet* del mismo. Los archivos independientes serán mencionados y posteriormente incluidos en los anexos de este documento.

4.4.1 ADC

Los microcontroladores son utilizados para procesar información del mundo natural, decidir una línea de acción basada en la información recopilada y, a continuación, emitir señales de control para poner en práctica la decisión. Como la información del mundo natural es analógica o continua (formas de onda), y el microcontrolador es un procesador digital, se necesita un método para convertir una señal analógica en digital.

El módulo ADC (Analog-to-Digital Converter), es el encargado de esto, pudiendo convertir los valores a binario, y posteriormente a su escala propia (si hubiese).

En nuestro sistema, ha sido usado para la conversión de los dos sensores análogicos, el *LM35* y el *GrayScale Sensor* y el *Shake Sensor*.

Como se explicó previamente, en el DSP F28335, el módulo ADC tiene 16 canales, configurables como dos módulos independientes de 8 canales (Multiplexados en dos grupos *ADCINA0-7* y *ADCINB0-7*). Ambos módulos pueden ser configurados en cascada para formar un único módulo de 16 canales. Cuenta con un único convertidor para los valores analógicos.

Cuenta además con 16 registros de resultados y un reloj independiente basado en el reloj de la CPU.

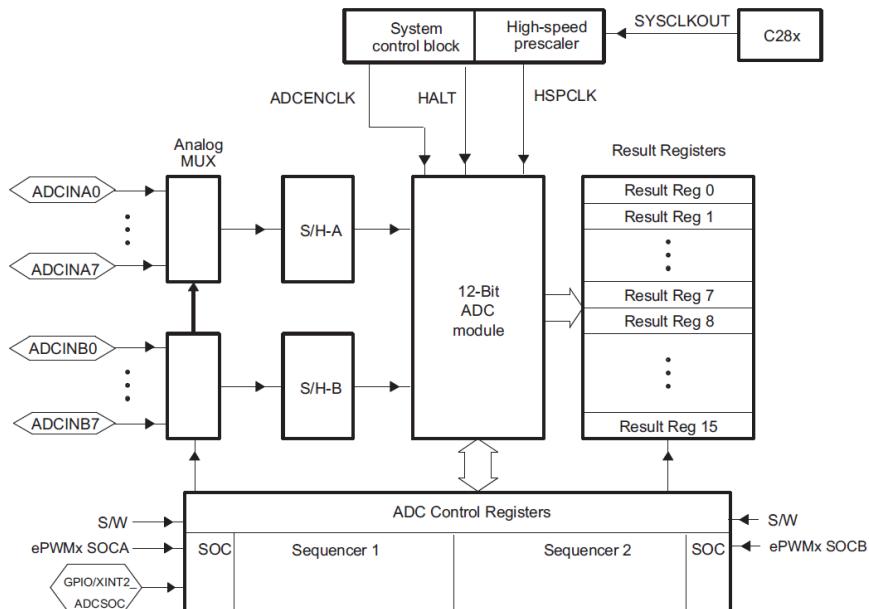


Figura 4.14 Diagrama de bloques del módulo ADC.

El rango de tensión de entrada es de 0 a 3V, y al tener en cuenta que tiene una resolución de 12 bits, obtenemos una resolución de 0,73 mV. Está compuesto por dos unidades **Sample and Hold**, que pueden ser configuradas en *Modo secuencial* (muestro de señales de una en una), o *Modo simultáneo* (muestreo de dos señales a la vez). La selección de ese modo, será con el registro *AdcRegs.ADCTRL3.bit.SMODE_SEL* (a 0 para modo secuencial o 1 para modo simultáneo)

La frecuencia de conversión (**ADCCLK**) está limitada a 12.55 MHz, y el tiempo de muestreo (*sampling time*) o tiempo en el que el Sample and Hold retiene el valor capturado, está configurado como el doble del periodo de conversión, y viene configurado por el registro *AdcRegs.ADCCTRL1.bit.ACQ_PS*.

Por otra parte, el Auto-Secuenciador, puede estar definido como un único secuenciador (modo cascada) o 2 secuenciadores (modo dual), en nuestro caso como modo dual. Una vez especificado el modo, tendremos que establecer cual es el número total de canales a convertir, con el registro *AdcRegs.ADCMAXCONV.all*, y siendo ese valor el siguiente: $MAX_CONV = N_{conversiones} - 1$. Por último, configuraremos el inicio de la secuencia de la siguiente conversión con el siguiente registro *AdcRegs.ADCCTRL1.bit.CONT_RUN*, poniéndolo a 0 para que espere a la siguiente señal de inicio para lanzar la siguiente secuencia.

Una vez convertido los datos, es el momento de almacenar los resultados, mediante el registro *AdcRegs.ADCCHSELSEQ1.bit.CONVxx*, almacenando en orden de conversión, y especificando que canal se debe guardar en cada resultado. Para finalizar deberemos habilitar la interrupción, y configurar el modo de interrupción para ser generada al final de cada secuencia de conversión (EOS - *End Of Sequence*).

Una vez dentro de la interrupción, copiamos los valores almacenados de los resultados, en nuestras variables, y posteriormente, limpiamos la interrupción para que pueda volver a saltar en la siguiente conversión.

A continuación se muestra el código que se ha utilizado para la función de configuración del ADC:

Código 4.5 Función *InitAdcRegs*.

```
void InitAdcRegs(void)
{
    AdcRegs.ADCCTRL1.all = 0;
    AdcRegs.ADCCTRL1.bit.ACQ_PS = 1;      // Sample window = 2*(1/ADCCLK)
    AdcRegs.ADCCTRL1.bit.SEQ_CASC = 0;    // Modo dual
    AdcRegs.ADCCTRL1.bit.CPS = 1;         // CPS = 2 --> ADCCLK = FCLK/(CPS+1)
    AdcRegs.ADCCTRL1.bit.CONT_RUN = 0;    // Modo start-stop

    AdcRegs.ADCCTRL2.all = 0;
    AdcRegs.ADCCTRL2.bit.INT_ENA_SEQ1 = 1; // Se habilita interrupcion SEQ1INT
    AdcRegs.ADCCTRL2.bit.INT_MOD_SEQ1 = 0; // Modo de interrupcion cada EOS

    AdcRegs.ADCCTRL3.bit.SMODE_SEL = 0;   // Modo muestreo secuencial
    AdcRegs.ADCCTRL3.bit.ADCCLKPS = 1;    // ADCCLKPS = 3 --> FCLK = HSPCLK / 2 * ADCCLKPS
                                            // HSPCLK = 75 MHz
                                            // FCLK = 37.5 MHz
                                            // ADCCLK = 12.5 MHz

    AdcRegs.ADCMAXCONV.all = 0x0002;     // 3 conversiones por secuencia

    AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0; // Canal A0 para el sensor de temperatura (LM35)
    AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 2; // Canal A2 para el sensor de cercania (Gsc)
    AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 9; // CONVV02 = ACDINB1. (Canal B1 ShakeSensor)
}
```

Se muestra a continuación también la función de interrupción:

Código 4.6 Función de interrupción ADC.

```
interrupt void adc_SEQ1_isr(void)
{
    Vbin_LM35 = AdcMirror.ADCRESULT0; // Almacenar resultado del sensor de temperatura
    Vbin_Gsc = AdcMirror.ADCRESULT1; // Almacenar resultado del sensor de cercania
    Vbin_Shk = AdcMirror.ADCRESULT2; // Almacenar resultado del sensor de vibracion

    AdcRegs.ADCCTRL2.bit.RST_SEQ1 = 1; // Reset SEQ1
```

```

        AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1; // Clear INT_SEQ1 para la siguiente interrupcion
        PieCtrlRegs.PIEACK.bit.ACK1 = 1;    // Clear PIEACK para siguiente interrupcion
    }
}

```

Una vez convertidos estos valores a binarios, será necesario convertirlos a la escala correspondiente, siempre antes convirtiéndolos a “valores analógicos” (valores binarios en números naturales) para facilitar la lectura del usuario.

La resolución del ADC es de 12 bits, lo que indica una resolución de 0,73 mV por lo que hay 4095 posibles valores de voltaje que se pueden obtener. Una vez convertido este valor digital a mV, será necesario convertirlo a la escala correspondiente:

Código 4.7 Conversión de binario a "analógico".

```

// Conversion a valores analogicos
Vana_LM35 = 3000 * (float)Vbin_LM35 / 4095; // mV
Temp = Vana_LM35 / 10;                      // Conversion a temperatura en °C
Vana_Gsc = 3000.0*(float)Vbin_Gsc/4095.0;   // Conversion de binario a analogico en mV
Vana_Shk = 3000.0*(float)Vbin_Shk/4095.0;   // Conversion de binario a analogico en mV

```

4.4.2 PWM

El módulo *Enhanced Pulse Width Modulation* (A partir de ahora ePWM) se encarga de la modulación del ancho de pulso, su uso principal, recae en el control de velocidad de motores eléctricos y convertidores de potencia.

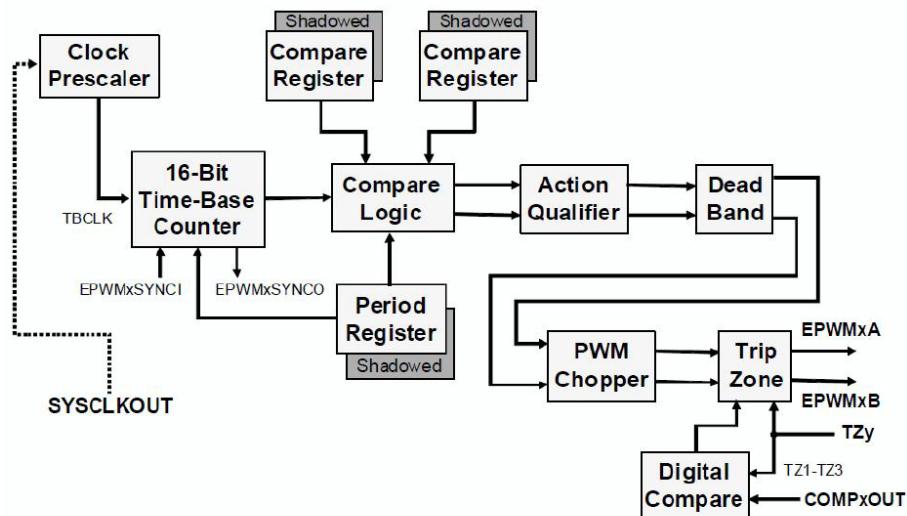


Figura 4.15 Diagrama de bloques del módulo PWM.

En concreto, el F28335 dispone de 6 unidades ePWM, cada una de las cuales puede generar 2 señales PWM. Los canales para el ePWM están multiplexados en algunos pines GPIO y consta de 7 módulos configurables: Time Base, Counter compare, Action qualifier, Dead band, PWM-Chopper, Trip Zone y Event trigger, de los cuales únicamente los tres primeros serán configurados estando los restantes limitados al control de convertidores de potencia.

Time Base

Usado para la temporización del ePWM, creando una señal para la generación de PWM. La frecuencia será definida y esta relacionada con las dos señales generadas, *EPWMxA* y *EPWMxB*.

Dispone de un registro contador de 16 bits **TBCTR**, cuyo valor varia con la frecuencia definida a partir de los registros **EPwm1Regs.TBCTL.bit.CLKDIV** y **EPwm1Regs.TBCTL.bit.HSPCLKDIV** de la siguiente forma:

$$TBCLK = \frac{SYSCLKOUT}{HSPCLKDIV * CLKDIV} \quad (4.2)$$

$$TBPRD = \frac{1}{TBCLK} \quad (4.3)$$

Una vez definido el valor del contador, deberemos delimitar la forma de contar teniendo tres opciones diferentes **Count up-down** (10- De 0 a TBPRD y luego disminuyendo hasta 0), **Count up**(00- De 0 a TBPRD) y **Count down** (01- De TBPRD a 0), siendo configurado el modo con el registro **EPwm1Regs.TBCTL.bit.CTRMODE**, en nuestro caso Up-Down.

De forma adicional contamos con un registro de "Shadow", que actúa como un buffer y se usa para evitar comportamientos erróneos debidos a asincronismos del registro TBPRD, y también un registro que proporciona la capacidad de sincronizar varias unidades ePWM entre sí, en nuestro caso desactivado.

Counter Compare

Este segundo bloque se encarga de establecer los puntos de comparación en el periodo TBPRD donde las señales ePWM cambian su valor. En nuestro caso, este valor a comparar, será el máximo valor del contador, modificado por el registro **EPwm1Regs.CMPA.half.CMPA**. Teniendo en cuenta que la onda es de modo Up-Down, tendremos dos puntos de comparación. Al igual que el bloque Time Base cuenta con la propiedad de "Shadow", en nuestro caso también será usada.

Action Qualifier

El último de los bloques a configurar, realiza la modificación de la señal EPWMxA dependiendo de los eventos de comparación que se establecen, pudiendo tomar 4 valores posibles estos registros: 00 (acción desactivada), 01 (nivel bajo), 10 (nivel alto) y 11 (intermitente entre alto y bajo) y definido por el registro **EPwm1Regs.AQCTLA**.

Han sido definidas 3 funciones diferentes, sin embargo, el único cambio entre una y otra es el pin GPIO escogido. Por ello, el código corresponde a la función del PWM1 (Altavoz).

Código 4.8 Función *Setup_ePWM*.

```

void Setup_ePWM1(void)
{
    // Configuración del modulo Time-Base
    EPwm1Regs.TBCTL.bit.CLKDIV = 0;           // Pre-escalador CLKDIV = 1
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = 1;         // Pre-escalador HSPCLKDIV = 2
    EPwm1Regs.TBCTL.bit.CTRMODE = 2;           // Modo up - down
    EPwm1Regs.TBCTL.bit.PRDLD = 0;             // Shadow activado
    EPwm1Regs.TBCTL.bit.SYNCSEL = 3;            // Sincronización deshabilitada
    EPwm1Regs.TBPRD = 37500;                  // Periodo para 1KHz de fPWM (150MHz/(2*2*37500))

    // Configuración del modulo Compare-Counter
    EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD; // Inicialmente al 0%
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = 0;        // shadow activado para CMPA
    EPwm1Regs.CMPCTL.bit.LOADAMODE = 0;          // shadow de CMPA cargado en TBCTR = 0

    // Configuración del modulo Action Qualifier
    EPwm1Regs.AQCTLA.all = 0;
    EPwm1Regs.AQCTLA.bit.CAD = 1;                // EPWM1A a low en CMPA down
    EPwm1Regs.AQCTLA.bit.CAU = 2;                // EPWM1A a high en CMPA up
}

```

El PWM, ha sido configurado a nivel bajo, por lo tanto, para seleccionar un funcionamiento activo al máximo, habrá que hacer un duty cycle del 0 %. Se muestra a continuación la configuración del PWM1 del altavoz:

Código 4.9 Modificación del PWM.

```
// Alternar entre 25% y 75% de duty cycle cada 500 ms
if (duty_cycle_25==0) {
    EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD * 0.1; // Duty cycle al 90%
} else {
    EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD * 0.75; // Duty cycle al 25%
}
duty_cycle_25 = !duty_cycle_25; // Alternar la señal
Delay_ms(50); // Esperar 500 ms
```

4.4.3 SCI

El último de los módulos es el de la comunicación serial o *Serial Communication Interface* (SCI a partir de ahora). Consiste en un puerto de comunicación serie bidireccional para la comunicación asíncrona entre la CPU del F28335 y otros dispositivos. En el caso de nuestro DSP, se disponen de dos unidades SCI, *SCIA* y *SCIB*. Cada una de estas unidades dispone de 2 líneas de comunicación, una de transmisión (TX) y otra de recepción (RX), en nuestro caso han sido escogidos los pines **GPIO28** y **GPIO29** respectivamente.

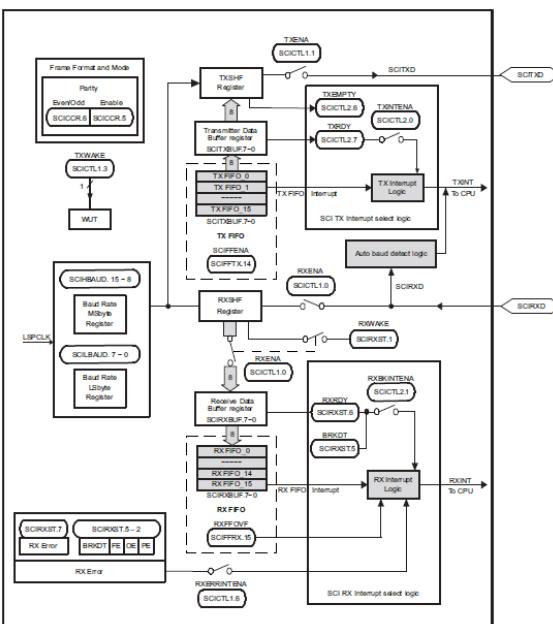


Figura 4.16 Diagrama de bloques del módulo SCI.

El primer paso será la configuración del formato del dato a transmitir/recibir. Este dato será una palabra, es decir, una variable de tipo *char* que podrá tener entre 1 y 8 bits, siendo manejado por el registro *SciaRegs.SCICCR.bit.SCICHAR* (en nuestro caso son 8 bits). Una vez determinado el tamaño, deberemos especificar si el dato también incluye el bit de paridad (indicar si es par o impar), el bit de dirección (indica el dispositivo al que deben llegar los siguientes marcos de dirección), y por último, si tiene uno o dos bits de Stop, marcado por el registro *SciaRegs.SCICCR.bit.STOPBITS*, lo estableceremos como un único bit de stop.

Una vez configurado, y antes de terminar, deberemos únicamente habilitar o deshabilitar las funciones de recepción y transmisión, así como las funciones de interrupción. Una vez habilitadas, definiremos dos funciones, una para la transmisión y otra para la recepción que almacenen o lean en una variable *pData*, los valores transmitidos y/o recibidos.

Por simplicidad y legibilidad a la hora de la redacción del código, estas funciones han sido declaradas en un archivo .c diferente mostrado a continuación:

Código 4.10 Archivo *sci.c*.

```
#include "sci.h"

void SCIA_init() {
    SciaRegs.SCICCR.all = 0;
    SciaRegs.SCICCR.bit.SCICHAR = 7;      // 8 bits de datos
    SciaRegs.SCICCR.bit.ADDRIDDLE_MODE = 0; // Sin bit de dirección
    SciaRegs.SCICCR.bit.PARITYENA = 0;     // Sin bit de paridad
    SciaRegs.SCICCR.bit.STOPBITS = 0;       // 1 bit de stop

    SciaRegs.SCICTL1.all = 0;              // Se pone a cero para hacer el RESET del SCIA
    SciaRegs.SCICTL1.bit.RXENA = 1;        // Se habilita la línea de recepción
    SciaRegs.SCICTL1.bit.TXENA = 1;        // Se habilita la línea de transmisión

    SciaRegs.SCIHBAUD = 487 >> 8;        // Highbyte
    SciaRegs.SCILBAUD = 487 & 0x00FF;      // Lowbyte

    SciaRegs.SCICTL2.bit.TXINTENA = 1;
    SciaRegs.SCICTL2.bit.RXBKINTENA = 1;

    SciaRegs.SCICTL1.bit.SWRESET = 1;      // Se libera el SCI del RESET para poder empezar a operar
}

void SCIA_Transmit(char const* pData, const uint16_t size){
    uint16_t index=0;

    while(index < size){
        while(!SciaRegs.SCICTL2.bit.TXRDY);
        SciaRegs.SCITXBUF = pData[index++];
    }
}

void SCIA_getchar(char *pData){
    *pData=SciaRegs.SCIRXBUF.bit.RXDT;
}
```

Un aspecto a comentar es que, debido a la forma de codificación de la función *SCIA_getchar*, el sistema se encuentra continuamente monitorizando la llegada de datos; por lo tanto, cuando recibe un dato, manualmente, deberemos comprobar si es un dato nuevo, para que el sistema solo se modifique si hay cambios en el dato recibido.

Código 4.11 Comprobación dato nuevo SCI.

```
// Actualiza 'dato_nuevo' si hay un cambio en 'dato' respecto a 'dato_anterior'
if (dato != dat_ant) {
    dat_nuevo = 1;          // Activa dato_nuevo para permitir un nuevo mensaje
    dat_ant = dato;         // Actualiza dato_anterior al nuevo valor de dato
}
```

Bluetooth

Una vez que pudimos modelar nuestro sistema, y comprobar el funcionamiento de ciertos aspectos, se modificó la forma en la que nuestro sistema se iba a comunicar a través del módulo SCI. En un principio, nuestro sistema iba a comunicarse por el usuario a través de UART por cable, con una aplicación de Terminal de comunicación serial, sin embargo, con el objetivo de dotar a nuestro sistema de una mayor funcionalidad y independencia, se planteó la idea de hacer esta comunicación vía "Bluetooth".

Para ello, lo primero que necesitamos fue un módulo bluetooth, específicamente el *HC-05*. Puede ser configurado como maestro o como esclavo, y se conecta directamente a los pines de recepción y transmisión del microcontrolador, teniendo un funcionamiento parecido con la comunicación por UART.

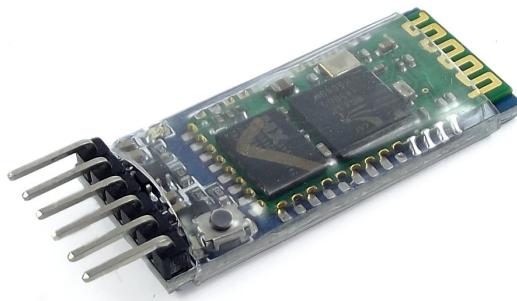


Figura 4.17 Módulo de Bluetooth HC-05.

Este módulo puede ser alimentado desde los 3V6 hasta los 6V. Cuenta con un pin de recepción (RX) y otro de transmisión (TX), y un pin de tierra. Los otros dos pines no serán utilizados. Tiene un rango de conexión de 10 metros y una velocidad de transmisión de 1200bps hasta 1.3Mbps[29].

Programación del módulo

Para la codificación de este módulo, hemos usado como base el código anteriormente mostrado para el SCI, y su archivo. Añadiendo la lectura de valores recibidos por Bluetooth para modificar el estado de nuestro sistema, y al mismo tiempo, comunicarnos con el dispositivo móvil al que puede ser conectado nuestro módulo.

Código 4.12 Comunicación Bluetooth.

```
//Mensajes para el terminal
const char * hola_msg = "*_Bienvenido!!\n\r*";
const char * conf1_msg = "*_Que modo quieres activar?\n\r*";
const char * conf2_msg = "*_c) Modo Confort\n\r*";
const char * conf3_msg = "*_a) Modo Anti-Robo\n\r*";
const char * conf4_msg = "*_\n\r*";
const char * normal_on_msg = "*_Modo confort ACTIVADO\n\r"
                            "\n\r*";
const char * robo_on_msg = "*_Modo Anti-Robo ACTIVADO\n\r"
                           "\n\r*";
const char * robo_det = "*_ROBO DETECTADO!!\n\r"
                        "\n\r*";
//Mensajes de control
const char * alarm_on = "*SV99*";
const char * alarm_off = "*SV00*";
const char * led_on = "*LR255G0B0*";
const char * led_off = "*LR0G0B0*";
const char * vent_max = "*VR0G255B0*";
const char * vent_med = "*VR255G255B0*";
const char * vent_off = "*VR0G0B0*";
const char * reset_temp = "*n_*";

// ...
interrupt void scia_rx_isr(void)
{
    if(SciaRegs.SCIRXST.bit.RXERROR) {
        SciaRegs.SCICTL1.bit.SWRESET = 0;
        SciaRegs.SCICTL1.bit.SWRESET = 1;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
        return;
    }
    SCIA_getchar(&modo);

    if(modo == 'c') {
        modo_normal = 1;
        alarm_active = 0; // Reseteo alarma cuando cambiamos a modo confort
        if (dat_nuevo) {

```

```

        SCIA_Transmit(normal_on_msg, strlen(normal_on_msg));
        dat_nuevo = 0;
    }
}
else if (modo == 'a') {
    modo_normal = 0;
    first_time = 1;
    if (dat_nuevo) {
        SCIA_Transmit(robo_on_msg, strlen(robo_on_msg));
        dat_nuevo = 0;
    }
}
else {
    alarm_active = 0;
    Mov_detect = 0;
    robo_msg_enviado = 0;
    peopl_det = 0;      // Reseteamos detec. de personas
    accel_sample_counter = 0;
    EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD;
    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    SCIA_Transmit(alarm_off, strlen(alarm_off));
    SCIA_Transmit(led_off, strlen(led_off));
}
if (modo != dat_ant) {
    dat_nuevo = 1;
    dat_ant = modo;
}
PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}

```

Bluetooth Electronics

Para el control de nuestro sistema, y el manejo y monitorización de variables, se ha utilizado la aplicación para dispositivos móviles Bluetooth Electronics, únicamente disponible en dispositivos con un sistema operativo que permita utilizar *Google PlayStore* como tienda de aplicaciones.

Una vez descargada, la conexión con nuestro módulo es extremadamente sencilla ya que es del tipo "*plug-and-play*", lo que indica que no necesita una configuración previa para comenzar a funcionar. Para la comunicación con los diferentes actuadores dentro de esta aplicación, en el momento de enviar datos a la aplicación, necesitamos que empiecen y acaben con un asterisco "*" para que pueda saber cuando empieza y cuando acaba. Igualmente para cada uno de los actuadores (p. ej. LEDs a encender), configuraremos la letra inicial para saber con cual queremos comunicarnos.

Se muestra a continuación la pantalla del usuario así como una pequeña tabla con los comandos utilizados:

Tabla 4.2 Comandos para Bluetooth Electronics.

Actuadores	Comandos en CCS
Selec. Confort (RX)	c
Selec. Antirrobo (RX)	a
Temperatura (TX)	*n...*
Estado ventilador (TX)	*VR255G0B0* (Rojo) *VR255G255B0* (Naranja) *VR0G0B0* (OFF)
Sensor proximidad (Sonido) (TX)	*SV99* (ON) *SV00* (OFF)
Sensor proximidad (LED) (TX)	*LR255G0B0* (Rojo) *LR0G0B0* (OFF)

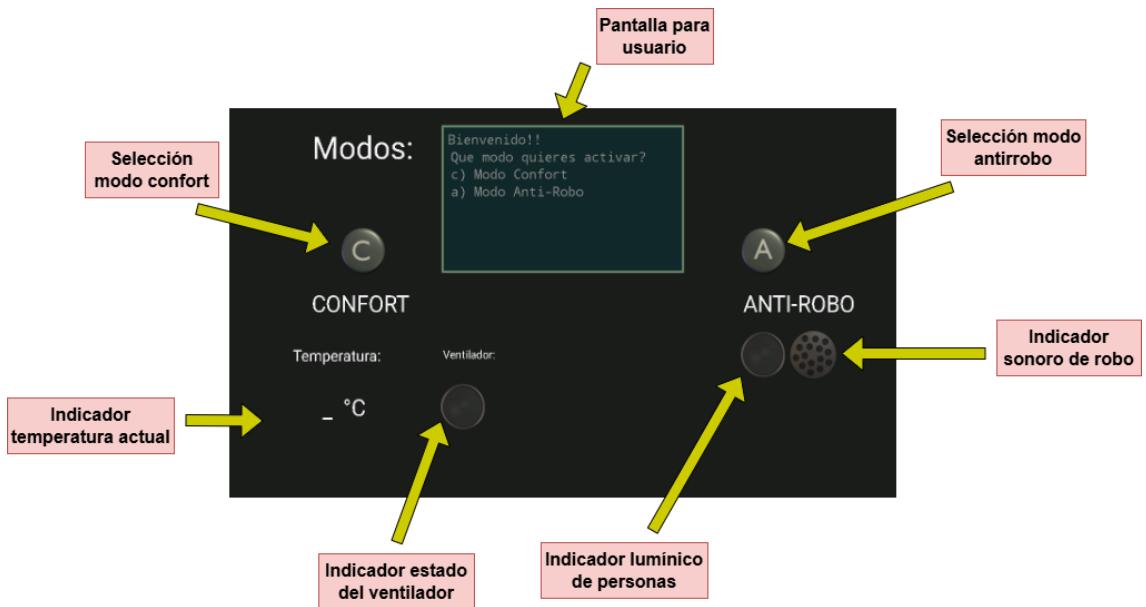


Figura 4.18 Pantalla usuario Bluetooth Electronics.

4.5 Selección y evaluación de sensores alternativos

Una vez configurado nuestro sistema, el funcionamiento es correcto, pero ampliamente mejorable. En el modo de antirrobo, ninguno de los dos sensores refleja un funcionamiento real, por ello se buscó alternativas para poder mejorar el sistema y alcanzar un funcionamiento más real.

4.5.1 Motivos de la búsqueda

Como comentado previamente, a pesar de ambos sensores estar caracterizados para nuestro sistema, no reflejaban un comportamiento real debido a las limitaciones que poseen.

En primer lugar el **shake sensor**, tiene un inconveniente que llama la atención, y es que la detección de vibraciones se reduce a un único eje, por lo que en caso de que vibre en otro eje, o incluso se mueva de forma suave, el sensor no tendría la capacidad de detectarlo.

En segundo lugar, el **grayscale sensor**, tiene a su vez, unas grandes limitaciones. Puesto que este sensor no está diseñado para la detección de personas, y simplemente detecta la intensidad de luz recibida, su caracterización y modulación para nuestro sistema se complica mucho. A ello hay que sumarle, que al tener un receptor de luz del que varía la medida, dependiendo de la luz del ambiente, e incluso del color de la superficie, no conseguimos tener un resultado fiable. En primera instancia, se buscó una solución provisional que fue la de poner este sensor boca abajo en un folio, de tal forma que la luz iba a ser siempre la misma o semejante, sin embargo, esta solución no refleja un comportamiento real.

4.5.2 Primera modificación: Acelerómetro

La primera propuesta, es el cambio del sensor de vibraciones por un acelerómetro, en este caso, en vez de detectar cambios bruscos en un eje, podemos ajustar la precisión para detectar pequeños cambios en la inclinación, en los tres ejes del espacio, con el que se consigue aumentar la eficiencia y fiabilidad de la medida.

En nuestro caso, el sensor escogido ha sido el *MMA7361 Triple Axis Accelerometer*, de la familia de DFRobot[30]. Es de tipo analógico, y tiene tres salidas independientes con las mediciones en cada uno de los ejes. Esta lectura de los datos se realiza haciendo uso del módulo ADC de nuestro DSP, y la lógica de control se efectuará en nuestro programa principal.



Figura 4.19 Sensor de aceleración escogido.

Cuenta con un selector en la parte superior con el que podremos indicar la precisión deseada, en el caso más favorable (1.5g), obteniendo una precisión de 800 mV/g. Debido a que el código para la configuración del ADC, es el mismo que el ya expuesto, únicamente se comentará la lógica de detección de movimiento y/o inclinación del sensor.

El procedimiento seguido se compone de una lectura inicial de los sensores, y la posterior comparación de los valores en cada instante de tiempo con el instante de tiempo anterior. Inicialmente, se planteó establecer el valor medio de la medida (Va desde 0 hasta 4095 por lo tanto 2048), como referencia de comparación, sin embargo, invalidaba los casos en los que el sensor inicialmente no estuviera sobre la mesa, y aún no sabiendo cual sería la disposición en el espacio de los sensores, se descartó este método.

En caso de que el sensor detecte un valor de comparación relativamente preciso, se activará la variable que indica que ha habido un movimiento anómalo.

Código 4.13 Lectura de datos del acelerómetro.

```
Vana_x = 3000.0*(float)Vbin_x/4095.0;
Vana_y = 3000.0*(float)Vbin_y/4095.0;
Vana_z = 3000.0*(float)Vbin_z/4095.0;

float delta_x = fabs(Vana_x - x_ant);
float delta_y = fabs(Vana_y - y_ant);
float delta_z = fabs(Vana_z - z_ant);
mov_total = delta_x + delta_y + delta_z;

if (accel_sample_counter >= ACCEL_SAMPLE_PERIOD) {
    x_ant = Vana_x;
    y_ant = Vana_y;
    z_ant = Vana_z;
    accel_sample_counter = 0; // Reset the counter
}

if (mov_total > 100 && Mov_detect == 0 && first_time == 0) {
    Mov_detect = 1;
}
if (mov_total < 120 && Mov_detect == 1) {
    Mov_detect = 0;
    robo_msg_enviado = 0;
}
```

Debido a que inicialmente las tres medidas se encuentran a cero, y nada más entrar en ese modo adquieren el valor medio, se procedió a retrasar el primer valor dos segundos para asegurar una medición real y evitar una falsa alarma.

Código 4.14 Contador para medida inicial del acelerómetro.

```
volatile unsigned int accel_sample_counter = 0;
const unsigned int ACCEL_SAMPLE_PERIOD = 20; // 20 * 100ms = 2 sec
// ...
interrupt void cpu_timer0_isr(void)
{
// ...
    accel_sample_counter++;
// ...
}
```

4.5.3 Segunda modificación: Sensor de infrarrojos

La segunda de las modificaciones, es el cambio del sensor de escala de grises por un sensor de infrarrojos. Este sensor en cambio es de tipo digital, y en caso de detectar personas manda una señal positiva al microcontrolador.



Figura 4.20 Sensor de infrarrojos escogido.

El nuevo sensor escogido ha sido el *SEN0018 Digital Infrared motion sensor*, también de la familia de DFRobot[31]. Su funcionamiento es simple. Una vez conectado, en uno o dos segundos, hace un escaneado del espacio en el que se encuentra, y si posteriormente detecta movimiento, manda un aviso al microcontrolador.

De forma adicional, cuenta con un pequeño potenciómetro, para ajustar la frecuencia de muestreo entre 0.5s y 50s. También tiene un jumper con el que poder escoger si el trigger se puede repetir, o una vez detectado movimiento ya lo marca indefinidamente como '1', hasta volver a ser desactivado.

Tiene un rango de detección de 7 metros, y 110°, lo que nos proporciona un grado de confianza relativamente amplio en la medida. Otro de los aspectos positivos, es su peso y tamaño, al contrario de lo que puede parecer, tiene unas dimensiones de 28mm×36mm y un peso de 25g, lo que cumple con las expectativas de nuestro sistema.

Para poder conseguir una medida fiable, y una respuesta real, en caso de detectar movimiento activa una variable auxiliar. Sin embargo, en caso de dejar de detectar movimiento durante 1 segundo, vuelve a desactivar la variable, pudiendo volver a saltar en caso de detectar movimiento.

Código 4.15 Control del sensor de infrarrojos.

```
int IR_counter = 0; // Contador de tiempo sin detección de IR
const int IR_no_motion_time = 10; // 10 * 100ms = 1 sec
char IR_active = 0; // Estado del sensor IR

// ...

interrupt void cpu_timer0_isr(void)
{
```

```

// ...
// Logica detección IR
if (GpioDataRegs.GPADAT.bit.GPIO20 == 1) {
    IR_active = 1;
    IR_counter = 0;
} else {
    IR_active = 0;
    if (IR_counter < IR_no_motion_time) {
        IR_counter++;
    }
}

// Actualizamos si hay personas o no
peopl_det = (IR_counter < IR_no_motion_time);
// ...
}

```

Sin embargo, una vez experimentado con este sensor, las pruebas realizadas con el dispositivo generan ciertas dudas de su aplicabilidad en el proyecto:

- En primer lugar, el sensor propuesto no permite la activación del modo de antirrobo in situ, ya que al alejarnos del dispositivo activaríamos nosotros mismos la alarma.
- Por otro lado, y en el caso de que el intruso no realizara ningún movimiento, la alarma dejaría de sonar, teniendo que forzar la desactivación de la misma a un mecanismo manual.

Por estas razones, se han buscado otros sensores que puedan cumplir con nuestras expectativas de forma más amplia.

4.5.4 Última modificación: Sensor de ultrasonido

Tras una búsqueda en internet, conseguimos obtener una idea de una posible solución para este segundo sensor, un sensor de ultrasonido, cuyo funcionamiento es intuitivo, y su nivel de adaptabilidad al sistema, alto.

Se basa en el principio de que las ondas ultrasónicas reflejan cuando encuentran algún obstáculo, este sensor cuenta con un emisor y un receptor de estas ondas. La medición es posible contando el intervalo de tiempo en el que la onda es emitida, y vuelve. Una vez recibida la onda, la medición de tiempo se para, y teniendo en cuenta que en el aire, estas ondas se propagan a la velocidad de la luz ($v = 340m/s$), podremos obtener con precisión la distancia a la que se encuentra el objeto.

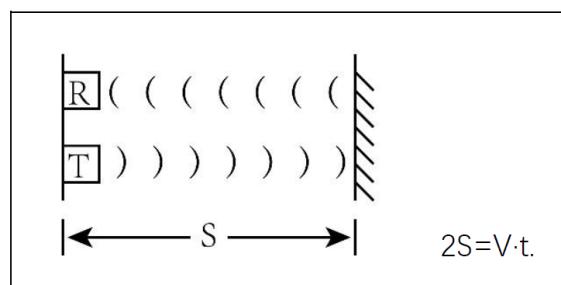


Figura 4.21 Cálculo de la distancia mediante ondas ultrasónicas.

Una vez sabido esto, el funcionamiento es simple, la mayoría de sensores ultrasónicos cuentan con dos pines (además del pin de alimentación y tierra), uno para dar el disparo o *trigger*, y otro para recibir las ondas. Para comenzar a medir, mandaremos un flanco de subida durante 10 segundos, y posteriormente lo desactivaremos. En ese momento, el pin del *Echo* se activa y cuando reciba la señal de vuelta, volverá a desactivarse, siendo la duración del estado activo del pin de *Echo*, el tiempo total de la onda ultrasónica desde la transmisión hasta la recepción ($s = v * t/2$).



Figura 4.22 Sensor HC-SR04 escogido.

En nuestro caso, el sensor escogido ha sido el *HC-SR04*, un sensor de precio reducido, y que cuenta con una alta producción, lo que ha facilitado su obtención. Su voltaje de operación es de 5V, y tiene un rango de medición desde los 2cm hasta los 400cm, con una precisión de $\pm 3\text{mm}$, y un ángulo de apertura de 15º[32].

Código 4.16 Sensor HC-SR04.

```
distance = getSonar();
if (distance <= 40) { //Alguien a menos de 40cm
    peopl_det = 1;
} else { peopl_det = 0;
}
//...
float getSonar(void)
{
    // Enviamos el pulso de trigger
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    DELAY_US(2);
    GpioDataRegs.GPASET.bit.GPIO6 = 1;
    DELAY_US(10);
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    // Medimos la duración del pulso de echo
    unsigned long pingTime = 0;
    unsigned long timeout = 0;

    while (GpioDataRegs.GPADAT.bit.GPIO7 == 0) { //Esperamos flanco de subida en echo
        timeout++;
        DELAY_US(2);
        if (timeout > 30000) return MAX_DISTANCE; // Return max distance if timeout
    }
    timeout = 0;
    while (GpioDataRegs.GPADAT.bit.GPIO7 == 1) // Medimos cuanto tiempo está alto
    {
        pingTime++;
        DELAY_US(2);
        if (pingTime > 30000) break; // 30ms timeout
    }
    // Convertimos a cm (58 = Velocidad_Sonido / 2 / 10000;)
    float distance = (float)pingTime / 58.0;
    return distance;
}
```

Es importante comentar que hemos usado la función *DELAY_US*, definida en el archivo *DSP2833x_Examples.h*, para ello, hemos cambiado el include al principio del programa por el siguiente: *#include "DSP28x_Project.h"*, que ya incluye tanto *DSP2833x_Examples.h*, como *DSP2833x_Device.h*.

De forma adicional, y tras ciertas pruebas, hemos tenido que añadir una salida de emergencia del primero de los bucles while, ya que en caso de que el sensor no se encuentre bien conectado, o el pulso de trigger no se envíe correctamente, nunca detectará el flanco de subida en Echo.

4.5.5 Diagrama de flujo

Después de la modificación de estos sensores, y su adaptación al código, quedan como definitivos, el acelerómetro, y el sensor de ultrasonido, además de los nuevos comandos para el Bluetooth. Se elimina, también, el vibrador, debido al poco uso real que puede tener en nuestro sistema. El diagrama de flujo que queda es el siguiente

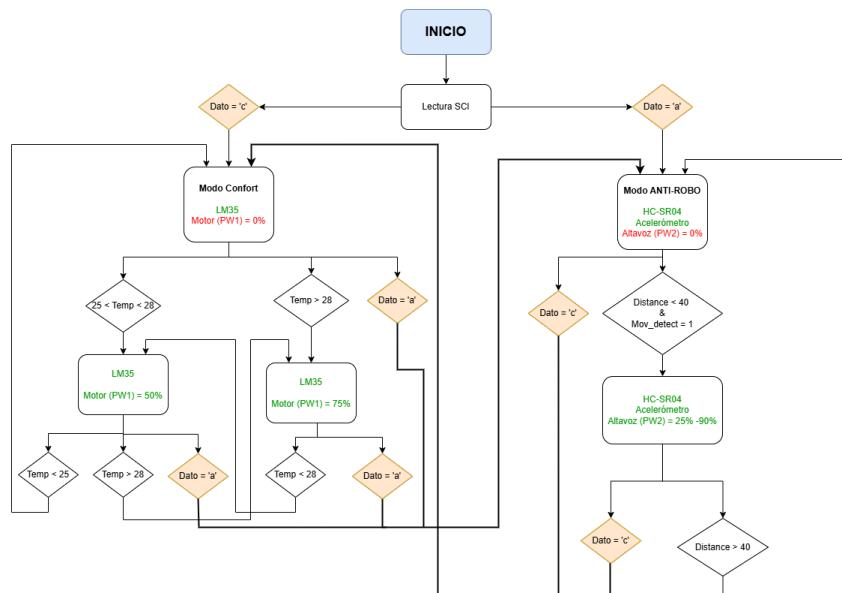


Figura 4.23 Diagrama de Flujo definitivo.

Se muestra a continuación también el diagrama de pines definitivo de nuestro sistema, con la inclusión de los nuevos sensores, y el módulo bluetooth:

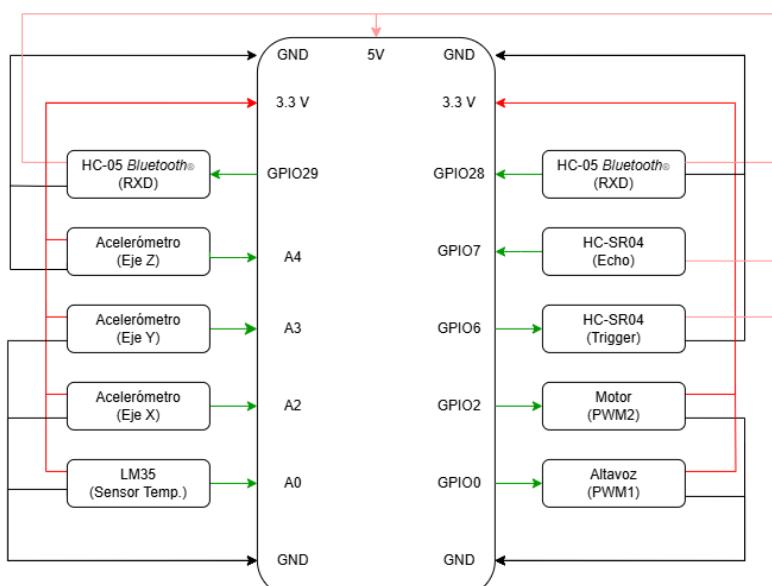


Figura 4.24 Esquema de conexionado definitivo.

4.6 Conclusión de necesidades del sistema

Para cumplir con el objetivo de este apartado, procedemos a enumerar de manera simple las necesidades de nuestro sistema, el número de pines que necesitamos, tanto analógicos como digitales, la alimentación necesaria para cada uno de ellos, y los módulos a considerar.

- Número de pines analógicos: 4 pines (1 para el sensor de temperatura y 3 para el acelerómetro)
- Número de pines digitales: 5 pines digitales (LED, Altavoz, Motor y 2 pines para el HC-SR04; Trigger y Echo)
- Alimentación necesaria: en principio 3V3 y 5V, sin embargo todos los sensores y actuadores pueden actuar ambas tensiones, reduciendo únicamente la precisión del HC-SR04 para el caso de 3V3, pudiendo añadir una calibración para asegurar que las medidas son las correctas.
- Módulos disponibles: ADC, PWM y a ser posible Bluetooth incorporado, o SCI para poder conectar el módulo HC-05.

5 Desarrollo del prototipo del sistema wearable

Todo debe hacerse tan simple como sea posible, pero no más sencillo.

ALBERT EINSTEIN, 1933

En este Capítulo se detalla el proceso de adaptación del sistema original al entorno de un dispositivo portátil o wearable, cumpliendo así con uno de los objetivos principales del proyecto. Para ello, se tendrán en cuenta las especificaciones obtenidas en el capítulo 4, y enumeradas en la sección 4.6.

A continuación se estudiarán las posibles opciones de microcontrolador de tamaño reducido, los nuevos componentes para el prototipo en caso de que los hubiera, la adaptación del código, y el montaje de la maqueta.

5.1 Selección de la placa de desarrollo

Como ha sido mencionado anteriormente, para poder hacer el prototipo de un sistema electrónico de videovigilancia, necesitaremos escoger un microcontrolador con un tamaño reducido, y las características mencionadas anteriormente.

Será necesario que cuente con 4 pines analógicos, uno de ellos para el sensor de temperatura, y los otros tres para las tres coordenadas del acelerómetro. En cuanto a pines digitales, serán necesarios 5 pines (LED, altavoz, ventilador, y dos para el HC-SR04; Trigger y Echo). Tendrá que ser capaz de alimentar a 3V3, o 5V, así como contar con módulos de ADC, PWM, y Bluetooth o SCI para poder conectar el HC-05.

Estudiaremos las principales opciones del mercado, haciendo una comparación entre ellos.

5.1.1 Raspberry Pi Pico W

La Raspberry Pi Pico W es una placa de desarrollo y de bajo consumo basada en el microcontrolador RP2040 desarrollado por Raspberry Pi. A diferencia del modelo original (Raspberry Pi Pico), el modelo Pico W añade conectividad Wi-Fi, lo que permite la creación de aplicaciones IoT inalámbricas. [33]

Basado en el RP2040, conocido por su alto rendimiento y bajo coste. Cuenta con 264 kB de SRAM y 2 MB de memoria flash. Tiene 26 pines GPIO, de los cuales 3 de ellos pueden ser utilizados como entradas analógicas.

Añade dos módulos de UART, dos controladores de SPI, y otros dos de I2C, además de 16 canales PWM. En cuanto a la conectividad, posee Wi-Fi 2.4 GHz con antena incorporada y Bluetooth 5.2, compatible con BLE y Bluetooth Classic.

La tensión de entrada puede variar desde 1.8 hasta 5V5, y tiene una temperatura de operación entre -20 °C y +70 °C. El aspecto más limitado de esta placa, son sus dimensiones, no llegando a ser totalmente compacto, siendo de 51 x 21 mm. Por último, su precio, se encuentra en los 11.90€ [34] (A partir de ahora, todos los precios incluirán gastos de envío, impuestos y han sido actualizados a día de escritura de este documento).

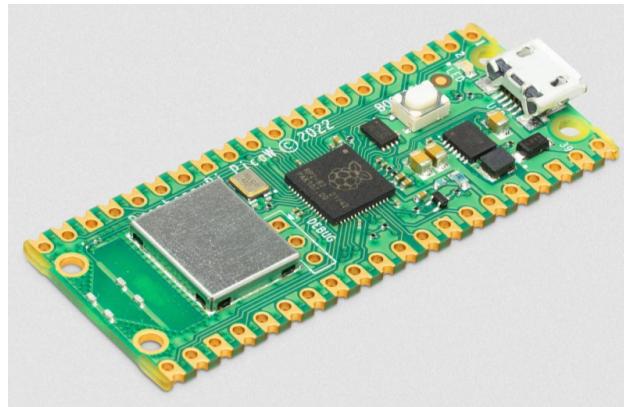


Figura 5.1 Raspberry Pi Pico W.

5.1.2 Arduino LilyPad

La LilyPad Arduino Main Board es una placa de desarrollo de bajo consumo orientada a proyectos de e-textiles y wearables, gracias a su diseño circular que permite coserla directamente sobre tejidos mediante hilo conductor. Fue diseñada por Leah Buechley en colaboración con SparkFun Electronics. Actualmente, se considera un producto retirado del mercado. [35]

Está basada en los microcontroladores ATmega168V o ATmega328V, versiones de bajo consumo de la familia ATmega. Dispone de 14 pines digitales, 6 canales PWM, y 6 entradas analógicas, lo que la convierte en una opción flexible para proyectos que requieren interacción con sensores o actuadores portátiles.

Ofrece una memoria flash de 16 KB (de los cuales 2 KB están ocupados por el bootloader), 1 KB de SRAM, y 512 bytes de EEPROM, funcionando a una frecuencia de 8 MHz. Cada pin puede entregar hasta 40 mA, y puede alimentarse con tensiones comprendidas entre 2.7 y 5.5 V.

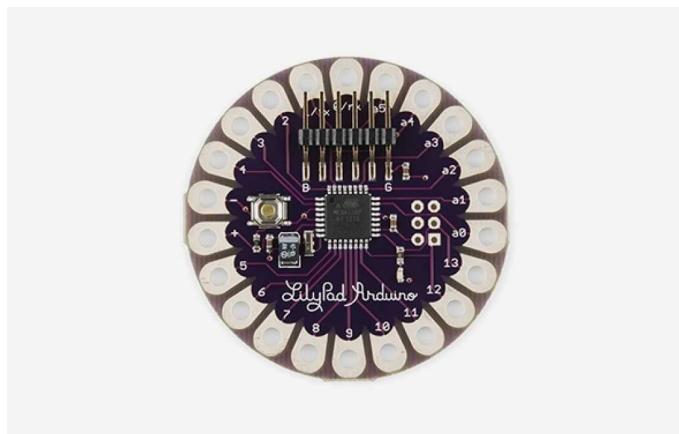


Figura 5.2 Arduino LilyPad Main Board.

El principal atractivo de la LilyPad es su adaptabilidad a entornos textiles, ya tiene una forma circular con un diámetro de 50 mm y 8 mm de espesor. Por el contrario, la mayor limitación es su retirada del mercado, a pesar de poder encontrarla en algunas tiendas de electrónica por 29,51€[36].

5.1.3 Adafruit FLORA

La Adafruit FLORA es una placa de desarrollo especialmente orientada a proyectos wearables. Diseñada por Adafruit Industries, está basada en el microcontrolador ATmega32u4, que incluye soporte USB, lo que permite su programación directa desde el ordenador sin necesidad de convertidores adicionales. [37]

Cuenta con una memoria flash de 32 KB, 2.5 KB de SRAM, y funciona a una frecuencia de 8 MHz. Dispone de 8 pines de los cuales cuatro pueden configurarse como entradas analógicas además de conectividad SPI así como un LED integrado en el pin D7 y un NeoPixel conectado al pin D8 en su versión v2. Su tamaño es de 45 mm de diámetro, lo que permite integrarla fácilmente en superficies textiles y responde al objetivo principal del proyecto.

La alimentación puede realizarse a través de un conector JST con una tensión entre 3.5 V y 9 V, contando con protección de polaridad y un regulador integrado que proporciona 3.3 V de salida (hasta 150 mA). También admite alimentación por USB (entre 4.5 V y 5.5 V), que a su vez sirve para la programación. La placa no incluye cargador de baterías LiPo, lo que permite el uso seguro con diferentes tipos de batería.

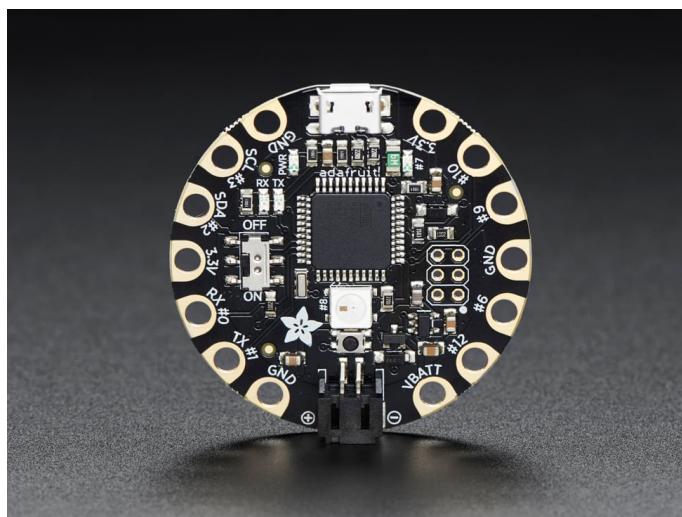


Figura 5.3 Adafruit FLORA Board.

Al igual que la placa de desarrollo del apartado 5.1.2, destaca por su tamaño y forma siendo fácilmente integrable en elementos textiles, sin embargo destacan negativamente, el bajo número de pines, así como el elevado precio de mercado encontrándose actualmente en 37,09€ a través de su página oficial[38].

5.1.4 SeeedStudio XIAO ESP32-C3

La última de las placas consideradas ha sido la SeeedStudio XIAO ESP32-C3, una versión de tamaño reducido del famoso ESP32, conocido por su gran memoria, rapidez en la ejecución de procesos, alto número de pines GPIO disponibles, e incorporación de Bluetooth y Wifi[39].

Este modelo cuenta con 400 KB de SRAM y 4MB de memoria flash, así como 11 pines digitales, todos ellos configurables como PWM, y tres de ellos pueden ser usados como pines analógicos. Además cuenta con los módulos principales tales como UART, I2C y SPI.

Proporciona conexión Bluetooth 5 (BLE) y Wi-Fi, y es capaz de alimentar a 3V3 y 5V. Está basado en el microcontrolador 32-bit RISC-v single core processor, que tiene una frecuencia máxima de 160 MHz, destacando así por su rapidez. Sus medidas son 25 x20 mm, siendo así el más pequeño de los hasta ahora estudiados. Su precio se encuentra en 7,02€[40].



Figura 5.4 SeeedStudio XIAO ESP32-C3.

5.1.5 Comparación entre las diferentes opciones y elección final

Se muestra a continuación una tabla comparativa entre las diferentes opciones, destacando en negrita la que tenga mejores condiciones para cada característica.

Tabla 5.1 Comparativa entre distintas placas de desarrollo.

Características	Raspberry Pi Pico W	Arduino LilyPad	Adafruit FLORA	SeeedStudio XIAO ESP32-C3
Microcontrolador	RP2040	ATmega168V / AT-mega328V	ATmega32u4	ESP32-C3 (RISC-V)
Frecuencia de reloj	133 MHz	8 MHz	16 MHz	160 MHz
Memoria SRAM	264 KB	1 KB	2.5 KB	400 KB
Memoria Flash	2 MB	16 KB	32 KB	4 MB
Pines GPIO	26 (3 analógicos)	14 (6 analógicos)	8 (4 analógicos)	11 (3 analógicos)
PWM	16 canales	6 canales	4 canales	11 canales
Módulos de comunicación	UART(2), SPI(2), I2C(2)	UART, SPI, I2C	UART, SCI	UART, SPI, I2C
Conectividad	USB, Wi-Fi 2.4GHz, Bluetooth 5.2	No	USB	USB, Wi-Fi 2.4GHz, Bluetooth 5 (BLE)
Tensión de entrada	1.8 - 5.5 V	2.7 - 5.5 V	3.5 - 9 V (USB 4.5 - 5.5 V)	3.3 V - 5 V
Tensión de salida	3V3	2.7 - 5.5 V	3V3 (máx 100 mA)	3.3 V (máx 700 mA) y 5 V
Dimensiones	51 x 21 mm	50 mm Ø x 8 mm	45 mm Ø	25 x 20 mm
Precio	11,90 €	29,51 €	37,09 €	7,02 €

Tras analizar las opciones comentadas, se decidió continuar con la placa SeeedStudio XIAO ESP32-C3 (a partir de ahora ESP32-C3), debido a su reducido precio, gran capacidad de memoria, variedad en la alimentación, posibilidad de conexión Wi-Fi y Bluetooth, y sobre todo, sus reducidas medidas, pudiendo así, hacer un prototipo lo más portátil posible.

Como desventaja de esta elección, tendremos que renunciar a uno de los pines analógicos, siendo

escogida la componente Z del acelerómetro. De esa forma podremos medir variaciones en un plano de 2-D, y quedando como posible ampliación la búsqueda de un acelerómetro con conexión I2C o digital.

Para la codificación de esta placa de desarrollo, se presentan dos opciones, Arduino, o MicroPython, quedando pendiente seleccionar posteriormente el lenguaje más adecuado.

5.2 Elección e integración de nuevos componentes

Tras la adaptación del sistema a la nueva placa de desarrollo, se han añadido nuevos componentes, y se han reducido algunos de ellos.

Módulo Bluetooth HC-05

El primero de los componentes en ser sustituido ha sido el módulo de Bluetooth HC-05, explicado en la sección 4.4.3. La motivación para ello, ha sido la búsqueda de una placa de desarrollo con sistema de conexión inalámbrica integrada en el sistema. Esta solución nos permitirá un abanico más amplio de opciones para la interfaz de comunicación con el sistema, pudiendo evitar el software utilizado en el proceso de evaluación de necesidades, conocido como *Bluetooth Electronics*.

Una vez establecido el protocolo de comunicación con la placa, figurará como un dispositivo más, con la ventaja de poder comunicarnos con una página web a través de este método.

Pantalla LCD

Una vez estamos en el proceso de adaptación del código, motivado por la ayuda para la depuración, conseguimos a través de esta, monitorizar el estado de nuestro sistema y de las variables usadas para el control del ambiente.

El modelo escogido ha sido un LCD1602, de la marca *Freenove*, una pantalla de cristal líquido ampliamente utilizada en proyectos electrónicos debido a su simplicidad y versatilidad. Tiene dos filas con 16 columnas cada una, permitiendo mostrar números, letras, símbolos y caracteres ASCII.

Cuenta con múltiples conexiones (16 pines), sin embargo al utilizar un módulo adaptador basado en comunicación I2C, conseguimos reducir significativamente el cableado. La interfaz I2C utilizará solo dos líneas de comunicación principales:

- SDA (Serial Data Line): Línea que lleva los datos.
- SCL (Serial Clock Line): Línea que transmite el pulso del reloj sincronizador.

El chip del módulo I2C serie-paralelo utilizado en este módulo es el PCF8574T (PCF8574AT)[41], siendo su dirección predeterminada 0x27(0x3F). Se consigue así convertir los datos serie recibidos por la interfaz I2C a señales paralelas, simplificando la comunicación con el microcontrolador. De este modo, el manejo de las 16 líneas originales del LCD se reduce a solo cuatro conexiones: alimentación (VCC y GND) y las dos líneas I2C (SDA y SCL).



Figura 5.5 LCD1602.

Dentro de las pantallas LCD1602, se comercializan dos tipos de ellas, una simple, con la propia pantalla y el chip I2C, y otra que de forma adicional incluye un potenciómetro para poder controlar la retroiluminación, este último ha sido el modelo utilizado. Por otra parte, con el objetivo de la simplificación del código y su legibilidad, se ha usado una librería externa llamada *LiquidCrystal I2C*.

5.3 Elección del software de desarrollo

A la hora de pensar en la adaptación del código surge la primera decisión a tomar, y es la de la elección del lenguaje que será usado. Nuestra placa, la ESP32-C3, puede ser codificada tanto usando el lenguaje Arduino, como MicroPython. Por ello, previamente, se hará una pequeña comparativa entre ambos.

Tabla 5.2 Comparativa entre Arduino y MicroPython.

Aspecto	Arduino	MicroPython
Lenguaje	C/C++ simplificado	Python simplificado
Curva aprendizaje	Moderada-fácil (más bajo nivel)	Muy fácil (más alto nivel)
Velocidad	Alta (compilado)	Media (interpretado)
Facilidad de uso	Alta	Muy alta
Compatibilidad	Gran cantidad de bibliotecas y ejemplos	Buena, pero menos bibliotecas
Comunidad	Amplia y consolidada	Creciendo rápidamente
Facilidad de depuración	Más difícil, requiere técnicas específicas	Más fácil debido al intérprete directo

Teniendo en cuenta lo arriba mencionado, se ha optado por elegir el **lenguaje de programación Arduino**, principalmente debido a su semejanza con el código realizado en la evaluación de necesidades (Lenguaje C), su gran cantidad de bibliotecas, su velocidad de compilación, y la gran comunidad, especialmente relevante para una posible ampliación futura.

5.4 Diagrama de pines ESP32-C3

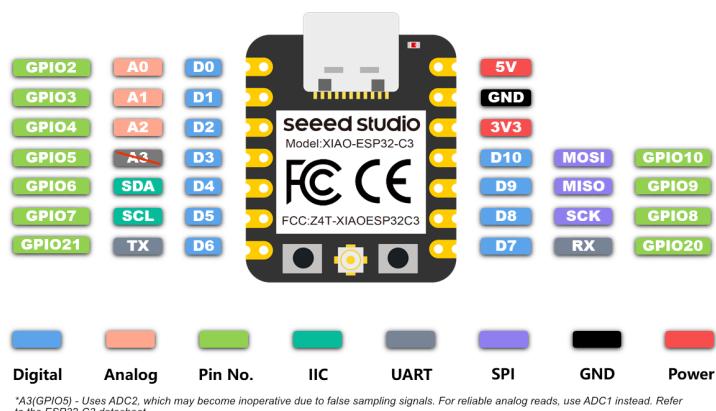


Figura 5.6 Diagrama de pines ESP32-C3.

El siguiente paso será el de la elección de pines para cada uno de los actuadores y sensores utilizados en la evaluación de necesidades, así como los utilizados únicamente para el prototipo. Tendremos en cuenta los pines propios de la placa con una función asignada (I2C y analógicos) y el

resto se asignarán de forma arbitraria pero permitiendo simplificar al máximo el diseño de la PCB y la ubicación de los componentes en la misma.

Un aspecto importante a comentar es que inicialmente, el ESP32-C3, contaba con 4 pines analógicos, sin embargo no se recomienda el uso de uno de ellos como analógico pero si como digital, reduciendo el número de pines disponibles para señales analógicas a 3, y motivando así la elección de no tener en cuenta uno de los ejes del acelerómetro.

La razón de no usar los 4 pines analógicos, es que uno de ellos, el pin A3, utiliza el ADC2 [42], el segundo de los convertidores con los que cuenta la placa, no estando calibrado de fábrica. Tras experimentar usando este canal, no se obtienen medidas razonables y se optó por descartar su uso.

Una vez analizado el diagrama de pines, se decidió asignar los pines GPIO6 y GPIO7, para la comunicación I2C de la pantalla, al mismo tiempo que se asignan el resto de pines únicamente con el objetivo de poder colocar los componentes de la forma más eficiente y con las pistas de menor tamaño en el PCB.

Inicialmente, la alimentación de todos los sensores y actuadores, será desde el pin de los 5V. Esta alimentación, viene directamente distribuida desde la alimentación recibida por la placa a partir del USB como puede observarse en el esquemático[43]. Se muestra por tanto, a continuación, el diagrama de asignación de pines definitivo.

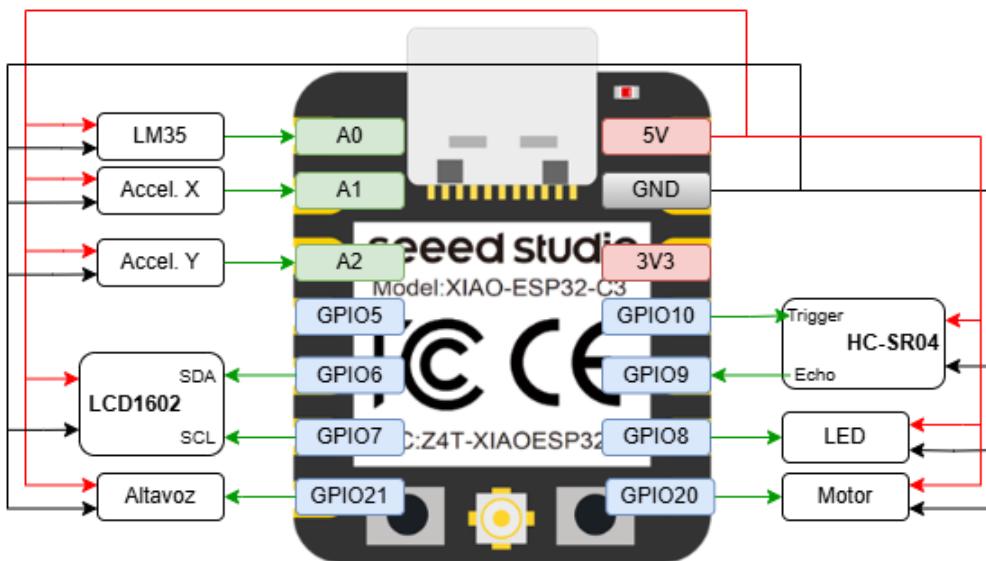


Figura 5.7 Asignación de pines ESP32-C3.

5.5 Funcionamiento del sistema

El sistema diseñado opera en dos modos principales: el **Modo Confort** y el **Modo Antirrobo**, controlados mediante comandos recibidos ahora a través de una conexión Bluetooth. Cada modo responde a condiciones específicas detectadas por sensores, detallándose a continuación los aspectos modificados y resumidos respecto al diseño original. A diferencia de como ocurría en la evaluación de necesidades con el F28335, no hay un modo predefinido, ofreciendo al usuario un mensaje de bienvenida, y esperando la selección de uno de los modos. Se muestra a continuación un breve resumen del funcionamiento, así como una mención especial a los cambios implementados.

Modo Confort

El *modo Confort* mantiene su objetivo de control ambiental mediante el sensor de temperatura LM35. Este modo se activa al recibir el comando ‘c’ vía Bluetooth. El funcionamiento es similar al original, salvo los siguientes cambios en el control de temperatura:

- $T < 28^{\circ}\text{C}$: Motor apagado (0 %).
- $28^{\circ}\text{C} \leq T < 32^{\circ}\text{C}$: Motor al 50 %.
- $T \geq 32^{\circ}\text{C}$: Motor al 75 %.

Estos cambios han sido motivados por el aumento de la temperatura durante la realización de este proyecto.

Modo Antirrobo

El *Modo Antirrobo* se activa al recibir el comando ‘a’ por Bluetooth. Este modo mantiene la estructura general del diseño original, mediante una monitorización del ambiente realizada por un sensor ultrasónico HC-SR04 y un acelerómetro analógico. Su operación se resume en lo siguiente:

- **Inicio:** Igual al diseño original, con actuadores inicialmente apagados.
- **Activación manual de la alarma:** Para que se genere una señal de alarma, el usuario debe activarla manualmente mediante un comando Bluetooth. Igualmente, puede desactivarla sin necesidad de cambiar de modo.
- **Condiciones de alarma:** La alarma se activa únicamente si:
 - El sensor ultrasónico HC-SR04 detecta una persona a menos de 50 cm.
 - Simultáneamente, el acelerómetro analógico detecta movimiento.
- **Respuesta de alarma:** Permanece igual al diseño original (altavoz con señal PWM y LED intermitente).
- **Transición y desactivación automática:** La señal de alarma únicamente se desactiva si el sensor ultrasónico deja de detectar presencia dentro del rango de 50 cm. El modo antirrobo se mantiene activo hasta recibir nuevamente el comando ‘c’ mediante Bluetooth, volviendo entonces al *Modo Confort*.

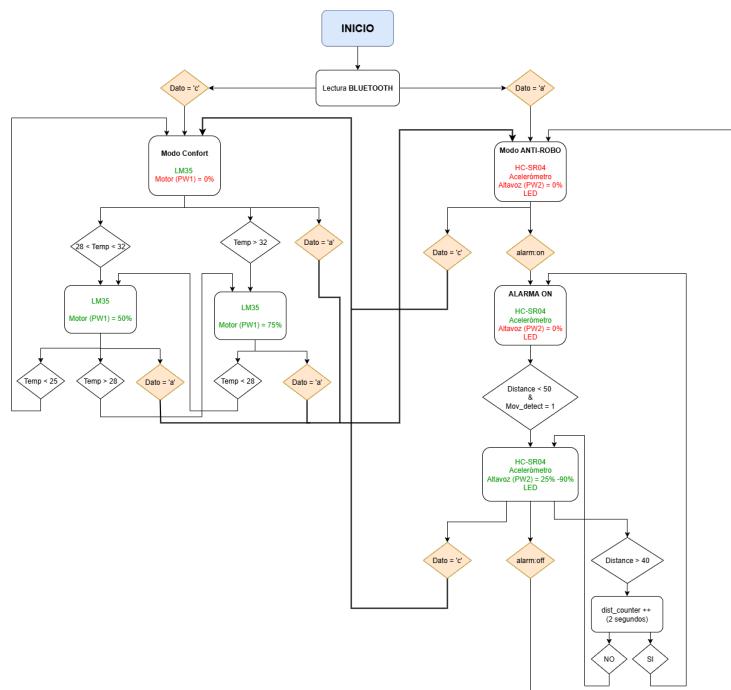


Figura 5.8 Diagrama de flujo.

De forma adicional, el usuario, puede monitorizar el estado mediante el LCD, el cual muestra el modo actual, así como la temperatura en el modo de confort, y en el modo antirrobo el estado de la alarma, (encendida o no, y si se ha activado). Con estos últimos cambios, conseguimos un funcionamiento del sistema acorde al esperado, asemejándose lo máximo posible a un sistema y circunstancias reales.

5.6 Implementación

5.6.1 ADC

El primero de los módulos a implementar será el convertidor analógico-digital (ADC). El microcontrolador, integra dos convertidores 12-bit SAR ADC, es decir, de registros de aproximación sucesiva, muy común en este tipo de sistemas, conocido por su rapidez y eficiencia.

Cuenta con una resolución de 12bits, es decir con valores entre 0-4095, sin embargo solo uno de ellos, el ADC1, esta calibrado de fábrica, lo que nos limita el número de pines de entrada analógica a 3, como ya ha sido comentado.

Además, será importante tener en cuenta cual es el voltaje efectivo que podremos obtener en la lectura del ADC. En el propio datasheet, especifica que dependiendo de la atenuación, se puede asegurar la linealidad del convertidor hasta 2.8V como máximo. Por defecto, viene configurada una atenuación de 11dB, pudiendo obtener así el valor máximo de esta medida. En algunos documentos más recientes la atenuación máxima es de 12 dB e incluso se garantiza la linealidad hasta los 3V3 [44]. Sin embargo, independientemente de esto, la influencia en nuestro sistema es prácticamente nula, debido a que el voltaje máximo del sensor de temperatura es de 1.5V y en el acelerómetro solo necesitaremos medir grandes variaciones, no obtener el valor exacto en cada ángulo.

Un aspecto considerable a estudiar, ha sido analizar cual es la salida de cada uno de los sensores para asegurar no tener problemas e introducir más tensión de la que nuestro microcontrolador puede manejar (funciona a 3V3).

El primero de ellos, el LM35, fue identificado rápidamente, ya que a partir de la página de información del componente [22], sabemos que tiene una sensibilidad de 10mV/°C, siendo la salida proporcional a la temperatura. Sin embargo, teniendo en cuenta que su rango de operación se encuentra entre 0 y 150°C, la máxima tensión que podría aportar, como ya ha sido comentado es de 1.5V, seguro para el sistema.

La lectura de este sensor se hace de la siguiente manera, utilizando las bibliotecas a nuestra disposición en Arduino y obteniendo ya directamente un valor natural analógico.

Código 5.1 Lectura del LM35 en Arduino.

```
float readTemperature() {
    uint16_t val = analogRead(TEMP_PIN);
    return ((double)val * (3.3 / 4095.0) * 100.0); // 10mV/C, 12-bit ADC, 3.3V ref
}
```

El otro sensor, es el acelerómetro, en ambas señales, tanto para el eje X, como el eje Y. En la página del producto, se especifica como puede tener un voltaje de entrada entre 3V3 y 8V [30]. Aunque este voltaje inicialmente no podría ser leído por el ADC, el propio componente tiene un regulador LDO fijo de 3V3, por lo que la salida estará siempre limitada a ese voltaje, y no supondrá riesgo alguno.

Código 5.2 Lectura del acelerómetro en Arduino.

```
Vbin_x = analogRead(ACCEL_X_PIN);
Vbin_y = analogRead(ACCEL_Y_PIN);
Vana_x = 3300.0 * (float)Vbin_x / 4095.0;
Vana_y = 3300.0 * (float)Vbin_y / 4095.0;
float delta_x = fabs(Vana_x - x_ant);
float delta_y = fabs(Vana_y - y_ant);
mov_total = delta_x + delta_y;
```

5.6.2 PWM

En nuestro nuevo sistema, la modulación por ancho de pulso es relativamente más simple que en el caso del F28335. El ESP32-C3, permite controlar cualquiera de los pines digitales por este método con hasta un máximo de 6 canales distintos a un mismo tiempo[45].

Por el contrario del DSP anterior, en este caso el PWM, es vagamente configurable, únicamente podremos configurar la frecuencia y la resolución, siendo el canal, escogido de forma aleatoria entre los disponibles. Usando el comando *ledcAttach(pin, pwmFreq, pwmResolution)*, podemos escoger el pin que queremos controlar, la frecuencia, que en nuestro caso estará fijada para un valor de 5 kHz y la resolución, que para este proyecto será de 8 bits, es decir entre 0 y 255, siendo 255 el 100% del duty cycle.

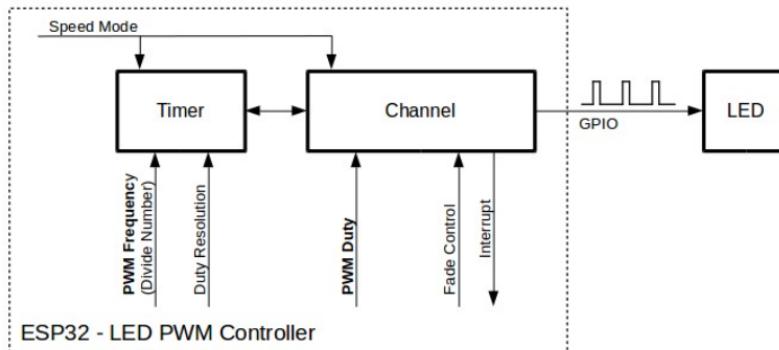


Figura 5.9 Estructura módulo PWM.

Es importante comentar que el nombre del módulo es por decisión del fabricante y no limita su uso únicamente a LEDs. Se muestra a continuación la programación del mismo en el código:

Código 5.3 Configuración PWM.

```
void setup() {
  //...
  ledcAttach(PWM_FAN, pwmFreq, pwmResolution); // 5kHz, 8-bit resolution
  ledcAttach(PWM_SPEAKER, pwmFreq, pwmResolution); // 5kHz, 8-bit resolution

  ledcWrite(PWM_FAN, 0); // Channel 0, 0% duty cycle
  ledcWrite(PWM_SPEAKER, 0); // Channel 1, 0% duty cycle
}
//...
void loop(){
  //...
  if (temp >= 30 && temp <= 32) {
    fanLevel = 1;
    ledcWrite(PWM_FAN, 128); // Channel 0, 50% duty cycle;
  } else if (temp > 32) {
    fanLevel = 2;
    ledcWrite(PWM_FAN, 255); // Channel 0, 100% duty cycle;
  } else {
```

```

    fanLevel = 0;
    ledcWrite(PWM_FAN, 0); // Channel 0, 0% duty cycle;
}
}

```

Para el altavoz, ya que únicamente queremos un sonido similar al de una alarma, su valor variará entre 20 % y 70 %, y para el caso del ventilador, está activado a nivel bajo (apagado), medio (50 %) y alto(100 %)

5.6.3 Bluetooth Low Energy

El siguiente de los módulos a explicar, será el de la conexión Bluetooth, uno de los grandes atractivos para la elección de esta placa de desarrollo. Este es el Bluetooth Low Energy (BLE) que integra un controlador link layer de hardware, un bloque RF/módem y varios protocolos software. Es compatible con las funciones básicas de Bluetooth 5.

Radio Bluetooth LE y capa física (PHY)

La radio del ESP32-C3, opera a una frecuencia de 2,4 GHz, y soporta varias velocidades de transmisión; 1 Mbps estándar, 2 Mbps para mayor velocidad, y modos de largo alcance a 125 Kbps o 500 Kbps. El dispositivo puede seleccionar entre las varias antenas con el uso de interruptores RF controlados por los GPIO para optimizar la calidad de la señal y el alcance.

Controlador de Enlace (Link layer controller)

Es el encargado de gestionar los protocolos básicos de comunicación BLE, tales como la publicidad (advertising), el escaneo (scanning), el establecimiento de conexiones y la transferencia de datos. Incluye además funciones de cifrado y seguridad a nivel de enlace.

Procedimiento de comunicación

A continuación se muestra una breve explicación del mecanismo de comunicación BLE:

1. Advertising y escaneo: En general, los dispositivos transmiten periódicamente paquetes publicitarios con información de identificación y servicios
2. Conexión e intercambio de datos: Una vez el dispositivo detecta a otro, puede iniciarse la conexión, en el caso del ESP32-C3, permite la conexión simultánea a varios dispositivos al mismo tiempo.
3. Técnicas de adaptación: Mientras se encuentra conectado, el dispositivo emplea saltos de frecuencia adaptativos y evaluación de canales para evitar interferencias en la banda de 2.4 GHz

A continuación se procederá a explicar la codificación de este módulo de comunicación en nuestro sistema. En primer lugar, será necesario, usar las librerías disponibles para poder implementar un servidor BLE en el ESP32-C3.

Código 5.4 Librerías BLE.

```

#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

```

El siguiente paso será el de definir las variables y los objetos BLE, donde *pServer* será el objeto que representa al servidor BLE, *pControlCharacteristic*; la característica BLE para enviar y recibir datos, *deviceConnected* y *oldDeviceConnected*; los flags para detectar conexiones y desconexiones y por último *SERVICE_UUID* y *CONTROL_CHAR_UUID* que serán identificadores únicos para distinguir el servicio y la característica BLE

Código 5.5 Variables y objetos BLE.

```

BLEServer* pServer = nullptr;
BLECharacteristic* pControlCharacteristic = nullptr;
bool deviceConnected = false;
bool oldDeviceConnected = false;

// UUIDs
#define SERVICE_UUID      "19b10000-e8f2-537e-4f6c-d104768a1214"
#define CONTROL_CHAR_UUID "19b10002-e8f2-537e-4f6c-d104768a1214"

```

El siguiente paso, sera el de definir los **callbacks**. Estos serán funciones definidas que serán entregadas a otra parte del código, en este caso librería para que la ejecute automáticamente cuando ocurre un evento.

Hay varios tipos de ellos, el primero el *callback* del servidor, donde se definirá que hacer cuando un dispositivo se conecta o desconecta, cambiando los flags para que el programa pueda detectar el cambio de estado.

Otro, será el *callback* de la característica de control, donde se define que sucede cuando desde el control remoto se envía un comando a la característica de control. Un claro ejemplo de este tipo es el del cambio de modo, o la activación o desactivación manual de la alarma

Código 5.6 Callbacks BLE.

```

class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) override { deviceConnected = true; }
    void onDisconnect(BLEServer* pServer) override { deviceConnected = false; }
};

class ControlCallbacks : public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic* characteristic) override {
        String value = pControlCharacteristic->getValue();
        // ... Lógica de control...
        updateLCDStatus();
        lcd_counter = 0;
    }
};

```

Continuaremos con la función de inicialización propia del BLE, donde configuraremos el dispositivo BLE, bajo el nombre *ESP32*, crearemos un servidor BLE y un servicio con el UUID definido previamente. Además será necesario una característica que pueda recibir datos (write) y enviar notificaciones (notify), asociando el callback de control a esta característica. Por otro lado añadiremos el descriptor BLE estándar (BLE2902) el cual nos permitirá activar o desactivar las notificaciones. Y por último iniciaremos el servicio y anunciaremos el BLE, para que sea visible a otros dispositivos y estos puedan conectarse a él.

Código 5.7 Función de inicialización BLE.

```

void setupBLE() {
    BLEDevice::init("ESP32");
    pServer = BLEDevice::createServer();

    BLEService *pService = pServer->createService(SERVICE_UUID);

    pControlCharacteristic = pService->createCharacteristic(
        CONTROL_CHAR_UUID,
        BLECharacteristic::PROPERTY_WRITE | BLECharacteristic::PROPERTY_NOTIFY
    );
    pControlCharacteristic->setCallbacks(new ControlCallbacks());
    pControlCharacteristic->addDescriptor(new BLE2902());

    pService->start();
    BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
    pAdvertising->addServiceUUID(SERVICE_UUID);
}

```

```

pAdvertising->setScanResponse(false);
pAdvertising->setMinPreferred(0x0);
BLEDevice::startAdvertising();
}

```

Una vez dentro de la función `voidloop()`, el ESP32-C3 enviará notificaciones a través de Bluetooth con información de los diferentes estados, dependiendo del modo en el que se encuentre nuestro sistema, para que puedan ser recibidos por el usuario en tiempo real.

Código 5.8 Uso de BLE dentro de loop().

```

// En el modo confort, cada cierto tiempo:
pControlCharacteristic->setValue(tempStr);
pControlCharacteristic->notify();
// ...
pControlCharacteristic->setValue(fanStatus.c_str());
pControlCharacteristic->notify();

// En el modo antirrobo, si cambia el estado de la alarma:
pControlCharacteristic->setValue(motionJson.c_str());
pControlCharacteristic->notify();

```

Por último, se realizará la gestión de las conexiones BLE, donde se permite que si se desconecta el usuario, el ESP32-C3, vuelve a quedar disponible para nuevas conexiones, con el siguiente código.

Código 5.9 Gestión de conexiones BLE.

```

if (!deviceConnected && oldDeviceConnected) {
    // Se desconectó el dispositivo, volvemos a anunciar el BLE
    pServer->startAdvertising();
    oldDeviceConnected = deviceConnected;
}
if (deviceConnected && !oldDeviceConnected) {
    // Se conectó un dispositivo
    oldDeviceConnected = deviceConnected;
}

```

Con este procedimiento, conseguiremos crear un servidor BLE en el ESP32-C3, definir un servicio y una característica para el intercambio de datos, enviar notificaciones sobre el estado del sistema, y gestionar la reconexión de forma automática en caso de desconexión.

5.6.4 LCD

Continuaremos con otro de los aspectos a configurar, la pantalla LCD, en nuestro caso el modelo LCD1602. Este, cuenta con un adaptador I2C, permitiendo reducir el número de salidas de 16 a únicamente 2, como ha sido comentado previamente, (SCA Y SCL). Para su implementación, usaremos la librería disponible <[LiquidCrystal_I2C.h](#)>, ofrecida por el propio fabricante.

Inicialmente, configuraremos la pantalla, comprobando que se encuentra en la dirección esperada (0x27), inicializa el LCD con el comando `lcd.init()`, enciende la retroiluminación y llama a la función principal de uso de la pantalla.

Esta función, limpiará y actualizará las dos líneas disponibles del LCD, según el modo en el que se encuentre y será llamada de forma recurrente en nuestro programa en la función `loop()` para poder actualizar la pantalla. Es importante comentar, que la tasa de refresco de la pantalla, se ha definido en 500 ms para evitar parpadeos rápidos y mejorar la legibilidad.

Código 5.10 Pantalla LCD.

```
// LCD INIT
if (!i2CAddrTest(0x27)) {
    lcd = LiquidCrystal_I2C(0x3F, 16, 2);
}
lcd.init();
lcd.backlight();
updateLCDStatus();
\\...
void updateLCDStatus() {
    lcd.setCursor(0, 0);

    if (currentMode == MODE_CONFORT) {
        lcd.print("Modo Confort ");
        lcd.setCursor(0, 1);
        lcd.print("Temp: ");
        float temp = readTemperature();
        lcd.print(temp, 1);
        lcd.print((char)223); // Símbolo de grado
        lcd.print("C ");
    } else if (currentMode == MODE_ANTIRROBO) {
        lcd.print("Modo Antirrobo ");
        lcd.setCursor(0, 1);
        if (!alarmOn) {
            lcd.print("Alarm OFF      ");
        } else if (alarm_triggered) {
            lcd.print("ALERT!!      ");
        } else {
            lcd.print("Mov: ");
            lcd.print(mov_total, 1);
            lcd.print("      ");
        }
    } else {
        lcd.print("Bienvenido      ");
        lcd.setCursor(0, 1);
        lcd.print("Seleccione modo ");
    }
}
```

5.6.5 HTML

Para poder visualizar el proyecto y poder interactuar con la placa de desarrollo, se decidió crear un archivo de tipo *.html*, donde el usuario pueda conectarse al dispositivo, seleccionar el modo de funcionamiento y poder observar el estado de variables como la temperatura, el estado del ventilador y si se ha activado la señal de alarma o no, además de la interacción directa pudiendo modificar el estado del ventilador, o activar y/o desactivar manualmente el modo antirrobo. Para el diseño de esta página web, se ha usado como referencia un proyecto disponible en internet y realizado por Rui Santos en *Random Nerd Tutorials*[46].

Este archivo HTML constituye una interfaz web para la monitorización y control de un sistema basado en ESP32-C3. Integra un panel principal donde el usuario puede conectarse al dispositivo por Bluetooth. La página permite seleccionar entre dos modos de funcionamiento (“Confort” y “Antirrobo”), mostrando para cada uno información relevante: temperatura del ambiente, control del ventilador o estado de la alarma y detección de movimiento. Además, incluye una consola de eventos donde se registran las acciones y mensajes del sistema.

El JavaScript integrado gestiona la lógica de conexión con el ESP32 mediante Web Bluetooth, permitiendo el envío de comandos y la recepción de datos en tiempo real. Dependiendo de la información recibida, la interfaz es actualizada para reflejar el estado actual del sistema y sus diferentes variables. Aunque esta interfaz web facilita la interacción y visualización para el usuario final, se considera un complemento al núcleo del proyecto, centrado principalmente en el desa-

rrollo del software y hardware del sistema y las comunicaciones entre el microcontrolador y los sensores/actuadores.

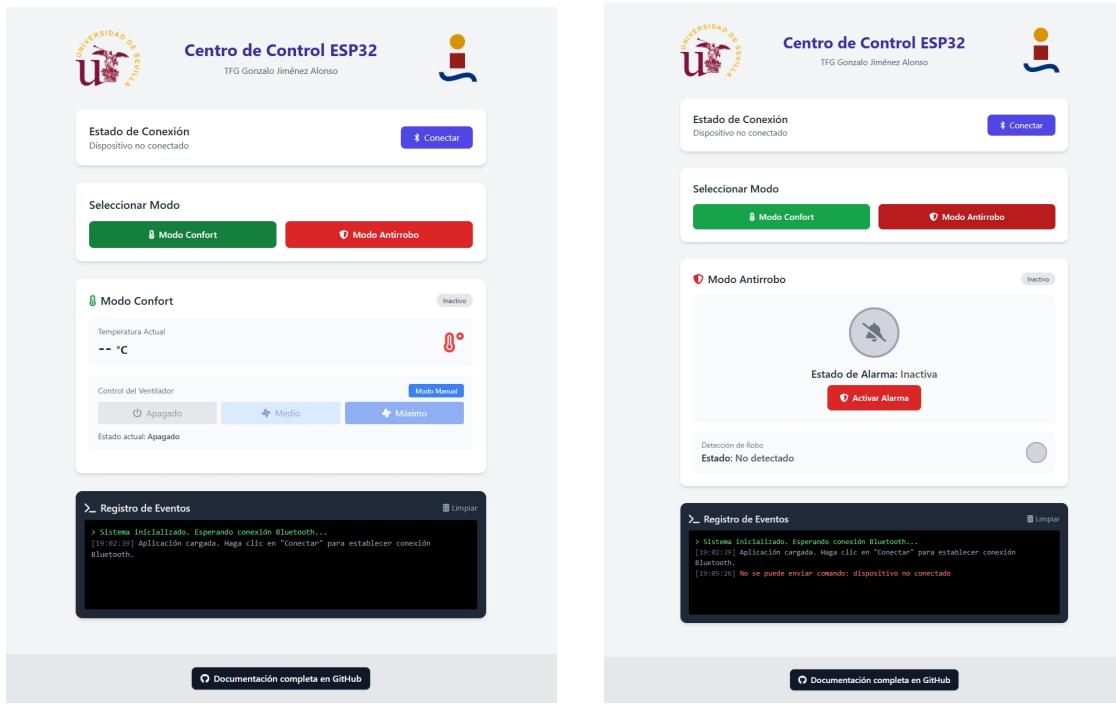


Figura 5.10 Web para control del usuario.

5.7 Diseño de la PCB

El siguiente paso, fue el diseño de la PCB, para poder evitar usar una placa de pruebas, y tener el sistema de la forma más compacta posible. Para ello, se ha usado el software KiCad, de licencia gratuita, que nos permitirá realizar el esquemático de forma sencilla.

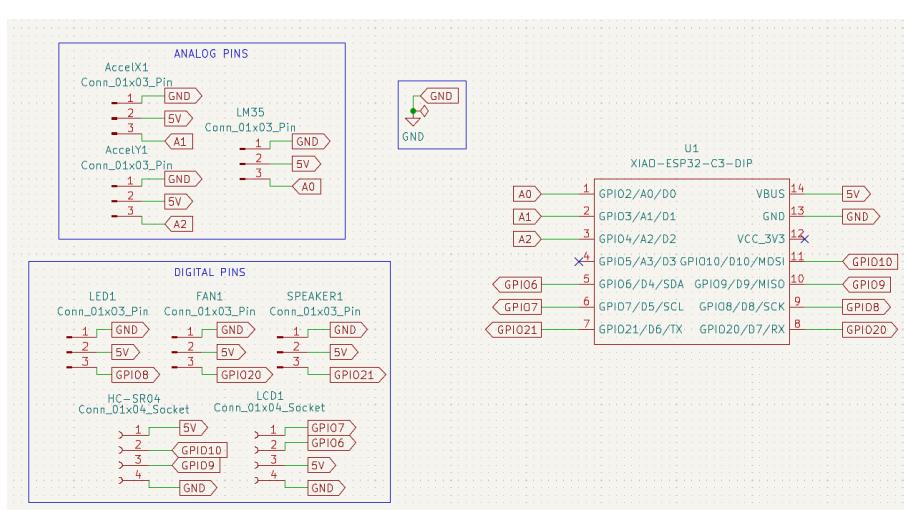


Figura 5.11 Esquemático en KiCad.

Para el diseño del esquemático, se ha utilizado el símbolo (symbol) disponible en la página web

del fabricante del ESP32-C3, y se han realizado las conexiones con los pines deseados así como con alimentación y tierra teniendo en cuenta que los conectores de los sensores, son todos conectores hembra de 3 pines, excepto el LCD y el HCSR-04, que también necesitan alimentación y tierra.

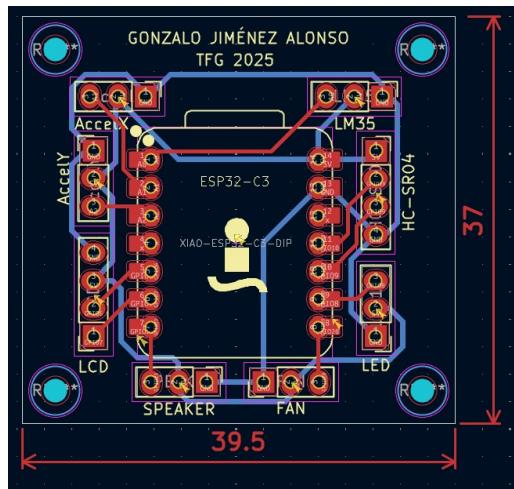


Figura 5.12 Diseño de la PCB.

Tras comprobar con el ERC (Electrical Rule Checker) que todas las conexiones son correctas, se pasa al diseño propio de la PCB conectando los pines entre ellos. Se ha usado un ancho de pista de 0.35 mm para las señales y 0.5 mm para la alimentación y tierra. Es importante comentar que toda la información para los sensores, provendrá del pin de 5V, alimentación extraída directamente de la señal del USB.

Además con el objetivo de hacer las pistas lo más cortas posibles, y evitar cruces entre ellas, se ha restringido la capa superior para señales, y la capa inferior para alimentación y tierra.

Y por último, se comprobó que el diseño de las pistas era eficiente y cumplía todas las reglas exigidas por el fabricante de PCBs, mediante la herramienta DRC (Design Rules Checker). Una vez fue comprobado, se mandó imprimir obteniendo el siguiente resultado.

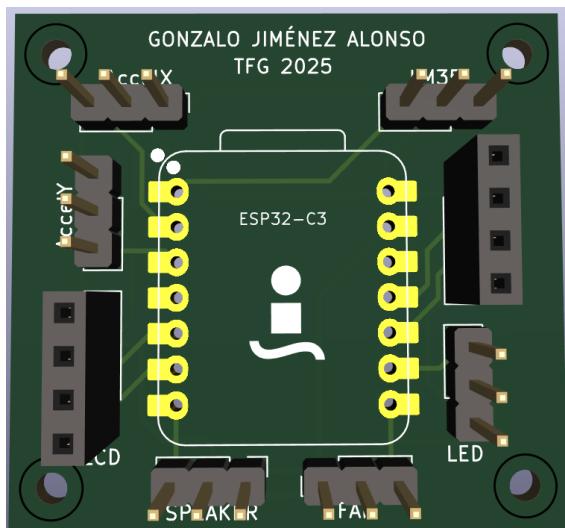


Figura 5.13 PCB en 3D.

5.8 Montaje y diseño físico de la maqueta

Por último, se procedió al montaje y diseño físico de la maqueta, para ello, se ha usado un software de modelado en 3D, así como una impresora 3D para poder obtener una caja siguiendo las especificaciones del proyecto. Para el propio modelado se ha usado el software de *AutoDesq Inventor*, y la impresora *Prusa Steel Original 3D*

Se buscó poder hacer una caja, con las menores dimensiones posibles en las que estuvieran visibles para el usuario, tanto la pantalla LCD, como el sensor HCSR-04, así como el LED, el altavoz y el ventilador. La principal limitación de esta maqueta fue la colocación de los componentes, debido a que el propio tamaño de la PCB, es muy reducido (39.5 x 37 mm), sin embargo hay que tener en cuenta algunas medidas especiales como la pantalla (80 x 36 mm) y el ventilador (Diámetro de 70 mm).

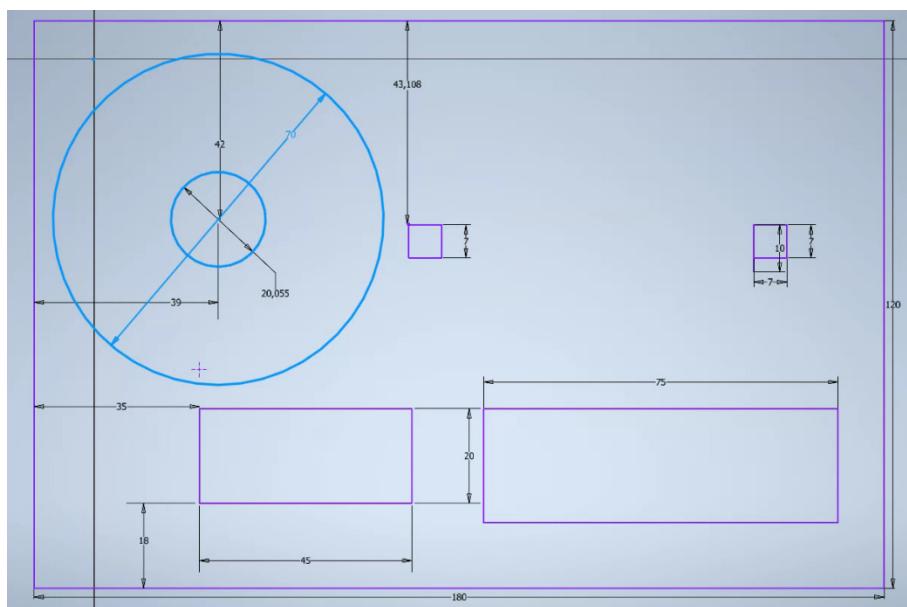


Figura 5.14 Medidas cara frontal maqueta.

Para el ventilador, debido a que el motor tiene muy poca fuerza y cualquier roce con la superficie de la caja provocaba que dejara de funcionar, se decidió hacer una hendidura en la cara frontal, reduciendo el espesor en esa zona hasta los 2 mm. De este modo, solo las aspas y el eje de giro sobresalen por el hueco, mientras que el motor queda completamente alojado en el interior de la caja. Esta solución permitió eliminar interferencias mecánicas y garantizar el giro libre de las aspas, asegurando así el correcto funcionamiento del sistema de ventilación. Además, se cuidó el diseño del rebaje para mantener la solidez estructural y facilitar un flujo de aire adecuado, contribuyendo a la refrigeración del conjunto sin aumentar de forma significativa el volumen de la maqueta.

Además en la parte superior, se le añadió dos orificios, uno para el LED como señal visual, y otro para el altavoz. El recipiente cuenta con 5 mm de espesor con el objetivo de hacerla lo más ligera posible. De forma adicional, tiene un orificio lateral para la entrada del cable de alimentación de la placa, y cuenta con dos soportes para la pantalla del ordenador portátil, o monitor, de forma que se asemeja y cumple con las expectativas.

Con esas medidas, se realizó la siguiente disposición:

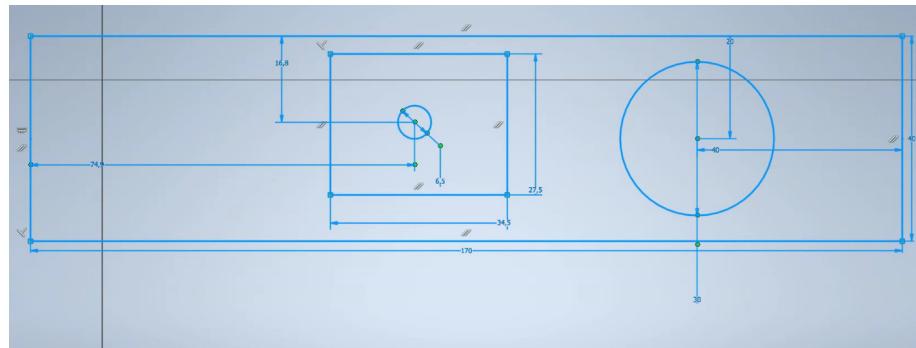


Figura 5.15 Medidas cara superior maqueta.

Para garantizar la funcionalidad y durabilidad de la maqueta, se optó por un filamento PLA de alta calidad en la impresión 3D, debido a su resistencia suficiente y facilidad de trabajo, además de su bajo coste. Tras el primer prototipo, se realizaron varios ajustes en el diseño, corrigiendo tolerancias para encazar con precisión los componentes y permitiendo su montaje y desmontaje sin dañar las conexiones. También se validó la ventilación mediante pruebas de funcionamiento, comprobando que la disposición del ventilador y los orificios reducía eficazmente la acumulación de calor en el interior del recinto. Estas modificaciones aseguraron que la caja final cumpliera con los requisitos de ergonomía, estética y accesibilidad definidos para el proyecto.

El resultado final se muestra a continuación:



Figura 5.16 Maqueta final en 3D.

5.9 Pruebas de funcionamiento e integración

Una vez obtuvimos el sistema completo, llegó el momento de la comprobación del funcionamiento, y la integración de todos los componentes de forma conjunta, para asegurar su funcionamiento conforme a lo esperado. En primer lugar, y tras soldar la placa, se pudo comprobar que todas las conexiones se encontraban en buen estado.

En segundo lugar, se procedió a la solución de un problema planteado de forma posterior al desarrollo de la PCB. En relación al sensor ultrasónico HCSR-04, en el datasheet, se menciona como tiene que estar alimentado a 5V, y el pulso que genera y envía a la placa es también de 5V. Sin embargo, nuestra placa, la ESP32-C3, no acepta más que 3V3 (se menciona como el voltaje de entrada puede llegar hasta los 3.6V como máximo).

Por ello, se procedió a la soldadura del divisor resistivo con valores de resistencia que permitieran reducir el voltaje hasta los 3V3. Después de ser soldado, se comprobó con el pin de 5V, y el multímetro, marcaba 3.33V, cumpliendo con el resultado esperado. No obstante, al conectar el divisor a la señal de *Echo*(Pulso de salida del sensor y entrada a la placa), no se consiguió leer nada.

Al ser este sensor un componente genérico, no se dispone del datasheet específico del mismo, sin embargo se encuentran en internet otros similares. Para poder analizar el problema de la lectura con el divisor resistivo, se procedió a observar el pulso que generaba esta señal en uno de los osciloscopios del laboratorio del departamento, y el resultado no se correspondió con lo esperado.



Figura 5.17 Señal Echo en osciloscopio.

Al parecer, el pulso que este sensor, manda a la placa de desarrollo tiene un valor de 3.36V, valor aceptado por el microcontrolador, y muy posiblemente debido a que el sensor, ya cuenta con un regulador de tensión que permite su uso con microcontroladores que no acepten 5V.

Una vez asegurada la integridad de la señal enviada por parte del sensor ultrasónico, así como el rango de tensión en el que se encuentra, aceptable por el microcontrolador, se procedió a evaluar el sistema en situaciones reales, dentro de la maqueta. Comprobando tanto el modo de confort como el antirrobo, así como las situaciones concretas para cada uno de ellos, activación automática del ventilador según temperatura del ambiente y control manual, y detección de intrusos y movimiento con la alarma en modo detección.

De forma adicional, y para poder comprobar su funcionamiento en todas las situaciones posibles, se procedió a examinar el sistema en situaciones de posibles fallos, tales como el cambio entre modos con alguno de los actuadores (altavoz o ventilador) activo, para poder comprobar su desactivación, así como la activación y desactivación del modo detección de la alarma con los actuadores propios de este sistema activos, siendo el resultado el esperado y respondiendo de forma eficiente.

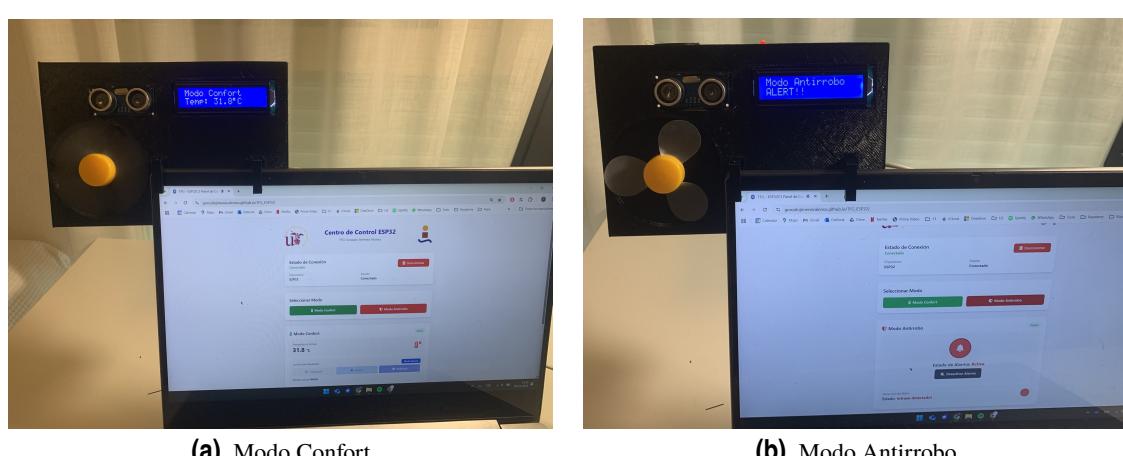
Se muestran a continuación imágenes de la página web de control cuando el sistema se encuentra en funcionamiento.

(a) Modo Confort.

(b) Modo Antirrobo.

Figura 5.18 Pruebas de funcionamiento Web de control.

Se añaden también a continuación imágenes del sistema incluyendo el hardware y la maqueta conectada a la red, junto con la web de control del usuario:

**Figura 5.19** Pruebas de funcionamiento sistema completo.

6 Conclusiones y futuro trabajo

Lo que sabemos es una gota de agua; lo que ignoramos es el océano.

ISAAC NEWTON, S. XVIII

En este trabajo se ha desarrollado un sistema electrónico portátil orientado a la videovigilancia y la supervisión, cumpliendo los objetivos marcados desde el diseño conceptual hasta la validación en prototipo físico. A lo largo del desarrollo, se ha abordado de forma estructurada el estudio de las necesidades de nuestro sistema, así como la elección de la placa de desarrollo más adecuada, priorizando un tamaño reducido, conectividad inalámbrica integrada y un bajo consumo, lo que finalmente motivó la elección del ESP32-C3.

Se ha demostrado la viabilidad de integrar sensores analógicos y digitales en un entorno compacto, resolviendo limitaciones técnicas como la reducción de entradas analógicas mediante selección de ejes relevantes o la compatibilidad de niveles de tensión de los sensores. Además, se ha implementado comunicación Bluetooth Low Energy, ofreciendo una interfaz de usuario sencilla mediante una página web que permite el control y monitorización en tiempo real, acercando el sistema a un posible uso práctico y flexible.

El diseño de la PCB y la caja impresa en 3D ha permitido validar la integración física del sistema, cumpliendo requisitos de accesibilidad, estética y funcionalidad. Las pruebas realizadas han demostrado un comportamiento correcto tanto en el modo Confort como en el modo Antirrobo, permitiendo un funcionamiento fiable y estable incluso en condiciones de cambio de estado o posibles fallos de activación.

Como líneas de mejora futuras, se sugiere estudiar la implementación de sensores digitales para liberar entradas analógicas y poder tener resultados más precisos, optimizar el diseño del PCB para permitir mayor flexibilidad en la distribución de pines y evaluar otras opciones de alimentación que doten al sistema de mayor autonomía, siendo una batería externa una de las posibles soluciones para ello.

En definitiva, este trabajo demuestra que es posible desarrollar un sistema portátil de seguridad adaptable, integrando hardware y software de forma eficiente y dejando la puerta abierta a futuras ampliaciones y mejoras orientadas a su uso real.

Apéndice A

Tabla de componentes utilizados

A continuación se muestra una tabla con todos los componentes que han sido analizados en este trabajo, tanto los definitivos, como los utilizados en la evaluación de necesidades con el F28335.

Tabla A.1 Listado de componentes.

Componente	Referencia	Proveedor
Microcontrolador F28335	TMS320F28335	Texas Instruments
Microcontrolador ESP32-C3	XIAO ESP32-C3	SeedStudio
Sensor de temperatura (LM35)	LM35 Linear Temperature Sensor v4	DFRobot
Ventilador	Gravity 130 DC Motor SKU DFR0411	DFRobot
Sensor de infrarrojos	SEN0018 Digital Infrared motion sensor	DFRobot
Sensor de vibraciones	SEN0289 Gravity Digital Shake sensor	DFRobot
Vibrador	DFR0440 Gravity Vibration Module	DFRobot
Altavoz	FIT0449 DFRobot Speaker v1.0	DFRobot
LED	DFR0021 Digital Red LED Module v2	DFRobot
Módulo Bluetooth	HC-05 Bluetooth Module	Laboratorio
Acelerómetro	MMA7361 Triple Axis Accelerometer	DFRobot
Sensor de Ultrasonido	HCSR-04 Ultrasound sensor	Freenove
Pantalla LCD	LCD1602	Freenove

Apéndice B

Código en Code Composer Studio

Código B.1 Código completo Code Composer Studio.

```
#include "DSP28x_Project.h"

#include "menu.h"
#include "sci.h"
#include "math.h"
#include <stdint.h>

// Prototipado de funciones externas
extern void InitAdc(void);
extern void InitSysCtrl(void);
extern void InitCpuTimers(void);
extern void ConfigCpuTimer(struct CPUTIMER_VARS *, float, float);
extern void InitPieCtrl(void);
extern void InitPieVectTable(void);
extern void SCIA_init();

// Prototipado de funciones internas
void InitAdcRegs(void);
void Gpio_select(void);
void Setup_ePWM1(void);
void Setup_ePWM2(void);
void MensajesInicio (void);
void ModoConfort(void);
void ResetConfort(void);
void ModoAntirrobo(void);
void ResetAntirrobo(void);
float getSonar(void);
void Delay_ms(unsigned long ms);

// Prototipado de funciones de interrupción
interrupt void cpu_timer0_isr(void);
interrupt void adc_SEQ1_isr(void);
interrupt void scia_rx_isr(void);

#define MAX_DISTANCE 400 // Maximum sensor distance in cm

// Variables globales
int duty_cycle_25 = 0; // Indicador de duty cycle actual
volatile int modo_normal = 2; // Indicador de activación del modo normal
unsigned int Vbin_LM35;
unsigned int Vbin_x;
unsigned int Vbin_y;
unsigned int Vbin_z;
float Vana_LM35;
float Vana_x;
```

```

float Vana_y;
float Vana_z;
float x_ant = 0.0, y_ant = 0.0, z_ant = 0.0;
int peopl_det = 0;
float mov_total = 0;
float Temp;
char temp_str[8];
int dat_nuevo = 0; //1 Para dato nuevo / 0 para dato antiguo
char modo = 'x'; // 'c' para confort, 'a' para antirrobo
char dat_ant = '\0';
int robo_msg_enviado = 0;
int alarm_active = 0;
int alarm_toggle_counter = 0;
const int alarm_toggle_interval = 5; // 5 * 100ms = 500ms
volatile unsigned int accel_sample_counter = 0;
const unsigned int ACCEL_SAMPLE_PERIOD = 20; // 20 * 100ms = 2 sec
float distance = 0;
char first_time = 1;

//Mensajes para el terminal
const char * hola_msg = "*_Bienvenido!!\n\r*";
const char * confi_msg = "*_Que modo quieres activar?\n\r*";
const char * conf2_msg = "*_c) Modo Confort\n\r*";
const char * conf3_msg = "*_a) Modo Anti-Robo\n\r*";
const char * conf4_msg = "*_\n\r*";
const char * normal_on_msg = "*_Modo confort ACTIVADO\n\r"
    "\n\r*";
const char * robo_on_msg = "*_Modo Anti-Robo ACTIVADO\n\r"
    "\n\r*";
const char * robo_det = "*_ROBO DETECTADO!!\n\r"
    "\n\r*";

//Mensajes de control
const char * alarm_on = "*SV99*";
const char * alarm_off = "*SV00*";
const char * led_on = "*LR255G0B0*";
const char * led_off = "*LROG0B0*";
const char * vent_max = "*VR0G255B0*";
const char * vent_med = "*VR255G255B0*";
const char * vent_off = "*VROG0B0*";
const char * reset_temp = "*n_*";

int Mov_detect = 0;

//#####
//          Código Principal
//#####
void main(void)
{
    InitSysCtrl();           // Inicialización del reloj del sistema SYSCLKOUT=150MHz,
    // deshabilitación del watchdog y
    // habilitación de los relojes de los periféricos

    DINT;                   // Deshabilitación de todas las interrupciones

    Gpio_select();           // Selección de los GPIOs

    Setup_ePWM1();           // Inicialización de la unidad ePWM1

    Setup_ePWM2();           // Inicialización de la unidad ePWM2

    InitCpuTimers();         // Inicialización del Timer0

    ConfigCpuTimer(&CpuTimer0,150,100000); // Se inicializa Timer0 para periodo de 100 ms
}

```

```

InitPieCtrl();                                // Se deshabilitan todas las interrupciones PIE y la
INTM

InitPieVectTable();                           // Inicializacion de la tabla PIE de direcciones

InitAdc();                                   // Inicialización básica del ADC

InitAdcRegs();                              // Inicialización resto de registros del ADC

SCIA_init();                                // Inicializacion de las interrupcion de TX y RX

EALLOW;
PieVectTable.TINT0 = &cpu_timer0_isr;    // Se direcciona la rutina de interrupcion del Timer0
en la PIE Table
PieVectTable.SEQ1INT = &adc_SEQ1_isr;
PieVectTable.SCIRXINTA = &scia_rx_isr;
EDIS;

PieCtrlRegs.PIEIER1.bit.INTx7 = 1;          // Se habilita la interrupcion a nivel de PIE
PieCtrlRegs.PIEIER1.bit.INTx1 = 1;          // Habilitación de interrupción por línea INT1.6 (ADC,
SEQ1)
PieCtrlRegs.PIEIER9.bit.INTx1 = 1;          // SCIA RX

IER |= M_INT1; // Timer0 y ADC
IER |= M_INT9; // SCI

EINT;                                       // Se habilita la interrupcion global INTM

CpuTimer0Regs.TCR.bit.TSS = 0;

GpioDataRegs.GPACLEAR.bit.GPIO17 = 1; // Apagar LED

MensajesInicio();

while(1)
{
    if(modo_normal==1)
    {
        ResetAntirrobo();
        ModoConfort();
    }

    if (modo_normal == 0)
    {
        ResetConfort();
        ModoAntirrobo();
    }
}

void Gpio_select(void)
{
    EALLOW;
    GpioCtrlRegs.GPAMUX2.bit.GPIO17 = 0; // GPIO17 como pin de propósito general
    GpioCtrlRegs.GPDIR.bit.GPIO17 = 1; // Configuración como salida: LED
    GpioCtrlRegs.GPAMUX2.bit.GPIO20 = 0; // GPIO20 como pin de propósito general
    GpioCtrlRegs.GPDIR.bit.GPIO20 = 0; // Configuración como entrada: IR
    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1; // GPIO0 como EPWM1A (ALTAVOZ)
    GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1; // GPIO2 como EPWM2A (MOTOR)
    GpioCtrlRegs.GPAMUX2.bit.GPIO28 = 1; // GPIO28 como SCIRXDA
    GpioCtrlRegs.GPAMUX2.bit.GPIO29 = 1; // GPIO29 como SCITXDA (RX HC-05)
    GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0; // Set GPIO6 as GPIO (trigPin)
    GpioCtrlRegs.GPDIR.bit.GPIO6 = 1; // Set GPIO6 as output
    GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0; // Set GPIO7 as GPIO (echoPin)
    GpioCtrlRegs.GPDIR.bit.GPIO7 = 0; // Set GPIO7 as input
    EDIS;
}

void InitAdcRegs(void)

```

```

{
    AdcRegs.ADCTRL1.all = 0;
    AdcRegs.ADCTRL1.bit.ACQ_PS = 1;      // Sample window = 2*(1/ADCCLK)
    AdcRegs.ADCTRL1.bit.SEQ_CASC = 0;    // Modo dual
    AdcRegs.ADCTRL1.bit.CPS = 1;         // CPS = 2 --> ADCCLK = FCLK/(CPS+1)
    AdcRegs.ADCTRL1.bit.CONT_RUN = 0;     // Modo start-stop

    AdcRegs.ADCTRL2.all = 0;
    AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1; // Se habilita interrupción SEQ1INT
    AdcRegs.ADCTRL2.bit.INT_MOD_SEQ1 = 0; // Modo de interrupción cada EOS

    AdcRegs.ADCTRL3.bit.SMODE_SEL = 0;   // Modo muestreo secuencial
    AdcRegs.ADCTRL3.bit.ADCCLKPS = 1;    // ADCCLKPS = 3 --> FCLK = HSPCLK / 2 * ADCCLKPS
    // HSPCLK = 75 MHz
    // FCLK = 37.5 MHz
    // ADCCLK = 12.5 MHz

    AdcRegs.ADCMAXCONV.all = 0x0003;    // 4 conversiones por secuencia

    AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0; // Canal A0 para el sensor de temperatura (LM35)
    AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 2; // Canal A2 para el acelerómetro (Eje X)
    AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 3; // Canal A3 para el acelerómetro (Eje Y)
    AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 4; // Canal A4 para el acelerómetro (Eje Z)

}

void Setup_ePWM1(void)
{
    // Configuración del módulo Time-Base
    EPwm1Regs.TBCTL.bit.CLKDIV = 0;          // Pre-escalador CLKDIV = 1
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = 1;        // Pre-escalador HSPCLKDIV = 2
    EPwm1Regs.TBCTL.bit.CTRMODE = 2;          // Modo up - down
    EPwm1Regs.TBCTL.bit.PRDLD = 0;            // Shadow activado
    EPwm1Regs.TBCTL.bit.SYNCSEL = 3;          // Sincronización deshabilitada
    EPwm1Regs.TBPRD = 37500;                 // Período para 1kHz de fPWM (150MHz/(2*2*37500))

    // Configuración del módulo Compare-Counter
    EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD; // Inicialmente al 0%
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = 0;        // shadow activado para CMPA
    EPwm1Regs.CMPCTL.bit.LOADAMODE = 0;          // shadow de CMPA cargado en TBCTR = 0

    // Configuración del módulo Action Qualifier
    EPwm1Regs.AQCTLA.all = 0;
    EPwm1Regs.AQCTLA.bit.CAD = 1;              // EPWM1A a low en CMPA down
    EPwm1Regs.AQCTLA.bit.CAU = 2;              // EPWM1A a high en CMPA up
}

void Setup_ePWM2(void)
{
    // Configuración del módulo Time-Base
    EPwm2Regs.TBCTL.bit.CLKDIV = 0;          // Pre-escalador CLKDIV = 1
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = 1;        // Pre-escalador HSPCLKDIV = 2
    EPwm2Regs.TBCTL.bit.CTRMODE = 2;          // Modo up - down
    EPwm2Regs.TBCTL.bit.PRDLD = 0;            // Shadow activado
    EPwm2Regs.TBCTL.bit.SYNCSEL = 3;          // Sincronización deshabilitada
    EPwm2Regs.TBPRD = 37500;                 // Período para 1kHz de fPWM (150MHz/(2*2*37500))

    // Configuración del módulo Compare-Counter
    EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD; // Asegura el duty cycle en 0%
    EPwm2Regs.CMPCTL.bit.SHDWAMODE = 0;        // Shadow activado para CMPA
    EPwm2Regs.CMPCTL.bit.LOADAMODE = 0;          // Shadow de CMPA cargado en TBCTR = 0

    // Configuración del módulo Action Qualifier
    EPwm2Regs.AQCTLA.all = 0;                // Limpia AQCTLA
    EPwm2Regs.AQCTLA.bit.ZRO = 1;             // Forzar salida a bajo cuando el contador esté en 0
    EPwm2Regs.AQCTLA.bit.CAU = 2;              // Forzar salida a alto en comparación ascendente (CMPA)
    EPwm2Regs.AQCTLA.bit.CAD = 1;              // Forzar salida a bajo en comparación descendente (CMPA)
}

```

```

void MensajesInicio (void)
{
    SCIA_Transmit(hola_msg, strlen(hola_msg));
    SCIA_Transmit(conf1_msg, strlen(conf1_msg));
    SCIA_Transmit(conf2_msg, strlen(conf2_msg));
    SCIA_Transmit(conf3_msg, strlen(conf3_msg));
    SCIA_Transmit(conf4_msg, strlen(conf4_msg));

    SCIA_Transmit(reset_temp,strlen(reset_temp));
}

void ModoConfort(void){
    // Conversión a valores analógicos
    Vana_LM35 = 3000 * (float)Vbin_LM35 / 4095; // mV
    Temp = Vana_LM35 / 10;                      // Conversión a temperatura en °C

    // Convert float temperature to integers for whole and decimal parts
    int Temp_whole = (int)Temp;
    int Temp_dec = (int)((Temp - Temp_whole) * 100); // Two decimal places

    // Build temperature string character by character
    temp_str[0] = '*';
    temp_str[1] = '\n';
    temp_str[2] = (Temp_whole / 10) % 10 + '0'; // Tens
    temp_str[3] = Temp_whole % 10 + '0';        // Ones
    temp_str[4] = '.';
    temp_str[5] = (Temp_dec / 10) % 10 + '0'; // First decimal place
    temp_str[6] = Temp_dec % 10 + '0';         // Second decimal place
    temp_str[7] = '*';                         // Newline

    SCIA_Transmit(temp_str,strlen(temp_str));

    // Velocidad de ventilador según temperatura
    if (Temp >= 25 && Temp <= 28)
    {
        EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD * 0.5; // Duty cycle al 25%
        SCIA_Transmit(vent_med,strlen(vent_med));
    }
    else if (Temp > 28)
    {
        EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD * 0.25; // Duty cycle al 50%
        SCIA_Transmit(vent_max,strlen(vent_max));
    }
    else
    {
        EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD; // Apaga el PWM cuando la temperatura es menor
        a 25
        SCIA_Transmit(vent_off,strlen(vent_off));
    }
}

void ResetConfort(void)
{
    EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD; //Apagar motor
}

void ModoAntirrobo(void)
{
    if (first_time == 1) {
        // Convert ADC values to analog values
        Vana_x = 3000.0*(float)Vbin_x/4095.0;
        Vana_y = 3000.0*(float)Vbin_y/4095.0;
        Vana_z = 3000.0*(float)Vbin_z/4095.0;

        // Initialize accelerometer reference values
    }
}

```

```

x_ant = Vana_x;
y_ant = Vana_y;
z_ant = Vana_z;

// Get initial sonar reading and ignore it
getSonar(); // First reading might be unstable

// Reset detection flags
Mov_detect = 0;
peopl_det = 0;
alarm_active = 0;

// Mark initialization complete
first_time = 0;

// Skip the rest of the function this first time
return;
}

EPwm2Regs.CMPA.half.CMPA = EPwm2Regs.TBPRD; //Apagar motor
Vana_x = 3000.0*(float)Vbin_x/4095.0;
Vana_y = 3000.0*(float)Vbin_y/4095.0;
Vana_z = 3000.0*(float)Vbin_z/4095.0;

float delta_x = fabs(Vana_x - x_ant);
float delta_y = fabs(Vana_y - y_ant);
float delta_z = fabs(Vana_z - z_ant);
mov_total = delta_x + delta_y + delta_z;

if (accel_sample_counter >= ACCEL_SAMPLE_PERIOD) {
    x_ant = Vana_x;
    y_ant = Vana_y;
    z_ant = Vana_z;
    accel_sample_counter = 0; // Reset the counter
}

if (mov_total > 100 && Mov_detect == 0) {
    Mov_detect = 1;
}
if (mov_total < 120 && Mov_detect == 1) {
    Mov_detect = 0;
    robo_msg_enviado = 0;
}

distance = getSonar();

static int dist_counter = 0;

if (distance <= 40) { //Alguien a menos de 40cm
    peopl_det = 1;
    SCIA_Transmit(led_on, strlen(led_on));
    dist_counter = 0;
} else {
    peopl_det = 0;
    dist_counter++;
}

if ((peopl_det && Mov_detect == 1)&& !alarm_active) {
    alarm_active = 1;
    if (!robo_msg_enviado) {
        SCIA_Transmit(robo_det, strlen(robo_det));
        robo_msg_enviado = 1;
    }
}

if (alarm_active) {
    if (alarm_toggle_counter >= alarm_toggle_interval) {
        if (duty_cycle_25 == 0) {
            EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD * 0.1;
            SCIA_Transmit(alarm_on, strlen(alarm_on));
        }
    }
}

```

```

        GpioDataRegs.GPASET.bit.GPIO17 = 1;
    } else {
        EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD * 0.75;
        SCIA_Transmit(alarm_off, strlen(alarm_off));
        GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    }
    duty_cycle_25 = !duty_cycle_25;
    alarm_toggle_counter = 0;
}
}

if (dist_counter>=20)
{
    alarm_active = 0;
    EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD;
    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
    SCIA_Transmit(led_off, strlen(led_off));
    SCIA_Transmit(alarm_off, strlen(alarm_off));
}
}

void ResetAntirrobo(void)
{
    GpioDataRegs.GPACLEAR.bit.GPIO17 = 1; // Apagar LED
    SCIA_Transmit(alarm_off,strlen(alarm_off));
    SCIA_Transmit(led_off,strlen(led_off));
    EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD; //Apagar altavoz
    Mov_detect = 0;
}

float getSonar(void)
{
    // Enviamos el pulso de trigger
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    DELAY_US(2);
    GpioDataRegs.GPASET.bit.GPIO6 = 1;
    DELAY_US(10);
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;

    // Medimos la duración del pulso de echo
    unsigned long pingTime = 0;
    unsigned long timeout = 0;

    while (GpioDataRegs.GPADAT.bit.GPIO7 == 0) { //Esperamos flanco de subida en echo
        timeout++;
        DELAY_US(2);
        if (timeout > 30000) return MAX_DISTANCE; // Return max distance if timeout
    }
    timeout = 0;
    while (GpioDataRegs.GPADAT.bit.GPIO7 == 1) // Medimos cuanto tiempo está alto
    {
        pingTime++;
        DELAY_US(2);
        if (pingTime > 30000) break; // 30ms timeout
    }

    // Convertimos a cm (58 = Velocidad_Sonido / 2 / 10000;)
    float distance = (float)pingTime / 58.0;

    return distance;
}

void Delay_ms(unsigned long ms)
{
    unsigned long i;
    for (i = 0; i < (150000 * ms); i++); // Ajuste del tiempo de espera basado en SYSCLKOUT
}

interrupt void cpu_timer0_isr(void)
{

```

```

AdcRegs.ADCTRL2.bit.SOC_SEQ1 = 1; // Start ADC conversion

alarm_toggle_counter++;
accel_sample_counter++;

PieCtrlRegs.PIEACK.all = 0x0001;
}

interrupt void adc_SEQ1_isr(void)
{
    Vbin_LM35 = AdcMirror.ADCRESULT0; // Almacenar resultado del sensor de temperatura
    Vbin_x = AdcMirror.ADCRESULT1; // Acelerómetro eje X
    Vbin_y = AdcMirror.ADCRESULT2; // Acelerómetro eje Y
    Vbin_z = AdcMirror.ADCRESULT3; // Acelerómetro eje Z

    AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1; // Reset SEQ1
    AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1; // Clear INT_SEQ1 para habilitar la siguiente interrupción

    PieCtrlRegs.PIEACK.bit.ACK1 = 1; // Clear PIEACK para habilitar siguiente interrupción
}

interrupt void scia_rx_isr(void)
{
    if(SciaRegs.SCIRXST.bit.RXERROR) {
        SciaRegs.SCICTRL1.bit.SWRESET = 0;
        SciaRegs.SCICTRL1.bit.SWRESET = 1;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
        return;
    }

    SCIA_getchar(&modo);

    if(modo == 'c') {
        modo_normal = 1;
        alarm_active = 0; // Reseteo alarma cuando cambiamos a modo confort
        if (dat_nuevo) {
            SCIA_Transmit(normal_on_msg, strlen(normal_on_msg));
            dat_nuevo = 0;
        }
    }
    else if (modo == 'a') {
        modo_normal = 0;
        first_time = 1;
        if (dat_nuevo) {
            SCIA_Transmit(robo_on_msg, strlen(robo_on_msg));
            dat_nuevo = 0;
        }
    }
    else {
        alarm_active = 0;
        Mov_detect = 0;
        robo_msg_enviado = 0;
        peopl_det = 0; // Reseteamos detec. de personas
        accel_sample_counter = 0;
        EPwm1Regs.CMPA.half.CMPA = EPwm1Regs.TBPRD;
        GpioDataRegs.GPACLEAR.bit.GPIO17 = 1;
        SCIA_Transmit(alarm_off, strlen(alarm_off));
        SCIA_Transmit(led_off, strlen(led_off));
    }

    if (modo != dat_ant) {
        dat_nuevo = 1;
        dat_ant = modo;
    }
}

PieCtrlRegs.PIEACK.all = PIEACK_GROUP9;
}

//=====
// End of SourceCode.
//=====

```

Apéndice C

Código en Arduino IDE

Código C.1 Código completo Arduino IDE.

```
#include <LiquidCrystal_I2C.h>
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>

// --- LCD ---
LiquidCrystal_I2C lcd(0x27, 16, 2);

// --- Declaración de pines ---
const int TEMP_PIN = 2; // LM35 on A2 (GPIO4)
const int ACCEL_X_PIN = 3; // Accelerometer X analog input
const int ACCEL_Y_PIN = 4; // Accelerometer Y analog input
const int ledPin = 8;
const int echoPin = 9; // echoPin connected to GPIO7
const int trigPin = 10; // trigPin connected to GPIO6
const int PWM_FAN = 20; // PWM output (Fan)
const int PWM_SPEAKER = 21; // PWM output (Speaker)

// --- Configuración BLE ---
BLEServer* pServer = nullptr;
BLECharacteristic* pControlCharacteristic = nullptr;
bool deviceConnected = false;
bool oldDeviceConnected = false;

// UUIDs
#define SERVICE_UUID "19b10000-e8f2-537e-4f6c-d104768a1214"
#define CONTROL_CHAR_UUID "19b10002-e8f2-537e-4f6c-d104768a1214"

#define MODE_NONE 0
#define MODE_COMFORT 1
#define MODE_ANTIRROBO 2

uint8_t currentMode = MODE_NONE;
bool alarmOn = false;
bool inAntirroboMode = false;

// ***** Declaracion de variables *****/
//Modo Confort
unsigned int Vbin_x, Vbin_y;
float Vana_x, Vana_y;
float x_ant = 0.0, y_ant = 0.0;
bool fanManual = false; // true: manual, false: auto
int fanLevel = 0; // 0 = off, 1 = med, 2 = max

//Modo Antirrobo
```

```

float mov_total = 0;
float distance = 0;
#define MAX_DIST 700 // Maximo teórico 400-500cm.
//timeOut= 2*MAX_DISTANCE /100 /340 *1000000 = MAX_DISTANCE*58.8
float timeOut = MAX_DIST * 60;
char accelFirstTime = 1; // flag para ignorar la primera lectura
unsigned int accel_sample_counter = 0;
#define ACCEL_SAMPLE_PERIOD 20 // Frecuencia de actualización del acelerómetro (aprox. 500 ms)
unsigned long lastAlarmCheck = 0;
const unsigned long alarmCheckInterval = 500; // ms
int alarm_triggered = 0;

//Auxiliares para LCD
int lcd_counter = 0;
const int lcd_update_period = 25; // 500ms / 20ms = 25

//Auxiliares para HTML
unsigned long lastNotify = 0;
const unsigned long notifyInterval = 500; // ms

//Auxiliares PWM
int pwmFreqFan = 5000; // Frec ventilador: 5 kHz
int pwmFreqSpeaker = 500; // Frec altavoz: 500 Hz
int pwmResolution = 8; // 8-bit (0-255)

// --- Función actualización LCD ---
void updateLCDStatus() {
    lcd.setCursor(0, 0);

    if (currentMode == MODE_CONFORT) {
        lcd.print("Modo Confort ");
        lcd.setCursor(0, 1);
        lcd.print("Temp: ");
        float temp = readTemperature();
        lcd.print(temp, 1);
        lcd.print((char)223);
        lcd.print("C ");
    } else if (currentMode == MODE_ANTIRROBO) {
        lcd.print("Modo Antirrobo ");
        lcd.setCursor(0, 1);
        if (!alarmOn) {
            lcd.print("Alarm OFF      ");
        } else if (alarm_triggered) {
            lcd.print("ALERT!!      ");
        } else {
            lcd.print("Alarm ON      ");
        }
    } else {
        lcd.print("Bienvenido      ");
        lcd.setCursor(0, 1);
        lcd.print("Seleccione modo ");
    }
}

// --- Función Desactivar actuadores ---
void ActuadoresOff() {
    digitalWrite(ledPin, LOW);
    ledcWrite(FAN_PWM_CHANNEL, 0);
    ledcWrite(SPEAKER_PWM_CHANNEL, 0);
}

// --- BLE Callbacks ---
class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) override { deviceConnected = true; }
    void onDisconnect(BLEServer* pServer) override { deviceConnected = false; }
};

class ControlCallbacks : public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic* characteristic) override {
        String value = pControlCharacteristic->getValue();
    }
}

```

```

if (value == "c") {
    currentMode = MODE_CONFORT;
    alarmOn = false;
    ActuadoresOff();
} else if (value == "a") {
    currentMode = MODE_ANTIRROBO;
    alarmOn = false;
    ActuadoresOff();
}
//Modo Confort

else if (value == "fan:auto" && currentMode == MODE_CONFORT) {
    fanManual = false;
    fanLevel = 0;
} else if (value == "fan:off" && currentMode == MODE_CONFORT) {
    fanManual = true;
    fanLevel = 0;
} else if (value == "fan:med" && currentMode == MODE_CONFORT) {
    fanManual = true;
    fanLevel = 1;
} else if (value == "fan:max" && currentMode == MODE_CONFORT) {
    fanManual = true;
    fanLevel = 2;
}
//Modo Antirrobo
else if (value == "alarm:on" && currentMode == MODE_ANTIRROBO) {
    alarmOn = true;
    digitalWrite(ledPin, LOW);
    accel_first_time = 1; // Reseteamos el flag cada vez que se activa el modo
} else if (value == "alarm:off" && currentMode == MODE_ANTIRROBO) {
    alarmOn = false;
    ActuadoresOff();
}
//Actualizamos LCD
updateLCDStatus();
lcd_counter = 0;
}
};

// ***** Funciones auxiliares *****/
// Test I2C address for LCD
bool i2CAddrTest(uint8_t addr) {
    Wire.beginTransmission(addr);
    return (Wire.endTransmission() == 0);
}

//Lectura LM35
float readTemperature() {
    uint16_t val = analogRead(TEMP_PIN);
    float tempC = ((double)val * (3.3 / 4095.0) * 100.0); // 10mV/grad C, 12-bit ADC, 3.3V ref
    return tempC;
}

//Lectura HC-SR04
float getSonar() {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    unsigned long pingTime = pulseIn(echoPin, HIGH, timeOut);
    if (pingTime == 0) return -1;
    float distance = (float)pingTime / 58.0;
    return distance;
}

void checkAlarmSensors() {
    // Primera lectura para evitar falsos positivos
    if (accel_first_time == 1) {

```

```

Vbin_x = analogRead(ACCEL_X_PIN);
Vbin_y = analogRead(ACCEL_Y_PIN);
Vana_x = 3300.0 * (float)Vbin_x / 4095.0;
Vana_y = 3300.0 * (float)Vbin_y / 4095.0;
x_ant = Vana_x;
y_ant = Vana_y;
getSonar();
accel_sample_counter = 0;
accel_first_time = 0;
alarm_triggered = 0;
return;
}

// Lectura real de los sensores
Vbin_x = analogRead(ACCEL_X_PIN);
Vbin_y = analogRead(ACCEL_Y_PIN);
Vana_x = 3300.0 * (float)Vbin_x / 4095.0;
Vana_y = 3300.0 * (float)Vbin_y / 4095.0;
float delta_x = fabs(Vana_x - x_ant);
float delta_y = fabs(Vana_y - y_ant);
mov_total = delta_x + delta_y;

accel_sample_counter++;
if (accel_sample_counter >= ACCEL_SAMPLE_PERIOD) {
    x_ant = Vana_x;
    y_ant = Vana_y;
    accel_sample_counter = 0;
}

distance = getSonar();
// Lógica de activación de la alarma
if (distance >= 50) {
    alarm_triggered = 0;
    x_ant = Vana_x;
    y_ant = Vana_y;
} else if (distance > 0 && mov_total > 100) {
    alarm_triggered = 1;
}
}

// --- Funciones configuración BLE ---
void setupBLE() {
    BLEDevice::init("ESP32");
    pServer = BLEDevice::createServer();

    BLEService *pService = pServer->createService(SERVICE_UUID);

    pControlCharacteristic = pService->createCharacteristic(
        CONTROL_CHAR_UUID,
        BLECharacteristic::PROPERTY_WRITE | BLECharacteristic::PROPERTY_NOTIFY
    );
    pControlCharacteristic->setCallbacks(new ControlCallbacks());
    pControlCharacteristic->addDescriptor(new BLE2902());

    pService->start();
    BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
    pAdvertising->addServiceUUID(SERVICE_UUID);
    pAdvertising->setScanResponse(false);
    pAdvertising->setMinPreferred(0x0);
    BLEDevice::startAdvertising();
}

// ***** FUNCIONES PROPIAS ARDUINO *****
void setup() {
    //LCD
    Serial.begin(115200);
    delay(100);
    Wire.begin();
}

```

```

//analogSetAttenuation(ADC_ATTENDB_MAX);

//Pines como E/S
pinMode(ledPin, OUTPUT);
pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
pinMode(ACCEL_X_PIN, INPUT);
pinMode(ACCEL_Y_PIN, INPUT);
pinMode(PWM_SPEAKER, OUTPUT);
pinMode(PWM_FAN, OUTPUT);

//PWM
ledcAttach(PWM_FAN, pwmFreqFan, pwmResolution); // 5kHz, 8-bit resolution
ledcAttach(PWM_SPEAKER, pwmFreqSpeaker, pwmResolution); // 500Hz, 8-bit resolution

digitalWrite(ledPin, LOW);
ledcWrite(PWM_FAN, 0); // Channel 0, 0% duty cycle
ledcWrite(PWM_SPEAKER, 0); // Channel 1, 0% duty cycle

// LCD INIT
if (!i2CAddrTest(0x27)) {
    lcd = LiquidCrystal_I2C(0x3F, 16, 2);
}
lcd.init();
lcd.backlight();
updateLCDStatus();

// BLE INIT
setupBLE();
}

void loop() {
//Variables locales
static int last_alarm_triggered = -1;
static int lastFanLevel = -1;
static int lastFanManual = false;

// Lectura temperatura
float temp = readTemperature();

// --- BLE Notify ---
if (currentMode == MODE_CONFORT && (millis() - lastNotify > notifyInterval)) {
    updateLCDStatus();
    if (fanManual) {
        // Control Manual
        if (fanLevel == 1) {
            ledcWrite(PWM_FAN, 128); // Channel 0, 50% duty cycle;
        } else if (fanLevel == 2) {
            ledcWrite(PWM_FAN, 255); // Channel 0, 100% duty cycle;
        } else {
            ledcWrite(PWM_FAN, 0); // Channel 0, 0% duty cycle;
        }
    } else {
        // Control Automatico
        if (temp >= 28 && temp <= 32) {
            fanLevel = 1;
            ledcWrite(PWM_FAN, 128); // Channel 0, 50% duty cycle;
        } else if (temp > 32) {
            fanLevel = 2;
            ledcWrite(PWM_FAN, 255); // Channel 0, 100% duty cycle;
        } else {
            fanLevel = 0;
            ledcWrite(PWM_FAN, 0); // Channel 0, 0% duty cycle;
        }
    }
    //Envio de temperatura a HTML
    char tempStr[8];
    dtostrf(temp, 1, 1, tempStr);
}
}

```

```

pControlCharacteristic->setValue(tempStr);
pControlCharacteristic->notify();

//Envio de modo de ventilador a HTML
if (fanLevel != lastFanLevel || fanManual != lastFanManual) {
    String modeStr = fanManual ? "manual" : "auto";
    String levelStr = (fanLevel == 2) ? "max" : (fanLevel == 1) ? "med" : "off";
    String fanStatus = "{\"fan_mode\":\"" + modeStr + "\",\"fan_level\":\"" + levelStr + "\"}";
    pControlCharacteristic->setValue(fanStatus.c_str());
    pControlCharacteristic->notify();
    lastFanLevel = fanLevel;
    lastFanManual = fanManual;
}
lastNotify = millis();
}

if (currentMode == MODE_ANTIRROBO) {
    if (alarmOn && (millis() - lastAlarmCheck > alarmCheckInterval)) {
        checkAlarmSensors(); // Comprobamos sensores para actualizar alarm_triggered
        if (alarm_triggered != last_alarm_triggered) {

            //Envio notificación robo
            if (pControlCharacteristic != nullptr) {
                String motionJson = "{\"motion\":";
                motionJson += (alarm_triggered ? "true" : "false");
                motionJson += "}";
                pControlCharacteristic->setValue(motionJson.c_str());
                pControlCharacteristic->notify();
            }
            last_alarm_triggered = alarm_triggered;
        }

        if (alarm_triggered) {
            // Activamos LED y altavoz
            digitalWrite(ledPin, HIGH);
            ledcWrite(PWM_SPEAKER, 51); // Channel 1, 20% duty cycle;
            delay(200);
            ledcWrite(PWM_SPEAKER, 191); // Channel 1, 70% duty cycle;
            digitalWrite(ledPin, LOW);
            delay(80);
        } else {
            ledcWrite(PWM_SPEAKER, 0); // Channel 1, 0% duty cycle;
        }
        lastAlarmCheck = millis();
    } else if (!alarmOn) {
        alarm_triggered = 0;
    }
}

// Frecuencia actualización LCD
lcd_counter++;
if (lcd_counter >= lcd_update_period) {
    updateLCDStatus();
    lcd_counter = 0;
}

// --- Conexión BLE ---
if (!deviceConnected && oldDeviceConnected) {
    Serial.println("Device disconnected.");
    currentMode = MODE_NONE;
    alarmOn = false;
    fanManual = false;
    fanLevel = 0;
    ActuadoresOff();
    updateLCDStatus();
    delay(500);
    pServer->startAdvertising();
    Serial.println("Restart advertising");
    oldDeviceConnected = deviceConnected;
}
if (deviceConnected && !oldDeviceConnected) {
}

```

```
    Serial.println("Device Connected");
    oldDeviceConnected = deviceConnected;
}

delay(20);
}
```


Índice de Figuras

1.1	Internet de las cosas o IoT	1
2.1	Análisis de sistemas de seguridad electrónica	8
2.2	Esquema típico de un sistema integrado	8
2.3	Sensor analógico VS digital	10
2.4	Tipos de redes inalámbricas	11
2.5	Evolución de Internet	12
2.6	Tipos de sensores IoT	14
3.1	Cerradura primitiva en madera	16
3.2	Reconocimiento facial con IA en <i>Raspberry Pi</i>	17
3.3	Mecanismos de seguridad física	18
4.1	Sensor LM35	22
4.2	Sensor Grayscale	23
4.3	Sensor de vibraciones	23
4.4	Variable <i>mov_det</i> con <i>Int_Ext</i>	25
4.5	Funcionamiento del módulo eCAP	25
4.6	Variable <i>mov_detectado</i> con módulo eCAP	26
4.7	Variable <i>mov_detect</i> con módulo ADC	27
4.8	Motor	27
4.9	Altavoz	28
4.10	Vibrador	28
4.11	LED	28
4.12	Esquema de conexiones	29
4.13	Diagrama de flujo	31
4.14	Diagrama de bloques del módulo ADC	32
4.15	Diagrama de bloques del módulo PWM	34
4.16	Diagrama de bloques del módulo SCI	36
4.17	Módulo de Bluetooth HC-05	38
4.18	Pantalla usuario Bluetooth Electronics	40
4.19	Sensor de aceleración escogido	41
4.20	Sensor de infrarrojos escogido	42
4.21	Cálculo de la distancia mediante ondas ultrasónicas	43
4.22	Sensor HC-SR04 escogido	44
4.23	Diagrama de Flujo definitivo	45
4.24	Esquema de conexión definitivo	45

5.1	Raspberry Pi Pico W	48
5.2	Arduino LilyPad Main Board	48
5.3	Adafruit FLORA Board	49
5.4	SeeedStudio XIAO ESP32-C3	50
5.5	LCD1602	51
5.6	Diagrama de pines ESP32-C3	52
5.7	Asignación de pines ESP32-C3	53
5.8	Diagrama de flujo	54
5.9	Estructura módulo PWM	56
5.10	Web para control del usuario	61
5.11	Esquemático en KiCad	61
5.12	Diseño de la PCB	62
5.13	PCB en 3D	62
5.14	Medidas cara frontal maqueta	63
5.15	Medidas cara superior maqueta	64
5.16	Maqueta final en 3D	64
5.17	Señal Echo en osciloscopio	65
5.18	Pruebas de funcionamiento Web de control	66
5.19	Pruebas de funcionamiento sistema completo	66

Índice de Tablas

4.1	Esquema de conexiones	30
4.2	Comandos para Bluetooth Electronics	39
5.1	Comparativa entre distintas placas de desarrollo	50
5.2	Comparativa entre Arduino y MicroPython	52
A.1	Listado de componentes	69

Índice de Códigos

4.1 Función <i>init_ex_int</i>	24
4.2 Limpieza de la variable <i>mov_det</i>	24
4.3 Función <i>ConfigurarECAP1</i>	26
4.4 Conversión binario a mV	27
4.5 Función <i>InitAdcRegs</i>	33
4.6 Función de interrupción ADC	33
4.7 Conversión de binario a "analógico"	34
4.8 Función <i>Setup_ePWM</i>	35
4.9 Modificación del PWM	36
4.10 Archivo <i>sci.c</i>	37
4.11 Comprobación dato nuevo SCI	37
4.12 Comunicación Bluetooth	38
4.13 Lectura de datos del acelerómetro	41
4.14 Contador para medida inicial del acelerómetro	42
4.15 Control del sensor de infrarrojos	42
4.16 Sensor HC-SR04	44
5.1 Lectura del LM35 en Arduino	55
5.2 Lectura del acelerómetro en Arduino	56
5.3 Configuración PWM	56
5.4 Librerías BLE	57
5.5 Variables y objetos BLE	58
5.6 Callbacks BLE	58
5.7 Función de inicialización BLE	58
5.8 Uso de BLE dentro de loop()	59
5.9 Gestión de conexiones BLE	59
5.10 Pantalla LCD	60
B.1 Código completo Code Composer Studio	71
C.1 Código completo Arduino IDE	79

Bibliografía

- [1] Muhammad Ahmad Baballe, Abdullahi Abdullahi, Abubakar Sadiq Muhammad, and Yusuf Bello Saleh. The various types of sensors used in the security alarm system. *International Journal of New Computer Architectures and their Applications*, 2020.
- [2] Wikipedia contributors. Embedded system, February 2025.
- [3] Raul Camposano and Jorg Wilberg. Embedded system design. *Design Automation for Embedded Systems*, 1:5–50, 1996.
- [4] Process Sensing Technologies. Digital versus analog sensors, 2025.
- [5] Fortinet. Wireless network, 2025.
- [6] RedesZone. Diferencias entre full duplex y half duplex, 2025.
- [7] IONOS by 1and1. Los tipos de redes más conocidos, 2025.
- [8] ProtoExpress. Best electronic circuit design practices, 2025.
- [9] Pradyumna Gokhale, Omkar Bhat, and Sagar Bhat. Introduction to iot. *International Advanced Research Journal in Science, Engineering and Technology*, 2018.
- [10] Superllave Tenerife. La historia de la cerradura, 2025.
- [11] Blog Prodeincendio. Evolución histórica de los sistemas de seguridad, 2025.
- [12] Forbes España. Más del 80 % de los españoles tiene instalada alguna medida de seguridad, 2025.
- [13] Nikolay Radulov. Artificial intelligence and security. security 4.0. *International Scientific Journal “Security and Future”*, pages 3–5, 2019.
- [14] Core Electronics. Face identify with raspberry pi, 2025.
- [15] Telefónica. ¿quién inventó el primer teléfono móvil?, 2025.
- [16] Ámbito Financiero. iphone 17 air: cómo será el celular más delgado de la historia de apple, 2025.
- [17] Xing Yang, Lei Shu, Ye Liu, Gerhard P. Hancke, Mohamed Amine Ferrag, and Kai Huang. Physical security and safety of iot equipment: A survey of recent advances and opportunities. *IEEE Transactions on Industrial Informatics*, 18(7):4319–4332, 2022.

- [18] Apple Inc. Find my network and privacy, 2025.
- [19] Wikipedia contributors. Ley de moore, 2025.
- [20] Xataka. Samsung acelera la guerra de los 5nm con tecnología ultravioleta extrema, 2025.
- [21] Texas Instruments. *TMS320F2833x Delfino Technical Reference Manual*, 2019.
- [22] DFRobot. Dfrobot lm35 linear temperature sensor, 2025.
- [23] DFRobot. Analog grayscale sensor v2, 2025.
- [24] DFRobot. Gravity: Digital shake sensor, 2025.
- [25] DFRobot. Gravity: 130 dc motor, 2025.
- [26] DFRobot. Speaker module, 2025.
- [27] DFRobot. Gravity: Vibration module, 2025.
- [28] DFRobot. Digital white led light module, 2025.
- [29] ZS-040 Module Manufacturer. Hc-05 bluetooth module datasheet, 2018.
- [30] DFRobot. Triple axis accelerometer mma7361, 2025.
- [31] DFRobot. Digital infrared motion sensor, 2025.
- [32] Elecfreaks. Hc-sr04 ultrasonic sensor datasheet, 2019.
- [33] Raspberry Pi Ltd. Raspberry pi pico w datasheet, 2024.
- [34] TiendaTec. Raspberry pi pico w, 2025.
- [35] Arduino and SparkFun Electronics. Lilypad arduino main board, 2024.
- [36] BricoGeek. Arduino lilypad, 2025.
- [37] Adafruit Industries. Flora wearable electronic platform datasheet, 2018.
- [38] Adafruit Industries. Adafruit flora wearable electronic platform, 2025.
- [39] Seeed Studio. Getting started with seeed studio xiao esp32c3, 2025.
- [40] Seeed Studio. Seeed studio xiao esp32c3, 2025.
- [41] Waveshare. Lcd1602 module specification datasheet, 2020.
- [42] Espressif Systems. Esp32-c3 datasheet, 2024.
- [43] Seeed Studio. Seeduino xiao esp32-c3 schematic, 2024.
- [44] Espressif Systems. *ESP32-C3 Technical Reference Manual*, 2024.
- [45] Espressif Systems. Esp-idf programming guide: Led control (ledc), 2025.
- [46] Random Nerd Tutorials. Esp32 web bluetooth, 2025.