UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA INFORMÁTICA

Diseño e implementación de un módulo para la realización de prácticas en la asignatura *"Sistemas de Tiempo Real"* basado en el sistema operativo RTEMS

Proyecto fin de grado

César Rodríguez Cerro

Tutor: Javier Fernández Muñoz

En Madrid, a 26 de Septiembre de 2017

“El hombre encuentra a Dios detrás de cada puerta que la ciencia logra abrir”

Albert Einstein.

Índice de contenidos

Índice de tablas

Índice de ilustraciones

Capítulo 1: Introducción

El primer capítulo del presente proyecto incluye una visión general del contenido del trabajo, las principales motivaciones que han llevado a su realización y una descripción de los objetivos generales.

1.1 Visión general

El fin del presente proyecto es la creación de un entorno de trabajo que englobe las herramientas necesarias para el desarrollo de aplicaciones de tiempo real y que permita después, generar una aplicación concreta que forme una de las prácticas de la asignatura *"Sistemas de Tiempo Real"* correspondiente al Grado en Ingeniería Informática de la Universidad Carlos III de Madrid.

En concreto, en este proyecto se utilizará el sistema operativo de tiempo real RTEMS para la implementación de un módulo correspondiente a un servidor de control remoto y la plataforma Arduino para el desarrollo del módulo correspondiente a un sistema de control empotrado en un dispositivo concreto. Ambos módulos se comunicarán entre sí para lograr un objetivo final y conformarán la práctica que los futuros alumnos deben realizar.

Además se proporciona la documentación necesaria para la creación del entorno de trabajo así como se propone el enunciado y los criterios de evaluación para una posible práctica.

1.2 Motivación

La elección de este proyecto se basó en un primer momento, en el interés surgido tras realizar una de las prácticas de la carrera correspondiente a la asignatura *"Sistemas de Tiempo Real".* Como breve explicación sobre el contenido de la práctica, decir que, se basaba en el desarrollo de una aplicación de tiempo real que tenía como objetivo automatizar el control de un dispositivo concreto.

Si bien, este interés llevó a la reflexión sobre la importancia en aumento de los sistemas empotrados y más concretamente de los de tiempo real. Hoy en día y con la evolución de la tecnología, los sistemas informáticos están cada vez más presentes en el entorno de la sociedad. Mediante su aplicación es posible comprender, organizar e interactuar con el entorno y mejorar la calidad de vida. Precisamente, con este fin nacen los sistemas de tiempo real, un tipo específico de sistema empotrado, que tiene como objetivo automatizar buena parte de las facetas de la vida cotidiana, beneficiando en gran medida a diversos ámbitos, tales como la seguridad, el transporte, la industria, las telecomunicaciones, etc.

Surge de esta manera la motivación por conocer tanto los sistemas operativos específicos como las plataformas hardware que permiten el desarrollo de este tipo de aplicaciones. Dentro de las plataformas hardware, supusieron mayor atractivo para el desarrollo de este proyecto aquellas que son de hardware libre debido a la libertad de implementación que ofrecen. El mismo motivo fue utilizado para la elección del sistema operativo.

Otra motivación añadida al desarrollo del proyecto fue la de ampliar conocimientos sobre la electrónica, aspecto básico en el diseño de un sistema embebido.

Finalmente la realización de este proyecto permitía generar un entorno de trabajo y posteriormente, una aplicación de tiempo real que conformase una de las prácticas para futuros alumnos cursantes de la asignatura *"Sistemas de Tiempo Real".* Lo que supuso también una motivación clave en la decisión de seleccionar este proyecto como trabajo de fin de grado.

1.3 Objetivos

El objetivo general de este trabajo de fin de grado es el diseño y la implementación de un módulo para la realización de prácticas en la asignatura *"Sistemas de Tiempo Real"* basado en el sistema operativo de tiempo real RTEMS.

Dicho módulo debe incluir:

- El diseño y la integración de un entorno de trabajo o framework que incluya el sistema operativo de tiempo real RTEMS para el desarrollo de aplicaciones de tiempo real.
- El diseño y la implementación de una aplicación de sistema real utilizando el framework previamente integrado que constituya una de las prácticas de la asignatura.

En concreto, para la integración del entorno de trabajo es necesario:

- Realizar un estudio para la elección de la plataforma tanto hardware como software más adecuada en el desarrollo de aplicaciones de tiempo real.
- Realizar un estudio para la elección del entorno de desarrollo más favorable en el que incorporar las funcionalidades de RTEMS para facilitar la implementación de aplicaciones basadas en este sistema operativo.
- Realizar un estudio para seleccionar la interfaz de programación que mejor se ajuste a las necesidades.
- Integrar todas las herramientas en una máquina virtual, conformando así el entorno de trabajo.
- Proporcionar la documentación necesaria para la generación del entorno de trabajo.

Una vez el framework se encuentre disponible, se implementará una aplicación que sirva de práctica a los alumnos de la asignatura. Además y con el objetivo de que el tutor de la asignatura sea capaz de evaluar a los estudiantes será necesario:

- Diseñar el enunciado de la práctica.
- Concretar los criterios que serán aplicados en la corrección de la misma.

Capítulo 2: Estado de la cuestión

El segundo capítulo comienza con la descripción de los sistemas empotrados en general y se exponen específicamente algunas de las plataformas hardware y software libres. El capítulo cierra con un análisis sobre los sistemas de tiempo real y sobre sus principales sistemas operativos.

2.1 Sistemas Embebidos

Un sistema embebido o empotrado es un sistema computacional que reside integrado en el dispositivo que supervisa y controla con el objetivo de cubrir unas necesidades específicas. Está compuesto tanto por hardware como por software y normalmente, en conjunto, forman parte de un sistema más grande y complejo. En lo referente a su programación, existen diferentes modalidades: programación en lenguaje de ensamblador específico del procesador que incorporan o mediante la implementación de programas en lenguajes de programación como Java, C o C++ alojados en sistemas operativos en tiempo real.

2.1.1 Características básicas

- Número limitado funciones a ejecutar: Como se ha mencionado anteriormente, un sistema embebido tiene un objetivo específico y por lo tanto las funciones que implemente deben ser de acuerdo al mismo.
- Capacidad de cálculo limitada: Como consecuencia de la especialización, los procesadores pueden ser más específicos y aportar beneficios palpables tales como un menor consumo de energía. Las CPUs suelen trabajar con registros de 8 o 16 bits debido a que las tareas son relativamente sencillas.
- Memoria escasa: Puesto que las tareas son de control (el tamaño del código es pequeño), generalmente, los sistemas empotrados tienen pocos recursos de memoria.
- El software y el dispositivo se crean a la vez: Ambas partes se deben integrar fuertemente. Además el Firmware del dispositivo no suele verse modificado, y si lo hace se modifica en su totalidad.
- Fiabilidad y seguridad: Si el sistema es crítico, los fallos en él pueden conllevar graves consecuencias tales como pérdida de vidas humanas o fallos medioambientales. Por ello el producto final es testeado a conciencia. Algunos sistemas empotrados implementan protocolos criptográficos para proteger la información y hacen uso de una comunicación autenticada si su ámbito de operación así lo requiere.
- Eficacia y eficiencia: La comunicación con el sistema controlado ha de ser rápida, además todos los recursos son utilizados óptimamente.

- Bajo consumo: Debido al crecimiento de las comunicaciones inalámbricas, el objetivo de muchos sistemas empotrados ha sido la comunicación remota, surgiendo así la posibilidad de que estos sistemas se ubiquen en lugares que no disponen de energía eléctrica. De esta manera el consumo se convierte en un aspecto fundamental a optimizar.

- Tamaño reducido y bajo coste: Los avances en la industria de la electrónica junto con las técnicas de diseño de sistemas empotrados, han permitido la creación de sistemas con un tamaño más reducido adaptándose a los nuevos procesos industriales de fabricación. Estos procesos también han favorecido una reducción de los costes y automatización de los procesos (muchos empotrados son producidos en millones de unidades) [1] [2].

2.1.2 Componentes hardware

El núcleo de un sistema empotrado está constituido por la unidad central de procesamiento (CPU), encargada de controlar las funciones específicas para las que está programado. Como consecuencia del amplio abanico de aplicaciones embebidas existe también una gran variedad de procesadores utilizados con un rango que se extiende desde un bajo rendimiento hasta una mayor complejidad y uso.

El software que se ejecuta sobre el procesador necesita de algún tipo de memoria en el que poder almacenarse para su posterior ejecución.

Puesto que el objetivo de un sistema empotrado es controlar el sistema en el que se encuentra, intrínsecamente está obligado a interactuar con el entorno que le rodea y con una amplia variedad de dispositivos tanto analógicos como digitales. Esta interacción se produce mediante un módulo de entradas y salidas que hacen posible la comunicación con sensores (encargados de recopilar los datos del entorno y entregárselos al procesador) y actuadores (cuya misión es realizar las acciones que la CPU ordene tras el procesamiento de datos).

Tras un esbozo de los componentes comunes a la mayoría de sistemas embebidos se procede a una explicación en profundidad de estos componentes comunes y de los específicos.

Unidad Central de Procesamiento (CPU)

El formato en el cual se encuentra alojada la CPU en el sistema empotrado puede ser de diferentes tipos:

- Microprocesador: Se trata del componente electrónico complejo en el que se encuentra la CPU de un sistema computacional. Sus componentes principales son la unidad de procesamiento (UP) encargada del tratamiento de datos y que incluye un conjunto de registros y una unidad aritmético lógica y la unidad de control (UC) que establece y coordina la secuencia de etapas que debe llevar a cabo el procesador. Además necesita de otro componente para desempeñar su función: el módulo de entradas y salidas (E/S) que recibe y envía datos desde y hacia el exterior. El conjunto del esquema genérico UP + UC + E/S compone la denominada máquina de tratamiento de la información o algorítmica (esto es, que implementan un algoritmo).

Si además se incluye una memoria principal (MP) como un elemento adicional diferenciado en la que se almacena tanto el programa como los datos se obtiene una máquina programable. Su función puede verse transformada sin el esfuerzo de alterar las conexiones, sino simplemente mediante el acceso y modificación de una determinada zona de memoria (programación). Todos los componentes se conectan y comunican entre sí mediante buses. Considerando el esquema UP + UC + E/S + MP se obtiene la arquitectura Von Neumann. La memoria principal es la memoria RAM (Random Access Memory), en la que se cargan todas las instrucciones del programa temporalmente para su ejecución por parte del procesador. Además se incluye una memoria caché, situada ente la MP y el procesador, que está caracterizada por un acceso más rápido a los datos usados con más frecuencia. La principal ventaja de la arquitectura Von Neumann es la adaptación a diferentes aplicaciones mediante la modificación del programa almacenado en memoria. Gracias a esta característica su uso se ha globalizado y es la subyacente en la mayoría de microprocesadores.

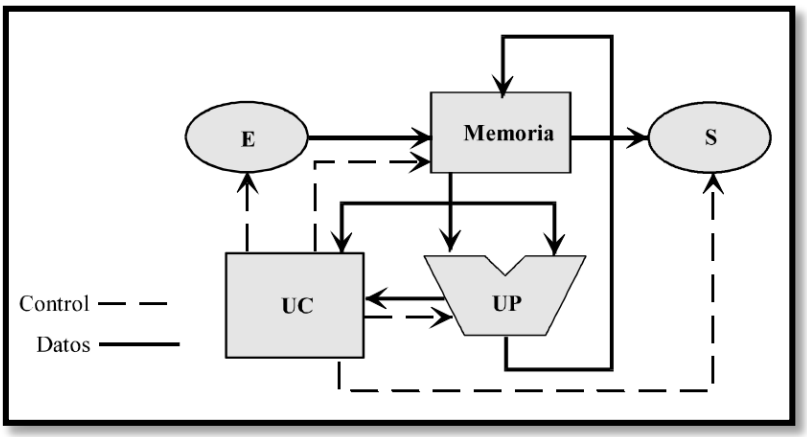


Ilustración 1. Máquina con arquitectura de Von Neumann

Los microprocesadores tienen gran potencia de cálculo, son capaces de manejar una cuantiosa cantidad de memoria y su velocidad de procesamiento es muy elevada.

Es por ello que son utilizados en los ordenadores personales y en las estaciones de trabajo, cuyo objetivo es el cálculo científico y el diseño en ingeniería. Los principales fabricantes de microprocesadores son las empresas *Intel Corporation* y *AMD* que abarcan casi en su totalidad el mercado actual. Ofrecen los procesadores más potentes y novedosos actualmente como puede ser el i9-7900X de Intel (familia Intel Core X-series) y el Ryzen 7 1800X (familia Ryzen 7), ambos con arquitecturas completamente nuevas y con frecuencias máximas de 4.30 GHz y 4 GHz respectivamente. ARM Holdings destaca con sus procesadores utilizados en dispositivos portátiles como smartphones y tabletas.

Ilustración 2. Intel i9-7900X

Ilustración 3. AMD Ryzen 7 1800x

- Microcontrolador: Es un tipo de microprocesador que contiene en su interior la arquitectura de von Neumann en su totalidad. Además de la CPU, se añade una cantidad de memoria y elementos de entrada y salida como dispositivos periféricos y puertos en el mismo circuito integrado. El microcontrolador ejecuta el programa que contiene en su propia memoria con una función específica. Esta memoria debe ser no volátil, de tipo ROM (Read Only Memory), que, según la aplicación destino del microcontrolador puede encontrarse en diferentes versiones: EPROM (Erasable Programmable Read Only Memory), EEPROM (Electrical Erasable Programmable Read Only Memory) y Flash. Además dispone de una memoria RAM en la que se almacenan datos y variables.

Este diseño ha permitido reducir el espacio de un sistema computacional y por ello son utilizados en sistemas empotrados con aplicaciones muy específicas que requieren pequeñas cantidades de memoria y una capacidad de cómputo inferior a la de un microprocesador. Su velocidad de operación por tanto, es menor en comparación con la de un microprocesador. Además se caracterizan por su reducido consumo energético, implementando incluso modos de reposo en la espera de un evento o interrupción.

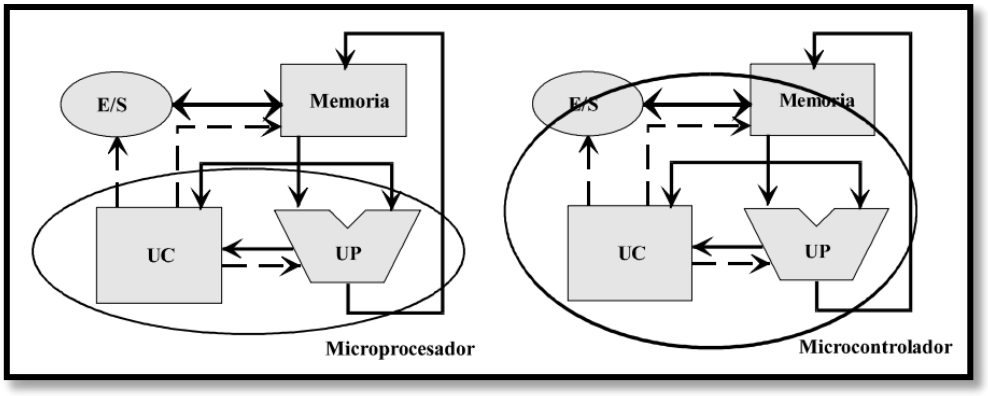


Ilustración 4. Microprocesador y microcontrolador

Existen microcontroladores de diferente tamaño dependiendo de la tarea que desempeñan, desde 4 y 8 bits para aplicaciones sencillas como el control de un electrodoméstico hasta de 32 y 64 bits en el procesamiento de imágenes y vídeo. Su ámbito de aplicación es muy extenso, son utilizados en automoción, tecnología médica, agricultura, industria militar y aeroespacial, seguridad, etc.

Existen múltiples empresas fabricantes de microcontroladores pero algunas de las principales son *Atmel, Freescale Semiconductor, Microchip Technology, Intel Corporation, Renesas o Texas Instruments* [3] [4].

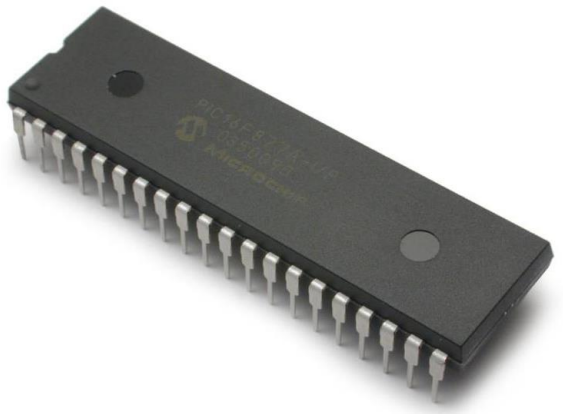


Ilustración 5. Microprocesador PIC16F877a de Microchip Technology

- Procesador Digital de Señales (DSP): Muchas aplicaciones embebidas realizan tareas de procesamiento digital de señales en tiempo real que requieren operaciones numéricas intensivas. Los procesadores diseñados específicamente para dichas tares son los DSP. Sus campos principales de actuación son las telecomunicaciones, la multimedia, la electrónica industrial y en general todos los que puedan ser relacionados con el procesamiento de señales. Entre los principales fabricantes destacan *Analog Devices, Hewlet Packard, Cirrus Logic, Creative Technology, Philips o Yamaha Corporation* [4].

Ilustración 6. DSP CS47L90 de Cirrus Logic

- Dispositivo Lógico Programable (Field Programmable Gate Array): Un FPGA es un dispositivo semiconductor programable que proporciona implementaciones de circuitos digitales de manera eficiente. Está formado por una matriz de bloques lógicos programables que, interconectados entre sí, permiten aumentar la funcionalidad tanto de las puertas lógicas más básicas como de circuitos combinacionales más complejos (por ejemplo un codificador y un multiplexor) o funciones matemáticas. La arquitectura general de una FGPA consiste en una matriz bidimensional de bloques lógicos programables denominados clústers lógicos que se comunican en horizontal (filas) y en vertical (columnas) mediante rutas o canales de conexión. En la intersección de los canales horizontales y verticales existe un bloque de conmutación que proporciona una conexión programable entre los mismos. Además, para la comunicación con los elementos exteriores disponen de un módulo de E/S por la que se reciben y envían las señales correspondientes.

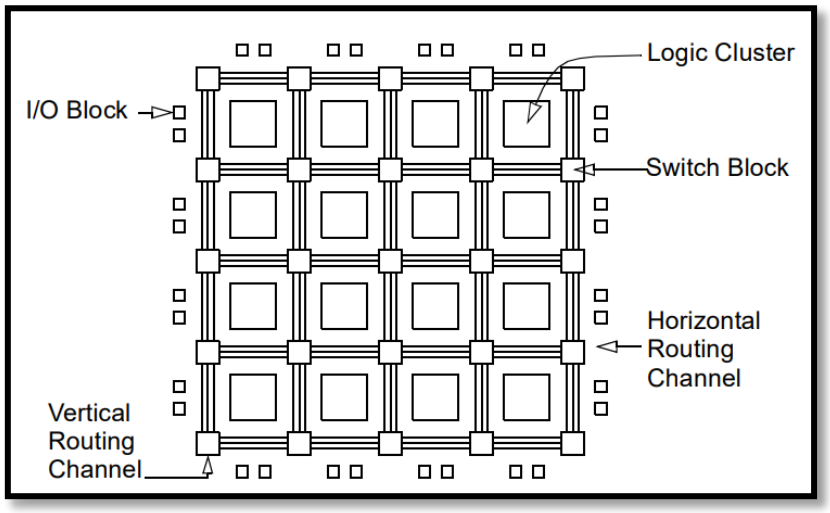


Ilustración 7. Arquitectura tradicional de las FGPA

Pueden incorporar dos tipos de memorias: no volátiles reprogramables (flash o EPROM) o no reprogramables (fusibles) y volátiles (RAM). Si hacen uso de una memoria RAM, necesita otra memoria que no sea volátil para configurar su arranque.

La peculiaridad de la programación de las FPGA es que no se realiza mediante código, si no que el programador debe establecer la función lógica mediante el diseño hardware que tendrá la misma. Para ello existen lenguajes de programación especiales denominados Hardware Description Languaje (HDL).

Entre sus principales ventajas destacan, su proceso de fabricación es rápido y tienen la capacidad de ser programadas infinidad de veces después de ser fabricadas, lo que permite mejorar el hardware o adaptarse a nuevos protocolos o especificaciones. Su coste de fabricación es bastante más reducido que el de los circuitos integrados de aplicaciones específicas (ASIC) para pequeñas producciones, ya que éstos necesitan instalaciones y procesos de fabricación mucho más especializados.

Gracias a la flexibilidad que ofrecen, están presentes en un gran número de aplicaciones, desde el procesamiento digital de señales hasta sistemas multimedia aplicados a la medicina.

También tienen desventajas, los circuitos que implementan son generalmente más grandes, más lentos y consumen más energía que los ASIC.

Los fabricantes de FPGA principales y más destacables son, entre otros, *Altera, Xilinx, Microsemi, Intel, Lattice Semiconductor, Atmel y Achronix Semiconductor* [5].

Reloj

Es necesario un módulo de reloj que se encargue de generar los pulsos de reloj necesarios para que el circuito digital esté provisto de una referencia temporal. Estas señales están generadas mediante un único oscilador principal, cuyas características son relevantes para la aplicación a la que esté destinado el sistema. Las principales características a tener en cuenta de un oscilador son la frecuencia, la precisión y el mantenimiento de la misma (cuando existan factores externos que puedan afectar) y el consumo eléctrico. Los osciladores que presentan unos factores más fiables de estas características son los basados en un resonador de cristal de cuarzo.

Comunicación

La comunicación es un aspecto principal en los sistemas empotrados. Hoy en día y gracias al auge de las conexiones inalámbricas no solo se dispone de la comunicación vía cable sino que también se cuenta con la comunicación inalámbrica. Típicamente la comunicación se ha realizado a través de interfaces de carácter estándar, entre las que se pueden diferenciar:

- Interfaces de comunicación Hombre-Máquina (HMI): Mediante estas interfaces el usuario puede comunicarse de una manera sencilla e intuitiva con el sistema embebido. Dispositivos como botones, monitores o mecanismos de introducción hacen capaz la comunicación.

- Interfaces eléctricas: Hacen posible la comunicación tanto entre los diferentes componentes internos como con otros dispositivos. Estándares como Serial Peripherial Interface (SPI), Circuito Interintegrado (I^2C) son algunos ejemplos.

- Interfaces de comunicación exterior: Son empleadas para la transmisión de datos entre diferentes equipos o dispositivos. Estas transmisiones pueden ser cableadas o inalámbricas. Los protocolos destacados en comunicación cableada son Recommended Standard 232 (RS-232), RS-485, el conocido Universal Serial Bus (USB), Ethernet, Puerto Paralelo Centronics o Local Interconnect Network (LIN). Las tecnologías y protocolos inalámbricos más utilizados son Global System for Mobile Communications (GSM), General Packet Radio Service (GPRS), Universal Mobile Telecommunications System (UMTS), Bluetooth o el más que conocido IEEE 802.11 Wi-Fi.

Módulo de Entradas y Salidas (I/O)

Los sistemas empotrados tienen la necesidad de conectar con hardware especializado del procesador que integran. Esta conexión se realiza a través del módulo de Entradas y Salidas, encargado de recibir o enviar las señales analógicas y digitales desde o hacia el hardware específico (sensores, actuadores, etc). El módulo I/O está formado por una serie de puertos o nodos que contienen un número determinado de registros para el almacenamiento temporal de los datos. Los principales puertos se pueden clasificar como:

- Puertos serie: Una manera efectiva de enviar y recibir datos es como secuencias de bits mediante un solo hilo de comunicación. Son denominados RS-232 (hacen uso de este protocolo) o puertos COM.

- Puertos paralelos: A diferencia de los puertos serie, transmite datos de forma paralela, es decir, a través de varios hilos de comunicación.

- Puertos universales: Permite conectar diferentes dispositivos con el estándar USB, con la ventaja de ofrecer mayor velocidad de transmisión.

- Puertos inalámbricos: No necesitan una conexión cableada si no que hacen uso de ondas electromagnéticas.

Sensores y Actuadores

La interacción con el entorno se hace efectiva a través de una serie de sensores y actuadores. Los sensores son los encargados de recopilar información del ambiente que rodea y desea controlar el sistema empotrado (sensores de temperatura, humedad, luminosidad, sísmicos, acústicos, magnéticos, etc). Una vez procesada la información, los actuadores hacen efectivas las órdenes del procesador, realizando las acciones necesarias en su ámbito de actuación (actuadores de sonido, motores eléctricos, diodos, etc).

Subsistema de presentación

Algunos sistemas empotrados integran un subsistema gráfico de comunicación con el usuario. Pueden ser dispositivos como diodos LED (Light-Emitting Diode), una pantalla LCD (Liquid Crystal Display) o las novedosas pantallas táctiles.

Módulo de Energía

Para alimentar a todos los componentes del sistema empotrado es necesario generar tensión y corriente mediante este módulo especializado. Normalmente se hace uso de una conexión a una fuente de alimentación (convertidor de potencia) si existe la posibilidad de acceso a la red eléctrica, para los dispositivos inalámbricos es necesario algún tipo de batería portátil e integrada en el sistema. Los valores de voltaje más utilizados en los sistemas empotrados suelen ser de 5, 9, 12 y 24 voltios.

Para algunas aplicaciones específicas alimentadas por batería, el consumo energético suele ser determinante, ya que la vida del sistema queda limitada por la misma [3] [6].

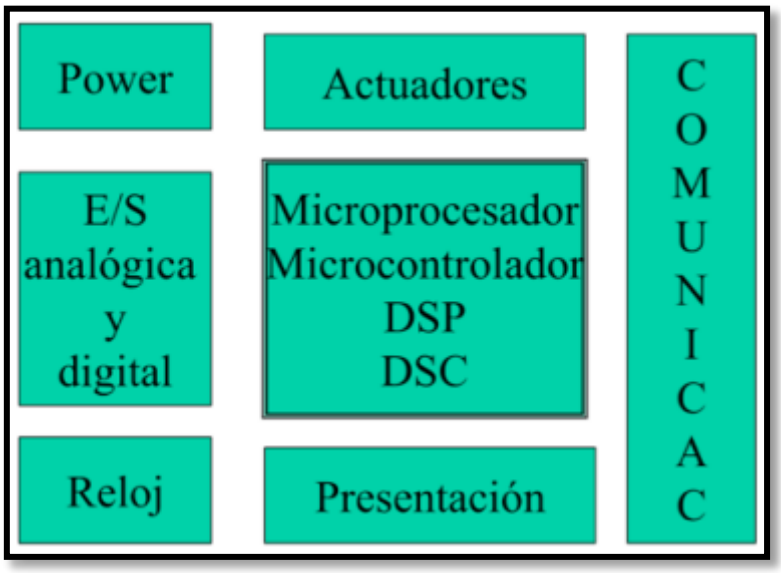


Ilustración 8. Diagrama de bloques de los componentes principales de un sistema embebido

2.1.3 Hardware y Software libre

La expresión software libre (open software) o hardware libre (open hardware) hace referencia al hardware o software cuyo diseño se encuentra publicado para que cualquiera tenga la posibilidad de uso, estudio, modificación, mejora e incluso posterior redistribución y venta. Obviamente cada término, respectivamente, hace uso de las mismas ideas pero aplicadas a su campo específico y esto provoca una serie de incompatibilidades en el caso del hardware que se analizan a lo largo de este apartado.

El open hardware incluye las especificaciones y diagramas necesarios para su comprensión. Los tres requisitos principales de un hardware abierto son:

1. Interfaz abierta: La interfaz con el hardware debe hacerse pública, para poder utilizarla libremente.
2. Diseño abierto: El diseño hardware debe ser público, para que exista la posibilidad de aprender de él e implementarlo.
3. Implementación abierta: Las herramientas necesarias para la creación del diseño deben ser libres, para poder desarrollar y mejorar el diseño.

Aplicar las mismas ideas del software abierto al hardware abierto es complicado puesto que existen ciertas diferencias sustanciales. A continuación se realiza una primera clasificación del hardware libre en base a su naturaleza y se enumeran los principales problemas.

Hardware estático

Comprende los componentes principales de un diseño hardware: el circuito esquemático, el circuito impreso, la información del diseño y la documentación asociada. Como consecuencia de la unión de todos estos elementos se produce un circuito físico final. Esta propiedad física no la tiene el software y como consecuencia, conlleva una serie de desventajas:

1. El diseño físico como tal es único: No existe el concepto de compartición de código fuente software de fácil duplicación. Para compartir un diseño físico es necesario un nuevo proceso de fabricación.
2. Coste de fabricación: Como todo proceso de fabricación, existe un coste asociado a los componentes necesarios y a la construcción del diseño.

3. Falta de material: Cabe la posibilidad de tener un problema con la disponibilidad de componentes, por ejemplo, si el país de origen del hardware es diferente al del diseñador que desea fabricarlo.

Queda patente de esta manera, las notables diferencias entre la divulgación del hardware y software libre.

*Har*dware dinámico o reconfigurable

Es el hardware que puede ser modificado mediante algún lenguaje de tipo HDL (Hardware Description Language) y que, por tanto, permite detallar su diseño y, en consecuencia, su funcionalidad. Su esencia es completamente diferente a la del hardware estático y además se desarrolla de manera parecida al software. Como ya se ha especificado anteriormente, un dispositivo FPGA programado con algún lenguaje HDL conforma el hardware reconfigurable. A partir de un fichero fuente se obtiene la secuencia de bits necesaria para que, al cargarla en el dispositivo FPGA, el diseño se materialice físicamente.

En el caso del hardware reconfigurable, no existen diferencias a la hora de ser compartido con respecto al software libre:

1. El uso, estudio, modificación y mejora se aplica al código generado mediante HDL, aplicándole una licencia GPL (General Public License).
2. Para desarrollar hardware en conjunto, surgen repositorios específicos en los que compartir código.

Entre las múltiples principales ventajas que ofrece el hardware libre se pueden destacar las siguientes:

- Permite que cualquier persona con cierto conocimiento sea capaz de elaborar proyectos complejos utilizando el hardware existente sin depender de empresas fabricantes, ahorrando de esta manera tiempo y dinero.
- Existe una competencia que asegura un hardware de calidad, eliminando la existencia de monopolios.
- Como consecuencia de su uso, se produce un aumento del conocimiento tecnológico enfocado a la electrónica, algo que es muy beneficioso por ejemplo en los centros de formación como colegios o universidades.

Como conclusión, disponer de una plataforma de hardware libre, proporciona una gran ventaja a los desarrolladores ya que no tienen la obligación de invertir en productos y patentes de alto coste y porque disponen de la autonomía necesaria para

la mejora de aplicaciones. Gracias al uso de estas herramientas, la innovación mundial se ha descentralizado (desde Silicon Valley o Nueva York por ejemplo) hacia múltiples partes del planeta que han sido dotadas con independencia tecnológica [7].

2.1.4 Plataformas de Hardware Libre

Arduino

Plataforma embebida de código abierto (open source) que permite la creación de prototipos basados en hardware y software libres. El hardware está formado por pequeñas placas de desarrollo programables basadas en microcontroladores y el software está compuesto por su entorno de desarrollo integrado (IDE) propio, Arduino IDE. No es necesaria ningún tipo de licencia para su uso, algo que permite el bajo coste de producción de sus placas en comparación con las de otras empresas.

- Hardware: Existen cantidad de placas o tarjetas Arduino con diferentes características y funcionalidades, que ofrecen un abanico muy amplio de implementaciones. Los microcontroladores más presentes en las tarjetas suelen ser de la familia AVR de Atmel, basados en 8 bits y en la arquitectura RISC (Reduced Instruction Set Computer). Actualmente incorporan microcontroladores de la nueva gama de Atmel con arquitectura ARM y basados en 32 bits. A continuación se realiza un análisis en función de las características principales (microcontrolador, voltaje de funcionamiento, Tensión, E/S digitales, E/E analógicas, etc).

  La propia compañía hace una clasificación de los diferentes modelos en base a las funcionalidades que ofrecen, diferenciándose de este modo: las placas de iniciación, las de funciones mejoradas, las destinadas al Internet de las Cosas y las destinadas al textil. De esta manera se detallan a continuación, algunas de las placas con características muy favorables a la iniciación.

Arduino UNO	
Microcontrolador	ATmega328P, AVR de 8 bits
Tensión de funcionamiento	5V
E/S Digitales	14 (6 de ellos proporcionan salida PWM)
Entradas Analógicas	6
Corriente DC por cada Pin E/S	20 mA
Corriente DC para el Pin de 3.3V	50 mA
Memoria Flash	32 KB (ATmega328P) 0.5 KB utilizados por el bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Velocidad de reloj	16 MHz

Tabla 1. Características Arduino UNO

Arduino UNO es una tarjeta recomendada en la familiarización con la electrónica y su programación, dispone de todo lo necesario para soportar el microprocesador. La alimentación viene dada por cable USB.

Arduino Leonardo	
Microcontrolador	ATmega32u4, AVR de 8 bits
Tensión de funcionamiento	5V
E/S Digitales	20
Canales PWM	7
Entradas Analógicas	12
Corriente DC por cada Pin E/S	40 mA
Corriente DC para el Pin de 3.3V	50 mA
Memoria Flash	32 KB (ATmega32u4) 4 KB utilizados por el bootloader
SRAM	2 KB (ATmega32u4)
EEPROM	1 KB (ATmega32u4)
Velocidad de reloj	16 MHz

Tabla 2. Características Arduino Leonardo

El modelo Leonardo es similar al UNO pero incorpora comunicación USB, lo que permite programar la placa como un dispositivo de entrada emulando un teclado o un ratón por ejemplo y un mayor número de entradas y salidas digitales.

Arduino Genuino 101	
Microcontrolador	Intel Curie, ARC de 32 bits
Tensión de funcionamiento	3.3V (Tolera 5V de E/S)
E/S Digitales	14 (4 de ellos proporcionan salida PWM)
PWM E/S Digitales	4
Entradas Analógicas	6
Corriente DC por cada Pin E/S	20 mA
Memoria Flash	196 KB
SRAM	24 KB
Velocidad de reloj	32 MHz
Otras características	Bluetooth y acelerómetro de 6 ejes

Tabla 3. Características Arduino Genuino 101

El Arduino Genuino 101 es uno de los pocos modelos que incorporan un microcontrolador Intel de 32 bits y su diseño se ha realizado mediante la colaboración de ambas compañías. Además incluye conexión Bluetooth y un acelerómetro/giroscopio de 6 ejes.

Se procede ahora a analizar parte de las placas que incorporan funcionalidades mejoradas o mayor capacidad de procesamiento.

Arduino Mega 2560	
Microcontrolador	ATmega2560, AVR de 8 bits
Tensión de funcionamiento	5V
E/S Digitales	54 (15 de ellos proporcionan salida PWM)
Entradas Analógicas	16
Corriente DC por cada Pin E/S	20 mA
Corriente DC para el Pin de 3.3V	50 mA
Memoria Flash	256 KB de los cuales 8 KB utilizados por el bootloader
SRAM	8 KB (ATmega328)
EEPROM	4 KB (ATmega328)
Velocidad de reloj	16 MHz

Tabla 4. Características Arduino Mega 2560

El Mega 2560 soporta proyectos más complejos, equipado con más pines de entrada y salida digitales y un tamaño mayor es el recomendado para proyectos de robótica.

Arduino Due	
Microcontrolador	AT91SAM3X8E, ARM de 32 bits
Tensión de funcionamiento	3.3V
E/S Digitales	54 (12 de ellos proporcionan salida PWM)
Entradas Analógicas	12
Salidas Analógicas	2 DAC
Corriente DC por cada Pin E/S	130 mA
Corriente DC para el Pin de 3.3V y de 5V	800 mA
Memoria Flash	512 KB para las aplicaciones
SRAM	96 KB (dos bancos: 64 KB y 32 KB)
Velocidad de reloj	84 MHz

Tabla 5. Características Arduino Due

El Arduino Due fue la primera placa Arduino basada en un microcontrolador ARM de 32 bits. Gracias a su elevado número de entradas y salidas digitales y analógicas es recomendado para proyectos de gran escala.

Arduino Genuino Zero	
Microcontrolador	ATSAMD21G18, ARM de 32 bits Cortex M0+
Tensión de funcionamiento	3.3V
E/S Digitales	20
Botones PWM	3, 4, 5, 6, 8, 9, 10, 11, 12, 13
Dispositivos UART	2 (Programación y nativo)
Entradas Analógicas	6, canal ADC de 12 bits
Salidas Analógicas	1, DAC de 10 bits
Interrupciones externas	Todos los pines menos el 4
Corriente DC por cada Pin E/S	7 mA
Memoria Flash	256 KB
SRAM	32 KB
Velocidad de reloj	48 MHz

Tabla 6. Características Arduino Genuino Zero

Se trata de un modelo que proporciona un mayor rendimiento gracias a su controlador de 32 bits destinado a proyectos de alta tecnología. Su característica más diferenciable es la incorporación de un Debugger Empotrado Atmel (EDGB) que ofrece una interfaz de depuración muy completa.

Arduino Mega ADK	
Microcontrolador	ATmega2560, AVR de 8 bits
Tensión de funcionamiento	5V
E/S Digitales	54 (15 de ellos proporcionan salida PWM)
Entradas Analógicas	16
Corriente DC por cada Pin E/S	40 mA
Corriente DC para el Pin de 3.3V	50 mA
Memoria Flash	256 KB de los cuales 8 KB utilizados por el bootloader
SRAM	8 KB (ATmega328)
EEPROM	4 KB (ATmega328)
Velocidad de reloj	16 MHz

Tabla 7. Características Arduino Mega ADK

La principal característica que diferencia al Arduino Mega ADK de los demás es la integración de una interfaz USB para conexión con teléfonos equipados con Android, de esta manera, amplía el muestrario de potenciales aplicaciones.

A continuación se analizan algunas de las principales placas orientadas al *"Internet of Things"* (IoT), que se diferencian del resto por su capacidad de conexión a internet y además, porque incorporan un microprocesador adicional. Por ende, su coste es más elevado que el de las demás.

Arduino Yún	
Microcontrolador	ATmega32U4, AVR de 8 bits
Tensión de funcionamiento	5V
E/S Digitales	20
Salidas PWM	7
E/S Analógicas	12
Corriente DC por cada Pin E/S	40 mA
Corriente DC para el Pin de 3.3V	50 mA
Memoria Flash	32 KB de los cuales 4 KB utilizados por el bootloader
SRAM	2.5 KB
EEPROM	1 KB
Velocidad de reloj	16 MHz
Microprocesador	Atheros AR9331, MIPS
Tensión de funcionamiento	3.3V
Ethernet	IEEE 802.3 10/100Mbit/s
WiFi	IEEE 802.11b/g/n
USB	2.0
Lector de tarjetas	Micro-SD
RAM	64 MB DDR2
Memoria Flash	16 MB
SRAM	2.5 KB
EEPROM	1 KB
Velocidad de reloj	400 MHz

Tabla 8. Características Arduino Yún

Este modelo está equipado además de con el microcontrolador correspondiente, con un microprocesador Atheros AR9331 basado en la arquitectura MIPS. Este procesador soporta una distribución de Linux llamada Linino OS, lo cual ofrece un ordenador con conexión a la red y con las funcionalidades de Arduino. Es capaz de comprender la programación de scripts (python y shell). Esta placa soporta tanto la conexión vía WiFi como Ethernet y el reconocimiento de tarjetas Micro-SD. Cabe destacar la memoria RAM DDR2 (Double Data Rate type two) de 64 MB muy potente que incorpora su procesador.

Arduino Tian	
Microcontrolador	SAMD21G18, ARM de 32 bits Cortex M0+
Tensión de funcionamiento	3.3V
E/S Analógicas	6
Corriente DC por cada Pin E/S	7 mA
Memoria Flash	256 KB
SRAM	32 KB
Velocidad de reloj	48 MHz
Microprocesador	Atheros AR9342, MIPS
Tensión de funcionamiento	3.3V
Ethernet	IEEE 802.3 10/100Mbit/s
WiFi	IEEE 802.11b/g/n
USB	2.0
RAM	64 MB DDR2
Memoria Flash	16 MB + 4GB eMMC (Multimedia Card)
Velocidad de reloj	560 MHz
Otras características	Bluetooth

Tabla 9. Características Arduino Tian

Tian, tiene características muy similares a la placa Yun, incorpora un microcontrolador de 32 bits y un procesador Atheros AR9342 con una distribución de Linux. Su principal característica es que dispone de hasta 4 GB adicionales a los 16 Mb incorporados de memoria flash, gracias a su adaptador para tarjetas eMMC.

Arduino Ethernet	
Microcontrolador	ATmega328, AVR de 8 bits
Tensión de funcionamiento	5V
E/S Digitales	14 (4 de ellos proporcionan salida PWM)
Entradas Analógicas	6
Corriente DC por cada Pin E/S	40 mA
Corriente DC para el Pin de 3.3V	50 mA
Memoria Flash	32 KB (ATmega328) de los cuales 0.5 KB utilizados por el bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad de reloj	16 MHz
Controlador Ethernet Empotrado	W5100 TCP/IP
Lector de tarjetas	Micro-SD

Tabla 10. Características Arduino Ethernet

Como su propio nombre indica dispone de conexión vía Ethernet que se diferencia en la integración de un módulo Wiznet Ethernet.

Arduino también se ha lanzado hacia la producción de pequeños dispositivos programables que podemos llevar literalmente "puestos encima" (pulseras, relojes, módulos cosidos a la ropa, etc). La siguiente placa está basada en esta característica.

Arduino LilyPad	
Microcontrolador	ATmega168 o ATmega328V, AVR basado en 8 bits
Tensión de funcionamiento	2.7-2.5V
E/S Digitales	14
PWM E/S Digitales	6
Entradas Analógicas	6
Corriente DC por cada Pin E/S	40 mA
Memoria Flash	16 KB de los cuales 2 KB utilizados por el bootloader
SRAM	1 KB
EEPROM	512 bytes
Velocidad de reloj	8 MHz

Tabla 11. Características Arduino LilyPad

Como se puede observar, son dispositivos con características y funcionalidades reducidas en comparación con los demás debido a las necesidades de su ámbito de aplicación [8] [9].

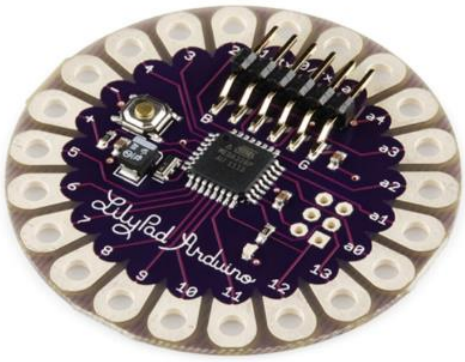


Ilustración 9. LilyPad Arduino

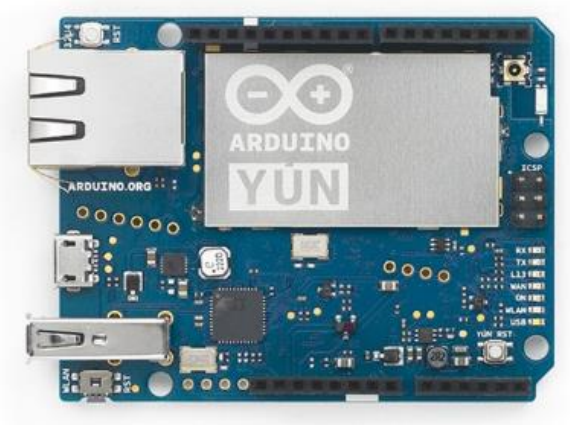


Ilustración 10. Arduino YÚN

- Software: Arduino dispone de su propio entorno de desarrollo integrado, Arduino IDE (open source) que proporciona todo lo necesario para programar el dispositivo. El entorno está basado en entorno de Processing (Processing Development Environment) y existen versiones para Linux, Windows y Mac OS X. El lenguaje de programación de Arduino está basado en C++ y se trata de una adaptación de avr-libc que proporciona una biblioteca C para aplicar GNU

Compiler Collection (GCC) en los microcontroladores que integra la marca (Atmel AVR).

El entorno de desarrollo permite implementar, compilar y subir el programa a la placa mediante comunicación serial y un programa de carga (bootloader) alojado en la memoria del microcontrolador de la placa. Los archivos de programación de Arduino se denominan "Sketches", y son escritos normalmente en C/C++, aunque también se pueden implementar mediante otros lenguajes de programación como .NET, Python, Node.js, C# o Java.

Obviamente Arduino dispone de su propia API (Application Programming Interface) que proporciona y facilita la programación de los aspectos más básicos de las placas, como por ejemplo escribir en el puerto serie y que se incluyen en la instalación del IDE. Algunas de estas librerías son: Ethernet (comunica las capas 2 y 3 del protocolo de red mediante Ethernet), Firmdata (comunicación serial), SD (lectura y escritura en tarjetas SD) o LiquidCrystal (control de dispositivos LCD). Además cualquiera puede implementar su propia librería en Arduino, mediante el uso de dos archivos, un archivo de cabecera (con extensión .h) que contiene las definiciones de la librería y un archivo con el código fuente (con extensión .cpp) [10] [11].

Raspberry Pi

Es un computador de placa simple (Single Board Computer) y bajo coste, que ejecuta como sistema operativo, una versión de Debian Linux denominada Raspbian. Fue diseñado originalmente para promover la enseñanza de las ciencias de la computación e inspirar a jóvenes programadores en escuelas y universidades. Desarrollado por la Fundación Raspberry Pi en Reino Unido, está destinado para servir de plataforma para aquellos interesados en la programación de periféricos a bajo nivel. A pesar de que diferentes empresas tienen contratos de venta y distribución de Raspberry Pi, el libre uso tanto del hardware como del software está permitido a nivel educativo y particular.

- Hardware: A pesar del pequeño tamaño de sus placas se trata de un computador completamente funcional. Todas las placas incorporan un SoC (System-on-a-Chip, un circuito integrado que integra todos los componentes de un computador en un solo chip) Broadcom que contiene un microprocesador ARM, una unidad de procesamiento gráfico VideoCore, un procesador digital de señales (DSP), un puerto USB y memoria SDRAM (compartida entre la CPU y la GPU). La versión del Broadcom es de diferente tipo dependiendo el modelo de placa. A continuación se realiza un análisis de las principales placas de Raspberry Pi actuales, centrado en sus características:

Raspberry Pi 1 Modelo A+	
SoC	700 MHz Broadcom BCM2835 (ARMv6Z 32 bits, single-core) CPU with 512 MB SDRAM. Broadcom VideoCore IV GPU
Tensión de funcionamiento	5V
E/S	40(GPIO)
USB 2.0	2
Video Input	Interfaz serial para cámara Raspberry Pi
Video Output	HDMI
Almacenamiento	Puerto MicroSD

Tabla 12. Características Raspberry Pi 1 Modelo A+

Se trata del primer modelo de Raspberry mejorado, con un procesador de un solo núcleo y 512 MB de memoria SDRAM.

Raspberry Pi 1 Modelo B+	
SoC	700 MHz Broadcom BCM2835 (ARMv6Z 32 bits, single-core) CPU with 512 MB SDRAM. Broadcom VideoCore IV GPU
Tensión de funcionamiento	5V
E/S	40(GPIO)
USB 2.0	4
Video Input	Interfaz serial para cámara Raspberry Pi
Video Output	HDMI
Almacenamiento	Puerto MicroSD
Adaptador de red	USB/Ethernet chip

Tabla 13. Características Raspberry Pi 1 Modelo B+

El modelo B+ es una variación del modelo A con la mejora de incorporar un puerto USB y un conector Ethernet.

Raspberry Pi 2 Modelo B	
SoC	900 MHz Broadcom BCM2837 (ARMv8-A 64 bits, quad-core) CPU with 1 GB SDRAM. Broadcom VideoCore IV GPU
Tensión de funcionamiento	5V
E/S	40(GPIO)
USB 2.0	4
Video Input	Interfaz serial para cámara Raspberry Pi
Video Output	HDMI
Almacenamiento	Puerto MicroSD
Adaptador de red	USB/Ethernet chip

Tabla 14. Características Raspberry Pi 2 Modelo B

Raspberry Pi 2 Modelo B ya incorpora un procesador más potente de 900 MHz y cuatro núcleos y 1GB de SDRAM.

Raspberry Pi 3 Modelo B	
SoC	1.2 GHz Broadcom BCM2837 (ARMv8-A 64 bits, quad-core) CPU with 1 GB SDRAM. Broadcom VideoCore IV GPU
Tensión de funcionamiento	5V
E/S	40(GPIO)
USB 2.0	4
Video Input	Interfaz serial para cámara Raspberry Pi
Video Output	HDMI
Almacenamiento	Puerto MicroSD
Red	USB/Ethernet chip, 802.11 Wireless, Bluetooth 4.1

Tabla 15. Características Raspberry Pi 3 Modelo B

Esta placa incluye un procesador de mayor frecuencia respecto a Raspberry Pi 2 Modelo B de 1.2GHz. Además incluye conexión Wi-Fi y Bluetooth.

Los modelos analizados a continuación forman parte de otra gama de placas más pequeñas y de menor potencia que las anteriores. Por el contrario, su precio es menor.

Raspberry Pi Zero	
SoC	1 GHz Broadcom BCM2835 (ARMv6Z 32 bits, single-core) CPU with 512 MB SDRAM. Broadcom VideoCore IV GPU
Tensión de funcionamiento	5V
E/S	40(GPIO)
Micro-USB	1
Video Input	Interfaz serial para cámara Raspberry Pi
Video Output	Mini-HDMI
Almacenamiento	Puerto MicroSD

Tabla 16. Características Raspberry Pi Zero

Raspberry Pi Zero W	
SoC	1 GHz Broadcom BCM2835 (ARMv6Z 32 bits, single-core) CPU with 512 MB SDRAM. Broadcom VideoCore IV GPU
Tensión de funcionamiento	5V
E/S	40(GPIO)
Micro-USB	1
Video Input	Interfaz serial para cámara Raspberry Pi
Video Output	Mini-HDMI
Almacenamiento	Puerto MicroSD
Red	USB/Ethernet chip, 802.11 Wireless, Bluetooth 4.1

Incluye conexión Wi-Fi y Bluetooth respecto al modelo Raspberry Pi Zero [12].

Tabla 17. Características Raspberry Pi Zero W

Ilustración 11. Raspberry Pi 3 Model B

Ilustración 12. Raspberry Pi Zero W

- Software: Como se ha mencionado anteriormente el sistema operativo creado para Raspberry Pi está basado en la distribución Debian Linux, se trata de la versión Raspbian. Las placas no solo soportan esta versión, si no que pueden incorporar otros sistemas operativos como Pidora, Ubuntu Snappy, OSCM (Open Service Catalog Manager) o incluso Windows 10.

Debido a que los procesadores integrados son todos de arquitectura ARM, las versiones de los sistemas operativos deben estar compiladas para dicha arquitectura [13].

BeagleBoard

BeagleBoard es una plataforma de código abierto (open source) tanto hardware como software dedicada a la computación embebida. Fue creada por la empresa norteamericana *Texas Instruments* con el objetivo de promocionar la educación, el diseño y el uso de software y hardware open source. Su distribución y venta está abierta a decenas de distribuidores de todo el mundo y cualquier persona puede hacer uso de la plataforma a nivel educativo o particular.

- Hardware: La solución hardware que proporcionan son computadores de placa simple (Single Board Computer) de bajo coste que integran un SoC con los propios procesadores fabricados por Texas Instruments correspondientes la serie de ARM, Cortex-A. A continuación se realiza un análisis de algunas placas de BeagleBoard actuales, centrado en sus características:

BeagleBone Blue	
Soc	1GHz ARM Cortex-A8 con 512MB RAM DDR3 y 4GB de Flash eMMC
Tensión de funcionamiento	5V
USB 2.0	1
Video Output	HDMI
Almacenamiento	Puerto MicroSD
Red	WiFi, Bluetooth
Sensores	Acelerómetro, giroscopio y barómetro

Tabla 18. Características BeagleBone Blue

BeagleBone Green SeeedStudio	
Soc	1GHz ARM Cortex-A8 con 512MB RAM DDR3 y 4GB de Flash eMMC. Aceleración de gráficos 3D.
Tensión de funcionamiento	5V
E/S Analógicas	2 cabezales de 46
USB 2.0	1
Video Output	HDMI
Almacenamiento	Puerto MicroSD
Red	Ethernet
Sensores	Acelerómetro, giroscopio y barómetro

Tabla 19. Características BeagleBone Green SeeedStudio

Como se puede observar en las características son placas potentes que se asemejan más a un ordenador convencional. Son destinadas a proyectos que necesitan mayor capacidad de cómputo y memoria.

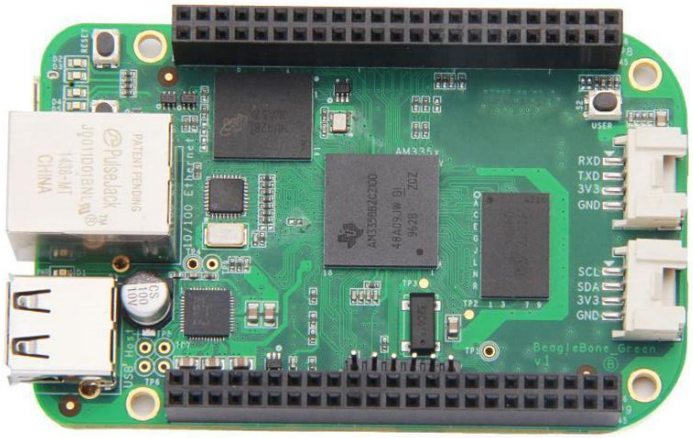


Ilustración 10. BeagleBone Green SeeedStudio

- Software: Las placas BeagleBoard soportan una cantidad importante de sistemas operativos como Ubuntu, QNX, Windows Embedded, Android, Fedora, RISC OS, etc, pero sin duda la más utilizada y recomendada por la propia organización es Debian [14].

2.2. Sistemas de Tiempo Real

Tras la explicación de las características y componentes principales de los sistemas empotrados, se detalla a continuación el tipo de sistema empotrado en el que se basa este proyecto, los sistemas de tiempo real (**STR**).

Un sistema de tiempo real es un sistema informático que interacciona con un entorno dinámico, es decir, que evoluciona con el transcurso del tiempo. Las respuestas de estos sistemas ante los estímulos recibidos deben ejecutarse dentro de un intervalo de tiempo establecido para que el funcionamiento sea correcto. Las aplicaciones de tiempo real abarcan diferentes ámbitos tales como la aviónica, el control del tráfico aéreo, las telecomunicaciones o la electrónica de consumo. Sus características principales son:

- Gran complejidad y tamaño: Hoy en día y debido a la continua evolución de los sistemas y plataformas de comunicación, los sistemas de tiempo real son cada vez más elaborados y complejos, mezclando incluso tareas muy diferentes entre sí.
- Seguridad y fiabilidad: Un fallo en un sistema crítico puede suponer consecuencias importantes como la pérdida de vidas humanas. Es por ello que los sistemas deben invertir en seguridad y ser fiables frente a averías.
- Comportamiento determinista: Es imprescindible que la actuación de un STR sea previsible para contemplar todos los casos posibles de ejecución. El sistema debe comportarse adecuadamente en todas las situaciones, para ello es necesario testear el sistema con ahínco, forzando los peores casos.
- Concurrencia: Las acciones del sistema que supervisan y controlan el dispositivo se ejecutan de manera concurrente.

Las actividades que implementan y ejecutan los STR se denominan tareas y están caracterizadas por los siguientes parámetros temporales:

- Periodo: Cada tarea se ejecuta de manera regular cada intervalo de tiempo determinado denominado periodo.
- Plazo de ejecución: Es el tiempo máximo que puede transcurrir entre el instante en el que debe activarse la tarea y su finalización.
- Instante de activación: Es el instante de tiempo a partir del cual puede comenzar la ejecución de la tarea.

- Arranque: Es el instante de tiempo en el que comienza la ejecución de la tarea.
- Terminación: Instante de tiempo en el que termina la ejecución de la tarea.
- Tiempo de ejecución o cómputo: Es el tiempo necesario de uso de la CPU por parte de la tarea hasta completar su ejecución. Obviamente este tiempo depende de la complejidad de la tarea y de la velocidad que aporte el procesador del sistema [15].

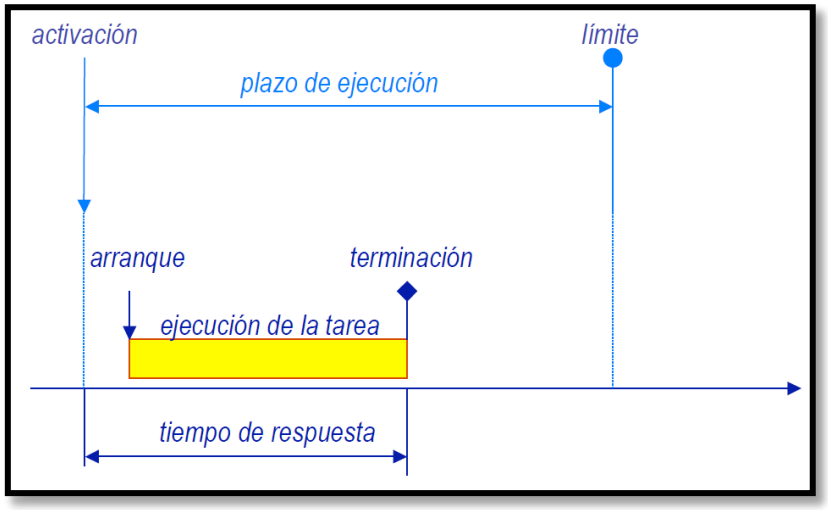


Ilustración 11. Diagrama de ejecución de una tarea de tiempo real

2.2.1 Tipos de Sistemas de Tiempo Real

Se puede realizar una clasificación de los STR atendiendo al nivel de tolerancia aceptado en la violación del plazo temporal asociado a cada actividad que ejecuta el sistema. Están caracterizados por sus requisitos temporales y de fiabilidad. Surgiendo así los sistemas críticos o estrictos (hard real-time) y los sistemas acríticos o laxos (soft real-time). En los sistemas críticos todas las acciones deben terminar dentro del plazo especificado y por tanto un plazo perdido supone un error y es suficiente para que el sistema falle por completo. Además las consecuencias del fallo del sistema son graves. Un requisito esencial de los sistemas críticos es la certificación mediante una entidad externa que garantice el cumplimiento de los requisitos. Por el contrario, en los sistemas laxos, la pérdida de un plazo no tiene por qué suponer un error.

Clasificándolos por su comportamiento en caso de avería encontramos los sistemas con parada segura (fail-safe) y los sistemas con degradación aceptable (fail-soft). En los sistemas con parada segura, cuando se detecta un fallo se produce una detención de manera segura, minimizando así las posibles consecuencias. En cambio cuando un fallo es detectado en un sistema con degradación aceptable, la ejecución del sistema continúa, pero con una limitación en las funcionalidades asociadas.

Distinguiendo los sistemas por su grado de determinismo temporal, surgen los sistemas con respuesta garantizada (guaranteed response systems) y los sistemas que hacen lo que pueden (best-effort systems). Los primeros están caracterizados por garantizar el comportamiento temporal de forma analítica, siendo necesario verificar la carga máxima de tareas y los posibles fallos. En el segundo grupo, por contra, no es necesaria dicha verificación de la carga y los fallos.

En base a la cantidad de recursos de los que dispongan, existe un primer grupo, los sistemas con recursos adecuados (resource-adequate systems), cuyo diseño tiene recursos suficientes para garantizar el comportamiento con carga máxima y en cualquier escenario de fallo. Por otro lado, se encuentran los sistemas con recursos inadecuados (resource-inadequate systems), que trabajan en base a la idea de que la provisión de los recursos necesarios para manejar todas las situaciones posibles tengan costes económicos elevados y por lo tanto realizan una asignación dinámica de los mismos según surgen los diferentes escenarios.

Mediante la forma de comienzo de la ejecución de sus tareas, se pueden diferencias los sistemas dirigidos por eventos (event-triggered systems) y los sistemas dirigidos por tiempo (time-triggered systems). El primer conjunto inicia sus actividades cuando se observa un cambio significativo de estado, es decir, cuando surge un evento diferente al de la señal del reloj (una interrupción). El segundo conjunto inicia sus tareas en instantes determinados de tiempo (marcados por una interrupción de reloj).

En base a la instalación del sistema de tiempo real, se diferencian por la forma de integrar e instalar el sistema de tiempo real en el anfitrión que lo albergue. De esta diferenciación surgen los sistemas convencionales, en los que el sistema se instala en una máquina convencional y por ello disfruta de todos sus mecanismos y recursos. Por otra parte existen los sistemas empotrados, en los que el computador es una parte más de un sistema mayor y no tiene aspecto de computador (no es perceptible ni alterable por el usuario, sus recursos son bastante limitados y su interfaz de usuario está muy caracterizada y totalmente orientada a la aplicación específica que implementa).

Existe otra clasificación de los sistemas de tiempo real basada en su forma de organización. Un sistema centralizado es aquel en el que todo el procesamiento de cómputo se produce en una máquina central que se encuentra conectada a los diferentes terminales y periféricos que controla. Sin embargo un sistema de tiempo real distribuido es aquel que está formado por recursos de computación, tanto hardware como software, que se encuentran físicamente distribuidos e interconectados a través de redes y que trabajan de manera conjunta para lograr un determinado objetivo.

2.2.2 Planificación de Sistemas de Tiempo Real

Para la ejecución de múltiples tareas en un mismo procesador, es necesaria una repartición de recursos entre ellas durante un cierto periodo de tiempo. Además, es necesario asegurarse de que el conjunto de tareas cumplirá los requisitos temporales que exige la aplicación. Para que esta planificación sea efectiva deben existir dos elementos fundamentales: un algoritmo de planificación, para coordinar la repartición de recursos (procesador) a las tareas y un método de análisis para determinar el comportamiento temporal del sistema.

Un algoritmo de planificación ordena la ejecución del conjunto de tareas en base a un criterio predefinido. La consideración de los requisitos temporales es necesaria para ejecutar las tareas de forma garantizada, y de esa manera lograr que el sistema operativo correspondiente tenga un comportamiento de fácil predicción. Obviamente los sistemas operativos de tiempo real cumplen con este tipo de comportamiento.

Como se ha expuesto anteriormente, también es necesario realizar un método de análisis para asegurar el cumplimiento de los requisitos temporales en cualquier situación. En este escenario surgen los planificadores de tareas. En función de cuándo se realice el análisis, los planificadores se pueden caracterizar en dos grupos: estáticos (off-line) cuando el análisis se produce antes de la ejecución (en la fase de diseño), y dinámicos (on-line) cuando el análisis tiene lugar durante la ejecución.

En lo que respecta a este proyecto el interés está centrado en los planificadores estáticos.

Planificación cíclica

Se basan en el análisis de la ejecución de las tareas dando por hecho una ejecución determinista y predecible establecida antes de la misma (off-line). En este tipo de planificación se establecen los instantes en los que se deben ejecutar las actividades, teniendo en cuenta su tiempo de ejecución. De esta manera el planificador únicamente tiene la tarea de ejecutar las tareas o actividades siguiendo el plan de ejecución definido previamente.

Este tipo de planificación es usado con sistemas de control en los que las actividades tienen periodos de ejecución iguales o múltiplos entre sí. Al estar el plan definido a priori, puede surgir el problema de indeterminación si una tarea excede el tiempo de ejecución normal, y por ello hay que tenerlo en cuenta y mitigarlo en la medida de lo posible (tolerancia a fallos).

Además, para realizar un plan de ejecución fijo es necesario que todas las tareas tengan naturaleza periódica. Por lo tanto el esquema de ejecución (formado por todas las actividades) se repite cada cierto tiempo, lo que se denomina ciclo principal que es calculado como el mínimo común múltiplo de los periodos de todas las actividades. El ciclo principal se divide en ciclos secundarios en los cuales se ejecutan secuencias de tareas distintas. La ejecución del ciclo principal se repetirá constantemente de manera periódica hasta que se produzca un cambio de modo de ejecución o finalice la misma.

Planificación basada en prioridades

Este tipo de algoritmos seleccionan la actividad a ejecutar en función de la prioridad de la misma. Dicha prioridad puede ser de naturaleza fija (no modificable y establecida por el diseñador) o dinámica (modificable y establecida por el sistema en sí).

En la planificación con prioridades fijas es necesario que las tareas se modelen como procesos concurrentes y que, cada una de ellas pueda encontrarse en diferentes estados:

- Ejecutándose: La actividad tiene el control del recurso (procesador).
- Lista: La actividad está preparada para tomar el control del procesador pero aún no lo tiene debido a que otra tarea de mayor prioridad lo está usando.
- Suspendida: La actividad finalizó su ejecución y está a la espera de activación.

De esta manera el planificador determina qué actividad ejecuta en un determinado instante del tiempo. El despacho de las tareas pude se expulsivo o no expulsivo. Si es expulsivo, cuando una tarea con mayor prioridad que la que en ese momento tiene en su poder el recurso (procesador), se activa y la reemplaza en la ejecución. En el caso contrario, en un planificador no expulsivo, una actividad no libera el recurso hasta que no termina o lo cede de manera voluntaria.

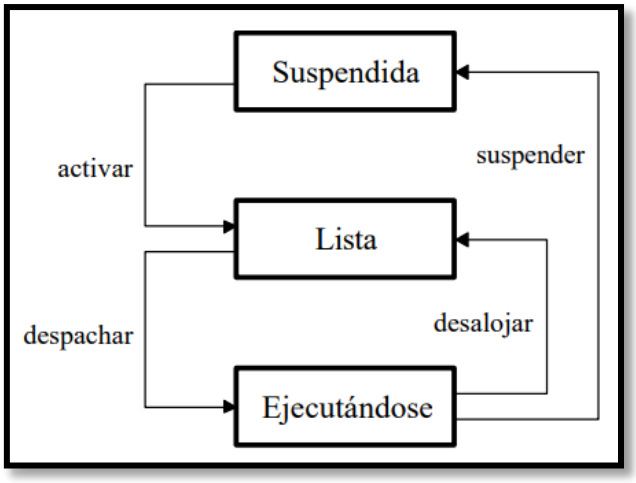


Ilustración 12. Estados de una tarea con prioridades fijas

El planificador de prioridades estáticas más comúnmente usado para tareas periódicas es el Planificador de Prioridades Monótonas en Frecuencia (Rate Monotonic), en el cual se establece mayor prioridad a las actividades de menor periodo. Los requisitos para que este planificador sea óptimo son los siguientes:

- Todas las tareas deben ser periódicas e independientes.
- Los plazos de las actividades deben ser iguales que sus respectivos periodos.
- Como se ha adelantado anteriormente, la asignación de prioridades debe realizarse de formo inversa al periodo, esto es, la tarea de menor periodo tendrá una mayor prioridad.

Cuando los plazos de las actividades puedan ser distintos a sus respectivos periodos (menores o iguales) se hace uso del Planificador de Prioridades Monótonas en Plazos (Deadline Monotonic):

- Todas las tareas deben ser periódicas e independientes.
- Los plazos de las actividades deben ser iguales o menores que sus respectivos periodos.
- La asignación de prioridades debe realizarse de formo inversa al plazo de finalización, esto es, la tarea de menor plazo de finalización tendrá una mayor prioridad [15] [16].

2.2.3 Entorno de ejecución de un sistema de tiempo real

El firmware de un dispositivo es el conjunto de instrucciones que se encuentra integrado directamente con el hardware del dispositivo y que ejerce el control de los circuitos del mismo. Entre sus principales funciones destacan la coordinación de las rutinas de funcionamiento, la gestión del arranque del dispositivo, la implementación de una interfaz para la configuración del sistema (no siempre es así). Por lo tanto, como es obvio, el firmware en los sistemas de tiempo real juega un papel fundamental.

Normalmente el firmware o entorno de ejecución de un sistema empotrado o de tiempo real suele estar formado por diferentes componentes:

- La aplicación: El programa implementado por el diseñador para una tarea específica (normalmente suele ser único).

- El núcleo de ejecución: El entorno que aporta la funcionalidad que el programa necesita (bibliotecas). El run-time suele ser o estar relacionado con un lenguaje de programación específico.
- El sistema operativo: Es el encargado de interactuar y controlar el hardware del dispositivo.

También debe cumplir una serie de requisitos:

- Simplicidad: Deben incluirse solo las funciones necesarias para el desarrollo de la aplicación de tiempo real, para así evitar un consumo innecesario de recursos y simplificar el software.
- Eficiencia: Es necesario garantizar el cumplimiento de los requisitos temporales, para de esta manera no transitar a modos de ejecución erróneos.
- Determinismo: Es necesario para poder analizar el comportamiento temporal.

Además el firmware de un sistema de tiempo real se crea a medida para la aplicación específica que se va a desarrollar. La metodología de desarrollo utilizada es el desarrollo cruzado en el que se utiliza un compilador que es capaz de crear código ejecutable para otra plataforma completamente distinta [16].

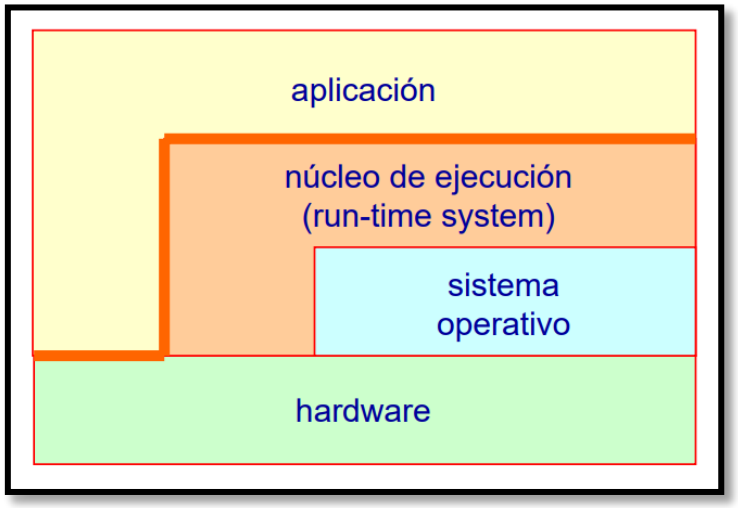


Ilustración 13. Entorno de ejecución de un sistema de tiempo real

Entorno de ejecución Ada

Ada es un lenguaje de programación que concibe la misma en base a una serie de instrucciones y condiciones que modifican el estado del programa para llegar a un objetivo final (lenguaje imperativo). Ada proviene de Pascal y se caracteriza por una programación estructurada en bloques y orientada a objetos y por ser un lenguaje fuertemente tipado y con repetidas comprobaciones en tiempos de ejecución. A continuación se detallan sus características principales:

- Tipado fuerte: Se trata de un lenguaje de programación con tipos estáticos y seguros que permiten al programador elegir el más adecuado para su objetivo y generar abstracciones que reflejan el mundo real de manera sencilla. Además esta característica permite detectar errores potenciales en el momento de la compilación.

- Reusabilidad: La extensión de tipo que permite el lenguaje se fácilmente reutilizable.

- Ocultación de la información: El lenguaje establece una clara separación entre interfaces e implementación.

- Portabilidad: El código fuente programado con ADA tiene una gran portabilidad entre las diferentes plataformas hardware existentes.

- Soporte de concurrencia y tiempo real: Es esencial para aplicaciones de tiempo real embebidas. La concurrencia y el tiempo real están directamente incluidos en el lenguaje.

- Soporte para el desarrollo software y estandarización: Ada contiene un amplio conjunto de normas (ISO) que son muy útiles para aplicar técnicas de validación y verificación del software. Las normas limitan el uso de ciertas características lingüísticas que pueden aportar indeterminismo. Existen cuatro versiones normalizadas de Ada:

    - Ada 83 (ISO 8652:1987).
    - Ada 95 (ISO 8652:1995)
    - Ada 2005 (ISO 8652:1995/Amd 1:2007)
    - Ada 2012 (ISO 8652:2012(E))

- Programación de bajo nivel: Se trata de un lenguaje que interacciona directamente con el hardware (programación a bajo nivel) y contiene sus propias interrupciones.

Ada es un lenguaje de programación específico para diseñar sistemas fiables de tiempo real embebidos y generalmente de grandes dimensiones. Los errores de ejecución son reconocidos como excepciones y se tratan de manera explícita.

Los programas de Ada se estructuran en una o varias unidades de programa, que pueden ser:

- Subprogramas: Definen algoritmos ejecutables.

- Paquetes: Definen conjuntos de entidades.
- Unidades de tarea: Definen cálculos concurrentes.
- Unidades protegidas: Definen operaciones para el intercambio coordinado de datos entre las diferentes tareas.
- Unidades genéricas: Definen paquetes y subprogramas de manera parametrizada.

Cada unidad de programa suele estar formada por dos partes diferenciadas:

- Especificación: Contiene la información que necesita ser visible para otras unidades diferentes.
- Cuerpo: Formado por la implementación en sí, que no debe ser visible para otras unidades.

Las unidades de programa pueden ser compiladas por separado, lo que permite junto a las características especificadas anteriormente diseñar, implementar y probar un programa como un conjunto de componentes software independientes.

Existe una biblioteca con paquetes ya predefinidos pero además, cualquiera puede implementar las suyas propias. La versión actual de Ada es la 2012 y los principales cambios y mejoras que aportó con respecto a las versiones anteriores son:

- Introducción de declaraciones dinámicas: Lo que permite fijar ciertas propiedades de las entidades en su declaración.
- Expresiones más flexibles: La introducción de precondiciones aumenta la necesidad de expresiones más poderosas. Se añaden de esta manera se añaden expresiones *"if"*, *"case"* y cuantificadas.
- Control de estructura y visibilidad: Se permite que las funciones tengan parámetros de entrada y salida, y una mayor flexibilidad con tipos incompletos.
- Mayor soporte para los sistemas de tiempo real: Se añaden nuevos paquetes para el control de tareas en sistemas multiprocesador y nuevas facilidades para las expulsiones no preventivas.
- Mejoras en la biblioteca estándar: Nuevos paquetes que incluyen mecanismos para el manejo de árboles y colas [17] [18] [19].

Ilustración 14. Logotipo de Ada

Entorno de ejecución Java

Java es un lenguaje de programación orientado a objetos cuyo objetivo de diseño es eliminar las dependencias de implementación con respecto a la máquina en la que se ejecute el programa.

Las características principales del lenguaje se detallan a continuación:

- Simplicidad: Su sintaxis es similar a la del lenguaje C y C++, pero evita semánticas que hacen el lenguaje complejo y confuso. Un aspecto destacable es el recolector de basura incorporado, que se encarga de gestionar la memoria a la hora de crear objetos y de eliminarlos.

- Programación orientada a objetos: La ventaja de este tipo de programación son: un mejor tratamiento de la complejidad de los problemas dividiéndola en subproblemas más sencillos, una capacidad de reutilización sencilla y una mayor facilidad para la corrección y la evolución. Además Java contiene un conjunto de clases que permiten gestionar cualquier objeto.

- Programación distribuida: Java ofrece la posibilidad de desarrollar aplicaciones con el modelo cliente/servidor en arquitectura distribuida puesto que implementa los protocolos de red necesarios.

- Independencia de las arquitecturas y portabilidad: El código, una vez implementado y compilado, puede ejecutar en cualquier otra máquina, es decir, no hace falta una compilación específica para cada procesador. Este comportamiento es posible gracias a que el programa se ejecuta sobre una

máquina virtual (Java Virtual Machine) en el sistema anfitrión, que es la encargada de reconocer e interpretar los ficheros de código compilados y adaptarlos al procesador particular utilizado. De este modo se logra la independencia de la máquina.

- Robustez: Se trata de un lenguaje fuertemente tipado. Se verifica el código tanto en el momento de la compilación como en el de ejecución, detectando de esta manera los errores prematuramente. La gestión automática de punteros es una gran ayuda para el programador y es también una forma de reducir errores (escrituras accidentales en determinadas zonas de memoria).

- Lenguaje securizado: El motor de ejecución de Java es el encargado de velar por la seguridad de las aplicaciones y sistemas. También se encarga de repartir los recursos (memoria) entre los diferentes objetos.

- Eficaz, multitarea y dinámico: La interpretación del código la realiza un proceso (Just In Time) en tiempo de ejecución, algo que permite el mismo rendimiento que un programa escrito en otro lenguaje como C. Además Java soporta la concurrencia y permite la programación de aplicaciones que implementen varios hilos de ejecución. El lenguaje es dinámico porque no es necesario editar manualmente los vínculos al modificar una o varias clases, ya que la comprobación de las clases se realiza también en tiempo de ejecución.

Como ya se ha expuesto la plataforma Java se distingue del resto en cuanto que es sólo software, capaz de ejecutarse en diferentes arquitecturas y sistemas operativos. Se compone de las siguientes partes:

- La máquina virtual Java (JVM).
- La interfaz de programación de aplicación Java (API Java).
- Herramientas destinadas al despliegue de las aplicaciones implementadas.
- Herramientas para la ayuda al desarrollo.

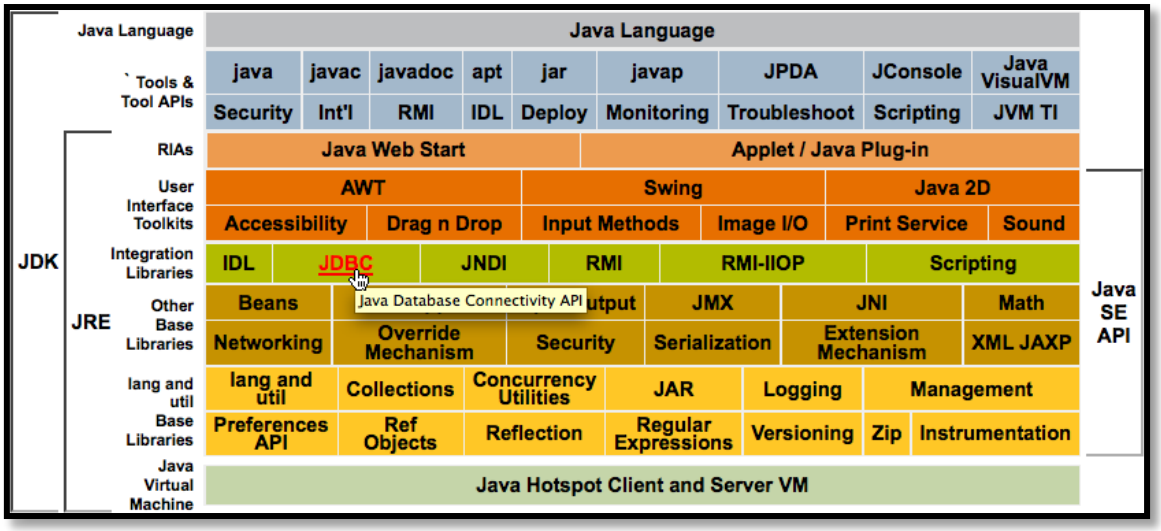


Ilustración 15. Plataforma Estándar Java

En Java existen diferentes plataformas:

- Java Standard Edition (JSE): Es un conjunto de APIs que contienen la funcionalidad básica de Java.
- Java Enterprise Edition (JEE): Sus APIs incluyen funcionalida extra sobre JSE, permitiendo desarrollar aplicaciones distribuidas.
- Java Micro Edition (JME): Se trata de una plataforma que ofrece un entorno flexible para aplicaciones ejecutadas en dispositivos móviles e integrados (microcontroladores, teléfonos móviles, PDAs…). Esta plataforma introduce nuevas características para soportar las implementaciones de tiempo real. Real Time Specification for Java es un conjunto de interfaces que proporcionan dichas características y que se basa en una máquina virtual extendida para los sistemas embebidos de tiempo real. Entre sus características destacan:
    - Soporte de programación para aplicaciones en tiempo real que contengan tanto tareas periódicas como dinámicas.
    - Soporte para plazos de tareas y los tiempos de reloj.
    - Herramientas para prevenir retrasos en la recolección de basura [20] [21] [22].

Ilustración 16. Logotipo de la plataforma Java Micro Edition 2

Entorno de ejecución POSIX

Posix (Portable Operating System Interface) es un estándar que define la interfaz estándar que debe tener un sistema operativo y su entorno para soportar la portabilidad de las aplicaciones a nivel de código fuente (código programado con el lenguaje de programación C). Fue creado por la Computer Society del Institute of Electrical and Electronics Engineers, la IEEE y está basado en UNIX. De esta manera el estándar define:

- La interfaz del sistema operativo: Conjunto de funciones y tipos que se encuentran agrupadas en ficheros de cabeceras.
- Un intérprete de comandos.
- Una serie de programas útiles para el desarrollador.

Los estándares POSIX se pueden agrupar en diferentes categorías, todas ellas recogidas en la norma IEEE 1003:

- Estándares base: Este tipo de estándares, como su propio nombre indica definen las interfaces del sistema relacionadas con los aspectos más básicos del sistema operativo. Al especificar tanto la sintaxis como la semántica a utilizar, los programas pueden invocar los servicios del sistema operativo directamente. Algunos de os estándares base de POSIX son:
  - POSIX.1: Interfaces del sistema.
  - POSIX.2: Shell y diferentes utilidades.
  - POSIX.4 a y b: Extensión de threads o varios flujos de control (a) y extensiones adicionales de tiempo real (b).
  - POSIX.6: Extensiones de seguridad.

    - POSIX.7: Administración del sistema.
    - POSIX.12: Interfaces de red independientes del protocolo utilizado.
    - POSIX.17: Servicios de directorios.
- Interfaces en diferentes lenguajes: Traducen a un lenguaje de programación concreto los estándares base:
    - POSIX.5: Interfaces Ada.
    - POSIX.19: Interfaces Fortran 90.
    - POSIX.20: Interfaces Ada específicas de las extensiones de tiempo real.
- Entornos de sistemas abiertos: Incluyen una guía del entorno POSIX y los perfiles de entornos de aplicación. Un perfil de aplicación es un conjunto de servicios, que especifican las opciones y parámetros para un entorno de programación determinado. Especifican un conjunto bien definido de implementaciones del sistema operativo para ámbitos de aplicación específicos.
    - POSIX.0: Guía al entorno POSIX de sistemas abiertos.
    - POSIX.10: Perfil de entorno de aplicaciones de supercomputación.
    - POSIX.13: Perfiles de entornos de aplicaciones de tiempo real.
    - POSIX.14: Perfil de entorno de aplicaciones multiprocesador.
    - POSIX.18: Perfil de entorno de aplicación de plataforma POSIX.

La estandarización de sistemas operativos embebidos y de tiempo real añaden al estándar POSIX básico los servicios necesarios para desarrollar aplicaciones de tiempo real. En este tipo de aplicaciones es obligado garantizar que el sistema tenga un comportamiento temporal que se pueda predecir. Por lo tanto es necesario que también los servicios del sistema operativo tengan un tiempo de respuesta definido. Los principales estándares POSIX destinados al tiempo real son:

- POSIX.4: Extensiones de tiempo real. Define las interfaces necesarias para soportar la portabilidad de aplicaciones de tiempo real.
- POSIX.4a: Extensión de threads: Definen interfaces destinadas a soportar varios flujos de ejecución dentro de cada proceso POSIX.
- POSIX.4b: Extensiones adicionales de tiempo real. Interfaces para soportar servicios adicionales de tiempo real.

- POSIX.13: Perfiles de entornos de aplicaciones de tiempo real: Listas de servicios para entornos de aplicación particulares.
    - PSE50: Forma un sistema de tiempo real mínimo, sin gestión de memoria ni ficheros ni terminal. Además solo utiliza hilos de ejecución.
    - PSE51: Controlador de tiempo real con sistema de ficheros y terminal.
    - PSE52: Sistema de tiempo real dedicado, con gestión de memoria y procesos pesados.
    - PSE53: Forma un sistema de tiempo real generalizado, un sistema completo con una gran diversidad de servicios.

Resumiendo, los principales criterios que cumple el estándar POSIX son:

- Entradas y salidas asíncronas: Para proporcionar entradas y salidas a nivel de usuario, el sistema de tiempo real debe soportar la entrega de las interrupciones desde los dispositivos de entrada y salida a un proceso de manera eficiente.
- Bloqueo de memoria: La capacidad de garantizar el almacenamiento de partes de un proceso que no ha sido referenciado recientemente en memoria secundaria.
- Semáforos: Sincronizar variables globales a las que acceden diferentes procesos de la aplicación.
- Memoria compartida: Asignar memoria compartida en espacios virtuales específicos.
- Planificación de ejecución: Capacidad para planificar múltiples tareas.
- Timers: Aumentan la funcionalidad y el determinismo del sistema.
- Comunicación entre procesos: Los métodos de comunicación más comunes que utiliza un RTOS son colas de mensajes.
- Archivos e hilos de ejecución de tiempo real: Capacidad para crear y acceder a archivos que tienen una naturaleza determinista. [23] [24] [25]

2.2.4 Sistemas Operativos de Tiempo Real

Un Sistema Operativo de Tiempo Real o en inglés Real Time Operating System (RTOS) es un tipo específico de sistema operativo que ofrece una serie de cualidades y

características específicas para dar soporte a aplicaciones de tiempo real. Las características que debe garantizar un sistema de tiempo real son las siguientes:

- Gestión del tiempo: Para el diseño de aplicaciones de tiempo real es esencial un sistema operativo que gestione el tiempo. Debe controlar un reloj monótono para permitir el control de los plazos temporales de las tareas. Los relojes deben ser de alta resolución y asociados a cada tarea para gestionar el tiempo de ejecución de las mismas.

- Multiprogramación: El sistema operativo tiene que dar soporte a diferentes tareas. Las aplicaciones de tiempo real son multitarea y necesitan de un núcleo que permita gestionarlas en base a sus plazos. Además, un RTOS debe manejar tanto múltiples niveles de interrupciones como de prioridades.

- Servicios predecibles: El coste de los servicios debe ser completamente predecible y su coste conocido. Para la planificación de una aplicación de tiempo real es necesario conocer el coste para despreciarlo o no dependiendo del coste de las tareas de aplicación.

- Garantía de ejecución: Para asegurar la correcta ejecución de las tareas, el sistema operativo de tiempo real tiene que seleccionar el acceso a los recursos (procesador) de las mismas en base a unos aspectos como la prioridad. Como se expuso anteriormente, la prioridad de una tarea puede ser determinada a priori por el usuario (prioridad fija) o ser calculada en tiempo de ejecución por el propio sistema (prioridad dinámica).

- Comunicación y sincronización de tareas: Es necesario que las tareas realicen estas funciones entre sí, y para ello, el sistema operativo debe proporcionar mecanismos y los protocolos adecuados (basados en herencia de prioridad) para sincronizar y comunicar segura y eficientemente las tareas correspondientes a la aplicación. Esquemas de memoria compartida y paso de mensajes son las técnicas utilizadas por el sistema operativo.

- Gestión de memoria: Es esencial que el espacio de direcciones que utiliza el kernel, se encuentre separado del que utiliza la aplicación. Si no fuera de esta manera y compartieran el mismo espacio de direcciones, podrían surgir problemas que afectaran directamente al sistema operativo. Además el núcleo debe proporcionar mecanismos a las aplicaciones para gestionar dinámicamente la memoria.

- Interfaz de programación (API): Como se ha explicado en los apartados anteriores, la interfaz de programación determina casi totalmente la portabilidad de las aplicaciones.
- Niveles de prioridad suficientes para la implementación: Cuando se trabaja con planificación de tareas con prioridad, el sistema operativo debe tener un número suficiente de niveles de prioridad.

En un RTOS son significativamente importantes las medidas de ciertas operaciones de bajo nivel que el sistema realiza frecuentemente, y que requieren un coste temporal reducido. Algunas medidas críticas son:

- Cambio de contexto acotado.
- Gestión eficiente del reloj y de los temporizadores.
- Tiempo de respuesta del hardware.
- Latencia de la gestión de interrupciones garantizada.
- Baja sobrecarga introducida por el planificador.

Para garantizar un producto final en tiempo real que cumpla todas las restricciones temporales, no basta solo con seleccionar un sistema operativo adecuado, sino que es necesario una correcta implementación software y un análisis sencillo.

Para implementar una aplicación de tiempo real no es solo necesario seleccionar un RTOS en base a sus características temporales, sino que además hay que considerar otros aspectos como los siguientes:

- Tipo de interfaz de llamadas al sistema (API) dependiendo del estándar que proporcionan la portabilidad de las aplicaciones.
- Lenguajes soportados.
- Certificación.
- Disponibilidad de un gran conjunto de drivers de dispositivos.
- Middleware para la comunicación y sincronización de aplicaciones.
- Otros aspectos como sistemas de ficheros o pilas de comunicaciones.

Por supuesto existen un gran número de kernels o sistemas operativos de tiempo real que cumplen los requisitos explicados. Entre ellos, se pueden definir dos tipos: los

comerciales (VxWorks, Windows CE) y los de código abierto (RTAI, Xenomai). A continuación, se detallarán en profundidad algunos de ellos [15].

2.2.5 Windows CE

Windows CE (conocido oficialmente como Windows Embedded Compact y anteriormente como Windows Embedded CE) se trata de un sistema operativo de tiempo real diseñado por *Microsoft*, modular y portable para dispositivos móviles, multimedia y consolas de vídeo o televisión con memoria reducida. Soporta la concurrencia y reparte el tiempo de CPU entre los diferentes threads, que ejecutan en modo kernel para optimizar el rendimiento. Se puede ejecutar en múltiples arquitecturas de procesador diferentes (ARM, MIPS x86, SH4, etc). Su interfaz de programación la componen todos los lenguajes incluidos en el framework .NET. La versión actual es Windows Embedded 8. [25][26] [27]

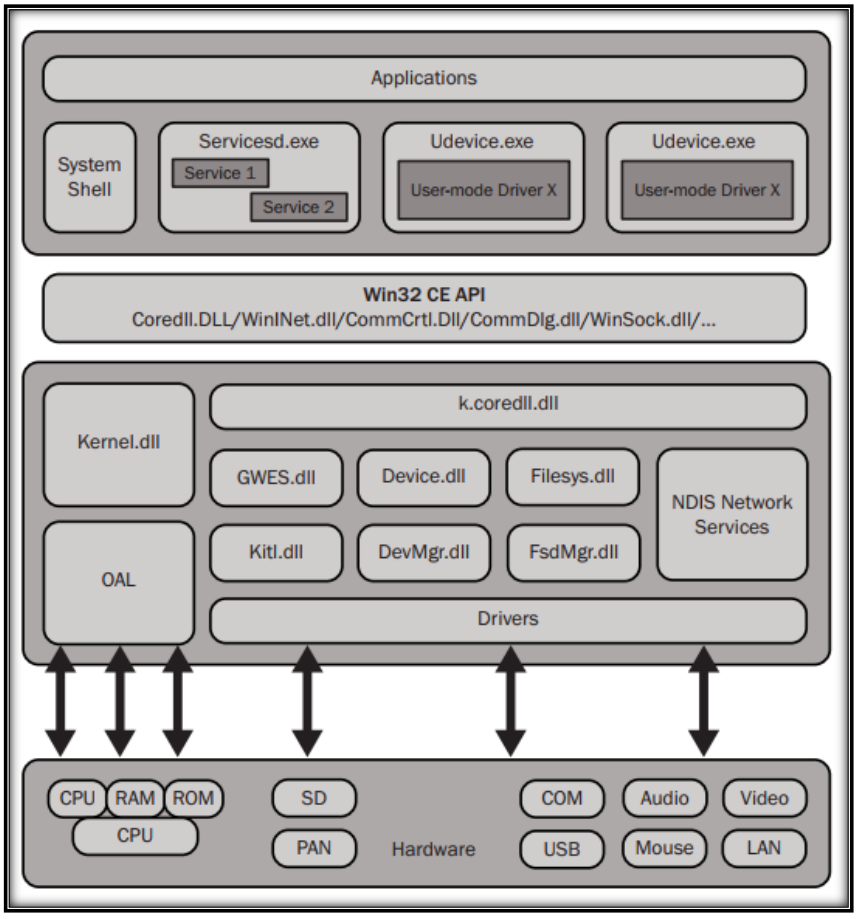


Ilustración 17. Arquitectura del sistema Windows CE

2.2.6 Real Time Linux (RT-Linux)

Es un sistema operativo de tiempo real basado, como su propio nombre indica, en Linux. Se trata de una modificación de su kernel para sistemas de tiempo real. En su interior, está formado por un minikernel en el cual Linux ejecuta como un thread de menor prioridad que las tareas de tiempo real. De esta manera, dichas tareas nunca se verán retrasadas por las operaciones del sistema operativo que no forman parte del tiempo real. Se puede ejecutar en las arquitecturas de procesador x86 y Power PC.

Cabe destacar que RTLinux no es código independiente, por lo tanto no es una nueva versión de Linux. Parte de su distribución es un parche sobre el propio código de Linux, y otra parte son módulos.

Actualmente está distribuido por *Wind River Systems* bajo el nombre Wind Real-Time Core para la versión Wind River Linux, cuyas características y mejoras principales son:

- Garantiza una serie de capacidades en tiempo real combinadas con un desarrollo sencillo y herramientas integradas, además de un rápido despliegue de Wind River Linux.
- Garantiza el comportamiento determinista de la aplicación de tiempo real, con tiempos de respuesta del orden de microsegundos y con un amplio abanico de arquitecturas.
- Las interfaces POSIX que implementa permiten que las aplicaciones de tiempo real sean casi indistinguibles de las aplicaciones UNIX estándar, algo que permite acelerar el tiempo de desarrollo.
- El espacio de usuario en tiempo real mejora debido a una capa de protección de la memoria de los subprocesos y, mejora también, el soporte del lenguaje de programación. De esta manera, ningún proceso ralentizará el sistema.
- Añade soporte de red en tiempo real para aquellas aplicaciones red que necesitan características deterministas. El módulo que añade esta funcionalidad permite que las aplicaciones envíen y reciban paquetes de red de una manera completamente determinista. [28] [29]

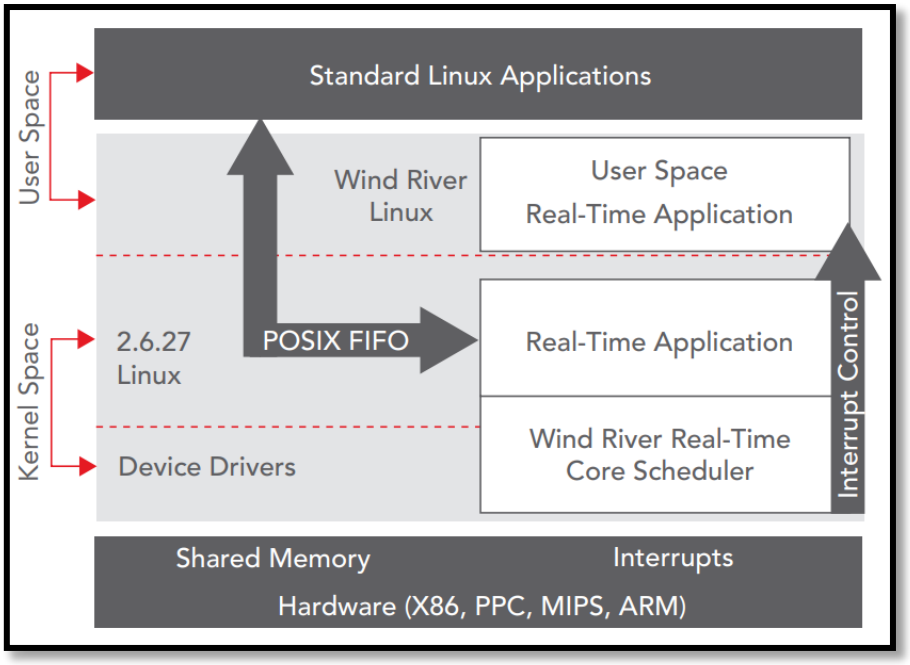


Ilustración 18. Arquitectura del sistema de Wind River Real Time Core

2.2.7 VxWorks

VxWorks es un sistema de tiempo real basado en UNIX, fabricado y distribuido por *Wind River Systems*. Incluye un minikernel multitarea como sucede con el sistema operativo anterior. Es uno de los sistemas de tiempo real más extendidos actualmente, destinado en múltiples aplicaciones robóticas, aeroespaciales, militares. Ejecuta en múltiples arquitecturas de procesadores como ARM, x86 y Power Architecture.

Las principales características de su versión más reciente son:

- Arquitectura modular: Gracias a la cual puede adaptarse a nuevos avances tecnológicos y evolucionar. El kernel de VxWorks está separado de los protocolos y aplicaciones, permitiendo que las actualizaciones e incorporación de nuevas características se realices de forma más rápida y con un testeo mínimo del sistema.
- Escalabilidad: VxWorks implementa desde aplicaciones sencillas hasta grandes sistemas inteligentes.
- Amplia conectividad: Debido a que la conectividad es un aspecto muy importante en las aplicaciones del Internet de las Cosas (IoT), VxWorks proporciona soporta los principales protocolos de red como Routing Information Protocol (RIP) y Quality of Service (QoS).

- Seguridad: VxWorks ofrece un conjunto completo de características para proteger de forma eficiente y eficaz los dispositivos, los datos y la propiedad intelectual.
- Soporte para el runtime de Java: Además de implementar aplicaciones en C/C++, proporciona kit de desarrollo software Java (SDK) para desarrollar aplicaciones Java específicas en dispositivos empotrados. Además ofrece un entorno de desarrollo cruzado basado en Eclipse. [28] [30]

2.2.8 RTEMS

RTEMS (Real-Time Executive for Multiprocessor Systems) es un sistema operativo de tiempo real para sistemas multiprocesador y de código abierto que permite tanto interfaces de programación estándar abiertas como POSIX. Se puede ejecutar en diferentes arquitecturas de procesador como x86, ARM, PowerPC, Blackfin o MIPS. RTEMS proporciona la mayoría de servicios POSIX relacionados con la asignación de memoria y la creación de procesos. Las aplicaciones implementadas con RTEMS son de distinto ámbito, desde las relacionadas con la producción industrial hasta militares y aeroespaciales utilizadas por la NASA (National Aeronautics and Space Administration). RTEMS es distribuido bajo una licencia GNU (General Public License).

Sus principales características son:

- Capacidad multitarea.
- Programación de tareas basada en prioridades y en eventos.
- Comunicación y sincronización entre tareas.
- Gestión de interrupciones.
- Gestión de memoria dinámica.
- Alto nivel de configuración de usuario. [31]

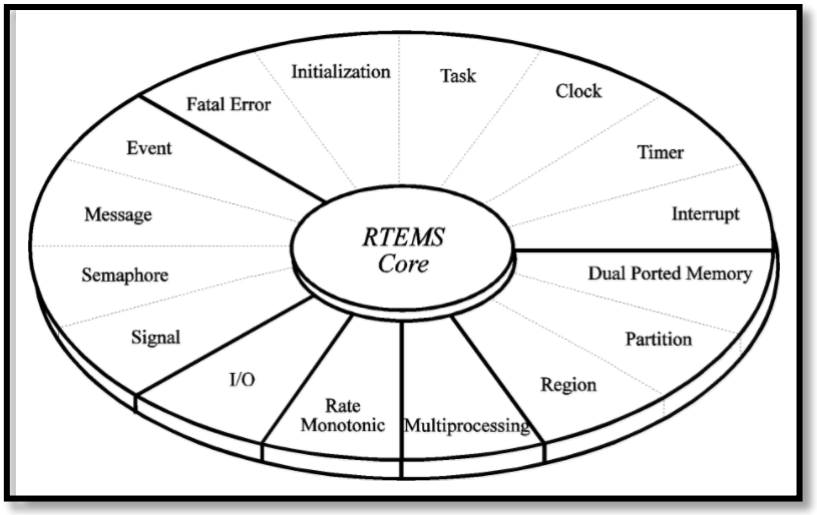


Ilustración 19. Arquitectura en capas de RTEMS

Capítulo 3: Análisis

Se definen a continuación, los requisitos tanto de usuario como de sistema, que se corresponden con las necesidades del proyecto. Para asegurar su debido cumplimiento se hace uso de matrices de trazabilidad.

3.1 Requisitos de usuario

En este apartado se especifican los requisitos de usuario, en los cuales se especificarán las funcionalidades que debe incorporar el proyecto ajustadas a las necesidades del usuario cliente.

Los requisitos de usuario quedan divididos en requisitos de usuario de capacidad y requisitos de usuario de restricción que, por comodidad, son representados en tablas cuyo formato es el especificado a continuación:

- Identificador: Código unívoco que identifica al requisito cuya nomenclatura viene dada por el formato **XX_RU_YY**:
  - **XX** será sustituido por ***'CA'*** si se trata de un requisito de capacidad o por ***'RE'*** si se trata de un requisito de restricción.
  - **RU** se corresponde con las siglas de Requisitos de Usuario.
  - **YY** será sustituido por el número del requisito que corresponda dentro de su categoría empezando en **01** y aumentándose en una unidad por requisito.
- Descripción: Se compone de la especificación del requisito de manera detallada. Es necesario que se realice de forma simple y precisa para evitar ambigüedades.
- Necesidad: Parámetro que establece la importancia de incluir el requisito dentro del proyecto. Los valores que puede adoptar son ***'Esencial'*** y ***'Deseable'***.
- Prioridad: Parámetro que establece la importancia de implementar el requisito en el proyecto. Puede tomar los valores ***'Baja'***, ***'Media'*** y ***'Alta'***.
- Estabilidad: Indica el grado de variación que puede tener el requisito a lo largo del ciclo de vida del proyecto. Puede adoptar los valores ***'Estable'*** o ***'Inestable'***.
- Verificabilidad: Establece la dificultad que existe a la hora de verificar el cumplimiento del requisito. Sus valores pueden ser ***'Alta'***, ***'Media'*** o ***'Baja'***.

3.1.1 Requisitos de capacidad

Los requisitos de capacidad indican las funcionalidades o características que debe incorporar la solución ofrecida para cumplir las necesidades que se desean cubrir.

Identificador: CA_RU_01	
Descripción	Seleccionar el sistema operativo de tiempo real para el desarrollo de aplicaciones de tiempo real más adecuado.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 20. Requisito de usuario CA_RU_01

Identificador: CA_RU_02	
Descripción	Seleccionar la plataforma hardware para el desarrollo de aplicaciones de tiempo real más adecuada.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 21. Requisito de usuario CA_RU_02

Identificador: CA_RU_03	
Descripción	Seleccionar la plataforma software para el desarrollo de aplicaciones de tiempo real más adecuada.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 22. Requisito de usuario CA_RU_03

Identificador: CA_RU_04	
Descripción	Seleccionar el entorno de desarrollo más adecuado en el que se integre la programación con el sistema operativo de tiempo real para facilitar el desarrollo de aplicaciones.
Necesidad	Deseable
Prioridad	Media
Estabilidad	Estable
Verificabilidad	Alta

Tabla 23. Requisito de usuario CA_RU_04

Identificador: CA_RU_05	
Descripción	Seleccionar el emulador para la ejecución las aplicaciones de tiempo real más adecuado.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 24. Requisito de usuario CA_RU_05

Identificador: CA_RU_06	
Descripción	El entorno de desarrollo, la plataforma hardware/software y el emulador deben estar integrados en un mismo framework de desarrollo dentro de una máquina virtual.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 25. Requisito de usuario CA_RU_06

Identificador: CA_RU_07	
Descripción	Diseñar una aplicación de tiempo real para el control de un sistema específico.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 26. Requisito de usuario CA_RU_07

Identificador: CA_RU_08	
Descripción	La aplicación debe constar de dos módulos o sistemas de tiempo real diferenciados: un módulo correspondiente a un servidor de control y otro correspondiente a un sistema electrónico de control empotrado en un sistema concreto.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 27. Requisito de usuario CA_RU_08

Identificador: CA_RU_09	
Descripción	El módulo correspondiente al servidor de control debe ser implementado con el sistema operativo de tiempo real y el API de POSIX.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 28. Requisito de usuario CA_RU_09

Identificador: CA_RU_10	
Descripción	El módulo correspondiente al sistema electrónico de control debe ser implementado con la plataforma software/hardware.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 29. Requisito de usuario CA_RU_10

Identificador: CA_RU_11	
Descripción	La comunicación de ambos módulos se realiza mediante mensajes de petición y respuesta a través de un puerto serie.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 30. Requisito de usuario CA_RU_11

Identificador: CA_RU_12	
Descripción	El módulo correspondiente al sistema electrónico de control debe comunicarse una serie de sensores y actuadores.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 31. Requisito de usuario CA_RU_12

Identificador: CA_RU_13	
Descripción	La aplicación debe implementar en ambos módulos un planificador cíclico de tareas simples.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 32. Requisito de usuario CA_RU_13

Identificador: CA_RU_14	
Descripción	La creación de la imagen de la aplicación de tiempo real debe realizarse mediante el uso de un script.
Necesidad	Deseable
Prioridad	Media
Estabilidad	Estable
Verificabilidad	Alta

Tabla 33. Requisito de usuario CA_RU_14

Identificador: CA_RU_15	
Descripción	Proporcionar la documentación necesaria para la integración del framework y la implementación de la aplicación.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 34. Requisito de usuario CA_RU_15

3.1.2 Requisitos de restricción

Los requisitos de restricción indican cómo se deben realizar las funcionalidades establecidas en los requisitos de capacidad y cómo no. Se trata de los requisitos que amplían y detallan más en profundidad la información proporcionada por los de capacidad y que a su vez, definen los límites del proyecto.

Identificador: RE_RU_01	
Descripción	El sistema operativo de tiempo real tiene que ser open source y gratuito para la arquitectura x86.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 35. Requisito de usuario RE_RU_01

Identificador: RE_RU_02	
Descripción	La plataforma hardware para el desarrollo de aplicaciones de tiempo real tiene que ser open hardware.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 36. Requisito de usuario RE_RU_02

Identificador: RE_RU_03	
Descripción	La plataforma software para el desarrollo de aplicaciones de tiempo real tiene que ser open source y gratuita.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 37. Requisito de usuario RE_RU_03

Identificador: RE_RU_04	
Descripción	El entorno de desarrollo en el que se integre la programación con el sistema operativo de tiempo real debe ser gratuito.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 38. Requisito de usuario RE_RU_04

Identificador: RE_RU_05	
Descripción	El emulador debe ser open source y gratuito.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta

Tabla 39. Requisito de usuario RE_RU_05

Identificador: RE_RU_06	
Descripción	La estructura de la implementación de la aplicación se especifica en un código fuente proporcionado.
Necesidad	Deseable
Prioridad	Media
Estabilidad	Estable
Verificabilidad	Alta

Tabla 40. Requisito de usuario RE_RU_06

3.2 Casos de uso

Los casos de uso representan gráficamente los requisitos mediante la representación de las acciones o funcionalidades que el usuario final puede realizar con la solución del proyecto.

En el caso del presente proyecto, y como se ha expuesto al principio de este capítulo, la solución proporcionada tiene dos destinatarios claramente diferenciables:

- Tutor: Haciendo uso de la solución del proyecto, debe ser capaz de integrar el framework de desarrollo, proponer a los alumnos la implementación de la aplicación de tiempo real y evaluar la misma en base a los criterios descritos.
- Alumno: Partiendo del framework de desarrollo y el enunciado de la aplicación proporcionado por el profesor, debe ser capaz de implementar la misma.

Al igual que los requisitos y por comodidad cada caso de uso se representa con una tabla que dispone del siguiente formato:

- Identificador: Código unívoco que identifica al caso de uso cuya nomenclatura viene dada por el formato **CU_X_YY**:
    - **X** será sustituido por ***'T'*** si se trata de un caso de uso del tutor o por ***'A'*** si se trata de un caso de uso del alumno.
    - **CU** se corresponde con las siglas de Caso de Uso.
    - **YY** será sustituido por el número del requisito que corresponda dentro de su categoría empezando en ***01*** y aumentándose en una unidad por requisito.
- Nombre: Identificación nominal en base a la naturaleza del caso de uso.
- Usuario: Adoptará el valor ***'Tutor'*** o ***'Alumno'*** dependiendo del tipo de usuario que realice el caso.
- Objetivo: Meta que persigue el usuario con la realización del caso de uso.
- Condiciones previas: Conjunto de requisitos previos a la ejecución del caso de uso.
- Acciones: Secuencia de pasos llevados por el usuario en el caso de uso.
- Condiciones posteriores: Hecho que acontece si la ejecución del caso de uso es satisfactoria.

Con los dos usuarios bien diferenciados se expone el diagrama de casos de uso y se especifican los casos para cada uno de ellos en particular. Los casos de uso de consulta no se encuentran detallados, puesto que no suponen ninguna dificultad.

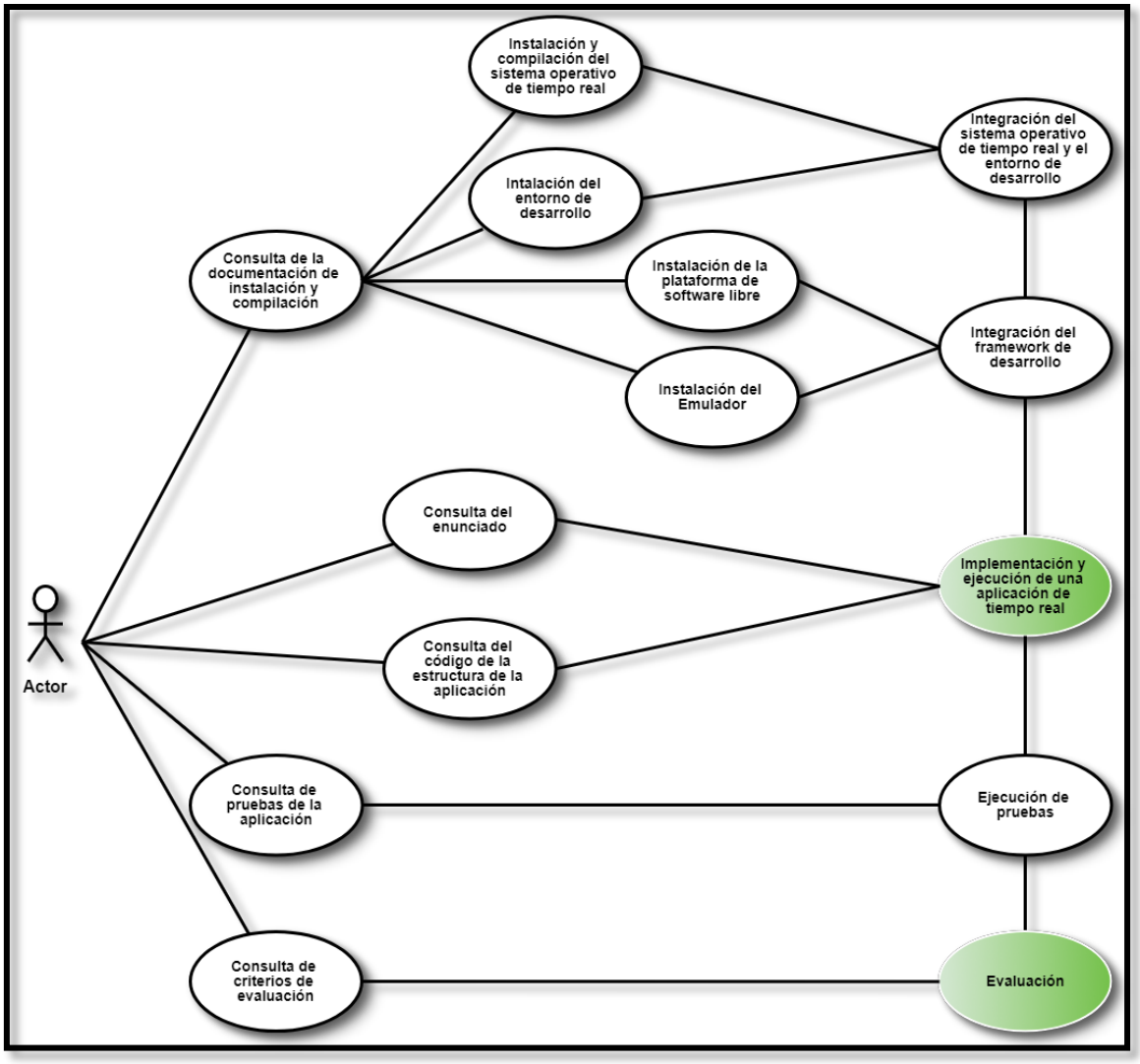


Ilustración 20. Diagrama de casos de uso

Identificador: CU_T_01	
Nombre	Instalación del sistema operativo de tiempo real.
Usuario	Tutor
Objetivo	Instalación del sistema operativo de tiempo real adecuado para el desarrollo de aplicaciones de tiempo real.
Condiciones previas	1. Debe existir conexión a Internet. 2. Se debe disponer de memoria suficiente para realizar la instalación. 3. Documentación previa.
Acciones	1. Descarga del código fuente desde el repositorio. 2. Instalación y compilación del sistema operativo.
Condiciones posteriores	El sistema operativo se encuentra instalado y compilado.

Tabla 41. Caso de uso CU_T_01

Identificador: CU_T_02	
Nombre	Instalación del entorno de desarrollo.
Usuario	Tutor
Objetivo	Instalación del entorno de desarrollo adecuado para la integración con el sistema operativo de tiempo real.
Condiciones previas	1. Debe existir conexión a Internet. 2. Se debe disponer de memoria suficiente para realizar la instalación. 3. Documentación previa.
Acciones	1. Descarga del código fuente desde el repositorio. 2. Instalación del entorno de desarrollo.
Condiciones posteriores	El entorno de desarrollo se encuentra instalado.

Tabla 42. Caso de uso CU_T_02

Identificador: CU_T_03	
Nombre	Instalación de la plataforma de software libre.
Usuario	Tutor
Objetivo	Instalación plataforma de software libre adecuada para el desarrollo de aplicaciones de tiempo real.
Condiciones previas	1. Debe existir conexión a Internet. 2. Se debe disponer de memoria suficiente para realizar la instalación. 3. Documentación previa.
Acciones	1. Descarga del código fuente desde el repositorio. 2. Instalación de la plataforma de software libre.
Condiciones posteriores	La plataforma de software libre se encuentra instalada.

Tabla 43. Caso de uso CU_T_03

Identificador: CU_T_04	
Nombre	Instalación del emulador.
Usuario	Tutor
Objetivo	Instalación del emulador adecuado para simular una aplicación de tiempo real.
Condiciones previas	1. Debe existir conexión a Internet. 2. Se debe disponer de memoria suficiente para realizar la instalación. 3. Documentación previa.
Acciones	1. Descarga del código fuente desde el repositorio. 2. Instalación del emulador.
Condiciones posteriores	El emulador se encuentra instalado.

Tabla 44. Caso de uso CU_T_04

Identificador: CU_T_05	
Nombre	Integración del sistema de tiempo real y el entorno de desarrollo.
Usuario	Tutor
Objetivo	Integración para disponer de un entorno de desarrollo de aplicaciones de tiempo real con un determinado sistema operativo.
Condiciones previas	1. El sistema operativo debe estar instalado y compilado. 2. El entorno de desarrollo debe estar instalado. 3. Documentación previa.
Acciones	1. Incluir las características del sistema operativo en el entorno de desarrollo.
Condiciones posteriores	La integración se hace efectiva.

Tabla 45. Caso de uso CU_T_05

Identificador: CU_T_06	
Nombre	Integración del framework de desarrollo.
Usuario	Tutor
Objetivo	Integración para disponer de un framework entero y adecuado para el desarrollo de aplicaciones de tiempo real.
Condiciones previas	1. La integración del sistema operativo de tiempo real y el entorno de desarrollo debe ser efectiva. 2. La plataforma de software libre debe estar instalada. 3. El emulador debe estar instalado.
Acciones	1. Realizar la integración.
Condiciones posteriores	La integración se hace efectiva.

Tabla 46. Caso de uso CU_T_06

Identificador: CU_T_07	
Nombre	Ejecución de pruebas.
Usuario	Tutor
Objetivo	Ejecutar las pruebas en la aplicación implementada por los alumnos.
Condiciones previas	1. La aplicación debe estar implementada en su totalidad. 2. Documentación sobre las pruebas de evaluación.
Acciones	1. Ejecutar las pruebas sobre la aplicación.
Condiciones posteriores	Las funcionalidades que contiene la aplicación son conocidas por el tutor.

Tabla 47. Caso de uso CU_T_07

Identificador: CU_T_08	
Nombre	Evaluación de la aplicación.
Usuario	Tutor
Objetivo	Evaluación de la aplicación implementada por los alumnos.
Condiciones previas	1. Las pruebas deben de haberse ejecutado. 2. Documentación sobre los criterios de evaluación.
Acciones	1. Evaluación en función de los criterios especificados.
Condiciones posteriores	La evaluación del alumno se hace efectiva.

Tabla 48. Caso de uso CU_T_08

Identificador: CU_A_09	
Nombre	Implementación de una aplicación.
Usuario	Alumno
Objetivo	Desarrollo de una aplicación de tiempo real.
Condiciones previas	1. Se debe proporcionar el framework de desarrollo al alumno. 2. Consulta de la estructura de la aplicación. 3. Lectura del enunciado de la aplicación y documentación previa.
Acciones	1. Implementación del módulo correspondiente al servidor de control remoto. 2. Implementación del módulo correspondiente del sistema electrónico empotrado. 3. Integración de ambos sistemas de tiempo real.
Condiciones posteriores	La aplicación de tiempo real queda implementada.

Tabla 49. Caso de uso CU_A_09

3.3 Requisitos del sistema

Una vez definidos los requisitos de usuario y tras un análisis en profundidad de las características necesarias, surgen los requisitos de sistema, en los que quedan reflejadas de manera más detallada las funcionalidades que debe aportar la solución del proyecto.

Los requisitos de sistema quedan divididos en requisitos funcionales y requisitos no funcionales. Por comodidad, los requisitos son representados en tablas cuyo formato es el especificado a continuación:

- Identificador: Código unívoco que identifica al requisito cuya nomenclatura viene dada por el formato **XX_RE_YY**:

- **XX** será sustituido por ***'F'*** si se trata de un requisito funcional o por ***'NF'*** si se trata de un requisito no funcional.
- **RS** se corresponde con las siglas de Requisitos de Sistema.
- **YY** será sustituido por el número del requisito que corresponda dentro de su categoría empezando en ***01*** y aumentándose en una unidad por requisito.

- Descripción: Se compone de la especificación del requisito de manera detallada. Es necesario que se realice de forma simple y precisa para evitar ambigüedades.
- Necesidad: Parámetro que establece la importancia de incluir el requisito dentro del proyecto. Los valores que puede adoptar son ***'Esencial'*** y ***'Deseable'***.
- Prioridad: Parámetro que establece la importancia de implementar el requisito en el proyecto. Puede tomar los valores ***'Baja'***, ***'Media'*** y ***'Alta'***.
- Estabilidad: Indica el grado de variación que puede tener el requisito a lo largo del ciclo de vida del proyecto. Puede adoptar los valores ***'Estable'*** o ***'Inestable'***.
- Verificabilidad: Establece la dificultad que existe a la hora de verificar el cumplimiento del requisito. Sus valores pueden ser ***'Alta'***, ***'Media'*** o ***'Baja'***.
- Origen: Indica qué requisito de usuario queda contemplado por el requisito de sistema. Todos los requisitos de usuario deben estar recogidos, como mínimo, por un requisito de sistema.

3.3.1 Requisitos funcionales

Suponen una descripción más exhaustiva de los requisitos de usuario de capacidad.

Identificador: F_RS_01	
Descripción	Instalar y compilar el sistema operativo de tiempo real RTEMS.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_01

Tabla 50. Requisito de sistema F_RS_01

Identificador: F_RS_02	
Descripción	Utilizar la plataforma de hardware libre Arduino, en concreto el modelo Arduino UNO.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_02

Tabla 51. Requisito de sistema F_RS_02

Identificador: F_RS_03	
Descripción	Instalar la plataforma de software libre Arduino IDE.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_03

Tabla 52. Requisito de sistema F_RS_03

Identificador: F_RS_04	
Descripción	Instalar el entorno de desarrollo Eclipse IDE C/C++ Development Tooling (CDT) versión Galileo.
Necesidad	Deseable
Prioridad	Media
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_04

Tabla 53. Requisito de sistema F_RS_04

Identificador: F_RS_05	
Descripción	Instalar el emulador Qemu.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_05

Tabla 54. Requisito de sistema F_RS_05

Identificador: F_RS_06	
Descripción	Integrar el framework de desarrollo en una máquina virtual Ubuntu versión 14.04 LTS de 64 bits.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_06

Tabla 55. Requisito de sistema F_RS_06

Identificador: F_RS_07	
Descripción	La simulación de la imagen del sistema generada que contiene la implementación del servidor de control se realiza mediante el emulador Qemu.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_05

Tabla 56. Requisito de sistema F_RS_07

Identificador: F_RS_08	
Descripción	Implementar el módulo del servidor de control con el API de POSIX soportado por RTEMS, mediante la programación de un archivo fuente C el IDE Eclipse's C/C++ Development Tooling (CDT).
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_07, CA_RU_08, CA_RU_09

Tabla 57. Requisito de sistema F_RS_08

Identificador: F_RS_09	
Descripción	Implementar la parte software del sistema electrónico de control mediante la programación de un sketch Arduino en el entorno Arduino IDE.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_07, CA_RU_08, CA_RU_10

Tabla 58. Requisito de sistema F_RS_09

Identificador: F_RS_10	
Descripción	Implementar la parte hardware del sistema electrónico de control mediante la construcción de un prototipo con la placa Arduino UNO y los sensores y actuadores necesarios.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_07, CA_RU_08, CA_RU_10

Tabla 59. Requisito de sistema F_RS_10

Identificador: F_RS_11	
Descripción	Habilitar el puerto en el microcontrolador Arduino UNO serie para permitir la comunicación serial a unos baudios específicos.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_11

Tabla 60. Requisito de sistema F_RS_11

Identificador: F_RS_12	
Descripción	Identificar y habilitar el puerto serie en el módulo del servidor de control para permitir la comunicación serial a unos baudios específicos.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_11

Tabla 61. Requisito de sistema F_RS_12

Identificador: F_RS_13	
Descripción	Proporcionar una interfaz que aporte el código fuente necesario para habilitar la comunicación serial con la placa Arduino por parte del servidor.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_11

Tabla 62. Requisito de sistema F_RS_13

Identificador: F_RS_14	
Descripción	La aplicación recoge datos del entorno del sistema a través de los sensores y, tras el cómputo de los mismos, realizar las acciones pertinentes a través de los actuadores.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_12

Tabla 63. Requisito de sistema F_RS_14

Identificador: F_RS_15	
Descripción	La comunicación con los sensores y actuadores se realiza mediante los pines de entrada y salida incorporados en el microcontrolador Arduino UNO.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_12

Tabla 64. Requisito de sistema F_RS_15

Identificador: F_RS_16	
Descripción	Para la comunicación del sistema electrónico de control con los diferentes sensores y actuadores se debe configurar los pines del microcontrolador Arduino UNO como entrada o salida.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_12

Tabla 65. Requisito de sistema F_RS_16

Identificador: F_RS_17	
Descripción	La aplicación debe implementar un planificador cíclico simple para la ejecución de las tareas tanto en el módulo del servidor remoto como en el del sistema electrónico de control.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_13

Tabla 66. Requisito de sistema F_RS_17

Identificador: F_RS_18	
Descripción	El planificador cíclico simple de ambos módulos debe ser implementado mediante el algoritmo de planificación para tareas simples.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_13

Tabla 67. Requisito de sistema F_RS_18

Identificador: F_RS_19	
Descripción	La creación de la imagen del sistema a partir del ejecutable que genera RTEMS de la implementación del servidor de control se realiza mediante un bash script (.sh).
Necesidad	Deseable
Prioridad	Media
Estabilidad	Inestable
Verificabilidad	Alta
Origen	CA_RU_14

Tabla 68. Requisito de sistema F_RS_19

Identificador: F_RS_20	
Descripción	Proporcionar la documentación necesaria para instalar el sistema operativo de tiempo real e integrarlo con el entorno de desarrollo.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_15

Tabla 69. Requisito de sistema F_RS_20

Identificador: F_RS_21	
Descripción	Redactar el enunciado, los criterios y las pruebas para el diseño y corrección de la aplicación de tiempo real.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	CA_RU_15

Tabla 70. Requisito de sistema F_RS_21

3.3.2 Requisitos no funcionales:

Suponen una descripción más exhaustiva de los requisitos de usuario de restricción.

Identificador: NF_RS_01	
Descripción	Para la instalación de RTEMS descargar el código fuente desde su repositorio (https://git.rtems.org/) e instalarlo de forma manual mediante línea de comandos en un terminal.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	RE_RU_01

Tabla 71. Requisito de sistema NF_RS_01

Identificador: NF_RS_02	
Descripción	Configurar la placa Arduino UNO de para la conexión con Arduino IDE.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	RE_RU_02

Tabla 72. Requisito de sistema NF_RS_02

Identificador: NF_RS_03	
Descripción	Instalar Arduino IDE de forma manual mediante línea de comandos en un terminal.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	RE_RU_03

Tabla 73. Requisito de sistema NF_RS_03

Identificador: NF_RS_04	
Descripción	Para la instalación de Eclipse IDE C/C++ Development Tooling (CDT) versión Galileo, no es necesario instalar nada (versión portable), solo descargar la versión desde su repositorio (https://eclipse.org/downloads/packages/release/galileo/sr2)
Necesidad	Deseable
Prioridad	Media
Estabilidad	Estable
Verificabilidad	Alta
Origen	RE_RU_04

Tabla 74. Requisito de sistema NF_RS_04

Identificador: NF_RS_05	
Descripción	Instalar Qemu de forma manual mediante línea de comandos en un terminal.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	RE_RU_05

Tabla 75. Requisito de sistema NF_RS_05

Identificador: NF_RS_06	
Descripción	Para integrar las funcionalidades de RTEMS en el entorno de desarrollo de Eclipse, proporcionar el directorio de instalación del mismo e instalar el Plug-in RTEMS CDT Support.
Necesidad	Deseable
Prioridad	Media
Estabilidad	Estable
Verificabilidad	Alta
Origen	RE_RU_01, RE_RU_04

Tabla 76. Requisito de sistema NF_RS_06

Identificador: NF_RS_07	
Descripción	Implementar el código fuente que proporciona la estructura del módulo servidor en un archivo de programación C.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	RE_RU_06

Tabla 77. Requisito de sistema NF_RS_07

Identificador: NF_RS_08	
Descripción	Implementar el código fuente que proporciona la estructura del módulo correspondiente al sistema electrónico de control en un sketch de Arduino.
Necesidad	Esencial
Prioridad	Alta
Estabilidad	Estable
Verificabilidad	Alta
Origen	RE_RU_06

Tabla 78. Requisito de sistema NF_RS_08

3.4 Matriz de trazabilidad

Tras la definición y en análisis de los requisitos tanto de usuario como de sistema, es necesario asegurar que todos los requisitos de usuario están cubiertos por como mínimo un requisito del sistema. Las matrices de trazabilidad son el elemento que permite hacer dicha comprobación. Se trata de una matriz que contiene los identificadores de los requisitos de usuario en sus filas y los identificadores de los requisitos del sistema en sus columnas. De esta forma, se marca una intersección entre dos requisitos cuando uno de ellos, en concreto el requisito de usuario, está contemplado por el correspondiente requisito de sistema.

El campo "*Origen*" especificado en las tablas de los requisitos del sistema, ayudan a rellenar la matriz. Cuando la matriz se encuentre completa, es sencillo verificar si algún requisito de usuario no está contemplado por uno del sistema. En este caso, la implementación de la solución resultaría incompleta y habría que revisar la implementación.

En el caso de este proyecto se crean dos matrices de trazabilidad, la primera enfrenta los requisitos de usuario de capacidad y los requisitos de sistema funcionales, la segunda enfrenta los requisitos de usuario de restricción y los requisitos de sistema no funcionales.

	F_RS_01	F_RS_02	F_RS_03	F_RS_04	F_RS_05	F_RS_06	F_RS_07	F_RS_08	F_RS_09	F_RS_10	F_RS_11	F_RS_12	F_RS_13	F_RS_14	F_RS_15	F_RS_16	F_RS_17	F_RS_18	F_RS_19	F_RS_20	F_RS_21
CA_RU_01	X																				
CA_RU_02		X																			
CA_RU_03			X																		
CA_RU_04				X																	
CA_RU_05					X		X														
CA_RU_06						X															
CA_RU_07								X	X	X											
CA_RU_08								X	X	X											
CA_RU_09								X													
CA_RU_10									X	X											
CA_RU_11											X	X	X								
CA_RU_12														X	X	X					
CA_RU_13																	X	X			
CA_RU_14																			X		
CA_RU_15																				X	X

Tabla 79. Matriz de trazabilidad CA_RU – F_RS

	NF_RS_01	NF_RS_02	NF_RS_03	NF_RS_04	NF_RS_05	NF_RS_06	NF_RS_07	NF_RS_08
RE_RU_01	X					X		
RE_RU_02		X						
RE_RU_03			X					
RE_RU_04				X		X		
RE_RU_05					X			
RE_RU_06							X	X

Tabla 80. Matriz de trazabilidad RE_RU - NF_RS

Como se puede observar en ambas matrices todos los requisitos de usuario están al menos contemplados por un requisito de sistema, de manera que se puede afirmar que todas las necesidades del proyecto están bien cubiertas.

Capítulo 4: Diseño

Se procede a realizar previamente una comparativa de las principales características de los sistemas operativos de tiempo real, así como de las plataformas hardware, para posteriormente, seleccionar los que formarán parte de este proyecto.

4.1 Framework

Es necesario indicar que el proyecto consta de dos partes claramente diferenciables:

- Diseño e integración de un entorno de trabajo o framework para desarrollar aplicaciones de tiempo real: Como se ha expuesto en los requisitos, es necesario formar un entorno de desarrollo completo para aplicaciones de tiempo real que incluya, principalmente, un sistema operativo de tiempo real y una plataforma hardware y software que permitan dicho desarrollo.
- Diseño e implementación de una práctica de tiempo real: Una vez creado el framework de desarrollo se debe realizar una aplicación empotrada de tiempo real que controle un sistema específico.

De esta manera el diseño se realiza en dos apartados que corresponden con las dos partes del proyecto.

En concreto, para la creación del framework de desarrollo hay que integrar diferentes tecnologías harware y software existentes. Concretamente es necesario formar un entorno de trabajo que contenga un sistema operativo de tiempo real, una plataforma hardware/software y un entorno de desarrollo en el que poder integrar las funcionalidades del sistema operativo de tiempo real y de esta manera facilitar la programación.

4.1.1 Comparativa de sistemas operativos de tiempo real

Los criterios principales en la elección del sistema operativo son los siguientes:

- Por el motivo de que el proyecto está destinado a fines educativos, es preferible que el coste del mismo sea bajo o nulo.
- Es necesario que soporte la plataforma x86.
- Debe integrar la programación con el estándar POSIX.
- Debe proporcionar una amplia documentación para uso.

Tras el estudio de las principales características en el subapartado *Sistemas de tiempo real* de los sistemas operativos que ofrecen las funcionalidades más adecuadas para el desarrollo del proyecto, se realiza a continuación, una comparativa en forma de

tabla para la elección del mismo en base a las ventajas que ofrece cada uno y el acercamiento a las necesidades del proyecto.

Windows Embedded Compact	Real-Time LINUX	VxWorks	Embedded With RTEMS www.rtems.com
VENTAJAS			
• Los programadores pueden utilizar herramientas y desarrollar aplicaciones utilizando los mismos lenguajes y entornos que emplean con Windows para PC. • Multitarea. • Puede integrar Microsoft Visual Studio. • Escalabilidad, se pueden elegir los componentes necesarios de la imagen del sistema. • Documentación extensa y soporte en Internet.	• Gratuito, no es necesaria la adquisición de una licencia, se trata de un RTOS open source. • Programación flexible. • Es pequeño y estable. • Permite el uso de la interfaz POSIX. • Multitarea. • Gran portabilidad entre diferentes plataformas hardware. • Carga y descarga dinámica de módulos. • Instalación sencilla (aplicación de un parche al núcleo de Linux). • Gran cantidad de documentación y soporte en Internet.	• Bajo consumo de energía. • Utiliza poca memoria • Gestión potente de los eventos hardware. • Alta fiabilidad. • Multi-plataforma. • Multitarea. • Conjunto amplio de herramientas de comunicación entre procesos (señales, semáforos, mutex…) • Planificación basada en prioridades. • Reducción de los plazos en los que las interrupciones están deshabilitadas. • Permite el uso de la interfaz POSIX.	• Gratuito, no es necesaria la adquisición de una licencia, se trata de un RTOS open source. • Bajo consumo de energía. • Multitarea. • Gran portabilidad entre diferentes plataformas hardware y aislamiento del núcleo frente a los paquetes hardware específicos. • Está escrito en dos lenguajes: C y Ada. • Permite el uso de la interfaz POSIX. • Plug-in disponible para Eclipse. • Planificación basada en prioridades de 256 niveles. • Comunidad efectiva en Internet con amplia documentación y con listas de correos para discusiones técnicas y consultas.

Tabla 81. Comparativa de las ventajas de los sistemas operativos de tiempo real estudiados

Una vez analizadas las principales características de los RTOS estudiados, **el elegido para formar parte del framework es RTEMS** principalmente por ser gratuito, de código abierto y permitir la programación con el interfaz POSIX.

Tras la selección de RTEMS como sistema operativo de tiempo real y por la existencia de un *plugin* que hereda de CDT (conjunto de herramientas de desarrollo en C/C++) todas sus funcionalidades, **se decide utilizar Eclipse versión Galileo**, cuyo objetivo principal es facilitar la programación y compilación de las aplicaciones de tiempo real RTEMS.

Eclipse es una plataforma software multiplataforma y de código abierto desarrollada por IBM. Está destinada principalmente para crear entornos de desarrollo integrados (IDE) que permiten la programación, compilación, ejecución y depuración de aplicaciones. Su característica principal es la extensa comunidad de usuarios que deriva en una amplia colección de *plugins* que ofrecen diferentes funcionalidades y permiten desarrollar en un amplio abanico de lenguajes (Java, COBOL, C/C++). [32]

Ilustración 21. Logotipo de Eclipse

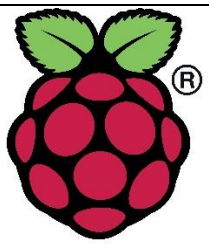
4.1.2 Comparativa de plataformas hardware/software

En la selección de una plataforma hardware/software y con el objetivo de ajustar de una manera eficiente los recursos a las necesidades del proyecto, se especifican los principales aspectos a tener en cuenta del mismo, que determinarán la plataforma seleccionada:

- Puesto que el proyecto tiene fines educativos, el precio de la plataforma utilizada no debe ser demasiado alto. Además, si es posible se elegirá un entorno de programación que disponga de amplia documentación y que facilite la tarea del desarrollador proporcionando una serie de librerías que cubran los aspectos básicos de la programación en tiempo real.
- La aplicación de tiempo real a implementar no será de gran tamaño, y por ello no se necesita una capacidad de procesamiento demasiado alta. Por la misma razón no es necesario que la cantidad de memoria (tanto principal como RAM) sea elevada.

- La aplicación no tiene la necesidad de que la placa incorpore batería porque siempre existe una conexión a red disponible.
- Debido a que la placa hardware interactuará con un número considerable de sensores y actuadores, es necesario que incorpore un número de entradas y salidas adecuado, y que sean tanto analógicas como digitales.
- La única conexión que necesita la aplicación entre sus diferentes módulos es la comunicación serial, por lo que no es necesario otro tipo de conexiones como WiFi, Bluetooth o Ethernet.
- No es necesario ningún sistema operativo corriendo en la plataforma, puesto que únicamente se debe cargar el código de aplicación directamente a la placa de desarrollo.

Tras el estudio de las principales plataformas open-source se realiza una comparativa en base a las características que presentan, haciendo una previa selección del modelo de cada empresa que mejor se adapta a las necesidades del problema. Los modelos elegidos son: Arduino Uno, Raspberry Pi 1 Model A+ y BeagleBone Green SeeedStudio.

		
	CARACTERÍSTICAS TÉCNICAS	
ARDUINO UNO	RASPBERRY PI 1 MODEL A+	BEAGLEBONE GREEN SEEEDSTUDIO
Microcontrolador ATmega 328P de 8 bits y 20MHz	Microprocesador ARMv6Z de 32 bits y 700MGHz	Microprocesador ARM AM335x de 32 bits y 1GHz
14 E/S digitales y 6 analógicas	40 E/S digitales o analógicas	92 E/S digitales o analógicas
Memoria Flash de 32KB y 2KB de SRAM	Memoria RAM de 512MB	Memoria Flash de 4GB y 512MB de RAM DDR3
Conexión USB, SPI e I2C	Conexión USB, USB micro HDMI, CSI, DSI, Micro SD	Conexiones USB (cliente y host), Ethernet
No soporta ningún sistema operativo, el programa se carga directamente a la placa vía serie	Sistema operativo oficial Raspbian	Soporta diferentes sistemas operativos
20 €	27€	38€

VENTAJAS		
• Programación sencilla con multitud de librerías, especialmente recomendada para la iniciación. • Bajo coste. • Bajo consumo. • Es multiplataforma, funciona en los sistemas operativos Windows, Linux y Macintosh OSX. • Gran comunidad en Internet, documentación, manuales, foros, ejemplos implementados, cursos, etc.	• Programación sencilla con multitud de librerías. • Potencia de computación para proyectos medianamente grandes. • Bajo coste. • Gran número de entradas y salidas. • Puede ejecutar diferentes de Linux. • Soporta protocolos de Internet. • Soporte para proyectos multimedia. • Capacidad de expansión mediante tarjeta Micro SD • Gran comunidad en Internet, documentación, manuales, foros, ejemplos implementados, cursos…	• Potencia computacional equivalente a un computador tradicional. • Procesador gráfico y aceleración 3D. • Conjunto muy extenso de entradas y salidas. • Sensores integrados (acelerómetro, giroscopio, barómetro) • Conexión Ethernet. • Puede ejecutar multitud de sistemas operativos. • Incorpora de serie un sistema operativo. • Buena documentación en la página web de la organización.
DESVENTAJAS		
• Baja potencia computacional. • No soporta la programación concurrente. • Memoria escasa, problemas con procesos pesados. • Programas en los que sólo hay que interactuar con sensores y actuadores.	• Máquina sin sistema operativo de serie, retrasa su uso. • Al no poseer memoria secundaria, necesita una Micro SD para cargar el sistema operativo. • Necesidad de instalar drivers para el control de periféricos. • Escasas interfaces para el control de sensores/actuadores.	• Comunidad menos extensa que la de Arduino o Raspberry Pi, mayor dificultad para encontrar ejemplos, proyectos o tutoriales. • Destinado para desarrolladores más experimentados. • Potencia computacional y cantidad de memoria excesiva para proyectos pequeños.

Tabla 82. Comparativa de las plataformas hardware estudiadas

Una vez realizado un completo análisis de las tres plataformas y de los modelos que más se acercan al proyecto, y en base a los criterios especificados anteriormente, **la plataforma escogida es Arduino**. El modelo de placa específico a utilizar es **Arduino UNO**.

4.1.3 Arquitectura del entorno de trabajo

A continuación se muestra la arquitectura del framework integrado para el desarrollo de aplicaciones de tiempo real. Consta de las siguientes partes:

- Sistema operativo de tiempo real RTEMS versión 4.12.
- Eclipse CDT versión Galileo y con el Plugin de RTEMS.
- Arduino IDE.
- Microcontrolador Arduino modelo UNO.

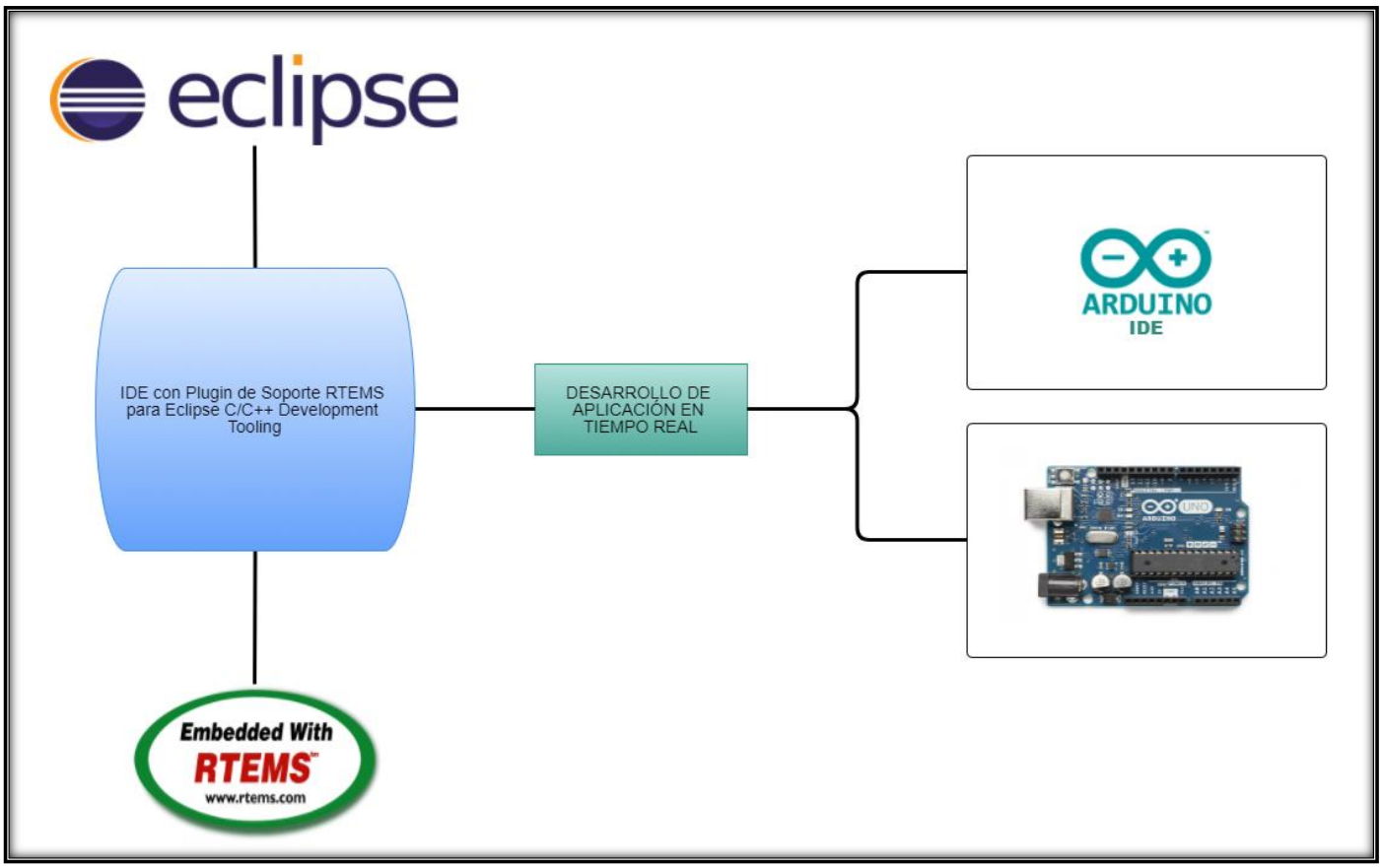


Ilustración 22. Arquitectura del entorno de trabajo integrado

4.2 Aplicación

Una vez integrado todo el entorno de desarrollo, es posible la implementación de aplicaciones de tiempo real basadas en RTEMS y la plataforma Arduino.

La aplicación a desarrollar es una aplicación de tiempo real empotrada para el control de un sistema o dispositivo con un objetivo determinado. Consta de dos sistemas de tiempo real diferenciados:

- Un sistema electrónico de control empotrado en el propio sistema.
- Un sistema de control remoto instalado en un servidor autónomo.

De esta manera y como se puede observar, la aplicación sigue un modelo Cliente-Servidor de arquitectura distribuida, en la que el servidor de control envía peticiones al sistema empotrado para recoger datos del dispositivo que controla, procesarlos y generar las acciones necesarias para alcanzar su objetivo. Ambas partes, por supuesto, son capaces de comunicarse entre sí.

La parte del servidor envía continuamente peticiones al sistema de control para conocer el estado del dispositivo que controla. Una vez recogidos los datos, hace los cálculos necesarios y envía de nuevo una petición para realizar la acción específica. Su implementación se realizará mediante programación C (interfaz POSIX) en el entorno de desarrollo para Eclipse CDT con el Plugin de RTEMS del entorno de trabajo.

En cuanto al sistema de control empotrado se encuentra escuchando continuamente peticiones del servidor remoto, las procesa y devuelve las respuestas con los datos correspondientes. Este sistema estará implementado mediante la plataforma Arduino, tanto software (Arduino IDE), como hardware (Arduino UNO). Para recoger los datos y realizar las acciones pertinentes del dispositivo controlado, el microcontrolador está conectado a una serie de sensores y actuadores. Las conexiones con los mismos se realizan a través del módulo de entradas y salidas que incorpora.

Ambas partes se comunican mediante una conexión serie UART estándar mediante el envío de mensajes de petición y respuesta de forma síncrona. Dicha conexión serie es posible gracias al puerto serie que se habilita al conectar la placa microcontroladora con el computador mediante el cable USB correspondiente.

Por lo tanto el esquema de la arquitectura de la aplicación de tiempo real queda definida de la siguiente forma:

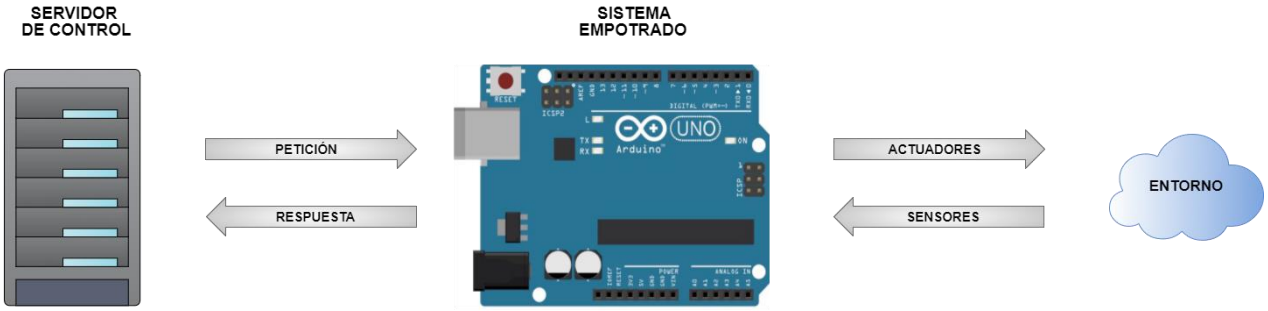


Ilustración 23. Arquitectura de la aplicación de tiempo real

A continuación se especifica la conexión del microcontrolador con cada uno de los sensores y actuadores a los que está conectado (se supone una cantidad relativamente grande de entradas y salidas, sin importar la ocupación de las mismas):

- Botón (sensor): Se trata de un pulsador que necesita mantener una presión para su funcionamiento. El estado inicial del botón no permite pasar la corriente, es cuando se mantiene pulsado cuando se cierra el circuito y fluye la corriente.

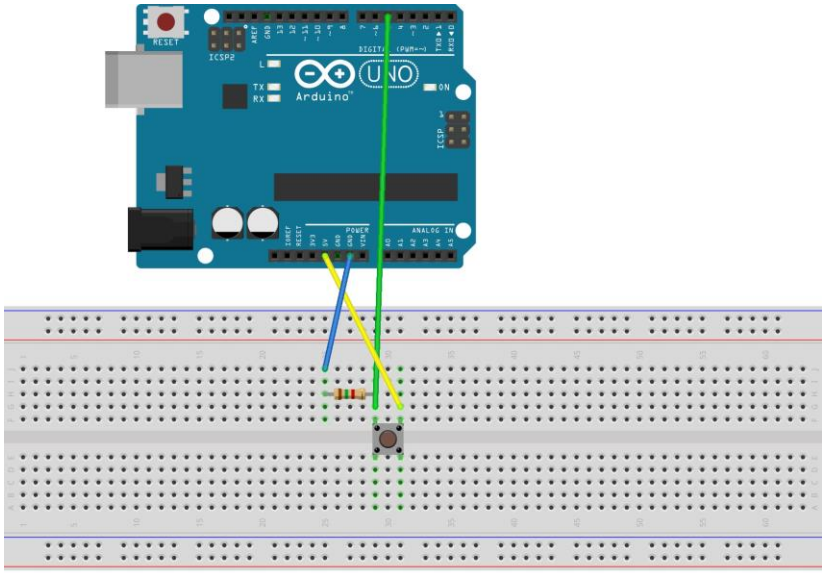


Ilustración 24. Esquema de conexión botón

- Sensor Infrarrojo modelo IR FC-51: Detecta la presencia de un objeto mediante la reflexión de la luz. Integra un LED emisor de infrarrojos y un fotodiodo que recibe la misma en caso de un posible obstáculo.

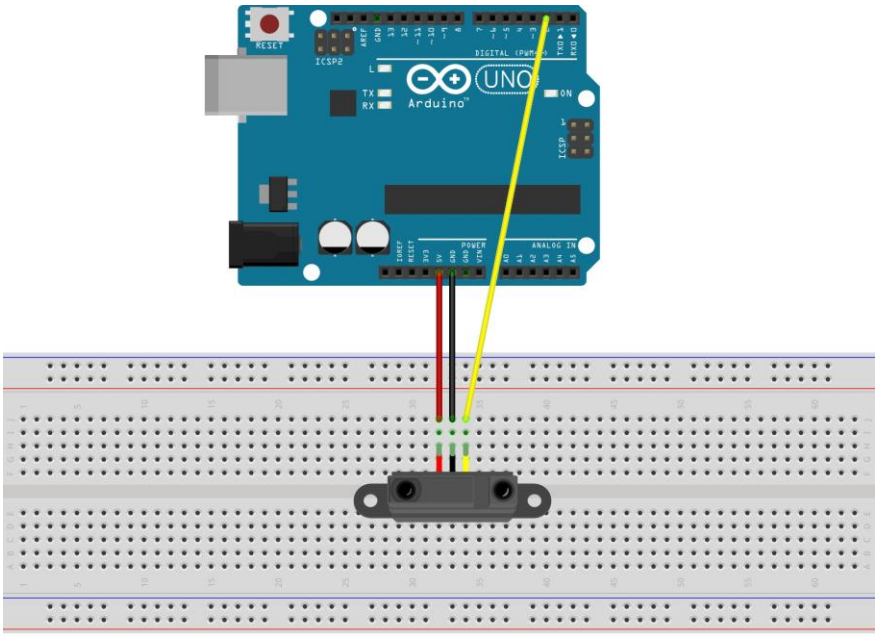


Ilustración 25. Esquema de conexión sensor infrarrojo

- Sensor de gas modelo MQ-2: Detecta mediante un sensor electro-químico Metano, butano, gas licuado del petróleo y humo.

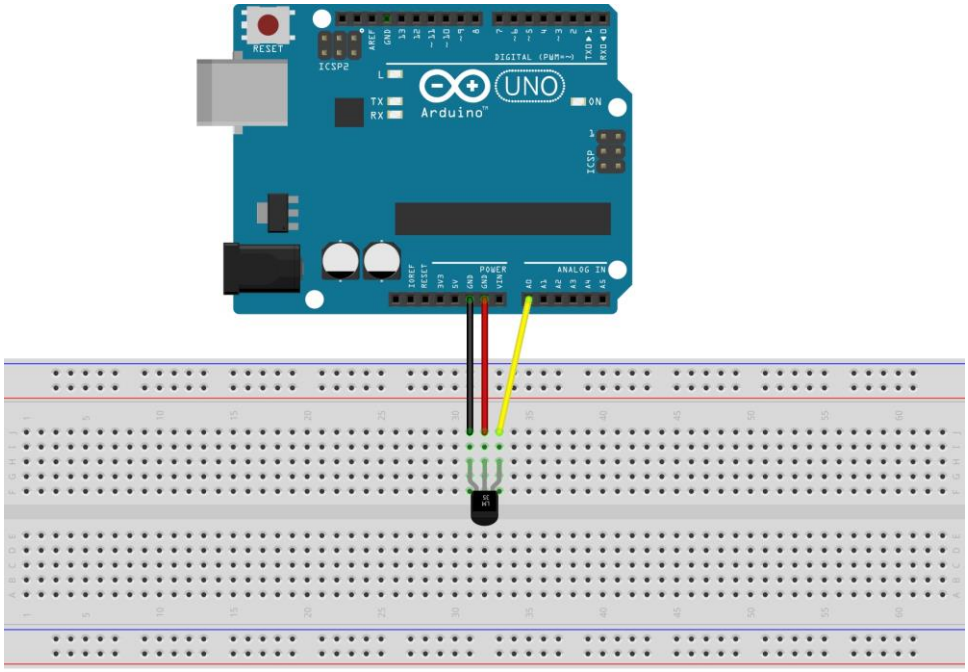


Ilustración 26. Esquema de conexión sensor de gas

- Led (actuador): Es un diodo que emite luz al paso de corriente eléctrica.

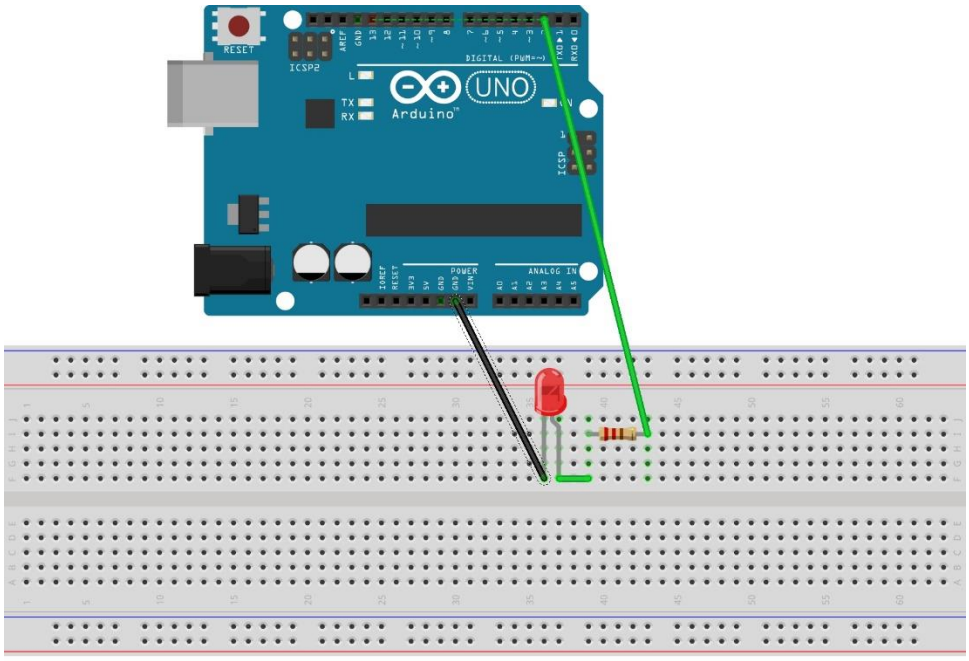


Ilustración 27. Esquema de conexión LED

- Sensor de llama modelo BSC-FLAME: Integran un sensor infrarrojo que detecta longitudes de onda desde los 760 a los 1100 nanómetros.

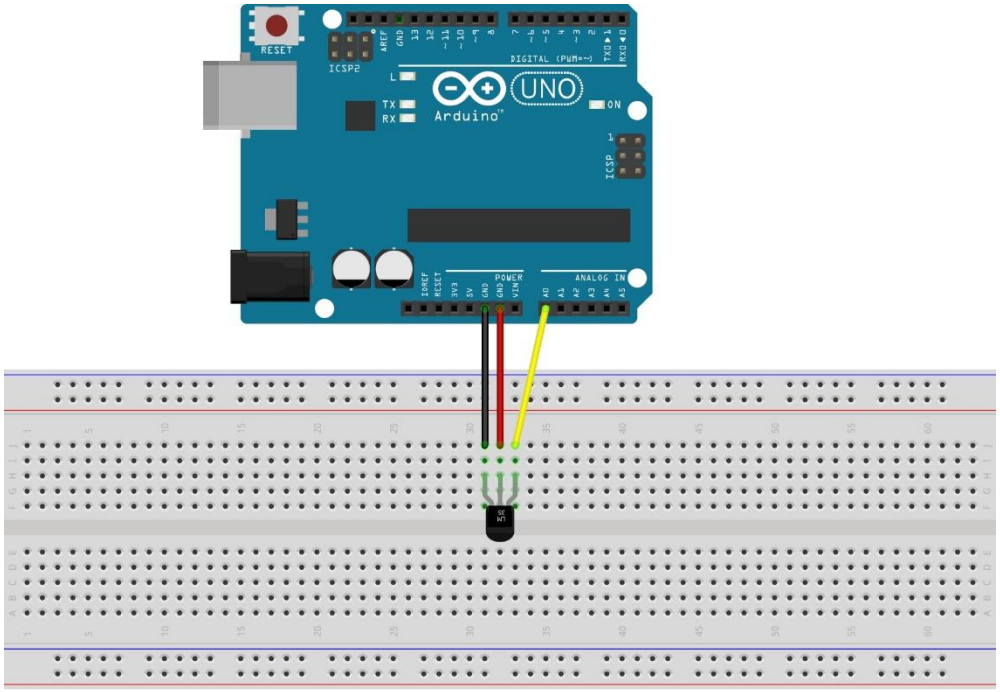


Ilustración 28. Esquema de conexión sensor de llama

- Buzzer: Se trata de un dispositivo que genera un sonido con una determinada frecuencia.

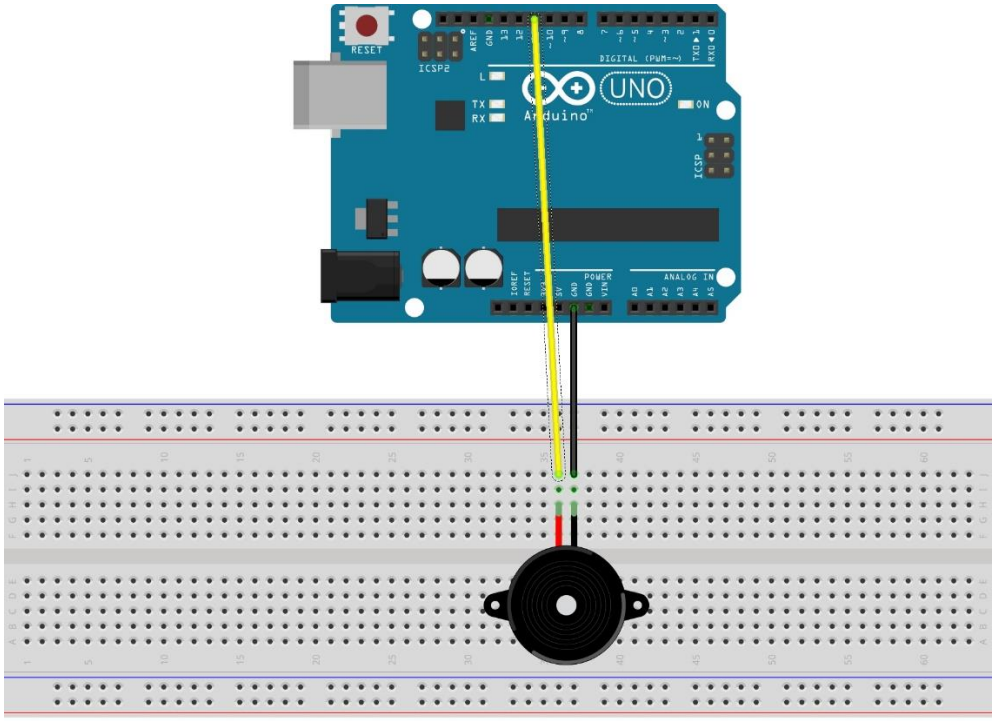


Ilustración 29. Esquema de conexión buzzer

- Sensor de temperatura LM35: Se obtiene la temperatura por la medición de la resistencia eléctrica que contiene, proporcionando un voltaje proporcional a la temperatura.

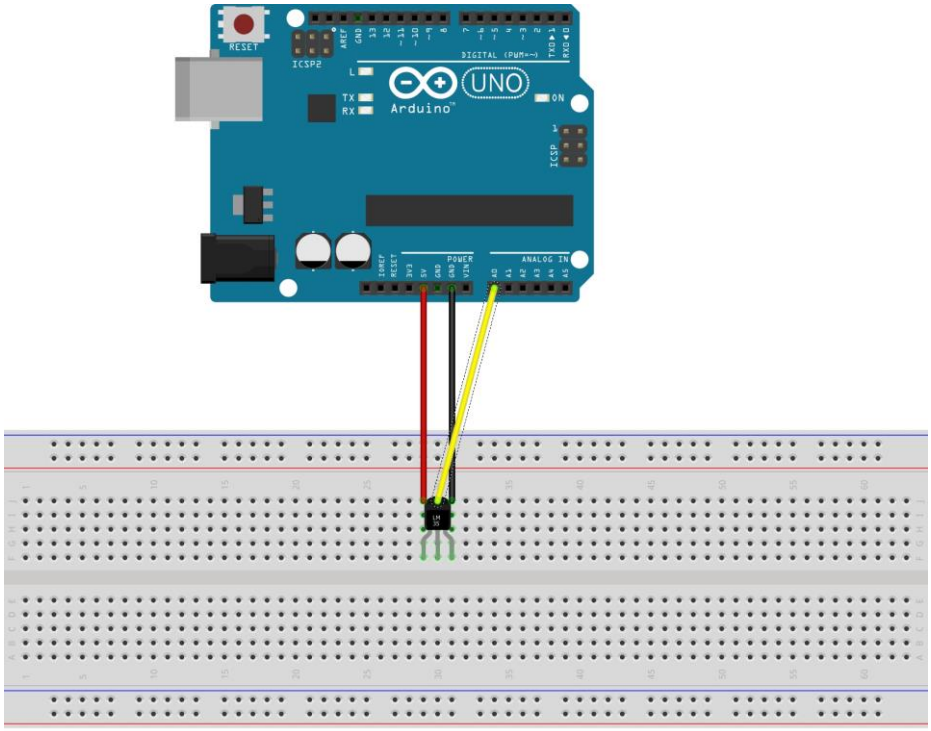


Ilustración 30. Esquema de conexión sensor de temperatura

- Interruptor magnético (sensor): Se trata de un interruptor que se activa en presencia de un imán.

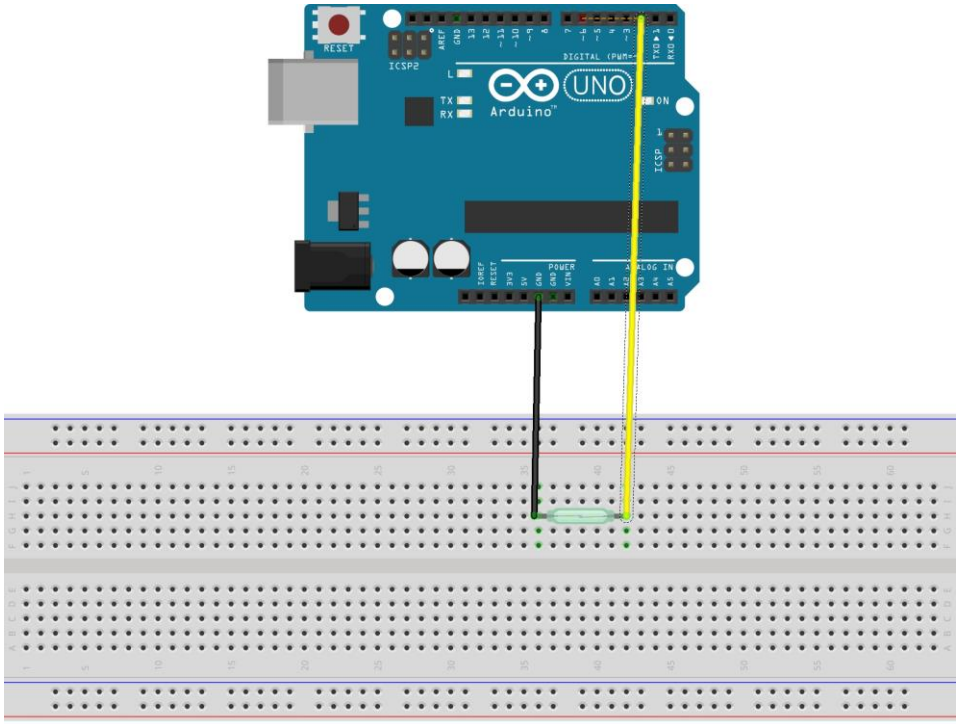


Ilustración 31. Esquema de conexión interruptor magnético

- Altavoz 0.5W (actuador):

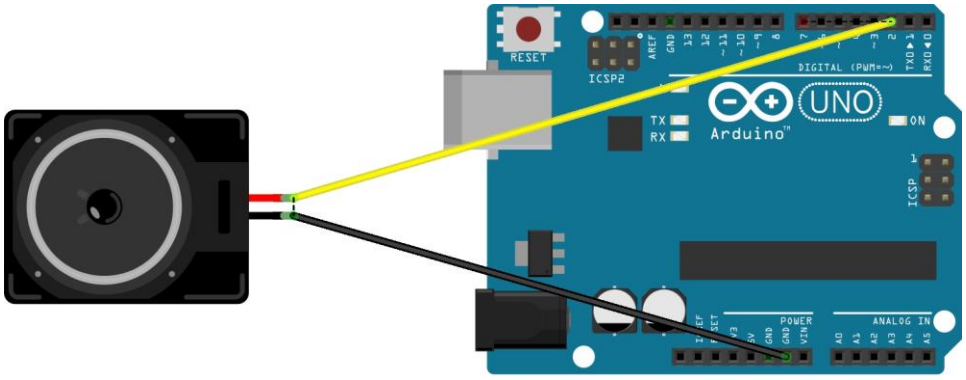


Ilustración 32. Esquema de conexión altavoz

- Ventilador 5V (actuador):

Ilustración 33. Esquema de conexión ventilador

- Pantalla LCD 16x2 (actuador) y potenciómetro (sensor): Es una pantalla con tecnología *Liquid Crystal Display.* Para calibrar el contraste es necesario acoplar un potenciómetro en el circuito.

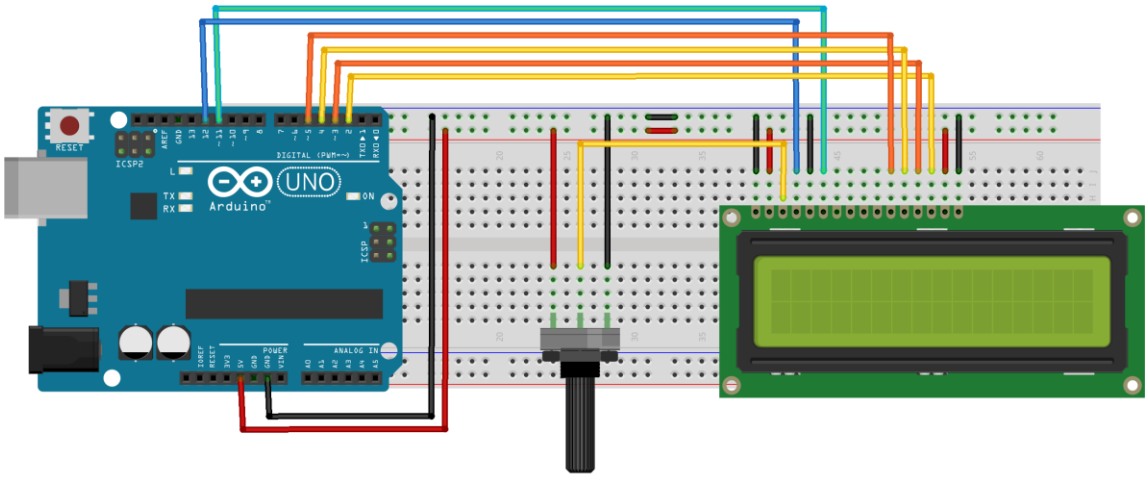


Ilustración 34. Esquema de conexión pantalla LCD

Capítulo 5: Implementación

En este capítulo se describe la integración del entorno de trabajo, así como la implementación de la aplicación concreta; destacándose solamente los aspectos relevantes o complejos.

5.1 Framework

Toda la implementación ha tenido lugar en una máquina virtual con las siguientes características:

Sistema Operativo	Ubuntu 14.04 LTS de 64 bits
Procesador	Intel® Core™ i5-4690 CPU @ 3.50GHz × 2
Memoria	4 Gb
Disco duro	60 Gb

Tabla 83. Características de la máquina virtual

Como se ha expuesto en apartados anteriores, la creación del framework para el desarrollo de aplicaciones en tiempo real integra tres elementos principales: RTEMS, la plataforma Arduino y Eclipse CDT. En esta sección únicamente se destacan y analizan los aspectos más importantes o que conllevan mayor complejidad de los tres componentes principales relacionados con su instalación, compilación e integración en el framework.

5.1.1 Instalación y compilación de RTEMS

El proceso de instalación y compilación del sistema operativo, se encuentra detallado el *Anexo B*.

La compilación del sistema operativo conlleva una serie de procesos previos de instalación. En primer lugar se necesita instalar el llamado *Source Builder*, una herramienta que permite crear los módulos que RTEMS necesita a partir de código fuente. Proporciona los detalles para construir dicho módulos de forma controlada y segura. En resumen, el *Source Builder* automatiza la instalación y proporciona el conjunto de herramientas necesario para RTEMS. Cabe destacar como aspecto positivo, que soporta un conjunto muy amplio de distribuciones en las que se puede instalar (*Archlinux, Centos, Fedora, FreeBSD, NetBSD, MacOS, Linux Mint, openSUSE, Raspbian, Ubuntu, Windows, Xubuntu*).

La instalación almacenará en un directorio de archivos toda la estructura el entorno de desarrollo RTEMS. Un aspecto fundamental para el desarrollo de aplicaciones con RTEMS es la modificación de la variable *PATH*. Es necesario especificar al sistema operativo host la ruta del nodo padre del cual colgará toda la estructura de RTEMS y en la que se almacenarán los programas.

Una vez instalado el *Source Builder*, el siguiente paso es la instalación del conjunto de herramientas de compilación (*Build Set*) específico para la arquitectura de procesador que se va a utilizar. El conjunto de herramientas se instala a través del *Source*

Builder. En el caso de este proyecto la arquitectura del procesador a utilizar es *i386* y por lo tanto es esencial que en todos los pasos de la instalación del *Build Set* que se requiera, se establezca dicha arquitectura.

En este momento ya es posible compilar el Kernel de RTEMS tras descargarlo de su repositorio y ejecutar un archivo de configuración. Es esencial compilar el kernel para un *Board support package* (BSP) particular, ya que contiene todos los drivers que permiten la comunicación de RTEMS con el hardware específico utilizado. Por lo tanto es completamente necesario indicar en la compilación el *BSP* específico con el *flag* correspondiente, en el caso de este proyecto, *pc386*. Además y como era requisito fundamental hay que habilitar la interfaz POSIX también con el *flag* correspondiente.

5.1.2 Integración de RTEMS en Eclipse Galileo

El proceso de instalación del Plugin RTEMS en Eclipse se encuentra detallado el *Anexo C*.

Para facilitar el desarrollo de las aplicaciones con RTEMS, es un aspecto positivo incluir sus características de programación en un entorno de desarrollo completo. Para ello se procede a la instalación de un *plugin* con soporte para RTEMS en Eclipse CDT versión Galileo.

El aspecto más importante de esta integración se encuentra en el ámbito de la configuración. Es necesario indicar al entorno de desarrollo Eclipse la ruta de instalación de RTEMS (nodo padre de la estructura de directorios) así como la que contiene el *BSP* específico. De esta manera se proporcionan dichos valores a los proyectos RTEMS que se creen.

5.1.3 Instalación y configuración de la plataforma Arduino

La instalación del entorno de desarrollo Arduino IDE no conlleva más complicación que una instalación tradicional, pero al conectar la placa microcontroladora para realizar la comunicación entre ambos y poder cargar el programa, es posible que el IDE no reconozca la misma. Esto se debe a un problema relacionado con los permisos si no se ha ingresado como el usuario *root* y para ello es necesario dar los permisos correspondientes a los demás usuarios:

Ubuntu reconoce y asigna a la placa un dispositivo de comunicación USB de tipo *ttyACMn*. Un dispositivo virtual almacenado en fichero en /dev. Para ver los permisos del fichero se ejecuta el siguiente comando:

```
$ ls -l /dev | grep ACM
```

Para otorgar permisos al resto de usuarios es necesario ejecutar el comando:

```
$ sudo chmod 777 /dev/ttyACMn
```

Tras realizar este ajuste, Arduino IDE ya es capaz de reconocer el microcontrolador.

5.2 Aplicación

La aplicación desarrollada consta de dos partes claramente diferenciables:

- Un sistema electrónico de control empotrado en el propio sistema: La parte hardware se implementa mediante Arduino UNO y los sensores y actuadores necesarios. La parte software se programa mediante Arduino IDE.
- Un sistema de control remoto instalado en un servidor autónomo: Solo consta de parte software y se implementa mediante programación C (interfaz POSIX) en el entorno de desarrollo para Eclipse CDT con el Plugin de RTEMS.

5.2.1 Sistema electrónico de control empotrado en el propio sistema

El hardware de Arduino está formado por pequeñas placas de desarrollo programables basadas en microcontroladores. Existen cantidad de placas con diferentes características y funcionalidades, que ofrecen un abanico muy amplio de implementaciones. Los microcontroladores más presentes en las tarjetas suelen ser de la familia AVR de Atmel, basados en 8 bits y en la arquitectura RISC (Reduced Instruction Set Computer). Actualmente incorporan microcontroladores de la nueva gama de Atmel con arquitectura ARM y basados en 32 bits. En el caso de este proyecto se utiliza el modelo Arduino UNO, cuyas características se recuerdan a continuación:

Arduino UNO	
Microcontrolador	ATmega328P, AVR de 8 bits
Tensión de funcionamiento	5V
E/S Digitales	14 (6 de ellos proporcionan salida PWM)
Entradas Analógicas	6
Corriente DC por cada Pin E/S	20 mA
Corriente DC para el Pin de 3.3V	50 mA
Memoria Flash	32 KB (ATmega328P) 0.5 KB utilizados por el bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Velocidad de reloj	16 MHz

Tabla 84. Características Arduino UNO

Como se puede observar se trata de un microcontrolador muy sencillo y con baja potencia computacional, además la memoria de la que dispone es considerablemente pobre. Sin embargo y como se ha analizado anteriormente, reúne las características necesarias para este proyecto.

Arduino IDE es el entorno de desarrollo encargado de proporcionar la lógica al microcontrolador para realizar las funciones de control necesarias. El lenguaje de programación de Arduino está basado en C++ y los archivos de programación en los que se encuentra la lógica programada se denominan *sketches* (extensión *.ino*). En ellos se puede diferenciar una estructura por defecto que adoptan todos los programas y que consta de dos partes:

- Configuración: Proporciona un método denominado *setup()* en el que se configuran todos los parámetros necesarios para implementación del programa (velocidad de comunicación serial, configuración de entradas y salidas, etc). Este método solo se ejecuta una vez.

- Bucle infinito: Se trata de una ejecución cíclica que proporciona respuesta continua a los eventos que se producen en la placa.

Una vez creado programada la lógica en el sketch es necesario compilarlo y subirlo a la placa mediante conexionado USB. El microcontrolador necesita almacenar el programa en algún sitio de la memoria, el *bootloader*. Es un gestor de arranque que almacena y ejecuta el programa en el microcontrolador.

En cuanto a la implementación hardware del sistema electrónico de control empotrado en el propio sistema ya se han detallado los diagramas de conexión del microcontrolador con cada uno de los sensores y actuadores que integra. A pesar de ello, es necesario destacar los siguientes aspectos:

- Los sensores infrarrojos IR FC-51, el sensor de gas MQ2 y el sensor de llama BSC-FLAME incorporan un potenciómetro para la regulación de un umbral a partir del cual se obtiene la lectura del fenómeno correspondiente. Por lo tanto es necesario regular el umbral.
- El sensor de temperatura LM35 debe tener una conexión a GND exclusiva puesto que en el caso de compartir la conexión el valor leído no corresponde con la realidad.

El skecth que contiene la lógica del microcontrolador se denomina *embeddedSystem.ino* y en lo referido a su implementación mediante el entorno de desarrollo Arduino IDE es necesario resaltar algunos aspectos:

La plataforma Arduino dispone de 14 entradas/salidas digitales (0-13) y de 6 entradas analógicas (A0-A5). Es posible convertir las entradas digitales en entradas y salidas digitales. Todas las entradas y salidas digitales han sido utilizadas en el desarrollo de la práctica y frente a la falta de una cantidad mayor tres de las entradas analógicas han sido utilizadas como digitales. La numeración de las entradas analógicas convertidas a digitales es por orden ascendente 14, 15,16 y 17. La instrucción necesaria utilizada para la conversión por ejemplo de la entrada analógica A0 a salida digital es:

```
pinMode(14, OUTPUT);
```

Para recibir las peticiones del servidor de control es necesario habilitar la comunicación por el puerto serie. La velocidad de transmisión de la información viene determinada en baudios, bits que se transfieren en un segundo. Tanto el computador como la placa deben coincidir en la velocidad de transmisión para no experimentar ni truncamiento ni pérdida de datos. Para la aplicación, la velocidad de transmisión se establece en Arduino con un valor de 9600 baudios mediante la siguiente instrucción:

```
Serial.begin(9600);
```

Además se proporciona un método de lectura y envío de mensajes mediante el puerto serie con un tamaño máximo igual a nueve caracteres (incluyendo el salto de línea *"\n"*).

El sensor de temperatura proporciona en la lectura un voltaje linealmente proporcional a la temperatura (10 mV por grado centígrado), por lo que es necesario hacer una conversión para obtener los grados centígrados correspondientes.

Las tareas de la aplicación tienen tiempos fijos de ejecución y gracias a esta característica es posible caracterizar la naturaleza determinista de la aplicación. Dicha naturaleza permite la programación de un planificador cíclico de tareas simples. Las

tareas como tal se programan por medio de métodos, de forma que cada tarea se corresponde con un método diferente. El planificador es el encargado de ejecutar las tareas en el instante de tiempo correspondiente a cada una.

5.2.2 Sistema de control remoto instalado en un servidor autónomo

El módulo correspondiente al servidor autónomo de control remoto solo se compone de implementación software, en concreto se trata de un programa C que utiliza el API de POSIX en el entorno de desarrollo para Eclipse CDT con el Plugin de RTEMS. El programa se denomina *"controlServer.c".*

RTEMS ejecuta en un simple proceso pesado con múltiples hilos, por lo tanto todos los hilos diferentes comparten el mismo espacio de memoria. Esto se debe a que los modelos simples de memoria implican un comportamiento más determinista e indicado para las aplicaciones de tiempo real.

RTEMS soporta diferentes interfaces estándar de aplicaciones, incluyendo POSIX. De esta manera puede hacer uso de rutinas orientadas a procesos POSIX, usuarios y grupos.

En primer lugar y para desarrollar el programa correspondiente al servidor de control hay que configurar el API de POSIX. La configuración consiste en añadir los parámetros necesarios de la aplicación a desarrollar, tales como el número máximo de hilos, mutex o señales.

El archivo de cabecera "*rtems/confdefs.h*" es el núcleo de la generación automática de la configuración del sistema y se basa en la idea del establecimiento de macros que definen los parámetros de configuración necesarios para la aplicación de tiempo real. Es el desarrollador el encargado de especificar los valores de dichos parámetros (por defecto el valor de los parámetros es 0).

Los parámetros necesarios para el desarrollo de la aplicación son los especificados a continuación:

- CONFIGURE_INIT: Se trata de una constante necesaria para poder instanciar las estructuras para la configuración de datos a través del archivo de cabecera "*rtems/confdefs.h*"**.** Solo se puede definir una vez por aplicación para no generar errores.
- CONFIGURE_APPLICATION_NEEDS_CONSOLE_DRIVER: Mediante la especificación de esta constante la aplicación incluirá un driver para el control

de la consola. Así, la consola se utilizará para los flujos de entrada, salida y error estándar reservando para ello tres descriptores de archivo.

- CONFIGURE_APPLICATION_NEEDS_CLOCK_DRIVER: Configura un driver para utilizar los tiempos de reloj. De este modo RTEMS será capaz de tener consciencia del tiempo y de poder manejar los tiempos de ejecución de las tareas que se implementen.
- CONFIGURE_MAXIMUM_POSIX_TIMERS: Define el número máximo de temporizadores activos simultáneamente en la aplicación.
- CONFIGURE_POSIX_INIT_THREAD_TABLE: Si la aplicación va a hacer uso de threads es necesario especificar esta constante. Permite la utilización una tabla de inicialización de hilos necesaria para su creación.
- CONFIGURE_MAXIMUM_POSIX_THREADS: Mediante la especificación de este parámetro se configura el número máximo de hilos que pueden ejecutar simultáneamente.
- CONFIGURE_MAXIMUM_POSIX_CONDITION_VARIABLES: Especifica el número máximo de variables de condición que pueden estar activas simultáneamente.
- CONFIGURE_MAXIMUM_POSIX_MUTEXES: Especifica el número máximo de mutex que pueden estar activos simultáneamente.
- CONFIGURE_MAXIMUM_POSIX_QUEUED_SIGNALS: Especifica el número máximo de señales que pueden estar activas simultáneamente.
- CONFIGURE_LIBIO_MAXIMUM_FILE_DESCRIPTORS: Especifica el número máximo de descriptores de archivo de la aplicación. Si CONFIGURE_NEEDS_CONSOLE_DRIVER está definido el valor predeterminado es 3, de lo contrario el valor predeterminado es 0.
- CONFIGURE_NUMBER_OF_TERMIOS_PORTS: Se establece en el número de puertos que utilizan la funcionalidad "*termios*".

Las constantes y parámetros definidos anteriormente son todos los necesarios para el funcionamiento de la aplicación.

Otra peculiaridad de las aplicaciones de tiempo real implementadas con RTEMS es la definición de un método "*POSIX_Init()".* Se trata del método que proporciona la entrada principal y la inicialización de la aplicación (hilo principal). Es un método obligatorio y solo se puede definir una vez por aplicación.

Para facilitar y dividir la programación del servidor remoto de control se proporciona un sistema que recibe los mensaje de petición y devuelve una respuesta simulada como si proviniera realmente del microcontrolador Arduino. Con este método es posible, antes de realizar la conexión real mediante el puerto serie con Arduino, asegurar el correcto formato de las peticiones y las respuestas.

Con el objetivo de realizar la comunicación serial con Arduino se hace uso de una librería que implementa las funciones principales para iniciar, escribir y leer puertos serie. Además y como establece Arduino de forma análoga hay que establecer la velocidad de comunicación. Puesto que la velocidad establecida en el microcontrolador es de 9600 baudios, en la parte servidor se establece la misma velocidad para evitar problemas en la comunicación o pérdida de datos. El Board Support Package intalado para RTEMS identifica el puerto serie como *//dev/com1*. De esta forma y mediante la función proporcionada en la librería para la comunicación con arduino, la instrucción para inicializar el puerto serie es la siguiente:

```
Serialport_init (“/dev/com1, 9600”);
```

El servidor de control remoto puede enviar dos tipos de peticiones al sistema de control empotrado:

- Peticiones de lectura: Para recopilar datos del entorno del dispositivo controlado a través de los sensores incorporados a Arduino.
- Peticiónes de actuación: Para enviar las órdenes necesarias que controlan el dispositivo por medio de los actuadores una vez realizados los cálculos pertinentes.

De la misma manera que en el programa correspondiente al sistema electrónico empotrado, las tareas se programan por medio de métodos, y existe un planificador cíclico de tareas simples que se encarga de ejecutarlas en el instante preciso.

El diagrama de secuencia de la aplicación se muestra en la siguiente imagen:

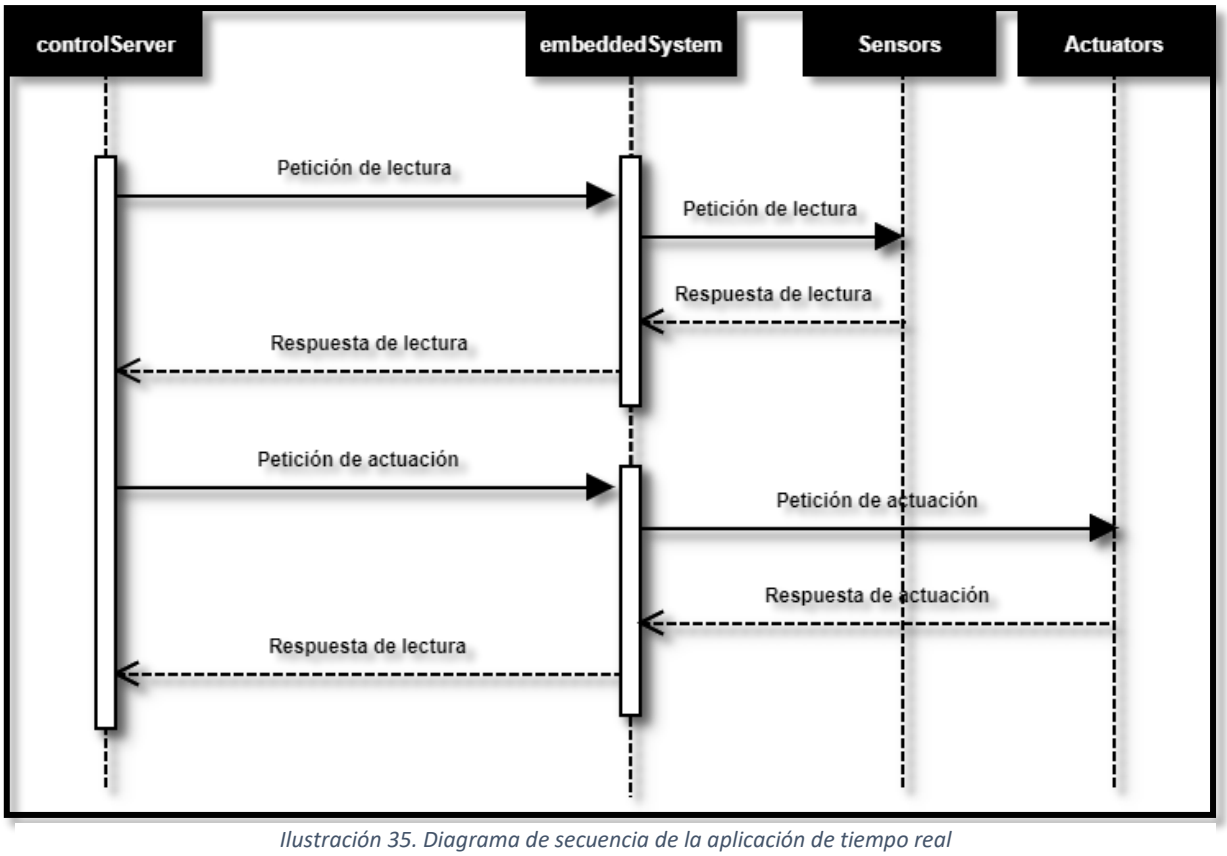


Ilustración 35. Diagrama de secuencia de la aplicación de tiempo real

5.3 Compilación y ejecución

Una vez implementados ambos módulos de la aplicación necesitan ser compilados para la comprobación de errores.

La compilación del módulo correspondiente al sistema de control empotrado en el dispositivo es altamente sencilla, puesto que el entorno de desarrollo de Arduino incluye su propio compilador, en concreto *avr-gcc*. La compilación genera un fichero binario hexadecimal (extensión *.hex*) entendido por el microcontrolador.

Una vez compilado el programa se carga el fichero binario en la memoria flash del microcontrolador y se ejecuta de manera inmediata, quedando a la espera de recibir peticiones del servidor remoto de control.

Por otra parte la compilación del programa del servidor remoto de control se realiza desde Eclipse mediante la forma habitual que ofrece este entorno.

El proceso de compilación y ejecución de una aplicación RTEMS se especifica en el *Anexo D*, pero es necesario resaltar una serie aspectos importantes:

- Si el programa no tiene ningún error y la compilación es satisfactoria, se generará un archivo ejecutable (extensión *.exe*) en el espacio de trabajo.
- Para ejecutar la aplicación es necesario generar una imagen de sistema RTEMS completa (extensión *.img*) a partir del archivo ejecutable. Para la creación de la imagen se proporciona un *bash script* que automatiza el proceso.
- Una vez obtenida la imagen del sistema RTEMS para poder ejecutar la aplicación hay que instalar la misma en una plataforma hardware. Por comodidad se hace uso del emulador Qemu que simulará la parte servidor de control remoto.

De esta forma la aplicación ejecuta de forma autónoma, mediante la comunicación de ambos sistemas de tiempo real.

Capítulo 6: Pruebas

Tras la implementación, se realizan a continuación una serie de pruebas para comprobar el funcionamiento tanto del framework de desarrollo como de la aplicación en sí.

6.1 Pruebas de funcionalidad

Todas las pruebas descritas son de funcionalidad y verifican las capacidades de los diferentes módulos que forman parte del proyecto.

Las pruebas son representadas en tablas para facilitar su lectura y comprensión. El formato de las mismas se define seguidamente:

- Identificador: Código unívoco que identifica a la prueba cuya nomenclatura viene dada por el formato **P_XX**, donde **XX** se sustituye por el número que corresponda con la prueba, comienza en ***01*** y se incrementa de manera unitaria por cada prueba.
- Módulo: Adquiere el formato **MX**, donde **X** se sustituye por ***"Framework"*** si la prueba se realiza sobre el módulo framework o ***"Aplicación"*** si la prueba se realiza sobre la aplicación.
- Elemento: Representa la unidad concreta de la prueba dentro del módulo correspondiente.
- Descripción: Explicación del propósito de la prueba.
- Requisitos previos: Conjunto de requisitos necesarios para la ejecución de la prueba.
- Acciones: Secuencia de pasos para la ejecución de la prueba.
- Resultado: Resultado de la prueba que adoptará los valores favorable o desfavorable.

Una vez definido el formato de la tabla que albergará cada prueba, se procede a la definición de las mismas.

Identificador: P_01	
Módulo	MFramework
Elemento	RTEMS
Descripción	Comprobación visual de los directorios correspondientes a la instalación y compilación del sistema operativo de tiempo real RTEMS.
Requisitos previos	1. Instalación y compilación completa de RTEMS
Acciones	1. Acceso al directorio "/home/user/development/rtems/4.12" y comprobación de los directorios creados por el *Build Set.* 2. Acceso al directorio "/home/cesar/development/rtems/kernel/rtems" y comprobación de los directorios creados en la descarga del kernel RTEMS. 3. Acceso al directorio "/home/cesar/development/rtems/kernel/pc386" y comprobación de los directorios creados por la compilación del *BSP* específico.
Resultado	Favorable

Tabla 85. Prueba P_01

Identificador: P_02	
Módulo	MFramework
Elemento	RTEMS
Descripción	Ejecutar un programa de ejemplo RTEMS proporcionado en su instalación.
Requisitos previos	1. Instalación y compilación completa de RTEMS.
Acciones	1. Acceso al directorio "/home/cesar/development/rtems/kernel/pc386/i386-rtems4.12/c/pc386/testsuites/samples/hello". 2. Ejecución del *bash script* para la creación de una imagen del sistema a partir del ejemplo *"hello.exe"*. 3. Emulación de la imagen mediante el emulador Qemu.
Resultado	Favorable

Tabla 86. Prueba P_02

Identificador: P_03	
Módulo	MFramework
Elemento	Eclipse CDT Galileo
Descripción	Comprobación manual de la instalación.
Requisitos previos	1. Instalación completa de Eclipse CDT Galileo.
Acciones	1. Acceso al entorno de desarrollo mediante el icono de escritorio. 2. Selección del espacio de trabajo. 3. Creación de un proyecto “Hello World ANSI C Project”. 4. Compilación y ejecución del mismo.
Resultado	Favorable

Tabla 87. Prueba P_03

Identificador: P_04	
Módulo	MFramework
Elemento	Eclipse CDT Galileo
Descripción	Comprobación manual de la instalación del Plugin RTEMS en el entorno de desarrollo.
Requisitos previos	1. Instalación y compilación completa de RTEMS. 2. Instalación completa de Eclipse CDT Galileo. 3. Instalación del Plugin RTEMS.
Acciones	1. Acceso al entorno de desarrollo mediante el icono de escritorio. 2. Selección del espacio de trabajo. 3. Creación de un proyecto Ejecutable “RTEMS Hello World Project”. 4. Compilación y ejecución del bash script para la creación de una imagen del sistema a partir del ejemplo “RTEMSHelloWorld.exe”. 5. Emulación de la imagen mediante el emulador Qemu.
Resultado	Favorable

Tabla 88. Prueba P_04

Identificador: P_05	
Módulo	MFramework
Elemento	Arduino IDE, Arduino UNO
Descripción	Comprobación manual de la instalación de Arduino IDE y de la correcta comunicación con la placa Arduino UNO.
Requisitos previos	1. Instalación completa de Arduino IDE. 2. Conexión de Arduino UNO vía USB. 3. Permisos de usuario sobre el puerto correspondiente ttyACMn.
Acciones	1. Acceso al entorno de desarrollo mediante el icono de escritorio. 2. Selección de un ejemplo incluido en la instalación del entorno. “Archivo – Ejemplos…” 3. Selección de la conexión con la placa desde Arduino IDE, “Herramientas – Puerto…” 4. Carga y ejecución del programa a la placa Arduino UNO.
Resultado	Favorable

Tabla 89. Prueba P_05

Identificador: P_06	
Módulo	MAplicación
Elemento	Servidor de control remoto
Descripción	Comprobación de la correcta inicialización de la aplicación.
Requisitos previos	1. Definición de las variables globales y parámetros necesarios para la aplicación. 2. Definición del método principal “POSIX_Init()”.
Acciones	1. Compilación y ejecución de “controlServer.c”.
Resultado	Favorable

Tabla 90. Prueba P_06

Identificador: P_07	
Módulo	MAplicación
Elemento	Servidor de control remoto
Descripción	Comprobación del formato de peticiones y respuestas mediante simulación.
Requisitos previos	1. Programación de las peticiones. 2. Uso del sistema de simulación proporcionado.
Acciones	1. Compilación y ejecución de “controlServer.c”.
Resultado	Favorable

Tabla 91. Prueba P_07

Identificador: P_08	
Módulo	MAplicación
Elemento	Servidor de control remoto
Descripción	Comprobación del cálculo de datos en función de las respuestas recibidas mediante simulación.
Requisitos previos	1. Programación de las peticiones. 2. Programación de la lógica en función de las respuestas recibidas. 3. Uso del sistema de simulación proporcionado.
Acciones	1. Compilación y ejecución de “controlServer.c”.
Resultado	Favorable

Tabla 92. Prueba P_08

Identificador: P_09	
Módulo	MAplicación
Elemento	Servidor de control remoto
Descripción	Comprobación de la ejecución del planificador cíclico.
Requisitos previos	1. Programación de las peticiones. 2. Programación de la lógica en función de las respuestas recibidas. 3. Uso del sistema de simulación proporcionado. 4. Programación del planificador cíclico de tareas simples.
Acciones	1. Compilación y ejecución de “controlServer.c”.
Resultado	Favorable

Tabla 93. Prueba P_09

Identificador: P_10	
Módulo	MAplicación
Elemento	Sistema de control empotrado.
Descripción	Comprobación del formato de peticiones y respuestas mediante el monitor serial que proporciona Arduino y que permite el envío de peticiones de forma manual.
Requisitos previos	1. Programación de las peticiones. 2. Definición de velocidad de conexión a 9600 baudios. 3. Carga del programa “embeddedSystem.ino” a la placa Arduino UNO.
Acciones	1. Acceso al monitor serial. 2. Envío de diferentes peticiones y observación de la respuesta.
Resultado	Favorable

Tabla 94. Prueba P_10

Identificador: P_11	
Módulo	MAplicación
Elemento	Sistema de control empotrado.
Descripción	Comprobación de la lectura de datos de los sensores mediante el monitor serial.
Requisitos previos	1. Programación de las peticiones. 2. Definición de velocidad de conexión a 9600 baudios. 3. Programación de los accesos a los sensores para la recopilación de datos. 4. Implementación del circuito eléctrico necesario. 5. Carga del programa “embeddedSystem.ino” a la placa Arduino UNO.
Acciones	1. Acceso al monitor serial. 2. Envío de diferentes peticiones y observación de la respuesta.
Resultado	Favorable

Tabla 95. Prueba P_11

Identificador: P_12	
Módulo	MAplicación
Elemento	Sistema de control empotrado.
Descripción	Comprobación de la acción de los actuadores mediante el monitor serial.
Requisitos previos	1. Programación de las peticiones. 2. Definición de velocidad de conexión a 9600 baudios. 3. Programación de los accesos a los actuadores para la ejecución de acciones. 4. Implementación del circuito eléctrico necesario. 5. Carga del programa “embeddedSystem.ino” a la placa Arduino UNO.
Acciones	1. Acceso al monitor serial. 2. Envío de diferentes peticiones y observación de la respuesta.
Resultado	Favorable

Tabla 96. Prueba P_12

Identificador: P_13	
Módulo	MAplicación
Elemento	Sistema de control empotrado.
Descripción	Comprobación de la ejecución del planificador cíclico.
Requisitos previos	1. Programación de las peticiones. 2. Definición de velocidad de conexión a 9600 baudios. 3. Programación de los accesos a los sensores para la recopilación de datos. 4. Programación de los accesos a los actuadores para la ejecución de acciones. 5. Programación del planificador cíclico de tareas simples. 6. Carga del programa “embeddedSystem.ino” a la placa Arduino UNO.
Acciones	1. Acceso al monitor serial. 2. Envío de diferentes peticiones y observación de la respuesta.
Resultado	Favorable

Tabla 97. Prueba P_13

Identificador: P_14	
Módulo	MAplicación
Elemento	Servidor de control remoto, sistema de control empotrado.
Descripción	Comprobación de la comunicación de ambos elementos mediante el puerto serie.
Requisitos previos	1. Especificación del puerto serie y definición de la velocidad de conexión a 9600 baudios en ambos extremos. 2. Uso de la interfaz de comunicación con Arduino proporcionada en el módulo del servidor remoto. 3. Conexión de Arduino UNO vía USB. 4. Programación de peticiones y respuestas en ambas partes.
Acciones	1. Carga del programa “embeddedSystem.ino” a la placa Arduino UNO. 2. Compilación y ejecución de “controlServer.c”.
Resultado	Favorable

Tabla 98. Prueba P_14

Identificador: P_15	
Módulo	MAplicación
Elemento	Servidor de control remoto, sistema de control empotrado.
Descripción	Comprobación del funcionamiento global de la aplicación.
Requisitos previos	1. Programación de la funcionalidad completa en ambos módulos 2. Conexión de Arduino UNO vía USB.
Acciones	1. Carga del programa “embeddedSystem.ino” a la placa Arduino UNO. 2. Compilación y ejecución de “controlServer.c”.
Resultado	Favorable

Tabla 99. Prueba P_15

6.2 Matriz de trazabilidad funcional

Mediante esta matriz de trazabilidad se comprueba que todos los requisitos software especificados en el capítulo de análisis se han implementado correctamente y cubren las necesidades del proyecto. Para comprobar la validación de los requisitos mediante las pruebas correspondientes realizadas, se marca la intersección entre ellos.

Todas las filas de la matriz deben tener al menos una intersección marcada, lo que significa que el requisito correspondiente a esa fila, ha sido cubierto por lo menos con una prueba funcional. En caso contrario, el requisito no se encontraría validado y sería necesario crear una prueba que lo validara.

	P_01	P_02	P_03	P_04	P_05	P_06	P_07	P_08	P_09	P_10	P_11	P_12	P_13	P_14	P_15
F_RS_01	X	X													
F_RS_02					X										
F_RS_03			X												
F_RS_04		X													
F_RS_05				X											
F_RS_06						X	X	X	X						
F_RS_07										X			X		
F_RS_08											X	X			
F_RS_09										X	X	X	X		
F_RS_10														X	
F_RS_11		X		X											
F_RS_12		X		X											
F_RS_13						X									
F_RS_14										X					
F_RS_15														X	X
F_RS_16											X	X			
F_RS_17											X	X			
F_RS_18									X				X		
F_RS_19						X	X	X	X						

Tabla 100. Matriz de trazabilidad NF_RS - P

Capítulo 7: Planificación

En este capítulo se detalla por un lado la planificación inicial del proyecto y por otro el tiempo realmente invertido.

Es complicado realizar una planificación en los proyectos como el estudiado en este documento debido a la naturaleza que les caracteriza. Aun así se muestra primero la planificación a priori y después el tiempo real que se ha invertido en el proyecto.

Para mejorar la visualización del tiempo de dedicación previsto y real en las diferentes actividades del ciclo de vida del proyecto, se hace uso de diagramas de Gantt.

El tiempo invertido en el proyecto ha sido aproximadamente de cuatro meses comprendidos en el periodo desde principios de Junio hasta mediados de Septiembre de 2017.

Las actividades correspondientes al ciclo de vida del proyecto se especifican a continuación:

- Propuesta del proyecto: Comprende la presentación del proyecto por parte del tutor y la posterior elección del mismo.
- Análisis: Consiste en el estudio de la solución tras la exposición de requisitos por parte del tutor. Incluye las siguientes tareas:
    - Documentación previa.
    - Definición de los requisitos.
    - Estudio y selección del sistema operativo de tiempo real.
    - Estudio y selección de la plataforma de hardware libre.
- Diseño: Incluye las siguientes tareas:
    - Diseño del proyecto en base a las tecnologías seleccionadas.
    - Adquisición de los medios técnicos necesarios para materializar el proyecto.
- Implementación: En la construcción del proyecto se diferencian 3 fases:
    - Instalación del sistema de tiempo real seleccionado.
    - Instalación del entorno de desarrollo de la plataforma open source.
    - Implementación de una práctica de tiempo real.
- Pruebas: Comprende el testeo del correcto funcionamiento del proyecto.
- Documentación: Generación de la memoria del proyecto.

A continuación se muestran los diagramas de Gantt correspondientes a la planificación del proyecto a priori y el tiempo real invertido en él. Tanto los fines de semana como la fiesta nacional del 15 de Agosto quedan excluidos de la planificación.

Actividad	Fecha de Inicio	Fecha de Fin	Duración
Propuesta del proyecto	05/06/2017	06/06/2017	2
Análisis	07/06/2017	30/06/2017	18
Documentación previa	07/06/2017	14/06/2017	6
Definición de los requisitos	14/06/2017	22/06/2017	6
Estudio y selección del sistema operativo de tiempo real	23/06/2017	27/06/2017	3
Estudio y selección de la plataforma de hardware libre	28/06/2017	30/06/2017	3
Diseño	03/07/2017	07/07/2017	5
Diseño del proyecto	03/07/2017	05/07/2017	3
Adquisición de medios	06/07/2017	07/07/2017	2
Implementación	10/07/2017	09/08/2017	23
Instalación del sistema de tiempo real seleccionado	10/07/2017	14/07/2017	5
Instalación del entorno de desarrollo de la plataforma open source	17/07/2017	21/07/2017	5
Implementación de una práctica de tiempo real	24/07/2017	09/08/2017	13
Pruebas	10/08/2017	14/08/2017	3
Documentación	16/08/2017	20/09/2017	26
TOTAL			**123**

Tabla 101. Planificación del proyecto a priori

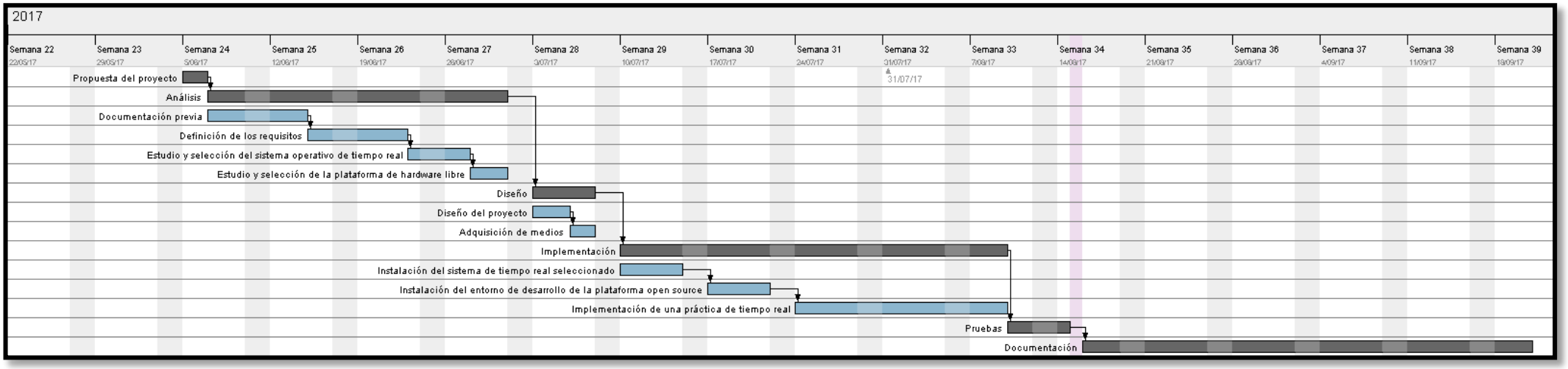


Ilustración 36. Diagrama de Gantt de la planificación a priori

Actividad	Fecha de Inicio	Fecha de Fin	Duración
Propuesta del proyecto	05/06/2017	06/06/2017	2
Análisis	07/06/2017	27/06/2017	15
Documentación previa	07/06/2017	12/06/2017	4
Definición de los requisitos	13/06/2017	16/06/2017	4
Estudio y selección del sistema operativo de tiempo real	19/06/2017	21/06/2017	3
Estudio y selección de la plataforma de hardware libre	22/06/2017	27/06/2017	4
Diseño	28/07/2017	05/07/2017	6
Diseño del proyecto	28/07/2017	29/07/2017	2
Adquisición de medios	30/07/2017	05/07/2017	4
Implementación	06/07/2017	11/08/2017	27
Instalación del sistema de tiempo real seleccionado	06/07/2017	14/07/2017	7
Instalación del entorno de desarrollo de la plataforma open source	17/07/2017	18/07/2017	2
Implementación de una práctica de tiempo real	19/07/2017	11/08/2017	18
Pruebas	14/08/2017	17/08/2017	4
Documentación	18/08/2017	22/09/2017	29
TOTAL			**131**

Tabla 102. Tiempo real invertido en el proyecto

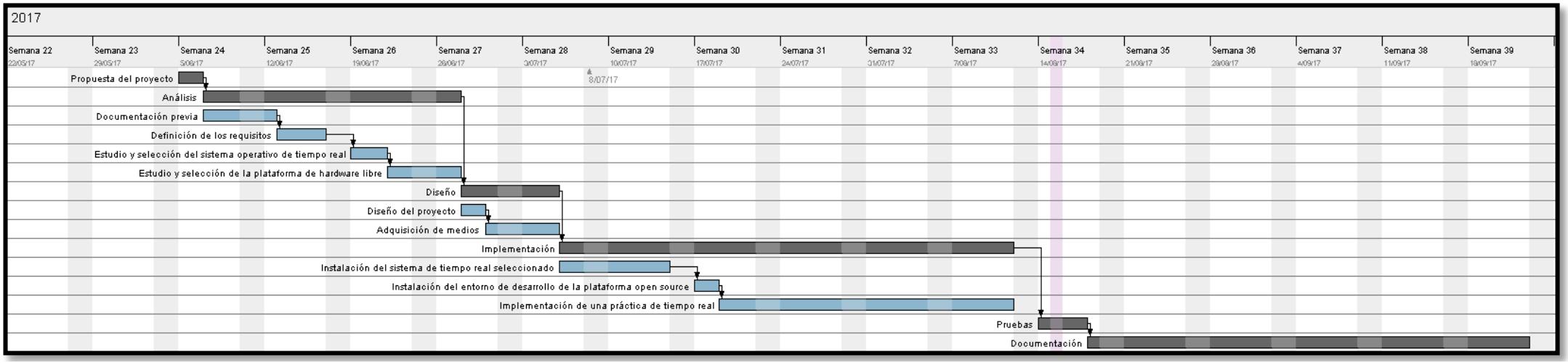


Ilustración 37. Diagrama de Gantt del tiempo real invertido en el proyecto

Como se puede observar en los diagramas, existe una diferencia entre la planificación inicial y el tiempo real invertido en el proyecto. Los principales motivos de esta diferencia se especifican a continuación:

- En la fase de análisis no fueron necesarios tantos días para la documentación en el ámbito del proyecto ni para la definición de requisitos ya que se indicaron de forma explícita por parte del tutor.
- El diseño de la práctica era muy claro, sin embargo para adquirir el material técnico necesario hubo que invertir más días que los previstos en un primer momento debido principalmente al mes de aquel momento (Julio) y el periodo correspondiente de vacaciones de muchos negocios.
- En cuanto a la implementación, se necesitaron más días de los provistos para la instalación del sistema operativo de tiempo real RTEMS debido a problemas técnicos encontrados. La implementación de la práctica de tiempo real también consumió más tiempo de lo previsto por el motivo principal de asegurar la calidad de la misma.

Capítulo 8: Marco regulador y entorno socio-económico

Se describe a continuación algunos aspectos fundamentales de la legislación, impactos socio-económicos, medioambientales y éticos y se explicita las aplicaciones prácticas del presente proyecto

8.1 Marco regulador

Hoy en día, en lo referido a la capacidad de generación de conocimientos intelectuales es fundamental conocer el marco legal los regula. Es de vital importancia definir la gestión de las herramientas legales disponibles para generalizar el conocimiento y garantizas los derechos de los usuarios.

En el ámbito del desarrollo software y hardware es necesario entender el marco regulador que regula la propiedad intelectual para ser conscientes de cómo podemos y debemos entregar los proyectos a los usuarios y qué derechos y obligaciones se deben cumplir al respecto.

En primer lugar, los derechos de autor son el utensilio legal básico para la protección de los proyectos intelectuales. Los derechos se regulan de modo que son únicamente para el creador original de la obra. Estos derechos son principalmente, los derechos de reconocimiento del autor y la divulgación de su proyecto, y los derechos de reproducción, modificación y distribución del mismo. En el caso de la creación de proyectos hardware o software los derechos de autor protegen la expresión del código fuente o los componentes hardware pero no los algoritmos o ideas que se encuentran en su interior.

La legislación de derechos de autor ha sido desarrollada en cada país con diferentes matices, pero, gracias a varios convenios internacionales se ha ido homogeneizando (destacan la Organización Mundial de la Propiedad Intelectual y la Organización Mundial del Comercio). Gracias a estos convenios, se garantiza al creador de un proyecto, los derechos de autor de su obra que son válidos en casi todo el mundo.

En cuanto a la cesión de los derechos de autor a terceros, el propietario del proyecto hace uso normalmente de un contrato formal. En él se determina como se produce la cesión de dichos derechos y en qué condiciones puede hacer de su uso. Este contrato se denomina licencia software. En el caso del software libre, y como ya se ha mencionado anteriormente, la licencia que determina los derechos y obligaciones para su uso es la *"General Public License"* (GLP).

Para proteger algoritmos, ideas o nuevos componentes hardware existen las patentes, las cuales conceden el monopolio de la invención al autor. El monopolio abarca la fabricación, distribución y comercialización. Al contrario que los derechos de autor que se conceden de forma gratuita, es necesario solicitar las patentes en la oficina correspondiente de cada país y abonar la cantidad que proceda. [36]

Existen otras formas de protección legal como el registro de una marca, pero en el caso del presente proyecto han quedado definidas las principales.

El objetivo de este proyecto es puramente académico. Únicamente tiene como fin un uso libre universitario para servir como herramienta de aprendizaje para el alumnado. Es de naturaleza open-source, permitiendo que otros desarrolladores, modifiquen o amplíen las funcionalidades ofrecidas.

En cuanto a la utilización de la solución del proyecto debe regularse mediante las mismas normas y obligaciones que cualquier otro material educativo perteneciente a la Universidad Carlos III de Madrid o a la educación en general.

Un aspecto fundamental en el desarrollo software es la garantía de los derechos de los datos de usuarios. La aplicación desarrollada en el presente proyecto no maneja la información de ningún dato personal, puesto que su único fin, como el de la gran mayoría de los sistemas de tiempo real es, en definitiva, automatizar o monitorizar un sistema para mejorar algún aspecto de la sociedad actual.

Debido a las consecuencias que pueden surgir a partir del fallo de un sistema de tiempo real, que incluso, pueden poner en peligro la integridad física de las personas, es esencial homologar y certificar exhaustivamente cualquier aplicación de este ámbito, asegurando de esta manera el correcto funcionamiento tanto de sus componentes por separado como el de su totalidad. Como ya se ha explicado, el fin del proyecto es educativo, pero si en algún momento se pretendiese hacer uso de la aplicación desarrollada en el entorno real, sería necesaria una certificación a cargo de la entidad de certificación correspondiente.

Finalmente es necesario mencionar que la implementación de aplicación de tiempo real ha sido basada en la norma *POSIX*, definida por la *IEE* (Institute of Electrical and Electronics Engineers). Esta norma define una interfaz estándar del sistema operativo cuyo objetivo principal es favorecer la portabilidad entre arquitecturas. [37]

8.2 Entorno socio-económico

Las tecnologías de la información y de la comunicación (TIC) se colocan en los últimos años dentro de los principales motores para el cambio de la economía, de la cultura y de la sociedad en general. Quizás sea su gran potencial para el manejo e intercambio de la información, el motivo de situar a estas tecnologías en los vértices del cambio socio-económico.

Cada vez, aparecen más problemas sin resolver que resaltan la urgente necesidad de generar constantemente, avances tecnológicos tanto del área hardware como software, y es que, el impacto socio-económico no es solo un impacto más dentro de un marco multi-criterio sino que, es precisamente la clave para valorar el éxito o fracaso de cualquier proyecto.

En esta situación en la que cada vez la ciencia intenta mejorar las condiciones de vida intra e interpersonales, surgieron dentro de las TIC, los sistemas de tiempo real, interactuando con el entorno para responder a una serie de estímulos dentro de un plazo limitado. A nivel socio laboral las ventajas de los sistemas de tiempo real son múltiples, desde la disminución del esfuerzo físico en la producción industrial, hasta la disminución de variables psicológicas como el esfuerzo atencional de un controlador aéreo. En cuanto al impacto económico, las ventajas de los sistemas de tiempo real no se quedan atrás. Pese a los costes de fabricación y certificación, la efectividad de este tipo de sistemas logra conseguir los efectos deseados en el menor tiempo posible y con la menor cantidad de recursos personales. En referencia al impacto medioambiental se debe tener en cuenta que el ámbito en que se aplican los diferentes STR es diverso, pudiendo interactuar de forma positiva o negativa con el ambiente. Por ejemplo, un sistema automático de fumigación agrícola, interactúa negativamente contaminando el ambiente pero, esta desventaja no es mayor que la que se produciría de forma tradicional. Por otro lado, si nos centramos en aplicaciones destinadas a la seguridad en los vehículos, no es solo que la interacción con el medio ambiente sea casi nula si no que las ventajas son tales como mantener la integridad física de los usuarios.

Visualizando el aspecto ético de la introducción de los STR en la sociedad, surge la incertidumbre del límite que se establece entre lo que debe ser responsabilidad de una máquina o del hombre, miedo a la reducción de puestos de trabajo, rechazo a la deshumanización de los procesos o su uso irresponsable, son algunos de los debates que aun surgen en torno a este tipo de sistemas.

En concreto, el proyecto que aquí se desarrolla, podría generar aplicaciones reales para diversos ámbitos (ocio, seguridad, transportes, procesos industriales, domótica…) suponiendo las ventajas tanto socio-laborales, medioambientales o personales previamente descritas. Además como aspecto reseñable, el proyecto tiene un gran impacto socio-educativo, pues supone una manera eficaz para el aprendizaje del desarrollo de sistemas de tiempo real, siendo el coste económico asequible para el tipo de usuario al que está destinado.

Capítulo 9: Presupuesto

En esta parte del documento se detalla una estimación del presupuesto que se destinará a la realización del proyecto desglosado en diferentes categorías *(Los costes parciales se especifican sin I.V.A, el cual es aplicado en los costes totales del proyecto).*

9.1 Costes de personal

El proyecto ha sido realizado por un solo ingeniero informático que ha adoptado diferentes roles a lo largo de las fases del mismo. Concretamente ha asumido los roles de Analista, Diseñador, Programador (incluye la realización de las actividades de implementación y pruebas) y Redactor. A continuación se detallan las horas invertidas en la realización de cada una de las actividades del proyecto teniendo presente el diagrama de Gantt del apartado anterior.

Análisis	*Diseño*	*Implementación*	*Pruebas*	*Documentación*
15 días	6 días	27 días	4 días	29 días

Tabla 103. Días invertidos en cada fase del proyecto

Actividad	Rol	Dedicación (Horas)	Coste Hombre/Hora (€)	Coste (€)
Análisis	Analista	15	35	525
Diseño	Diseñador	6	32	192
Implementación	Programador	27	37	999
Pruebas	Programador	4	37	148
Documentación	Redactor	29	25	725
			TOTAL	**2.589**

Tabla 104. Desglose del coste personal del proyecto

Resultando un coste de personal de ***Dos mil quinientos ochenta y nueve Euros***.

9.2 Costes hardware

A continuación se calculan los costes de los medios técnicos hardware necesarios para el desarrollo del proyecto (incluyendo la aplicación de tiempo real).

El coste imputable de cada dispositivo se calcula a partir de la fórmula de amortización:

$$\frac{Dedicación}{Periodo\ de\ depreciación} * Coste * \frac{Uso\ dedicado\ al\ proyecto}{100}$$

Concepto	Coste	% Uso dedicado al proyecto	Dedicación (meses)	Periodo de depreciación (meses)	Coste imputable
Lenovo IdeaPad 710S Plus-13IKB Intel Core i5-7200U (3.1GHz), 8GB RAM, 256GB Disco Duro SSD	905,59	40	4	60	24,14
Arduino UNO	19,95	60	4	24	1,99
Protoboard	8	100	4	12	2,66
Pantalla LCD 16x2	15,50	100	4	12	5,16
Buzzer	1,95	100	4	12	0,65
Botón	0,50	100	4	12	0,16
Resistencias	0,56	100	4	-	0,56
Potenciómetro	1,40	100	4	12	0,46
Sensor Infrarrojo IR FC-51	4,25	100	4	12	1,41
Sensor Gas MQ-2	5,90	100	4	12	1,96
Sensor Llama	4,20	100	4	12	1,4
Sensor Temperatura LM35	2,55	100	4	12	0,85
Detector de apertura magnético	4,00	100	4	12	1,33
Altavoz 0.5W	2,90	100	4	12	0,96
Ventilador 5V	2,65	100	4	12	0,88
Leds	1,95	100	4	-	1,95
				TOTAL	**46,52**

Tabla 105. Desglose del coste hardware del proyecto

Resultando un coste hardware de **Cuarenta y seis euros con cincuenta y dos céntimos.**

9.3 Costes software

Todo el software utilizado ha sido libre y gratuito, por lo tanto no existen costes asociados.

9.4 Resumen de costes

Una vez desglosados todos los costes asociados al proyecto, se presenta a continuación el coste global del mismo sumando los costes de personal, de hardware y de software. Además se añade un 10% de costes indirectos asociados a los riesgos del proyecto.

Concepto	Coste Total (€)
Costes de personal	2.589
Costes hardware	46,52
Costes software	0
Subtotal	2.635,52
Riesgo (10%)	263,55
Total sin I.V.A	2.899,07
Total con I.V.A (21%)	**3.507,87**

Tabla 106. Resumen de costes del proyecto

El coste total del proyecto antes de la aplicación de impuestos, resulta de ***DOS MIL OCHOCIENTOS NOVENTA Y NUEVE EUROS CON SIETE CÉNTIMOS***, y una vez aplicado el I.V.A correspondiente, la cantidad asciende a ***TRES MIL QUINIENTOS SIETE EUROS CON OCHENTA Y SIETE CÉNTIMOS***.

Capítulo 10: Conclusiones

El último capítulo del proyecto se destina a exponer las consecuencias tanto personales como relativas al proyecto que se deducen de su realización.

10.1 Conclusiones del proyecto y líneas futuras

Teniendo en cuenta los objetivos iniciales de este proyecto de fin de grado especificados en el capítulo de Análisis, se puede concluir que:

- Se ha diseñado e integrado un entorno de trabajo que incluye el sistema operativo de tiempo real RTEMS para el desarrollo de aplicaciones de tiempo real. Para ello ha sido necesario:
    - Realizar un estudio de las plataformas tanto hardware como software más adecuadas en el desarrollo de aplicaciones de tiempo real, seleccionando Arduino.
    - Realizar un estudio del entorno de desarrollo más adecuado en el que incorporar las funcionalidades de RTEMS para facilitar la implementación de aplicaciones basadas en este sistema operativo, siendo Eclipse CDT versión Galileo seleccionado.
    - Realizar un estudio de la interfaz de programación más adecuada, siendo el estándar POSIX seleccionado.
    - Integrar todas las herramientas en una máquina virtual Ubuntu versión 14.04 LTS de 64 bits.
    - Realizar una extensa documentación sobre todas las tecnologías.
- Se ha diseñado e implementado una aplicación de tiempo real utilizando el framework previamente integrado que constituya una de las prácticas de la asignatura, proporcionando al tutor de la misma la capacidad de evaluar a los estudiantes mediante:
    - La elaboración del enunciado para una práctica específica.
    - La delimitación de los criterios que serán aplicados en la corrección de la misma.

De esta manera, y con todos los objetivos iniciales cumplidos, se puede asegurar que el proyecto se ha desarrollado de manera satisfactoria. Sin embargo, tras la finalización del proyecto se pueden entrever diferentes posibilidades en su desarrollo. De esta forma, se pueden destacar como líneas futuras de trabajo:

- Utilización de otro sistema operativo de tiempo real para la implementación del módulo servidor remoto como puede ser *Real Time Linux*.

- Utilización de otra plataforma hardware para el módulo de control empotrado que ofrezca mayor potencial, por ejemplo *Raspberry Pi* o *BeagleBoard*.
- Desarrollo de aplicaciones más complejas y de diferente temática.

10.2 Conclusiones personales

En primer lugar me gustaría destacar que el desarrollo de este proyecto supone un trabajo muy completo. No solo consiste en desarrollar una aplicación y manejar componentes electrónicos, sino que es necesario en un primer lugar formar un espacio de trabajo completo para su posible desarrollo. Además, en la integración del entorno tuve que documentarme en profundidad de todas las tecnologías candidatas con el objetivo de analizar sus características y realizar una selección. Una vez seleccionadas las mismas, de nuevo tuve que instruirme para realizar la instalación o compilación correspondiente.

En el proceso de documentación, fui consciente del potencial que ofrecen las plataformas hardware y software open source. En el caso de las plataformas open hardware, aparte de que su adquisición no supone un coste para nada elevado en comparación con las funcionalidades que ofrecen, generan la posibilidad de formar prototipos muy específicos para el objetivo deseado mediante la adición de diferentes módulos de componentes. Por ejemplo, en el caso de Arduino, módulos tan diferentes como lector de huellas digital, cámara VGA, Joystick o adaptador de tarjeta SIM que permiten la realización de miles de proyectos.

Arduino es una plataforma magnífica para iniciarse en la programación de microcontroladores debido a la sencillez de su entorno de desarrollo, la amplia colección de librerías disponibles y la extensa documentación existente en Internet. Como inconveniente es necesario recalcar que la mayoría de sus microcontroladores no son demasiado potentes y tendrían dificultades en proyectos más complejos. Por estas razones, la realización del presente proyecto me ha motivado en la utilización de otras plataformas open hardware como Raspberry Pi en busca del diseño de proyectos más complejos.

También me gustaría destacar la importancia que adquieren los sistemas de tiempo real en nuestra sociedad a la hora de mejorar la calidad de vida. Gracias a la evolución de la tecnología cada vez están más presentes en nuestro entorno. El control de tráfico aéreo, de sondas espaciales, sistemas de navegación autónoma o sistemas de defensa militar son algunas de las aplicaciones de los sistemas de tiempo real. Es necesario recalcar las consecuencias que podría conllevar un mal funcionamiento de los

mismos, tanto materiales como personales, siendo de vital importancia una certificación exhaustiva de las aplicaciones.

En definitiva, este proyecto significa para mí, una gran ampliación de conocimientos, una mayor familiarización con plataformas hardware y software, un reto a nivel personal y profesional y sobre todo, un primer paso para seguir formándome en los sistemas empotrados.

ANEXO A: Abstract

Introduction

The purpose of this project is to create a working environment which includes necessary tools for the development of real-time applications and which allows to generate a specific application that forms one practice of the “Real Time Systems” subject at the University.

More specifically, the RTEMS real-time operating system will be used in this project for the implementation of a remote control server as a module. The Arduino platform will be used for the development of the module corresponding to a control system embedded in a specific device. Both modules communicate to each other in order to achieve the final goal and conform the project the future students should implement.

It also provides the necessary documentation for the creation of a working environment, as well as proposing a possible project definition and its evaluation criterion.

Goals

The general objective of this end-of-degree project is the design and implementation of a module for practices based on the RTEMS real-time operating system for the “Real Time Systems” subject.

This module should include:

- The design and integration of a development environment or framework that includes the RTEMS real-time operating system for real-time applications development.
- The design and implementation of a real system application using the mentioned framework with the aim of being one of the practices of the subject.

Specifically, for the integration of the framework it is necessary to:

- Conduct a study to choose the most appropriate hardware/software platform for real-time applications development.
- Conduct a study to choose the most appropriate development environment to incorporate RTEMS functionalities in order to facilitate the development of RTEMS-based applications.
- Conduct a study to choose the programming interface to be used.
- Integrate all the tools in a virtual machine, thus conforming the framework.
- Provide the necessary documentation for the generation of the development environment.

Once the framework is available, an application that serves the students as a practice of the subject will be implemented. In addition, the tutor of the subject should be able to evaluate the students. To achieve this objective it will be necessary to:

- Design the definition of the practice/project.
- Specify the criteria that will be applied in the project evaluation.

Art State

An embedded system is a computer system that is integrated in the device that supervises and controls with the goal of cover specific needs. It is made up of both hardware and software and usually, as a whole, is part of a larger and more complex system. In terms of programming, there are different modalities: processor-specific assembler programming or by implementing programs in programming languages such as Java, C or C ++ hosted on real-time operating systems.

The basic features of an embedded systems are:

- Limited functions to execute: Due to the specific aim.
- Limited computing performance and memory: Simple task execution.
- Reliability and safety: If the system is critical, failures could involve serious consequences such as human death or environmental disaster.
- Low power consumption: Most systems are placed in sites without energy, so it is necessary to incorporate batteries.
- Low production cost: The amount of production is very high, which entails a cost reduction.
- Software and device development take place at the same time.

Embedded systems' core is composed of the central processing unit (CPU), which is in charge of controlling the specific functions for which it is programmed. As a result of the wide range of embedded applications there is also a large variety of processors used with a range that extends from low performance to greater complexity and use.

The software that runs on the processor needs some type of memory to be stored.

Since the purpose of an embedded system is to control a system, it is intrinsically constrained to interact with the surrounding environment. This interaction is produced by an input and output module that enables communication with sensors (responsible for collecting the data from the environment and delivering it to the processor) and actuators (whose mission is to perform the actions that the CPU orders after the data processing).

Following, the main components of an embedded system are summarized:

- Central processing unit (CPU): There are two different types: microprocessor and microcontroller. Both types include the same components: arithmetic and logic unit (ALU), registers, control unit (CU), process unit (PU), input/output module and memory. The main difference between them is that microcontrollers contain all the architecture inside them, unlike microprocessors which include only control unit and process unit.

- Timer: A clock module is required to generate the clock pulses, necessary for the digital circuit that has to be provided with a time reference. These signals are generated by a single main oscillator, whose characteristics are relevant to the application for which the system is intended. The main characteristics to consider of an oscillator are the frequency, the precision and its maintenance when there are external factors that can affect and the electrical consumption.

- Communication is a major aspect in embedded systems. Nowadays and thanks to the boom of the wireless communications, not only the communication via cable but also the wireless communication are available. The mayor interfaces are: Human Machine Interface (HMI), serial communication interface such as Serial Peripheral Interface Bus (SPI) and RS-232.

- Input/Output module: Embedded systems have the requirement to connect with specialized hardware of the processor they integrate. This connection is made through the Input and Output module, responsible for receiving or sending analogue and digital signals to or from the specific hardware (sensors, actuators, etc.). The I/O module consists of a series of ports or nodes that contain a certain number of registers for the temporary storage of the data. Principal ports are: serial ports, parallel ports, universal ports, and wireless ports.

- Power module: To feed all components of the embedded system it is necessary to generate voltage. For some specific battery-powered applications, the energy consumption is usually decisive, since the life of the system is limited by it.

Open Hardware/Software

The term open hardware/software refers to the hardware or software whose design is published so that anyone has the possibility of use, study, modification, improvement and even redistribution and sale.

The three main requirements of open hardware are open interface, open design and open implementation. Applying the same ideas of open software to open hardware is complicated since there are some substantial differences.

Based on the nature of the open hardware, there are two possible classifications: static hardware and reconfigurable hardware. The first concerns the main design components such as schematic circuit, printed circuit, design details and documentation. The totality of them constitute the final physical circuit. Software does not have this physical property and as a consequence, it entails a series of disadvantages: singular physical design and manufacturing cost. On the contrary, reconfigurable hardware could be modified by some Hardware Description Language type (HDL).

In conclusion, having a free hardware platform provides a great advantage to developers since they do not have the obligation to invest in high cost products and because they have the necessary autonomy for the improvement of applications.

In this project, three free hardware platforms are studied: Arduino, Raspberry Pi and BeagleBoard.

Arduino hardware consists of small, programmable, microcontroller-based boards and the software is made up of its own integrated development environment (IDE), Arduino IDE. Most of their microcontrollers are from the Atmel AVR family, 8-bits and RISC architecture based. Arduino offers some advantages such as low price, cross compiling and simple programming environment.

Raspberry Pi is a Single Board Computer which runs a Debian-based Linux distribution. All models feature a Broadcom system on a chip (SoC), which includes an ARM microprocessor 32/64-bits based.

BeaggleBoard has similar features to Raspberry but with a very educative approach.

Real Time Systems

Real time systems (RTS) are a specific kind of embedded systems that interact with a dynamic environment, that is, that evolves over time. The responses of these systems to the received stimuli must be executed within a set time interval to ensure

proper functioning. RTS implement their functionality by programming and executing tasks.

For the execution of multiple tasks in the same processor, it is necessary a distribution of resources between them. In addition, it is necessary to ensure that the set of tasks will meet the temporary requirements required by the application. A scheduling algorithm orders the tasks set execution according to pre-defined criteria. Cyclic scheduling algorithm assumes a deterministic and predictable execution. Priority scheduling algorithm selects the activity to execute based on its priority.

Real-time firmware generally consists of different components: an operating system (OS), a run-time system and an application. The run-time is usually related to a specific programming language. The main run-time environments studied in this project are Ada, Java and POSIX.

Ada is a structured, imperative object-oriented programming language extended from Pascal. It is typically used to design reliable, real-time, embedded and generally large-scale systems.

Java is an object-oriented programming language whose goal is to limit implementation dependencies. Its main feature is that Java applications can run on any Java virtual machine (JVM) regardless of the computer architecture.

POSIX (Portable Operating System Interface) is a standard that defines the application programming interface (API), command line interpreter and useful development tools.

A real-time operating system (RTOS) is an operating system that offers a number of specific qualities and characteristics to support real-time applications. The characteristics that a real-time system must guarantee are: time management, multiprogramming, task synchronization, enough priority levels and memory management. RTOS studied in this project are Windows CE, Real Time Linux, VxWorks and RTEMS.

Windows CE (officially known as Windows Embedded Compact) is a real-time operating system designed by Microsoft, modular and portable for mobile devices, multimedia and video consoles or television with reduced memory.

Real Time Linux is a real-time operating system based on Linux. This is a modification of the Linux kernel for real-time systems. It is currently distributed by Wind River Systems under Wind Real-Time Core name for the Wind River Linux version.

VxWorks is a real-time UNIX-based system created and distributed by Wind River Systems. It is one of the most widespread real-time systems currently used in multiple robotic, aerospace and military applications.

RTEMS (Real-Time Executive for Multiprocessor Systems) is a real-time operating system for multiprocessor and open source systems that allows both standard open programming interfaces and POSIX. The applications implemented with RTEMS are of different scope, from those related to industrial production to military and aerospace used by NASA (National Aeronautics and Space Administration).

Analysis & Development

After an introduction to real-time systems and the study of main hardware and software platforms and real time operating systems, this section defines project requirements and use cases. They specify the main functionalities and constraints the final product must incorporate to cover the client´s necessity. Requirements must be quantifiable, relevant and detailed. Use cases graphically represent the requirements by representing the actions that the user can perform with the project solution. Both the requirements and the use cases are represented by tables to facilitate their understanding.

There are two different kinds of project requirements: user requirements and system requirements. User requirements specify the main functionalities and constraints the final product must incorporate and system requirements specify in more detail the user requirements.

After the analysis of the user's needs, a total of 21 user requirements were obtained. 9 use cases represent them graphically. As a consequence, 29 system requirements have been obtained through the final requirements' analysis.

After defining and analysing both user and system requirements, it is necessary to ensure that all user requirements are covered by at least one system requirement. Traceability matrices are the element that makes this verification possible. It is a matrix containing the identifiers of user requirements in its rows and identifiers of system requirements in its columns. In this way, an intersection between two requirements is marked when one user requirement in particular is covered by the corresponding system requirement.

In the case of this project two traceability matrices are created, the first one faces capacity user requirements and functional system requirements and the second one faces constraint user requirements and non-functional system requirements. In

conclusion, it is possible to ensure that all user requirements are contemplated at least by one system requirement, so all project needs are well covered.

Next, a study of the design of the project is made based on the hardware/software technologies analysed. It is necessary to emphasize that the project consists of two clearly differentiable parts:

- Design and integration of a framework to develop real-time applications. As stated in the requirements, it is necessary to generate a complete development environment for real-time applications that mainly includes a real-time operating system and a hardware/software platform that allow such development.
- Design and implementation of a real-time application. Once the development framework is created, a real-time embedded application that controls a specific system must be programmed.

The creation of the development framework is based on the integration of existing hardware/software technologies. First, the most suitable real-time operating systems comparison is made. As a result, RTEMS is the chosen one to be part of the framework, mainly for being free, open source and allowing POSIX programming. Because of this selection and plug in existence, it is decided to incorporate Eclipse Galileo, whose main objective is to facilitate RTEMS programming and compilation of real-time applications. Using the same comparison for the hardware/software platform, the chosen one is Arduino for being open source, and the specific board model to use is Arduino UNO.

Once the entire development framework is integrated, there is possible to implement real-time applications based on RTEMS and the Arduino platform. The application is a controlling system or device embedded real-time application with a specific purpose. It consists of two real-time systems: an embedded electronic control system in the device, and a remote control system installed on a stand-alone server. The application follows the distributed architecture of the Client-Server model, whose control server sends requests to the embedded system to collect data from the device it controls, process them and generate necessary actions to reach its goal.

The remote control system will be programmed through C (POSIX) in the development environment for Eclipse CDT with the RTEMS plugin. The embedded control system will be implemented on the Arduino platform, both software (Arduino IDE) and hardware (Arduino UNO). It also specifies microcontroller connections with each sensor and actuator it is connected to.

The RTEMS compilation involves some previous installation processes. First of all, the installation of Source Builder, a tool to aid building packages from source, used by the RTEMS project. Then, Build Set must be installed (i386) for the architecture. It describes a collection of packages defining a set of tools to be used when developing a software for RTEMS. Finally, the RTEMS kernel can be installed for a particular Board Support Package (pc386) which contains drivers to manage the hardware. Also, a RTEMS support plugin in Eclipse CDT and the Arduino IDE are normally installed.

Now, the application development is explained. The remote control system is programmed through C (*"controlServer.c"*). First, it is necessary to configure the POSIX API. The configuration consists of adding the necessary application parameters to be developed, such as maximum number of threads, mutexes or signals. The tasks are programmed by methods, and a simple tasks cyclic scheduler is responsible for executing them at the precise moment. The embedded control system is programmed by Arduino sketch ("*embeddedSystem.ino"*).

Results

The embedded control system module is compiled and uploaded to the Arduino UNO board. In the case of the remote server it is necessary:

- If the program compilation is successful, an executable file (.exe extension) will be generated in the workspace.
- To run the application it is necessary to generate a complete RTEMS system image (.img extension) from executable file. A bash script that automates the process is provided.
- For convenience, the emulator Qemu is used for server simulation.

Once the entire application is implemented, a test set is performed to verify the operation of both development framework and application. All the tests described are functionality tests and verify the different system's capabilities.

By a traceability matrix it is verified that all the software requirements specified in the analysis chapter have been correctly implemented and cover the project needs. All array rows must have at least one marked intersection, which means the requirement has been covered with at least one functional test.

Conclusions

Considering the initial objectives of this project, it can be concluded that:

- A working environment that includes RTEMS real-time operating system for the real-time applications' development has been designed and integrated. For this it has been necessary to:
    - Conduct a study of hardware and software platforms for real-time applications development, selecting Arduino.
    - Conduct a study of development environment in which to incorporate RTEMS functionalities to facilitate applications' implementation based on this operating system, being Eclipse CDT Galileo selected.
    - Study the most appropriate programming interface, with the POSIX standard selected.
    - Integrate all the tools in a 64 bits virtual machine Ubuntu version 14.04 LTS.
- A real-time application has been designed and implemented using the previously integrated framework that constitutes one of the practices/projects in the subject, providing the tutor with the capacity to evaluate students through:
    - The elaboration of the specific practice's definition.
    - The delimitation of the criteria that will be applied in the correction.

With all the initial objectives fulfilled, it can be assured that the project has been developed in a satisfactory way. However, after the completion of the project different possibilities may arise in its development. In this way, they can be highlighted as future work lines:

- To use another real-time operating system for the implementation of the remote server module, such as Real Time Linux.
- To use another hardware platform for the embedded control module that offers a greater potential, for example Raspberry Pi or BeagleBoard.
- More complex applications development.

Personally, I would like to emphasize that the development of this project meant a complete and multidisciplinary work. It does not only consist of developing an

application, but also it is necessary to create a complete workspace for its possible development in the first place. For integrating the environment, I had to research and document in depth all the candidate technologies in order to analyse their characteristics and make a selection. Once they have been selected, again I was instructed to perform the corresponding installation and compilation.

Ultimately, this project means to me a great extension of knowledge, a greater familiarization with hardware and software platforms, a personal and professional challenge and above all, a first step to continue building my knowledge and skills in embedded systems.

ANEXO B: Detalles de instalación de RTEMS

Máquina Virtual

Para la instalación, se utilizó una máquina virtual con las siguientes especificaciones:

Sistema Operativo	Ubuntu 14.04 LTS de 64 bits
Procesador	Intel® Core™ i5-4690 CPU @ 3.50GHz × 2
Memoria	4 Gb
Disco duro	60 Gb

Tabla 107. Características de la máquina virtual utilizada

La cantidad de memoria que requiere la instalación según la documentación oficial es la mostrada en la tabla:

Componente	*Memoria requerida*
Directorio de archivos	40 Mb
Herramientas descomprimidas	200 Mb
Cada directorio de compilación	Hasta 500 Mb
Cada directorio de instalación	Entre 20 y 200 Mb

Tabla 108. Requisitos de memoria para la instalación y compilación de RTEMS

Una vez instaladas todas las herramientas es necesario compilar e instalar RTEMS para un Board Support Package específico, aproximadamente 50 Mb adicionales.

RTEMS Source Builder

En primer lugar es necesario instalar el Source Builder. Se trata de una herramienta que permite toda la instalación del entorno de desarrollo de RTEMS. La instalación almacenará en un directorio de archivos toda la estructura el entorno de desarrollo RTEMS. El directorio padre del que colgarán todos los archivos necesarios del entorno de desarrollo será *$HOME/development/rtems.* Para que el sistema operativo encuentre los programas RTEMS una vez generados ello hay que definir un "prefix" que se define como el directorio en el que se almacenarán los ejecutables. Es necesario indicar la ruta al sistema operativo y para ello hay que modificar el archivo *etc/profile* y añadir la siguiente línea al final del mismo (el cambio es permanente):

```
$ export PATH=$HOME/development/rtems/4.12/bin:$PATH
```

Si no se dispone de permisos para modificar el archivo simplemente se ejecuta el mismo comando en un terminal. Será necesario ejecutar el comando cada vez que se inicie sesión puesto que de esta manera el cambio no es permanente.

A continuación se deben instalar algunas herramientas necesarias para la compilación del Source Builder ejecutando el siguiente comando:

```
$ sudo apt-get build-dep binutils gcc g++ gdb unzip git python2.7-dev
```

Si surgen problemas, instalar las herramientas una por una. A continuación hay que crear el directorio e instalar el Source Builder:

```
$ mkdir -p development/rtems/src

$ cd development/rtems/src

$ git clone git://git.rtems.org/rtems-source-builder.git

$ cd rtems-source-builder
```

Una vez instalado el mismo, se comprueban los requisitos necesarios para compilar RTEMS:

```
$ source-builder/sb-check
```

Si la instalación ha sido correcta, la salida debe ser:

```
RTEMS Source Builder - Check, v4.12.0
Environment is ok
```

Build Set

A continuación hay que instalar el conjunto de herramientas de compilación (build set) específico para la arquitectura del procesador a utilizar. En este caso el procesador utilizado es el i386. Para ver los procesadores soportados se ejecuta el siguiente comando.

```
$ cd rtems

$ ../source-builder/sb-set-builder --list-bsets
```

Una vez comprobado el soporte para el i386 se instalan las herramientas:

```
$ ../source-builder/sb-set-builder --log=l-i386.txt --prefix=$HOME/development/rtems/4.12 4.12/rtems-i386
```

Una vez ejecutado el comando, se procede a la instalación y compilación del conjunto de herramientas. Este proceso puede durar de 10 a 20 minutos aproximadamente. El flag *--prefix* indica dónde serán instaladas. Se produce un log.txt del proceso de compilación.

Una vez termine el proceso, comprobamos la instalación:

```
$ ls $HOME/development/rtems/4.12
```

La salida debe ser:

```
bin          include      lib          libexec      share
i386-rtems4.12
```

Compilación Kernel RTEMS

A continuación se descarga el Kernel de RTEMS y se ejecuta un script de configuración:

```
$ cd development/rtems

$ mkdir kernel

$ cd kernel

$ git clone git://git.rtems.org/rtems.git rtems

$ cd rtems

$./bootstrap -c && ./bootstrap -p &&
$HOME/development/rtems/src/rtems-source-builder/source-
builder/sb-bootstrap
```

Board Support Package

Ahora es necesario compilar el Board Support Package (BSP) para el procesador elegido (i386). Se habilita la interfaz POSIX y la depuración:

```
$ cd ..

$ mkdir pc386

$ cd pc386

$    $HOME/development/rtems/kernel/rtems/configure    --
prefix=$HOME/development/rtems/4.12 --target=i386-rtems4.12
--enable-rtemsbsp=pc386 --enable-posix --disable-networking
--enable-rtems-debug
```

Una vez terminado el proceso se compila RTEMS utilizando dos procesadores para reducir el tiempo:

```
$ make -j 2
```

```
$ make install
```

Para la instalación del Source Builder se siguieron los pasos descritos en un tutorial realizado por el principal desarrollador de RTEMS. [33]

Para la compilación del Kernel y el Board Support Package se utilizó el manual de usuario versión 4.11.99 (master). [34]

Toda la documentación de RTEMS actualizada se encuentra en el apartado de documentación de su web oficial. [35]

ANEXO C: Instalación del Plug-in RTEMS en Eclipse

Existe un *Plug-in* de RTEMS para Eclipse que hereda las características y herramientas de desarrollo CDT (C/C++ Development Tooling).

La instalación del *Plug-in* se ha realizado en la versión de Eclipse Galileo IDE for C/C++ Developers de 64 bits.

Instalación del Plug-in RTEMS

En la pestaña "*Help"* hay que seleccionar "*Install" New Software.*

Hay que indicar la siguiente URL en la etiqueta "*Work With":*

ftp://ftp.rtems.org/pub/rtems/eclipse/updates/

Se despliega el árbol especificado con "*ALL"* y se selecciona el check "*RTEMS CDT Support".*

Configuración

Una vez realizada la instalación del *Plug-in* es necesario especificar unas preferencias de RTEMS:

En la pestaña "*Window"* hay que seleccionar "*Preferences":*

En "*Base path"* hay que especificar la ruta en la que están instaladas el conjunto de herramientas (*Build Set*) de RTEMS. En "*BSP path"* hay que especificar la ruta en la que se encuentra instalado el *Board Support Package* de RTEMS (donde se encuentra el fichero *Makefile.inc*):

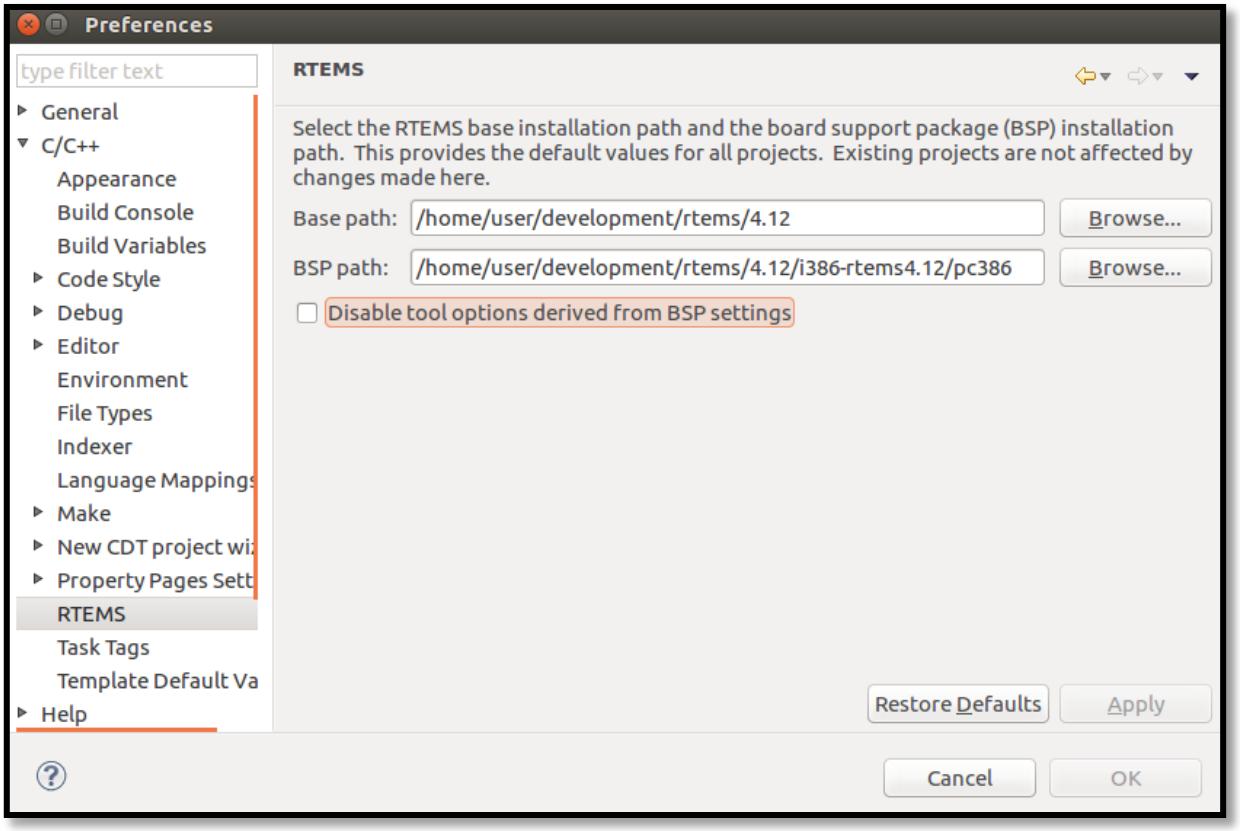


Ilustración 38. Configuración del Plugin RTEMS en Eclipse

Una vez aplicados los cambios ya es posible crear un proyecto RTEMS:

File → New → C Project → RTEMS Executable

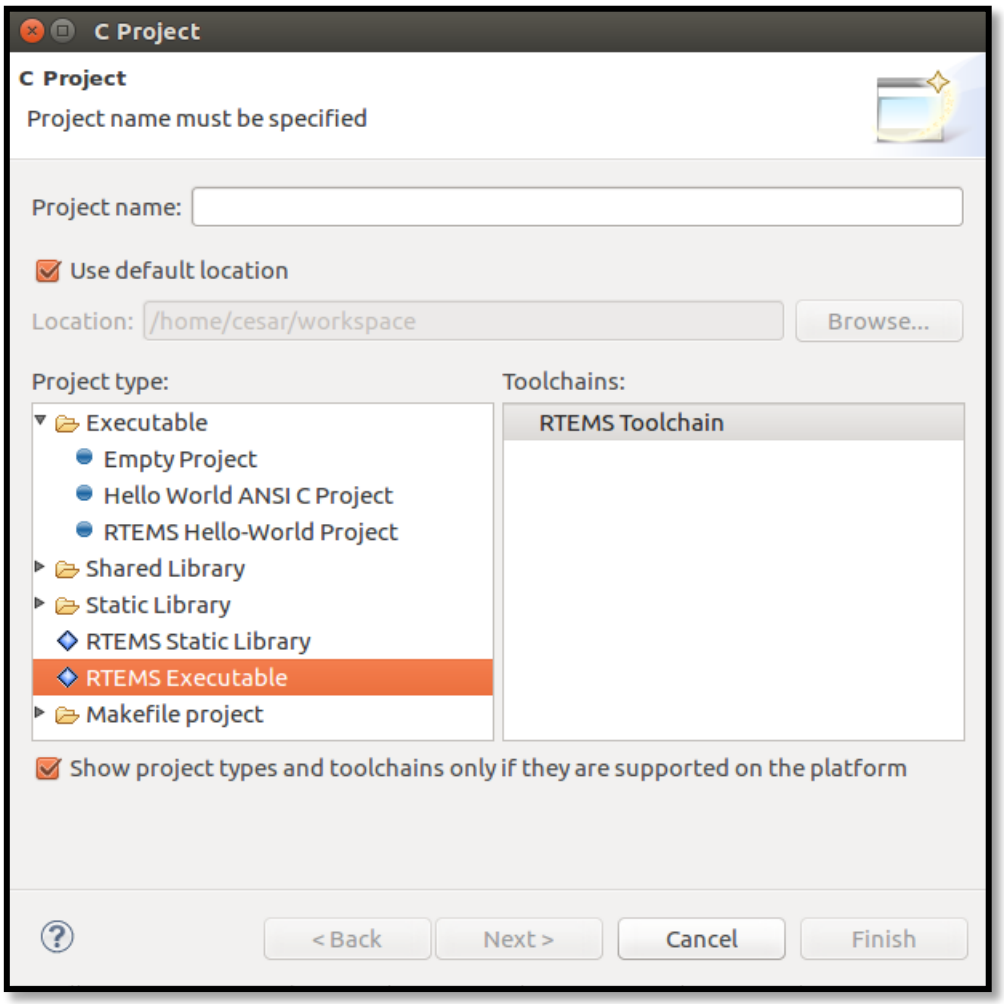


Ilustración 39. Creación de un proyecto RTEMS en Eclipse

Después de la creación del proyecto hay que incluir los directorios que contienen los archivos de cabeceras:

Botón derecho sobre el proyecto → C/C++ General → Paths and Symbols → Includes

Se deben incluir los siguientes directorios:

/home/user/development/rtems/4.12/i386-rtems4.12/pc386/lib/include

/home/user/development/rtems/4.12/lib/gcc/i386-rtems4.12/6.3.0/include

/home/user/development/rtems/4.12/lib/gcc/i386-rtems4.12/6.3.0/include-fixed

/home/user/development/rtems/4.12/i386-rtems4.12/include

Anexo D: Compilación y ejecución de una aplicación RTEMS

En este proyecto, para poder compilar un programa RTEMS y ejecutarlo es necesario incluir dos herramientas:

- Kpartx: Para poder generar una imagen del sistema a partir del ejecutable compilado por RTEMS. Para su instalación es necesario ejecutar el siguiente comando:

```
$ sudo apt-get install kpartx
```

- Qemu: Es un emulador necesario para poder simular la imagen del sistema y ejecutar el programa. Para su instalación es necesario ejecutar el siguiente comando:

```
$ sudo apt-get install qemu
```

La compilación del programa se realiza en Eclipse tras su implementación. Si la compilación es satisfactoria, se generará un archivo ejecutable (*.exe*) en la carpeta *RTEMS Executable Configuration* dentro del espacio de trabajo. Para compilar el programa:

Botón derecho sobre el proyecto → Build Configurations → Build → Select…

Se proporciona un *bash script* que genera la imagen del sistema a partir del ejecutable (El ejecutable debe ser copiado en la dirección que se especifique dentro del *script*). El comando necesario para la creación de la imagen es el siguiente:

```
$ sudo ./create_imagen.sh <nombreImagen>
```

Una vez creada la imagen, es necesario hacer uso del emulador Qemu para simularla y ejecutar la aplicación.

```
$ sudo qemu-system-i386 -hda <nombreImagen>
```

Para habilitar el puerto serie y para que Qemu sea capaz de utilizarlo hay que hacer uso del *flag --serial* seguido del dispositivo virtual. Linux identifica la placa Arduino como un dispositivo del tipo *ttyACMn* en el directorio */dev,* donde n es un número

identificativo. Para ver el nombre que ha proporcionado el sistema al puerto Arduino, se ejecuta el siguiente comando tras conectar el USB:

```
$ ls -l /dev | grep ACM
```

Una vez identificado el puerto (se supone *ttyACM0*) se puede ejecutar Qemu con el puerto serie habilitado mediante el comando:

```
$ sudo qemu-system-i386 -serial /dev/ttyACM0 -hda <nombreImagen>
```

ANEXO E: Enunciado de una posible práctica

Parte A

Se desea construir el sistema domótico de una casa controlado por computador desde un servidor externo. El objetivo del sistema es mantener unas condiciones agradables para el propietario y automatizar una serie de medidas de seguridad.

1. Características del entorno a controlar:

Se desea mantener una temperatura interior lo más próxima a 25 grados centígrados. Cualquier temperatura entre 20 y 30 grados se considera aceptable. Para ello se dispone de un sistema de aire acondicionado y un sistema de calefacción.

Para avisar al propietario de una posible fuga de gas o fuego en la casa se ubica un sistema de alarma que se activará cuando detecte ambos peligros y se desactivará en caso contrario.

2. Definición del sistema a controlar:

El sistema de control está formado por dos partes diferentes:

- El sistema electrónico de control instalado en el interior de la casa.
- El sistema de control remoto instalado en un servidor autónomo del centro de control.

Para la comunicación entre ambos sistemas se hará uso de una conexión serie UART estándar.

2.1. Definición del sistema electrónico de control instalado en la propia casa:

El sistema electrónico de la casa cuenta con diversos sensores y actuadores para recopilar la información del entorno e interactuar con él.

El hardware del sistema electrónico constará de una placa de microcontrolador modelo Arduino UNO. Los elementos sensores y actuadores se describen a continuación.

2.1.1. Lectura de la temperatura actual en el interior de la casa:

Para medir la temperatura se hace uso de un sensor de temperatura (modelo LM35). El esquema del circuito necesario para su implementación es el siguiente.

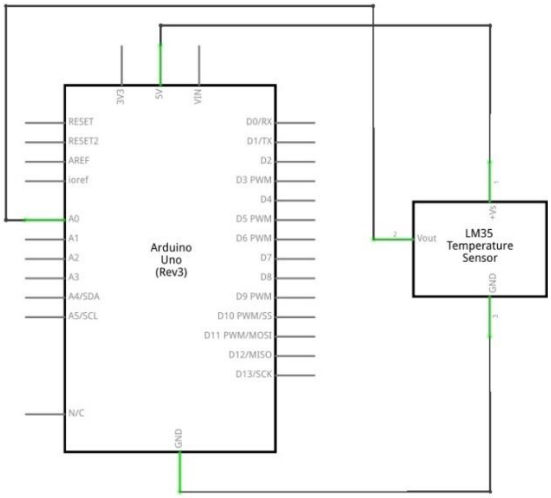


Ilustración 40. Esquema electrónico sensor de temperatura

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo el valor actual de la temperatura. El alumno deberá elegir un valor apropiado para dicho periodo.

2.1.2. Activación/desactivación del sistema de calefacción:

El sistema de calefacción debe implementarse mediante un LED que se encenderá cuando la temperatura se encuentre por debajo de los 25 grados. El esquema del circuito necesario para su implementación es el siguiente.

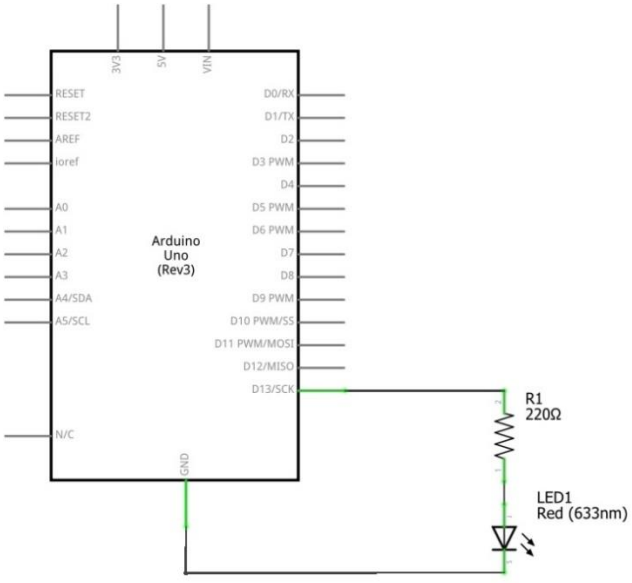


Ilustración 41. Esquema electrónico LED

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo el valor actual que debe tener el sistema de calefacción y que active o desactive el LED. El alumno deberá elegir un valor apropiado para dicho periodo.

2.1.3. Activación/desactivación del sistema de aire acondicionado:

El sistema de aire acondicionado debe implementarse mediante un pequeño ventilador (5 voltios) que se encenderá cuando la temperatura se encuentre por encima de los 25 grados. El esquema del circuito necesario para su implementación es el siguiente.

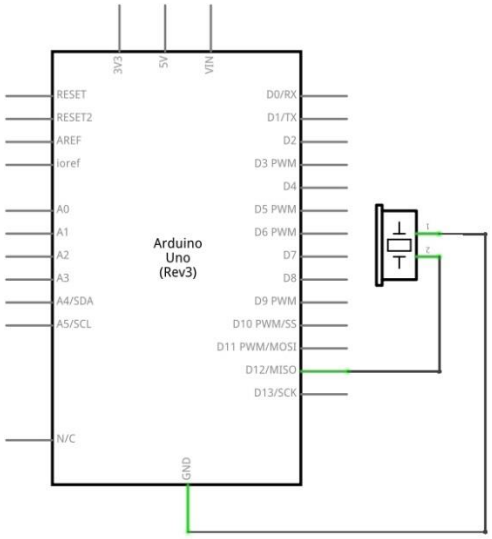


Ilustración 42. Esquema electrónico ventilador

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo el valor actual que debe tener el sistema de aire acondicionado y que active o desactive el ventilador. El alumno deberá elegir un valor apropiado para dicho periodo.

2.1.4. Lectura de una posible fuga de gas:

Para detectar una fuga de gas en la casa se hace uso de un sensor de gas (modelo MQ2). La lectura de gas se puede realizar digitalmente (si hay gas o no) o analógicamente (la concentración de gas). El esquema del circuito necesario para su implementación es el siguiente.

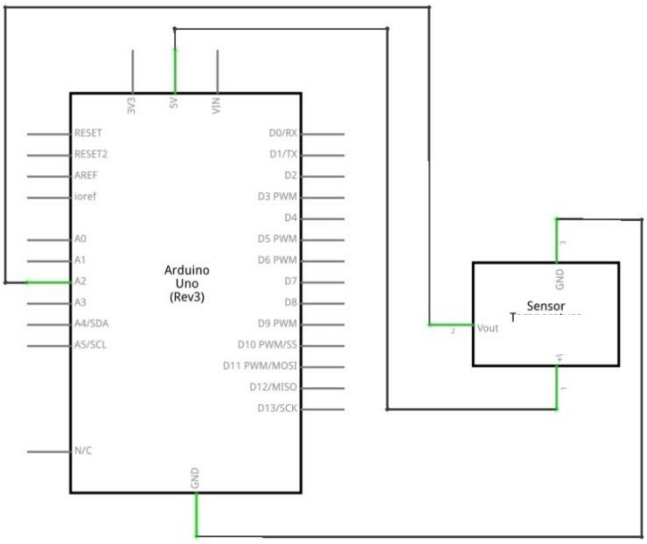


Ilustración 43. Esquema electrónico sensor de gas

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo el valor actual del sensor y determinar si existe gas en el ambiente o no. El alumno deberá elegir un valor apropiado para dicho periodo.

2.1.5. Lectura de un posible fuego:

Para detectar un fuego en la casa se hace uso de un sensor de llama (modelo BSC-FLAME). La lectura de fuego se puede realizar digitalmente (si hay fuego o no) o analógicamente (la concentración de llama). El esquema del circuito necesario para su implementación es el siguiente.

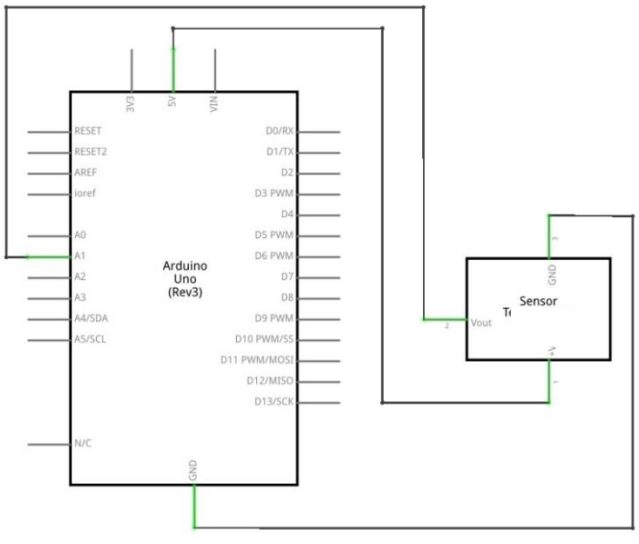


Ilustración 44. Esquema electrónico sensor de llama

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo el valor actual del sensor y determinar si existe fuego en el interior de la casa o no. El alumno deberá elegir un valor apropiado para dicho periodo.

2.1.6. Activación/desactivación de la alarma:

El sistema de alarma se representa mediante un zumbador que se activa cuando detecta gas o fuego en el interior de la casa. El esquema del circuito necesario para su implementación es el siguiente.

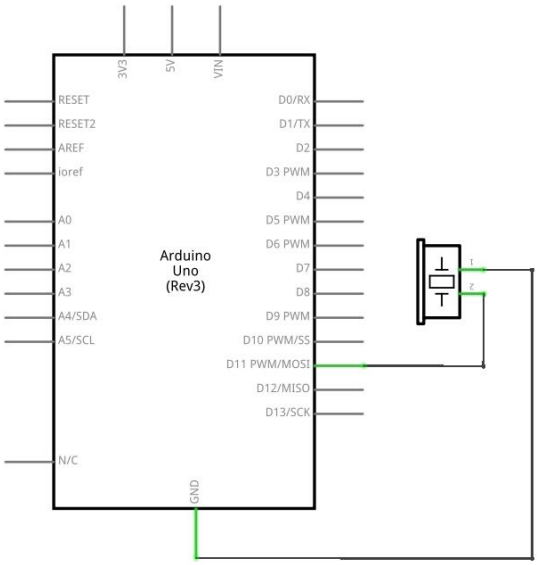


Ilustración 45. Esquema electrónico buzzer

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo el valor actual que debe tener la alarma de gas y fuego. El alumno deberá elegir un valor apropiado para dicho periodo.

2.1.7. Implementación del servidor de comunicaciones

El microcontrolador Arduino deberá incluir una tarea que, periódicamente, cada 200 milisegundos compruebe si se ha recibido una petición desde el sistema remoto. En caso afirmativo deberá realizar la acción correcta y enviar la respuesta correcta.

Tanto los mensajes de envío como los de recepción contendrán 9 caracteres (incluido el fin de línea). Esto permite detectar cuando se ha recibido un mensaje completo antes de empezar a leerlo.

Hay dos posibles tipos de comandos que se pueden recibir del sistema remoto.

a) Petición de lectura del valor de un sensor.

En este caso el sistema remoto envía un mensaje de petición indicando el sensor al que quiere acceder. El servidor de comunicaciones le envía un mensaje con el valor actual de dicho sensor que hay almacenado en la memoria del Arduino.

b) Petición de activar un elemento hardware.

En este caso el sistema remoto envía un mensaje indicado el elemento HW y si se quiere activar (SET) o desactivar (CLR). El servidor guardara la orden en memoria para que la tarea correspondiente la lleve a cabo y mandara un mensaje de OK.

En caso de que el mensaje enviado del sistema remoto no se pueda entender o haya cualquier tipo de error el servidor de comunicaciones enviará un mensaje de error.

La siguiente tabla incluye todos los mensajes del protocolo para cada tarea.

Petición	*Mensaje Petición*	*Mensaje Respuesta*	*Mensaje Error*
Leer Temperatura	TMP:<SP>REQ<CR>	TMP:00.0<CR>	MSG:<SP>ERR<CR>
Leer Fuego	FRE:<SP>REQ<CR>	FRE:<SP>YES<CR> FRE:<SP><SP>NO<CR>	MSG:<SP>ERR<CR>
Leer Gas	GAS:<SP>REQ<CR>	GAS:<SP>YES<CR> GAS:<SP><SP>NO<CR>	MSG:<SP>ERR<CR>
Activar/Desactivar Aire Acondicionado	AIR:<SP>SET<CR> AIR:<SP>CLR<CR>	AIR:<SP><SP>OK<CR>	MSG:<SP>ERR<CR>
Activar/Desactivar Calefacción	HOT:<SP>SET<CR> HOT:<SP>CLR<CR>	HOT:<SP><SP>OK<CR>	MSG:<SP>ERR<CR>
Activar/Desactivar Alarma	ALM:<SP>SET<CR> ALM:<SP>CLR<CR>	ALM:<SP><SP>OK<CR>	MSG:<SP>ERR<CR>

Tabla 109. Peticiones y respuestas correspondientes al apartado A de la práctica

2.2. Definición del sistema remoto:

El sistema remoto se encarga de obtener los datos del sistema electrónico instalado en la casa, toma las decisiones correspondientes y envía las peticiones de actuación de vuelta a la misma. El sistema remoto y propia casa estarán conectados por una conexión seria UART.

Además de lo anterior el sistema remoto dispondrá de un display para mostrar por pantalla el estado actual del sistema. Dicho display será accesible mediante un conjunto de llamadas a funciones.

Las tareas que debe realizar el sistema son las siguientes:

2.2.1. Lectura de la temperatura actual:

El sistema remoto deberá implementar una tarea periódica que envíe una petición a la casa para recibir el valor actual de la temperatura en el interior de la misma. Una vez recibida, la tarea debe mostrar la temperatura por el display utilizando la función correspondiente.

El periodo de esta tarea debe ser de 10 segundos.

2.2.2. Lectura de una posible fuga de gas:

El sistema remoto deberá implementar una tarea periódica que envíe una petición a la casa para recibir el valor actual de gas en el interior de la casa. Una vez recibida, la tarea debe mostrar el gas por el display utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

2.2.3. Lectura de un posible fuego:

El sistema remoto deberá implementar una tarea periódica que envíe una petición a la casa para recibir el valor actual de fuego en el interior de la casa. Una vez recibida, la tarea debe mostrar el fuego por el display utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

2.2.4. Activación/desactivación del sistema de aire acondicionado:

El sistema remoto deberá implementar una tarea periódica que calcule a partir a partir de los datos recibidos si debe activar el sistema de aire acondicionado o no. Una vez hecho deberá enviar una petición a la casa para activar o desactivar el mismo. Una vez realizada la acción, la tarea mostrará por el display si el sistema de aire acondicionado está activo o no utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

2.2.5. Activación/desactivación del sistema de calefacción:

El sistema remoto deberá implementar una tarea periódica que calcule a partir a partir de los datos recibidos si debe activar el sistema de calefacción o no. Una vez hecho deberá enviar una petición a la casa para activar o desactivar el mismo. Una vez realizada la acción, la tarea mostrará por el display si el sistema de calefacción está activo o no utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

2.2.6. Activación/desactivación de la alarma

El sistema remoto deberá implementar una tarea periódica que calcule a partir a partir de los datos recibidos si debe activar la alarma o no. Una vez hecho deberá enviar una petición a la casa para activar o desactivar la misma. Una vez realizada la acción, la tarea mostrará por el display si la alarma está activa o no utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

Las funciones de acceso al display del servidor remoto son las siguientes:

Elemento	*Función*	*Tiempo de Cómputo*
Temperatura	displayTemp (float temp)	< 0.5 s
Gas	displayGas (int gas)	< 0.5 s
Fuego	displayFire (int fire)	< 0.5 s
Aire Acondicionado	displayAir (int air)	< 0.5 s
Calefacción	displayHeat (int heat)	< 0.5 s
Alarma	displayAlarm (int alarm)	< 0.5 s

Tabla 110. Funciones display del apartado A

Se pide:

1. Diseñar un planificador cíclico que permita controlar el sistema descrito anteriormente para:

a) El sistema remoto

b) El control electrónico de la casa

2. Implementar dicho planificador cíclico para:

a) El sistema remoto utilizando el SSOO RTEMS y el código de apoyo

b) El control electrónico de la casa utilizando el microcontrolador Arduino UNO y hardware de apoyo.

El código de apoyo para el sistema a controlar se encuentra en **practica_1_sketch_2017-v1.zip**. Dicho zip contiene los siguientes ficheros:

- sketch.ino: Fichero de ejemplo del sketch de Arduino para el sistema a controlar.

El código de apoyo para el sistema remoto se encuentra en **practica_1A_RTEMS_2017-v1.zip**. Este contiene los siguientes ficheros:

- displayA.c: Incluye el código principal del display del apartado A.
- displayA.h: Incluye la cabecera de las funciones de control de display del apartado A.
- controladorA.c: Fichero de ejemplo para realizar el controlador del apartado A. Este fichero debe ser modificado y entregado como parte de la práctica.

Además se incluirá el código de apoyo de un módulo para conectar el sistema a controlar y el sistema remoto mediante un cable serie (**modulo_serie-v1.zip**). Este contiene los siguientes ficheros:

- Arduino-serial-lib.c: Código que permite comunicar una aplicación RTEMS con el Arduino mediante un puerto serie COM (a incluir en el programa del sistema remoto).

- Arduino-serial-lib.h: Fichero de cabeceras del código de enlace anterior (a incluir en el programa del sistema remoto).

Parte B

Se desea ampliar el sistema del apartado A para incluir el manejo automático de las luces de la casa y de un sistema multimedia.

1. Características del entorno a controlar:

El sistema a controlar es el mismo del Apartado A pero con la inclusión del control automático de luces cuando se detecte la presencia del propietario en la casa. Además, cuando se detecte al propietario en el salón se debe activar automáticamente un sistema multimedia.

El objetivo es activar tanto las luces como el sistema multimedia tan pronto como se detecte presencia en sus respectivas zonas. De la misma manera, deben desactivarse tan pronto como se deje de detectar la misma. Si no hay cambios no es necesario modificar sus estados. El sistema debe responder a los cambios tan pronto como sea posible, pero como mínimo, antes de 10 segundos.

Los nuevos componentes a controlar son:

- Sensor de presencia en la casa: Determina si existe o no presencia en la casa para encender o apagar las luces.

- Sensor de presencia en el salón: Determina si existe o no presencia en el salón para activar el sistema multimedia.

- Luces de la casa: Permiten la visibilidad del propietario dentro de la casa. Deben apagarse cuando no esté.

- Sistema multimedia: Reproduce un hilo musical cuando detecta al propietario en el salón. Debe apagarse cuando no se encuentre en el mismo.

2. Definición del sistema a controlar:

Al igual que en el apartado A sistema de control consta de dos partes diferenciadas:

- El sistema electrónico de control instalado en el interior de la casa.
- El sistema de control remoto instalado en un servidor autónomo del centro de control.

Para la comunicación entre ambos sistemas se hará uso de una conexión serie UART estándar.

2.1. Definición del sistema electrónico de control instalado en la propia casa:

Los requisitos del sistema electrónico de la casa serán los del apartado A pero añadiendo un sensor de presencia en la casa, un sensor de presencia en el salón, un activador de las luces y un activador del sistema multimedia. Los elementos sensores y activadores añadidos se describen a continuación.

2.1.1. Lectura del sensor de presencia en la casa:

El sensor de presencia se implementa mediante un sensor infrarrojo (IR FC-51). El esquema del circuito necesario es el siguiente.

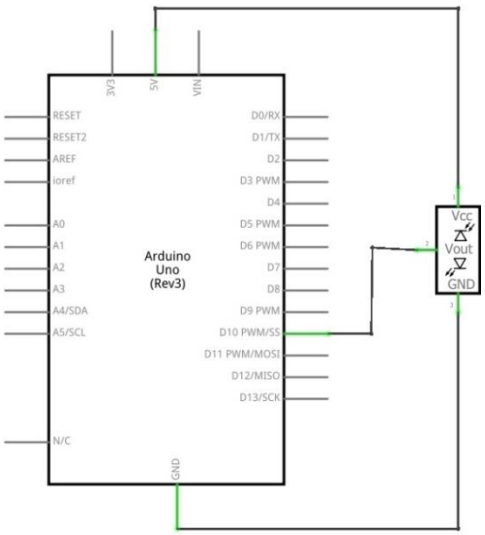


Ilustración 46. Esquema electrónico sensor infrarrojo

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo el valor actual que tiene el sensor y determinar si existe presencia en la casa o no. El alumno deberá elegir un valor apropiado para el periodo.

2.1.2. Lectura del sensor de presencia en el salón:

El sensor de presencia se implementa mediante un sensor infrarrojo (IR FC-51). El esquema del circuito necesario es el siguiente.

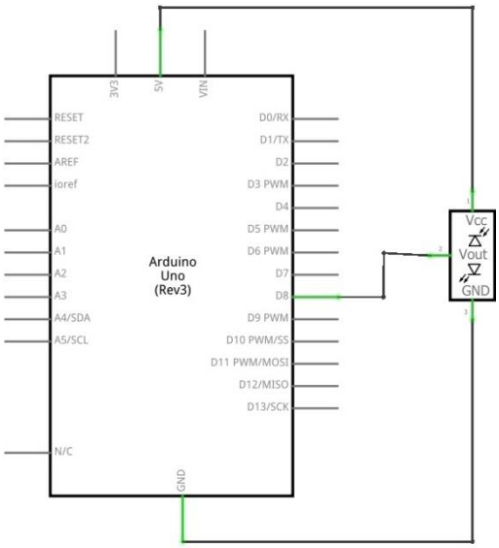


Ilustración 47. Esquema electrónico sensor infrarrojo

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo el valor actual que tiene el sensor y determinar si existe presencia en el salón o no. El alumno deberá elegir un valor apropiado para el periodo.

2.1.3. Activación/desactivación de las luces de la casa:

Las luces de la casa deben implementarse mediante un LED que se encenderá cuando se detecte presencia en la misma. El esquema del circuito necesario para su implementación es el siguiente.

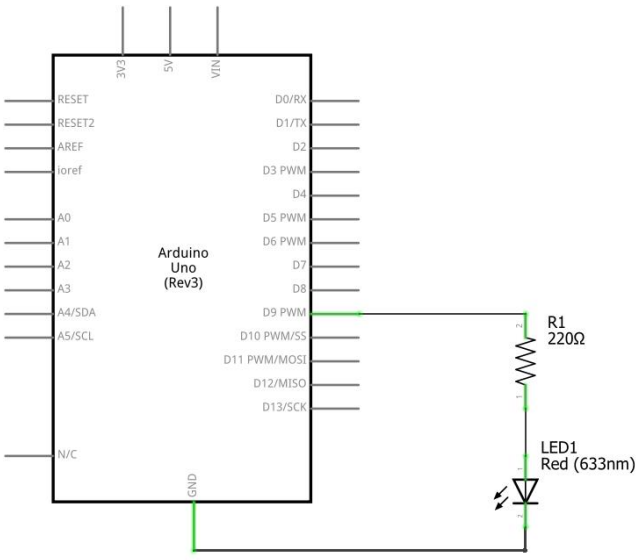


Ilustración 48. Esquema electrónico LED

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo el valor actual que deben tener las luces y que active o desactive las mismas. El alumno deberá elegir un valor apropiado para dicho periodo.

2.1.4. Activación/desactivación del sistema multimedia:

El sistema multimedia debe implementarse mediante un pequeño altavoz que reproducirá un hilo musical cuando se detecte presencia en el salón (el hilo musical puede ser un sonido simple o una canción, siempre median uso de frecuencias). El esquema del circuito necesario para su implementación es el siguiente.

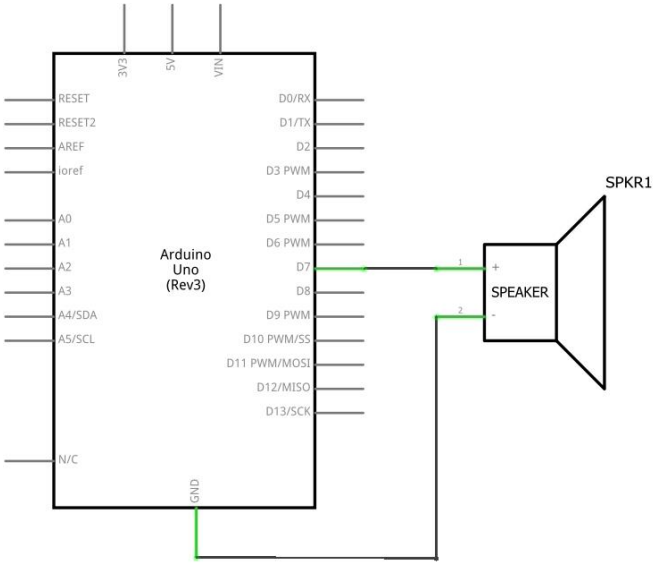


Ilustración 49. Esquema electrónico altavoz

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo el valor actual que debe tener el sistema multimedia y que active o desactive el altavoz. El alumno deberá elegir un valor apropiado para dicho periodo.

2.1.5. Implementación del servidor de comunicaciones:

El servidor de comunicaciones deberá modificarse para aceptar cuatro nuevas peticiones.

La siguiente tabla incluye todos los mensajes del protocolo para las nuevas tareas.

Petición	*Mensaje Petición*	*Mensaje Respuesta*	*Mensaje Error*
Leer Sensor Casa	DET:<SP>REQ<CR>	DET:<SP>YES<CR> DET:<SP><SP>NO<CR>	MSG:<SP>ERR<CR>
Leer Sensor Salón	HLL:<SP>REQ<CR>	HLL:<SP>YES<CR> HLL:<SP><SP>NO<CR>	MSG:<SP>ERR<CR>
Activar/Desactivar Luces	LIT:<SP>SET<CR> LIT:<SP>CLR<CR>	LIT<SP><SP>OK<CR>	MSG:<SP>ERR<CR>
Activar/Desactivar Sistema Multimedia	SPK:<SP>SET<CR> SPK:<SP>CLR<CR>	SPK<SP><SP>OK<CR>	MSG:<SP>ERR<CR>

Tabla 111.Peticiones y respuestas correspondientes al apartado B de la práctica

DET → Detection; HLL → Hall; LIT → Light; SPK → Speaker

2.2. Definición del sistema remoto

El sistema remoto es similar al de la parte A pero con cuatro nuevas tareas:

2.2.1. Lectura del sensor de presencia en la casa:

El sistema remoto deberá implementar una tarea periódica que envíe una petición a la casa para recibir el valor actual de la presencia en la casa. Una vez recibida, la tarea debe mostrar la presencia por el display utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

2.2.2. Lectura del sensor de presencia en el salón:

El sistema remoto deberá implementar una tarea periódica que envíe una petición a la casa para recibir el valor actual de la presencia en el salón. Una vez recibida, la tarea debe mostrar la presencia en el salón por el display utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

2.2.3. Activar/desactivar las luces:

El sistema remoto deberá implementar una tarea periódica que calcule a partir a partir de los datos recibidos si debe activar las luces o no. Una vez hecho deberá enviar una petición a la casa para activar o desactivar las mismas. Una vez realizada la acción, la tarea mostrará por el display si las luces están activas o no utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

2.2.4. Activar/desactivar el sistema multimedia

El sistema remoto deberá implementar una tarea periódica que calcule a partir a partir de los datos recibidos si debe activar el sistema multimedia o no. Una vez hecho deberá enviar una petición a la casa para activar o desactivar el mismo. Una vez realizada la acción, la tarea mostrará por el display si el sistema multimedia está activo o no utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

Las funciones de acceso al display del servidor remoto son las siguientes:

Elemento	*Función*	*Tiempo de Cómputo*
Presencia en casa	displayPresence (int presence)	< 0.5 s
Presencia en salón	displayHallPresence (int hallPresence)	< 0.5 s
Luces	displayLight (int light)	< 0.5 s
Altavoz	displaySpeaker (int speaker)	< 0.5 s

Tabla 112. Funciones display del apartado B

Se pide:

1. Diseñar un planificador cíclico que permita controlar el sistema descrito anteriormente para:

a) El sistema remoto

b) El control electrónico de la casa

2. Implementar dicho planificador cíclico para:

a) El sistema remoto utilizando el SSOO RTEMS y el código de apoyo.

b) El control electrónico de la casa utilizando el microcontrolador Arduino UNO y hardware de apoyo.

El código de apoyo para el sistema a controlar se encuentra en **practica_1_sketch_2017-v1.zip**. Dicho zip contiene los siguientes ficheros:

- sketch.ino: Fichero de ejemplo del sketch de Arduino para el sistema a controlar.

El código de apoyo para el sistema remoto se encuentra en **practica_1B_RTEMS_2017-v1.zip**. Este contiene los siguientes ficheros:

- displayB.c: Incluye el código principal del display del apartado B.
- displayB.h: Incluye la cabecera de las funciones de control de display del apartado B.
- controladorB.c: Fichero de ejemplo para realizar el controlador del apartado B. Este fichero debe ser modificado y entregado como parte de la práctica.

Además se incluirá el código de apoyo de un módulo para conectar el sistema a controlar y el sistema remoto mediante un cable serie (**modulo_serie-v1.zip**). Este contiene los siguientes ficheros:

- Arduino-serial-lib.c: Código que permite comunicar una aplicación RTEMS con el Arduino mediante un puerto serie COM (a incluir en el programa del sistema remoto).

- Arduino-serial-lib.h: Fichero de cabeceras del código de enlace anterior (a incluir en el programa del sistema remoto).

Parte C

Se desea ampliar el sistema del apartado A para incluir un sistema de alarma por robo.

1. Características del entorno a controlar:

El sistema a controlar es el mismo del Apartado B pero con la inclusión de una alarma que se activará cuando el propietario no esté en casa y la puerta se abra. En ese momento se activará un modo de alarma que finalizará cuando el propietario vuelva a casa y desactive la misma.

Para controlar dicho sistema se incluyen los siguientes cambios:

- Se incorpora un activador de alarma por robo de forma que cuando el propietario salga de casa se active la alarma.
- Se incorpora un sensor acoplado en la puerta principal que indicará, cuando el propietario no esté en casa, si se ha abierto la puerta o no.

Los objetivos buscados serán los siguientes:

- El sistema deberá ejecutar normalmente (como en la parte B) hasta que el propietario salga de casa y prepare la alarma por robo (debe comprobarse la preparación con la máxima frecuencia posible). En ese momento deberá cambiarse de modo de ejecución para detectar si la puerta se abre o no. Dicho proceso implica cambios respecto al modo de ejecución normal:
    - El sensor instalado en la puerta debe activarse (debe comprobarse su estado con la máxima frecuencia posible).
    - Los sensores de presencia que activan las luces de la casa y el sistema multimedia quedan desactivados.
    - El resto de sensores/actuadores deben funcionar igual que antes.
- El modo de alarma por robo preparada termina cuando el propietario regresa y lo desactiva o se produce una intrusión (en otro caso el modo no finalizará).En el primer caso se regresará al primer modo y en el segundo se cambiará al siguiente modo de ejecución para indicar que la alarma por intrusión ha saltado. Dicho proceso implica los siguientes cambios:
    - Se activa la señal de alarma permanentemente.

- Las luces de la casa parpadean.
- No es necesario regular la temperatura ni detectar posibles fugas de gas o fuego en el interior de la casa.
- Los sensores de presencia que activan las luces de la casa y el sistema multimedia quedan desactivados.
- Debe detectarse el momento en el que el propietario vuelve para desactivar la alarma. Esto debe realizarse con la máxima frecuencia posible.

- Tan pronto como el propietario desactive la alarma se volverá al modo de ejecución normal (en otro caso el modo de alarma por intrusión no finalizará).

2. Definición del sistema a controlar:

Al igual que en el apartado B sistema de control consta de dos partes diferenciadas:

- El sistema electrónico de control instalado en el interior de la casa.
- El sistema de control remoto instalado en un servidor autónomo del centro de control.

Para la comunicación entre ambos sistemas se hará uso de una conexión serie UART estándar.

2.1. Definición del sistema electrónico de control instalado en la propia casa:

Los requisitos del sistema electrónico de la casa serán los del apartado B pero añadiendo un sistema de preparación del modo alarma por robo, y el propio sistema de alarma por robo. En cualquier caso su funcionamiento dependerá del modo de operación que tenga el sistema electrónico en cada momento. Los modos de operación son los siguientes:

2.1.1. Modo de propietario en casa:

En este modo el sistema funcionara igual que el apartado B añadiendo las siguientes tareas y modificaciones a las tareas antiguas.

2.1.1.1. Lectura de la preparación de la alarma por robo:

La preparación de la alarma por robo se implementa mediante un botón. El esquema del circuito necesario es el siguiente.

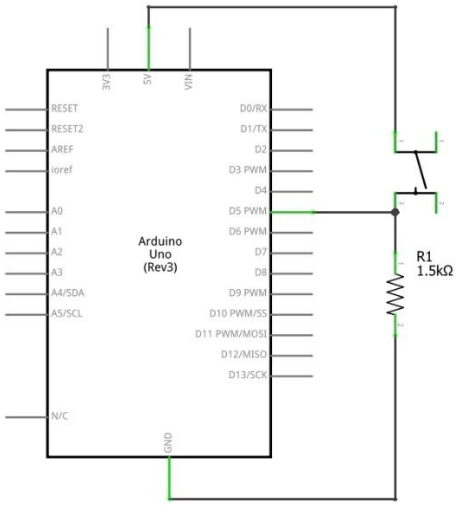


Ilustración 50. Esquema electrónico botón

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo cuando se ha producido una pulsación del botón (apretar y soltar el botón). En ese momento se cambiará al siguiente modo. El alumno deberá elegir un valor apropiado para el periodo.

2.1.1.2. Modificación del Servidor de Comunicaciones:

El servidor de comunicaciones deberá modificarse para aceptar una nueva petición que devuelva si se ha activado el modo de alarma por robo o no.

La siguiente tabla incluye todos los mensajes del protocolo para la nueva tarea.

Petición	*Mensaje Petición*	*Mensaje Respuesta*	*Mensaje Error*
Leer Activador Alarma Robo	BTN:<SP>REQ<CR>	BTN:<SP>PRS<CR> BTN:<SP><SP>NP<CR>	MSG:<SP>ERR<CR>

Tabla 113. Peticiones y respuestas (1) correspondientes al apartado C de la práctica

BTN-> Button; PRS-> Pressed; NP->No Pressed

2.1.2. Modo de alarma de robo preparada

En este modo el sistema funcionara igual que el apartado B a excepción de que los sensores de presencia (luces de la casa y sistema multimedia) quedan desactivados y añadiendo las siguientes tareas y modificaciones a las tareas antiguas.

2.1.2.1. Lectura del sensor de la puerta principal:

Para conocer el estado de la puerta principal se hace uso de un contacto magnético instalado en la misma. El esquema del circuito necesario para su implementación es el siguiente.

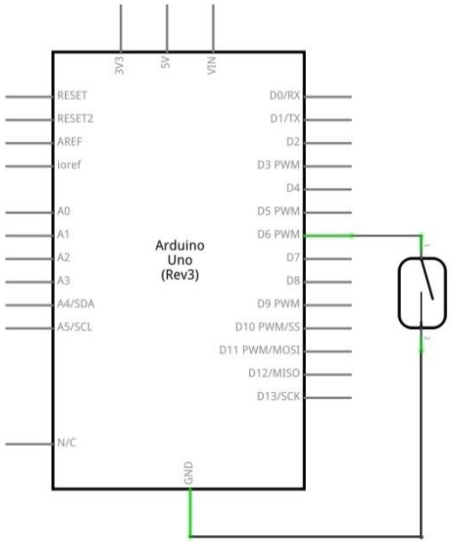


Ilustración 51. Esquema electrónico interruptor magnético

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo el sensor y determine si se ha abierto la puerta principal o no. En ese momento se cambiará al siguiente modo. El alumno deberá elegir un valor apropiado para el periodo.

2.1.2.2. Lectura del cambio al primer modo (propietario vuelve a casa):

El cambio al primer modo, es decir, cuando el propietario vuelve y no se ha producido ninguna intrusión se implementa mediante el mismo botón de preparación de la alarma.

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo cuando se ha producido una pulsación del botón (apretar y soltar el botón). En ese momento se cambiará al primer modo. El alumno deberá elegir un valor apropiado para el periodo.

2.1.2.3. Modificación del Servidor de Comunicaciones:

El servidor de comunicaciones deberá modificarse para aceptar una nueva petición que devuelva si la puerta se abre o no.

La siguiente tabla incluye todos los mensajes del protocolo para la nueva tarea.

Petición	*Mensaje Petición*	*Mensaje Respuesta*	*Mensaje Error*
Leer Sensor Puerta	MGN:<SP>REQ<CR>	MGN:<SP><SP>OP<CR> MGN:<SP><SP>CL<CR>	MSG:<SP>ERR<CR>

Tabla 114. Peticiones y respuestas (2) correspondientes al apartado C de la práctica

MGN → Magnetic; OP → Opened; CL →Closed

2.1.3. Modo de alarma por robo

En este modo el sistema ejecutará de la siguiente manera:

2.1.3.1 Activación de las luces de la casa:

Esta tarea utiliza el mismo hardware (led que simula las luces de la casa) que la tarea que enciende las luces por presencia. La diferencia es que en este modo, las luces deben parpadear constantemente.

Deberá programarse una tarea periódica en el microcontrolador para esta tarea. El alumno deberá elegir un valor apropiado para dicho periodo.

2.1.3.2 Activación del sonido de alarma:

Esta tarea utiliza el mismo hardware (zumbador que simula la alarma) que la tarea de alarma por gas o fuego. La diferencia es que en este modo, la alarma debe estar encendida permanentemente.

Deberá programarse una tarea periódica en el microcontrolador para esta tarea. El alumno deberá elegir un valor apropiado para dicho periodo.

2.1.3.3. Lectura de la desactivación de la alarma por robo.

El cambio al primer modo, es decir, cuando el propietario vuelve y desactiva la alarma se implementa mediante el mismo botón de preparación de la alarma.

Deberá programarse una tarea periódica en el microcontrolador que compruebe en cada periodo cuando se ha producido una pulsación del botón (apretar y soltar el botón). En ese momento se cambiará al primer modo. El alumno deberá elegir un valor apropiado para el periodo.

2.1.4. Visualización del modo actual:

Además en todos los modos se hace uso de un display LCD 16x2 que muestra el modo de ejecución actual y un potenciómetro (10kΩ) que regula el contraste de la pantalla. El esquema del circuito necesario para su implementación es el siguiente.

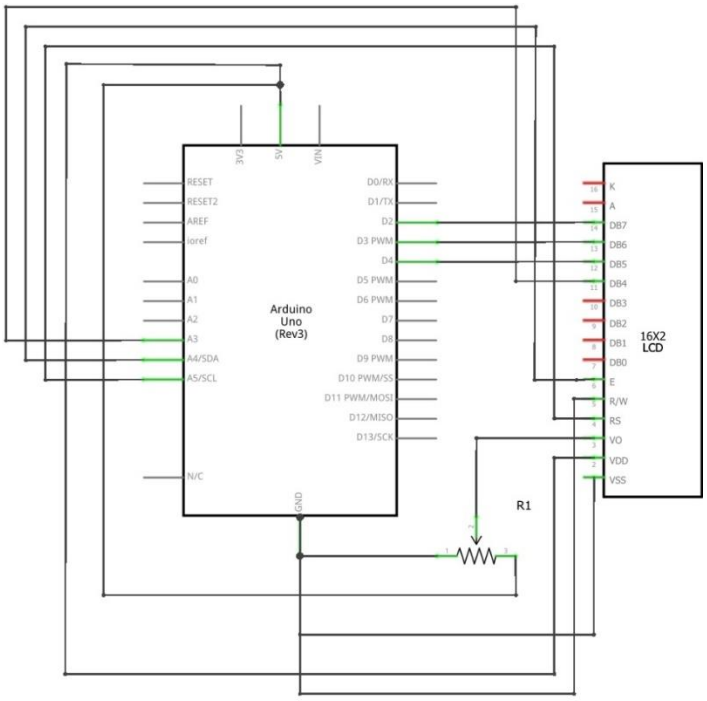


Ilustración 52. Esquema electrónico pantalla LCD

2.2. Definición del sistema remoto:

El sistema remoto es similar al de la parte 2 pero con nuevas tareas:

2.2.1. Modo propietario en casa

2.2.1.1. Lectura de la preparación de la alarma por robo:

El sistema remoto deberá implementar una tarea periódica que envíe una petición a la casa para conocer el valor del botón de preparación de la alarma por robo. Una vez recibida, la tarea debe mostrar el estado del botón por el display utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

2.2.2. Modo alarma por robo preparada

2.2.2.1. Lectura del sensor de la puerta principal:

El sistema remoto deberá implementar una tarea periódica que envíe una petición a la casa para conocer el estado de la puerta principal. Una vez recibida, la tarea debe mostrar el estado del sensor por el display utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

2.2.2.2. Lectura de cambio al primer modo (el propietario vuelve a casa):

El sistema remoto deberá implementar una tarea periódica que envíe una petición a la casa para conocer el estado del botón de preparación del modo de alarma por robo. Una vez recibida, la tarea debe mostrar el estado del botón por el display utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

2.2.3. Modo alarma por robo

2.2.3.1. Activación de las luces de la casa:

El sistema remoto deberá implementar una tarea periódica que calcule a partir a partir de los datos recibidos si debe activar las luces o no. Una vez hecho deberá enviar una petición a la casa para activar las mismas. Una vez realizada la acción, la tarea mostrará por el display si las luces están activas o no utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

2.2.3.2. Activación de la alarma:

El sistema remoto deberá implementar una tarea periódica que calcule a partir a partir de los datos recibidos si debe activar la alarma o no. Una vez hecho deberá enviar una petición a la casa para activar la misma. Una vez realizada la acción, la tarea mostrará por el display si la alarma está activa o no utilizando la función correspondiente.

2.2.3.3. Lectura de cambio al primer modo (el propietario vuelve a casa y desactiva la alarma):

El sistema remoto deberá implementar una tarea periódica que envíe una petición a la casa para conocer el estado del botón. Una vez recibida, la tarea debe mostrar el estado del botón por el display utilizando la función correspondiente.

El alumno deberá calcular el periodo correcto para esta tarea.

Las funciones de acceso al display del servidor remoto son las siguientes:

Elemento	*Función*	*Tiempo de Cómputo*
Botón preparación alarma	displayButtonStatus (int buttonStatus)	< 0.5 s
Sensor magnético puerta	displaydoorStatus (int doorStatus)	< 0.5 s
Luces por robo	displayLight2 (int light2)	< 0.5 s
Alarma por robo	displayAlarm2 (int alarm2)	< 0.5 s

Tabla 115. Funciones display del apartado C

Se pide:

1. Diseñar un planificador cíclico que permita controlar el sistema descrito anteriormente para:

a) El sistema remoto

b) El control electrónico de la casa

2. Implementar dicho planificador cíclico para:

a) El sistema remoto utilizando el SSOO RTEMS y el código de apoyo.

b) El control electrónico de la casa utilizando el microcontrolador Arduino UNO y hardware de apoyo.

El código de apoyo para el sistema a controlar se encuentra en **practica_1_sketch_2017-v1.zip**. Dicho zip contiene los siguientes ficheros:

- sketch.ino: Fichero de ejemplo del sketch de Arduino para el sistema a controlar.

El código de apoyo para el sistema remoto se encuentra en **practica_1C_RTEMS_2017-v1.zip**. Este contiene los siguientes ficheros:

- displayC.c: Incluye el código principal del display del apartado C.
- displayC.h: Incluye la cabecera de las funciones de control de display del apartado C.
- controladorC.c: Fichero de ejemplo para realizar el controlador del apartado C. Este fichero debe ser modificado y entregado como parte de la práctica.

Además se incluirá el código de apoyo de un módulo para conectar el sistema a controlar y el sistema remoto mediante un cable serie (**modulo_serie-v1.zip**). Este contiene los siguientes ficheros:

- Arduino-serial-lib.c: Código que permite comunicar una aplicación RTEMS con el Arduino mediante un puerto serie COM (a incluir en el programa del sistema remoto).

- Arduino-serial-lib.h: Fichero de cabeceras del código de enlace anterior (a incluir en el programa del sistema remoto).

Parte D

Se desea ampliar el sistema del apartado C para incluir un modo de emergencia en caso de un fallo en los tiempos del servidor remoto.

1. Características del entorno a controlar:

El sistema a controlar es el mismo del Apartado C. En este sistema las funciones de acceso al display pueden sufrir retraso más allá del tiempo de cómputo máximo previsto. En caso que se produzca un error en los tiempos de respuesta el sistema remoto debe ejecutarse un modo de emergencia que consiste en lo siguiente.

- Las luces de la casa parpadean.
- Por el altavoz musical se reproduce un sonido que indica el error del sistema.
- Los sensores de las alarmas quedan desactivados permanentemente.
- En el display se muestra un mensaje que indica el error del sistema.
- Se debe regular la temperatura todo lo posible.

2. Definición del sistema electrónico de control instalado en la propia casa:

Al igual que en el apartado A sistema de control consta de dos partes diferenciadas:

- El sistema electrónico de control instalado en el interior de la casa.
- El sistema de control remoto instalado en un servidor autónomo del centro de control.

Para la comunicación entre ambos sistemas se hará uso de una conexión serie UART estándar.

2.1. Definición del sistema electrónico de control instalado en la propia casa:

Los requisitos del sistema electrónico de la casa serán los del apartado C pero añadiendo un modo de emergencia que indica el fallo del mismo.

Además la tarea del servidor de comunicaciones deberá ser modificada en todos los modos para incluir la petición de salto al modo de emergencia.

2.2.1. Modo de emergencia:

En este modo el sistema incluirá las mismas tareas del apartado B de regulación de la temperatura. Los sensores de alarmas quedan desactivados, las luces de la casa deben parpadear, por el altavoz se reproduce un sistema que indica el error del sistema y en el display LCD se muestra un mensaje de error del sistema.

2.2.2. Modificación del servidor de comunicaciones:

El servidor de comunicaciones deberá incluirá una nueva petición para cambiar al modo de parada de emergencia.

La siguiente tabla incluye todos los mensajes del protocolo para las nuevas tareas.

Petición	*Mensaje Petición*	*Mensaje Respuesta*	*Mensaje Error*
Activar Modo Emergencia	ERR:<SP>SET<CR>	ERR:<SP><SP>OK<CR>	MSG:<SP>ERR<CR>

Tabla 116. Peticiones y respuestas correspondientes al apartado D de la práctica

2.2. Definición del sistema remoto:

El sistema remoto debe implementar el modo de emergencia indicado en la sección 1 (características del entorno a controlar).

Se pide:

1. Diseñar un planificador cíclico que permita controlar el sistema descrito anteriormente para:

a) El sistema remoto

b) El control electrónico de la casa

2. Implementar dicho planificador cíclico para:

a) El sistema remoto utilizando el SSOO RTEMS y el código de apoyo.

b) El control electrónico de la casa utilizando el microcontrolador Arduino UNO y hardware de apoyo.

El código de apoyo para el sistema a controlar se encuentra en **practica_1_sketch_2017-v1.zip**. Dicho zip contiene los siguientes ficheros:

- sketch.ino: Fichero de ejemplo del sketch de Arduino para el sistema a controlar.

El código de apoyo para el sistema remoto se encuentra en **practica_1D_RTEMS_2017-v1.zip**. Este contiene los siguientes ficheros:

- displayD.c: Incluye el código principal del display del apartado D.
- displayD.h: Incluye la cabecera de las funciones de control de display del apartado D.
- controladorD.c: Fichero de ejemplo para realizar el controlador del apartado D. Este fichero debe ser modificado y entregado como parte de la práctica.

Además se incluirá el código de apoyo de un módulo para conectar el sistema a controlar y el sistema remoto mediante un cable serie (**modulo_serie-v1.zip**). Este contiene los siguientes ficheros:

- Arduino-serial-lib.c: Código que permite comunicar una aplicación RTEMS con el Arduino mediante un puerto serie COM (a incluir en el programa del sistema remoto).
- Arduino-serial-lib.h: Fichero de cabeceras del código de enlace anterior (a incluir en el programa del sistema remoto).

ANEXO F: Corrección de la práctica propuesta

La práctica se evalúa sobre 10 puntos, pero existe la posibilidad, mediante la implementación de algunas funcionalidades concretas de sumar un punto extra. A continuación se desglosa la puntuación en casa apartado de la práctica. Cuando se emplee la expresión *"implementación de la funcionalidad"* se supone la implementación en ambos módulos de la práctica (sistema servidor remoto y sistema electrónico empotrado)

Parte 1 (3 puntos)

- Mensajes de petición y respuesta: La correcta implementación de todos los mensajes de petición y respuesta se puntúa con ***0,25 puntos***.
- Lectura de la temperatura: El valor de la temperatura debe ser medido en grados centígrados, realizando la conversión completa. La correcta implementación de esta funcionalidad se puntúa con ***1 punto.***
- Activación/desactivación del aire acondicionado: El correcto funcionamiento del LED se puntúa con ***0,25 puntos.***
- Activación/desactivación del ventilador: El correcto funcionamiento del ventilador se puntúa con ***0,25 puntos***.
- Lectura de gas: El funcionamiento de esta lectura se puntúa con ***0,5 puntos.***
- Lectura de fuego: El funcionamiento de esta lectura se puntúa con ***0,5 puntos.***
- Activación/descativación de la alarma por gas y fuego: El funcionamiento de la alarma se puntúa con ***0.25 puntos***.

La calibración de los sensores que incorporan potenciómetro debe de ser la adecuada.

La lectura analógica de los valores de gas y fuego y la posterior activación de la alarma por superación de un cierto umbral se puntúa con ***+0,25 puntos*** cada lectura. Esta puntuación pertenece al posible punto extra.

Parte 2 (2 puntos)

- Sensor de presencia en la casa: La lectura correcta del sensor se evalúa con **0,5 puntos**.
- Sensor de presencia en el salón: La lectura correcta del sensor se evalúa con **0,5 puntos**.
- Activación/desactivación de las luces de la casa: El correcto funcionamiento del LED se evalúa con ***0,25 puntos***.
- Activación/desactivación del altavoz: La correcta reproducción de sonido por el altavoz se puntúa con ***0,75 puntos***.

La calibración de los sensores que incorporan potenciómetro debe de ser la adecuada.

La reproducción de una canción mediante uso de frecuencias se puntuará con ***+0,25 puntos.*** Esta puntuación pertenece al posible punto extra.

Parte 3 (4 puntos)

- Botón de preparación alarmas: La implementación del botón se puntúa con ***0,25 puntos***.
- Lectura del sensor magnético instalado en la puerta: Su implementación se puntúa con ***0.25 puntos***.
- Display LCD 16x2: El correcto funcionamiento de la pantalla LCD se puntúa con ***0,5 puntos***.
- El funcionamiento de los 3 modos por separado se evalúa con ***1 punto***.
- El funcionamiento de los 3 modos en conjunto se evalúa con ***2 puntos.***

Parte 4 (1 punto)

- El correcto funcionamiento del modo de emergencia se evalúa con ***1 punto.***

Si la práctica funciona perfectamente en su totalidad se otorga ***+0,5 puntos***. Esta puntuación completaría el posible punto extra.

Bibliografía

[1] I. Vilajosana Guillén, "Introducción a los sistemas empotrados". **[En línea]. Disponible en**: https://www.exabyteinformatica.com/uoc/Informatica/Sistemas_empotrados/Sistemas_empotrados_(Modulo_1).pdf **Acceso:** junio 2017.

[2] J. Carretero Pérez & J. Fernández Muñoz, "Sistemas de Tiempo Real: Tema 1: Introducción". *Apuntes del curso 2015-2016*.

[3] B. Martín del Río, "Sistemas electrónicos basados en microprocesadores y microcontroladores", *Prensas Universitarias de Zaragoza*, 1999. **[En línea]. Disponible en**: https://www.researchgate.net/profile/Bonifacio_Martin-del-Brio/publication/31737513_Sistemas_electronicos_basados_en_microprocesadores_y_microcontroladores_B_Martin_del_Brio/links/5731d92b08ae9ace84047093/Sistemas-electronicos-basados-en-microprocesadores-y-microcontroladores-B-Martin-del-Brio.pdf. **Acceso:** junio 2017.

[4] E. Ashford Lee & S. Arunkumar Seshia, "Introduction to embedded systems: A cyber-physical systems approach". *MIT Press*, 2016. **[En línea]. Disponible en**: http://leeseshia.org/releases/LeeSeshia_DigitalV1_08.pdf. **Acceso:** junio 2017.

[5] A. Gean Ye, "Field-programmable gate array architectures and algorithms optimized for implementing datapath circuits", *University of Toronto*, 2004. **[En línea]. Disponible en**: http://www.eecg.toronto.edu/~jayar/pubs/theses/Ye/AndyYe.pdf. **Acceso:** junio 2017.

[6] B. Úbeda Miñarro, "Apuntes de: Sistemas embebidos". *Universidad de Murcia*, 2009. **[En línea]. Disponible en**: http://ocw.um.es/ingenierias/sistemas-embebidos/material-de-clase-1/ssee-t01.pdf. **Acceso:** junio 2017.

[7] R. Arango, A. Andrés Navarro & J. Bestier Padilla, "SISTEMAS OPEN HARDWARE Y OPEN SOURCE APLICADOS A LA ENSEÑANZA DE LA ELECTRÓNICA". *Journal of Research of the University of Quindio*, 2014, vol. 25, no 1. **[En línea]. Disponible en**: http://blade1.uniquindio.edu.co/uniquindio/revistainvestigaciones/adjuntos/pdf/1b18_126-133.pdf. **Acceso:** junio 2017.

[8] Web principal *Arduino* (Modelos de microcontroladores). **[En línea]. Disponible en**: https://www.arduino.cc/en/Main/Products. **Acceso:** julio 2017.

[9] Web principal *Arduino* (Introducción). **[En línea]. Disponible en**: https://www.arduino.cc/en/Guide/Introduction. **Acceso:** julio 2017.

[10] Web principal *Arduino* (Documentos). **[En línea]. Disponible en**: https://www.arduino.cc/en/Main/Docs. **Acceso:** julio 2017.

[11] Arduino. **[En línea]. Disponible en**: https://es.wikipedia.org/wiki/Arduino. **Acceso:** julio 2017.

[12] Web principal *Raspberry Pi*. **[En línea]. Disponible en**: https://www.raspberrypi.org/. **Acceso:** julio 2017.

[13] Ultimate Guide to Raspberry Pi. **[En línea]. Disponible en**: http://micklord.com/foru/Raspberry%20Pi%20Pages%20from%20Computer%20Shopper%202015-02.pdf. **Acceso:** julio 2017.

[14] Web principal *BeagleBoard* (Modelos de placas). **[En línea]. Disponible en**: https://beagleboard.org/boards. **Acceso:** julio 2017.

[15] A. Crespo & A. Alonso, "Una panorámica de los sistemas de tiempo real". *Revista Iberoamericana de Automática e Informática Industrial*, 2010, vol. 3, no 2, p. 7-18. **[En línea]. Disponible en**: https://www.researchgate.net/publication/28141966_Una_Panoramica_de_los_Sistemas_de_Tiempo_Real. **Acceso:** julio 2017.

[16] J. A. De la Puente, "Introducción a los Sistemas en Tiempo Real". **[En línea]. Disponible en**: http://isa.uniovi.es/docencia/TiempoReal/Recursos/Transparencias/Introduccion.pdf. **Acceso:** julio 2017.

[17] Ada Reference Manual. **[En línea]. Disponible en**: http://www.adaic.org/resources/add_content/standards/95lrm/RM.pdf. **Acceso:** agosto 2017.

[18] J. Garrido, J. Zamorano, J. A. de la Puente, A. Alonso & E. Salazar, "Ada, the programming language of choice for the UPMSat-2 satellite. Data Systems in Aerospace". *DASIA*, 2015. **[En línea]. Disponible en**: http://www.dit.upm.es/~str/papers/pdf/garrido&15a.pdf. **Acceso:** agosto 2017.

[19] J. Barnes, "Ada 2012 Rationale". 2013. **[En línea]. Disponible en**: http://www.ada-europe.org/manuals/Rationale_2012.pdf. **Acceso:** agosto 2017.

[20] T. Groussard. "Java 7: Los Fundamentos del lenguaje Java". *Ediciones Eni*, 2012.

[21] Web *The Real-Time Specification for Java.* **[En línea]. Disponible en**: http://www.rtsj.org/. **Acceso:** agosto 2017.

[22] Web *Oracle* (Java Platform Micro Edition). **[En línea]. Disponible en**: http://www.oracle.com/technetwork/java/embedded/javame/index.html. **Acceso:** agosto 2017.

[23] M. González Harbour, "POSIX de tiempo Real". *University of Cantabria*, Santander, 2004. **[En línea]. Disponible en**: https://www.istr.unican.es/publications/mgh-1993b.pdf. **Acceso:** agosto 2017.

[24] S. R. Walli, "The POSIX family of standards". *StandardView*, 1995, vol. 3, no 1, p. 11-17. **[En línea]. Disponible en**: http://stephesblog.blogs.com/papers/acm-posix.pdf. **Acceso:** agosto 2017.

[25] S. Meghanathan & N. Baskiyar, "A survey of contemporary real-time operating systems". *Informatica*, 2005, vol. 29, no 2. **[En línea]. Disponible en:** http://www.eng.auburn.edu/~baskiyar/MyArticles/Survey-of-RTOS-Informatica.pdf. **Acceso:** agosto 2017.

[26] Web *Microsoft* (Windows Embedded). **[En línea]. Disponible en**: https://www.microsoft.com/windowsembedded/en-us/windows-embedded.aspx. **Acceso:** agosto 2017.

[27] S. Pavlov & P. Belevsky, "Windows embedded CE 6.0 fundamentals". *Microsoft Press*, 2008. **[En línea]. Disponible en**: http://staff.ustc.edu.cn/~shizhu/WinCE/winCE6%20Fundamentals.pdf. **Acceso:** agosto 2017.

[28] Web *Windriver.* **[En línea]. Disponible en**: https://www.windriver.com/. **Acceso:** agosto 2017.

[29] Wind River Real-Time Core for Linux. **[En línea]. Disponible en**: https://www.windriver.com/products/product-overviews/PO_RTCore_for_LX_May2009.pdf. **Acceso:** agosto 2017.

[30] VxWORKS: The Safe and Secure RTOS for the Internet of Things. **[En línea]. Disponible en**: https://www.windriver.com/products/product-notes/pn_vxworks/VxWorks-PN.pdf. **Acceso:** agosto 2017.

[31] Web *RTEMS*. **[En línea]. Disponible en**: https://www.rtems.org/. **Acceso:** junio 2017.

[32] Web *Eclipse.* **[En línea]. Disponible en**: http://www.eclipse.org/home/index.php. **Acceso:** agosto 2017.

[33] Tutorial para la instalación de RTEMS Source Builder. **[En línea]. Disponible en**: *https://ftp.rtems.org/pub/rtems/people/chrisj/source-builder/source-builder.html*. **Acceso:** junio 2017.

[34] Manual de usuario de RTEMS. **[En línea]. Disponible en**: *https://docs.rtems.org/branches/master/user.pdf*. **Acceso:** junio 2017.

[35] Web *RTEMS* (Documentación). **[En línea]. Disponible en**: *https://docs.rtems.org/*. **Acceso:** junio 2017.

[36] J. MAS, "Marco jurídico y oportunidades de negocio en el software libre". *UOC Papers, Revista sobre la sociedad del conocimiento*, 2005, no 1. **[En línea]. Disponible en**: http://www.uoc.edu/uocpapers/1/dt/esp/mas.pdf. **Acceso:** agosto 2017.

[37] Estándar POSIX. **[En línea]. Disponible en**: http://pubs.opengroup.org/onlinepubs/9699919799/. **Acceso:** agosto 2017.