Escuela Técnica Superior de Ingeniería
y Sistemas de Telecomunicación
Campus Sur
Universidad Politécnica de Madrid

PROYECTO FIN DE CARRERA

Implementación de Sistema Domótico con Servidor Raspberry

Autores: Juan Angosto Herrmann

Fernando Salgado Álvarez

15 de septiembre de 2014

E.T.S.I.S. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

TEMA: Automatización y Sistemas de Control

TÍTULO: Implementación de Sistema Domótico con Servidor Raspberry

AUTOR: Salgado Álvarez, Fernando
Angosto Herrmann, Juan

TUTOR: Herrera Camacho, José Antonio **Vº Bº.**

DEPARTAMENTO: Sistemas Electrónicos y de Control

Miembros del Tribunal Calificador:

PRESIDENTE: García Hernando, Ana Belén

VOCAL: Herrera Camacho, José Antonio

VOCAL SECRETARIO: Jiménez Martínez, Francisco Javier

DIRECTOR:

Fecha de lectura:

Calificación: **El Secretario,**

RESUMEN DEL PROYECTO:

Se trata de diseñar y desarrollar un sistema de control ambiental distribuido para una vivienda utilizando hardware específico y un protocolo de comunicación inalámbrico destinado para ello. El sistema se compone de un servidor que hará de gestor de tareas y funciones, y de interfaz para el usuario. Este se implementará con un ordenador en miniatura llamado Raspberry Pi. Los controladores que recibirán las directivas de funcionamiento serán implementados por los alumnos utilizando microcontroladores de la familia C8051.

Escuela Universitaria de Ingeniería
Técnica de Telecomunicación
Universidad Politécnica de Madrid

RESUMEN EN CASTELLANO DEL PROYECTO
FIN DE CARRERA

Implementación de Sistema Domótico con Servidor Raspberry

Autores: Juan Angosto Herrmann

Fernando Salgado Álvarez

13 de septiembre de 2014

Este proyecto consiste en el diseño e implementación un sistema domótico que puede ser instalado en una vivienda para controlar distintas variables ambientales y conseguir así la máxima comodidad de los habitantes de manera automática o manual según los gustos y necesidades de los usuarios.

La característica principal de este sistema, es que cuenta con un funcionamiento distribuido donde entran en juego un servidor, encargado de tomar las decisiones generales para el comportamiento de la casa, y una serie de controladores esclavo cuya función es mantener constantes las variables ambientales con los valores fijados por el servidor. Así se consigue mantener la vivienda en una situación de bienestar constante para cualquier persona que se encuentre dentro.

El sistema ha sido pensado de manera que se intenta reducir al máximo el cableado para facilitar su instalación por lo que la comunicación entre los distintos dispositivos se hace de manera inalámbrica por medio de un protocolo descrito en la norma IEEE 802.15.4 llamado ZigBee. Para ello se ha utilizado un módulo de comunicación wireless llamado Xbee, el cual permite la comunicación entre dos dispositivos.

Para el control de dicho sistema distribuido se cuenta con una aplicación web, que mediante una interfaz gráfica permite al usuario controlar los distintos dispositivos dentro de la vivienda consiguiendo así controlar las variables ambientales a gusto del usuario. Dicha interfaz gráfica no depende de un software especifico, sino que sólo es necesario un cliente http como podría ser Internet Explorer, Mozilla Firefox, Google Chrome, etc.

Para integrar dicho sistema se ha usado un mini ordenador de bajo coste llamado RaspBerryPi, en el que se encuentra alojado un servidor Apache con el fin de gestionar y automatizar las variables ambientales. El control de los dipositivos encargados de modificar y estabilizar las variables ambientales

se realiza mediante unos controladores génericos implementados mediante μcontroladores 80C51F410, pertenecientes a la familia 80C51, y una serie de componentes y circuitería que permiten el correcto funcionamiento de éstos.

Existen dos tipos de controladores distintos, los cuales son:

- Controlador Sensor: Encargados de las tomas de valores ambientales como puede ser la luz y la temperatura.
- Controladores Actuadores: Encargados de actuar sobre los dispositivos que modifican y estabilizan las variables ambientales como pueden ser la calefacción, tiras de leds de iluminación, persianas, alarmas, etc.

El conjunto de la RaspBerryPi y los diferentes controladores forman el prototipo diseñado para este proyecto fin de carrera, el cual puede ser ampliado sencillamente para abarcar una amplia gama de posibilidades y funcionalidades dentro de la comodidad de una vivienda.

Escuela Universitaria de Ingeniería
Técnica de Telecomunicación
Universidad Politécnica de Madrid

RESUMEN EN INGLÉS DEL PROYECTO FIN DE CARRERA

Implementación de Sistema Domótico con Servidor Raspberry

Autores: Juan Angosto Herrmann

Fernando Salgado Álvarez

15 de septiembre de 2014

The project described in this report consisted designing and implementing a home automation system that could be installed in a house in order to control environmental variables and thus get the maximum comfort of the inhabitant automatically or manually according to their tastes and needs.

The main feature of this system consists in a distributed system, formed by a server which is responsible for making the main decisions of the actions performed inside the house. In addition, there are a series of slave controlers whose function consists in keeping the environmental variables within the values established by the server. Thus gets to keep the home in a situation of constant wellbeing to anyone who is inside.

The system has been designed in order to reduce the amount of wire needed for the inter-connection of the devices, by means of wireless communication. The devices chosen for the solution are Xbee modules, which use the Zigbee protocol in order to comunicate one between each other. The Zigbee protocol is fully described in the IEEE 802.15.4 standard.

A web application has been used to control the distributed system. This application allows users to control various devices inside the house and subsequently the different environmental variables. This implementation allows obtaining the maximum comfort by means of a very simple graphical interface.

In addition, the Graphical User Interface (GUI) does not depend on any specific software. This means that it would only be necessary a http client (such as Internet Explorer, Mozilla Firefox, Google Chrome, etc.) for handling the application

The system has been integrated using a low-cost mini computer called RaspBerryPi.This computer has an Apache server allocated which allows to

manage and to automatize the different environmental variables. Furthermore, for changing and stabilizing those variables, some generic controllers have been developed, based on μcontrollers 80C51F410.

There have been developed mainly two different types of controllers: Sensor Controllers, responsible for measuring the different environmental values, such as light and temperature; and Actuator Controllers, which purpose is to modify and stabilize those environmental variables by actuating on the heating, the led lamps, the blinders, the alarm, etc.

The combination of the RaspBerryPi and the different controllers conform the prototype designed during this project. Additionally, this solution could be easily expanded in order to intake further functionalities adapted to new needs that could arise in the future.

Índice General

Anexos

Índice de Figuras

Parte I

Introducción

Dentro del campo tecnológico, la vivienda y el hogar ha encontrado un amplio abanico de posibilidades. La integración de la tecnología en el diseño inteligente del hogar se conoce con el nombre de *Domótica*. Dicho término proviene de la unión de las palabras *domus* (derivada de la raiz domo que quiere decir casa en latín) y la palabra francesa *informatique* (de la que ha derivado la palabra informática).

Se puede decir que el origen de la *domótica* se remonta a la década de los 70. Es en dicha época cuando comienzan las primeras investigaciones y desarrollos de dispositivos enfocados a la automatización de las viviendas basados en la tecnología X-10(1) desarrollada entre 1976 y 1978 en Escocia. Posteriormente surgieron diversas tecnologías entre las cuales se pueden encontrar también KNX (2) y Zigbee(3). En los siguientes años comenzó a cobrar interés internacional dicho campo buscando así un modelo de vivienda ideal, lo cual inspiró cierto afán de investigación y desarrollo. A finales de la década de los 80 y principio de los 90 nacen los primeros PC's (Personal Computer) y comienza su integración en los edificios consiguiendo así un control de ciertos dispositivos electrónicos dando lugar a lo que hoy conocemos como edificios inteligentes. Podemos decir que en España el origen de la domótica se remonta a finales de la década de los 90. De la misma manera que a dias de hoy no es aceptable que una vivienda carezca de sistema eléctrico o agua corriente, en un futuro no muy lejano no se concebirá viviendas que no estén mínimamente domotizadas. Hoy en día el campo domótico está en contínuo crecimiento y desarrollo tecnológico.

El mundo de la *domótica* es complejo ya que trata de mezclar tecnología y avances tecnológicos con la interacción social del ser humano con las máquinas. Cabe mencionar que la *domótica* intenta satisfacer las necesidades del usuario, sean cuales fueren, teniendo en cuenta que dichos avances y facilidades ofrecidas han de ser *usables*. Aquí empieza otro mundo también muy complejo que es el de

la *experiencia de usuario y la usabilidad*(4) dentro de la *domótica*, que se encargará de un manejo cómodo, sencillo e intuitivo del sistema ofrecido a dicho usuario, el cual no tiene porque sentir ninguna afinidad con el campo tecnológico que esconde la *domótica*.

De ésta manera se entiende por *domótica* la integración de la automatización, la informática y las nuevas tecnologías dentro de los espacios habitables con los fines de mejorar el bienestar, el ahorro energético, la seguridad, la comunicación, la satisfacción y la comodidad del usuario de dicho espacio. (5)(6)

Debido a ésto se obtiene un sinfín de posibilidades entre las cuáles priman las diferentes aplicaciones de la *domótica*:

- Programación y ahorro energético.
- Comfort.
- Seguridad.
- Comunicaciones.
- Accesibilidad.

Analizando cada uno de estas aplicaciones se pueden determinar los servicios y ventajas que ofrece vivir en un hogar digital.

Programación y ahorro energético

El ahorro energético(7) es un punto muy importante ya que puede ser aplicado tanto al concepto económico como al concepto ecológico. Para poder tener un sistema eficiente el cual preste atención al ahorro energético no es necesario sustituir los dispositivos que se manejan dentro de una vivienda por

otros que consuman menos si no que basta con una gestión eficiente de los mismos como puede ser:

- Programación y zonificación de la climatización.
- Control de toldos y persianas.
- Gestión electrica, racionalización de cargas y control de tarifas.
- Uso de energías renovables.

Comfort

El confort conlleva todas las actuaciones que se puedan llevar a cabo que mejoren el confort en una vivienda. Dichas actuaciones pueden ser de carácter tanto pasivo, como activo o mixtas.

- Apagado general de todas las luces de la vivienda.
- Automatización del apagado / encendido en cada punto de luz.
- Regulación de la iluminación según el nivel de luminosidad ambiente.
- Automatización de todos los distintos sistemas / instalaciones / equipos dotándolos de control eficiente y de fácil manejo.
- Control vía Internet.
- Generación de programas de forma sencilla para el usuario y automatización.

Seguridad

Consiste en una red de seguridad encargada de proteger tanto los bienes, como la seguridad personal y la vida.

- Alarmas de intrusión para detectar la presencia de personas extrañas en la vivienda.
- Cierre de persianas puntual y seguro.
- Simulación de presencia.
- Detectores y alarmas de detección de incendios (detector de calor, detector de humo), detector de gas (fugas de gas, para cocinas no eléctricas), escapes de agua e inundación, etc.
- Alerta médica y teleasistencia.
- Acceso a cámaras.

Comunicaciones

Son los sistemas o infraestructuras de comunicaciones que posee el hogar.

- control tanto externo como interno, control remoto desde Internet, PC, mandos inalámbricos, etc.
- Teleasistencia.
- Telemantenimiento.
- Informes de consumo y costes.
- Transmisión de alarmas.
- Intercomunicaciones.

Accesibilidad

Esta aplicación es una de las mas complejas debido a la cantidad y variedad de limitaciones funcionales y discapacidades que existen hoy en dia. Cada usuario con su discapacidad necesitará unos u otros servicios por lo que éstos han de ser personalizados para cada usuario.

La *domótica* aplicada a favorecer la accesibilidad es un reto ético y creativo pero sobre todo es la aplicación de la tecnología en el campo más necesario, para suplir limitaciones funcionales de las personas, incluyendo las personas discapacitadas o mayores. El objetivo no es que las personas con discapacidad puedan acceder a estas tecnologías, porque las tecnologías en si no son un objetivo, sino un medio. El objetivo de estas tecnologías es favorecer la autonomía personal. Los destinatarios de estas tecnologías son todas las personas, ya sea por enfermedad, discapacidad o envejecimiento. (8)

El Sistema

Para poder realizar un sistema domótico en una vivienda como el propuesto en este proyecto son necesarios diversos elementos interconectados entre sí de tal manera que se base en una arquitectura centralizada en cuanto a que se dispone de varios pequeños dispositivos capaces de adquirir y procesar la información de múltiples sensores. Esta información es transmitida a un servidor central que es el encargado de la toma de decisiones en función de los datos almacenados a lo largo del tiempo y a su vez éste se encarga de transmitir dichas ordenes al resto de dispositivos distribuidos por la vivienda, los cuales actúan en consecuencia. Los elementos de una instalación domótica necesarios son:

- Central de gestión llamada servidor.
- Sensores o detectores.
- Actuadores.

- Soportes de comunicación.

Parte II

Ideas Generales del Proyecto

Capítulo 1

Resumen General Del Proyecto

En la especialidad de Sistemas Electrónicos y de Control, impartida en la carrera de Ingeniería Técnica de Telecomunicación, se estudian distintos campos que pueden dar lugar a tipos de proyecto diferentes. Uno de ellos en particular es el protagonista del trabajo presentado en esta documentación. Se trata de los sistemas de control, una ciencia especialmente apreciada por las personas que componen el equipo que ha desarrollado esta aplicación.

Este proyecto consiste en un sistema domótico que puede ser instalado en una vivienda para controlar distintas variables ambientales y conseguir así la máxima comodidad de los habitantes de manera automática.

Cabe mencionar de este sistema que cuenta con un funcionamiento distribuido donde entran en juego un servidor, que toma las decisiones generales para el comportamiento de la casa, y una serie de *controladores esclavo* cuya función es mantener constantes las variables ambientales con los valores fijados por el primero. Así se consigue mantener la casa en una situación de bienestar constante para cualquier persona que se encuentre dentro.

La comunicación entre los distintos dispositivos se hace de manera inalámbrica por medio de un protocolo descrito en la norma IEEE 802.15.4. Esta norma se utiliza para crear redes sin cables de transmisión de mensajes cortos. Para ello se ha utilizado un módulo de comunicación wireless llamado Xbee, que gracias a una adecuada configuración, permite la comunicación de 2 dispositivos, emulando una comunicación serie punto a punto por medio de una *UART*.

Además se dispone de una interfaz gráfica que permite al usuario controlar la vivienda sencillamente desde cualquier dispositivo que sea capaz de conectarse a una red local. Se trata de una aplicación web que puede ser controlada sin un software específico para su utilización, siendo necesario únicamente un cliente *http*.

Este proyecto ha sido integrado usando un Rasberry Pi(9). Se trata de un ordenador de dimensiones reducidas y muy bajo coste que se comercializó por primera vez en Febrero de 2012. Inicialmente fue lanzado al mercado para estimular el interés por la computación en los colegios. Actualmente se utiliza en todo tipo de aplicaciones, gracias a su bajo precio, dimensiones reducidas y la amplia gama de aplicaciones desarrolladas (9).

Capítulo 2

Sistema Completo

Para entender de manera sencilla la estructura y funcionamiento de este sistema, se ha dividido el mismo en 3 partes, teniendo en cuenta los distintos momentos de comunicación entre entes del sistema.

- Comunicación usuario → servidor.
- Comunicación servidor → controlador.
- Comunicación controlador → controlador.

Para explicar en detalle el funcionamiento de la aplicación se ha utilizado como ejemplo a un usuario tratando de mantener una temperatura constante en el salón de su casa por medio de nuestro sistema.

En primer lugar, el usuario debe acceder a la interfaz del sistema para configurar la temperatura. Esta comunicación de configuración se realiza con el servidor, dispositivo que toma las decisiones generales sobre las acciones de la vivienda. A esta comunicación se le ha denominado como *usuario → servidor*.

Una vez el servidor ha recogido los datos de configuración de la temperatura deseada en el salón, debe tomar decisiones y comunicarse con el controlador de la calefacción para que este tenga constancia de la temperatura deseada y la mantenga estable. A esta comunicación se la denomina como *servidor → controlador*.

Una vez el controlador tenga constancia de qué temperatura debe mantener, ha de medir la temperatura ambiental para saber si debe o no debe encender la calefacción. Para leer la temperatura ambiental, el controlador debe comunicarse con otro controlador. Este último controlador citado tiene acceso a los sensores localizados en toda la vivienda como pueden ser sensores de luz o temperatura. A esta comunicación se la denomina como *controlador → controlador*.

Por tanto el sistema completo, que como mínimo deberá estar formado por un servidor y dos controladores (dos controladores por habitación, uno sensor y otro actuador), debe tener un esquema general parecido al mostrado en la figura 2.1.

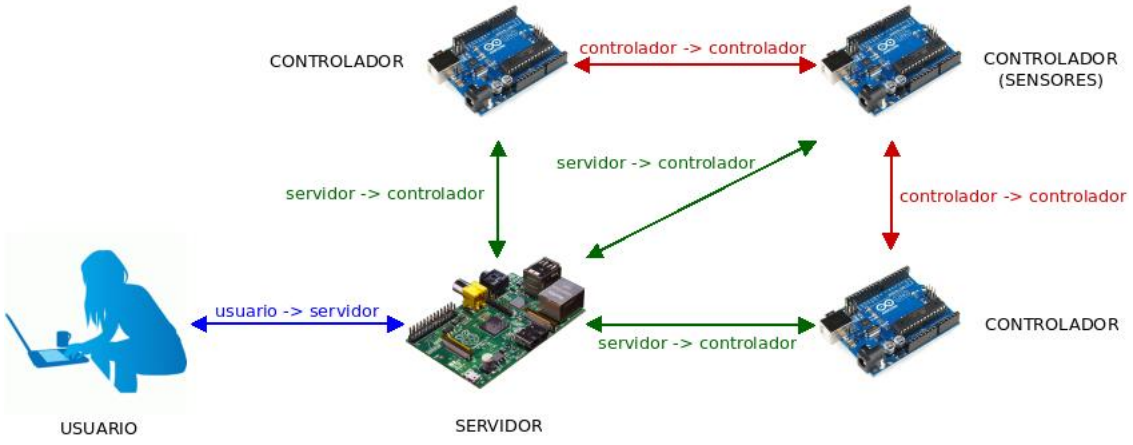


Figura 2.1: *Estructura de Comunicación del Sistema*

Después de describir el sistema en rasgos generales, a continuación se presenta un análisis de las funciones y comportamientos de cada una de las

partes.

2.1. Comunicación Usuario → Servidor

En este escenario entran en juego un usuario, que maneja una interfaz por medio de un dispositivo, y un servidor que ofrece esa interfaz y recoge los datos fijados por el usuario. Además de esto, el servidor almacena los datos de configuración introducidos en una base de datos.

En el servidor habrá instalado un sistema gestor de base de datos conocido como *Apache*(10) con los módulos necesarios para administrar dicha base de datos y para poder interpretar un lenguaje de programación diseñado para el desarrollo web conocido como PHP(11). Dicho lenguaje es el que ha sido utilizado para implementar la interfaz web.

El envío de los datos desde el navegador web se hace con aplicaciones llamadas *consultas Ajax* (Asynchronous Java Script and XML)(12). Esto se realizará mediante peticiones que permitan cambiar el contenido de la pagina web sin necesidad de recargarla. Para ello se utilizara una biblioteca, consistente en un único fichero, encargada de administrar dichas peticiones conocida como *JQuery* (13). Dichas peticiones son enviadas a un puerto determinado del servidor que estará de manera continuada a la escucha y recepción de datos (también llamado *demonio* en lenguaje de programación C). Una vez recibidos los mensajes del usuario, estos son procesados y son enviados al controlador correspondiente.

2.2. Comunicación Servidor → Controlador

Una vez procesados los datos recibidos por la interfaz web, el servidor toma las decisiones oportunas para que la configuración fijada sea llevada a cabo. Entre estas decisiones, la más importante es la de configuración del dispositivo destino del mensaje. Para la gestión de esta decisión se utiliza un direccionamiento implementado por los desarrolladores de este sistema. Debido al módulo de comunicación de comunicación wireless *Xbee* utilizado (el más sencillo y económico) las comunicaciones entre dispositivos únicamente se pueden hacer punto a punto o por difusión (también conocido como comunicación *Broadcast*). En el caso de establecer comunicaciones punto a punto, es necesario configurar el módulo wireless cada vez que se desee enviar un dato a un dispositivo determinado. Por tanto, la decisión tomada es la de enviar mensajes por difusión y añadir en la trama enviada una dirección. Esto hace que un dispositivo, que recibe un mensaje donde la dirección destino coincide con la suya propia, tome dicha cadena. En caso contrario lo descarta. El hecho de enviar los mensajes por difusión, permite transmitir ordenes comunes a todos los dispositivos a la vez, como por ejemplo un mensaje de sincronización de hora, o un mensaje de activación de alarmas de emergencia, etc...

Otro escenario en el que el servidor necesita comunicarse con el controlador es en el momento en el que el primero desea registrar el valor actual de una variable ambiental para guardarla en la base de datos y hacer así un historial del entorno. Para ello el servidor envía una petición de dato al *controlador sensor* para que este mida y envíe el valor como respuesta. En este caso el direccionamiento es el mismo que el del caso anterior.

Cada dispositivo que actúa en el sistema de comunicación por medio del módulo de comunicación wireless *Xbee*, tiene una dirección única que lo identifica

en la red.

2.3. Comunicación Controlador → Controlador

Las comunicaciones entre controladores del mismo tipo se realizan con el fin de poder mantener una variable ambiental constante. Para ello el controlador sensor compara el valor ambiental medido con el valor configurado y deseado por el usuario, y envía un mensaje de orden, mediante el módulo de comunicación wireless Xbee, al controlador actuador correspondiente para que este actúe en consecuencia. Se produce una toma de medidas periódicamente con el fin de mantener estable la variable ambiental.

Capítulo 3

Motivaciones del Proyecto

3.1. Raspberry Pi

La primera motivación para desarrollar este proyecto fué la utilización del dispositivo *Raspberry Pi* (RbP) (figura 3.1) un ordenador completo de bajo coste y de pequeñas dimensiones. Una de las características mas interesantes del Raspberry Pi es que dispone de interfaces físicas que nos permiten acceder a los diferentes periféricos de los que dispone el SoC de la placa. De esta manera se puede interactuar con circuitos externos de manera sencilla conectandolos a dicha interfaz.

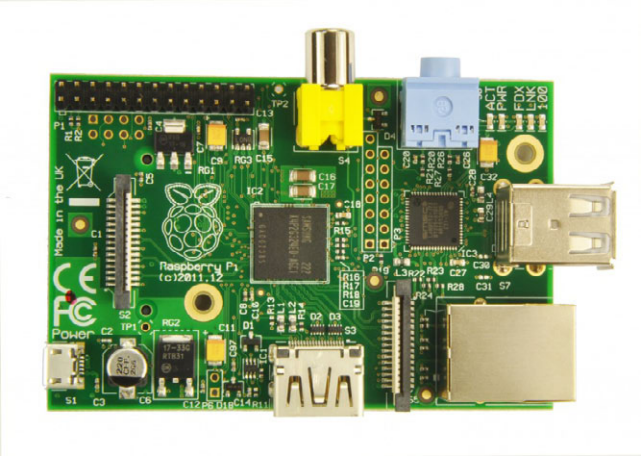


Figura 3.1: *Ordenador de Bajo Coste RaspBerry Pi*

3.2. Controladores

En un principio los controladores iban a ser implementados con placas de desarrollo llamadas *Arduino*(14). Se trata de unos dispositivos de hardware libre, es decir controladores de los cuales se dispondría de las especificaciones y diagramas esquemáticos de manera pública, que disponen de un μcontrolador Atmel y pines de Entrada/Salida. Se programan mediante un lenguaje propio basado en *Processing*. Son muy sencillos de programar, depurar y utilizar.

Finalmente se decidió complicarlo un poco y diseñar e implementar controladores parecidos a los ya utilizados en la asignatura cursada ISE (Ingeniería de Sistemas Electrónicos). Para ello se utilizó un μcontrolador C8051F410 de *Silicon Laboratories*. De esta manera, se ha podido hacer un proyecto más interesante además de conocer y practicar con el módulo de comunicación wireless Xbee. Cabe decir que los μcontroladores implementados en las placas son más potentes que los del arduino. La solución implementada, a diferencia de los arduinos, dispone de un sistema operativo que permite ejecutar tareas de manera concurrente.

3.3. Entorno Web

En lo referente al entorno web, se han querido aprovechar los conocimientos de desarrollo web de los que disponen los aspirantes a ingenieros, para poder demostrar la versatilidad de los mismos. Hay que tener en cuenta que en lo tiempos que corren es complicado encontrar un trabajo de desarrollo de sistemas electrónicos y por tanto, la implementación de aplicaciones informáticas es una buena alternativa profesional, debido al margen que este negocio ofrece, ya que la infraestructura necesaria es más fácil de conseguir y mantener.

La primera ventaja que ofrece la interfaz web que sirve para controlar el sistema es que es innecesario desarrollar una aplicación específica para cada dispositivo que pueda entrar en juego. Es bien sabido, que la estandarización de sistemas comerciales conlleva complicaciones, y que si una persona decide desarrollar una aplicación compatible con cualquier dispositivo móvil, tendrá que tener en cuenta 3 grandes sistemas operativos que son *Android, IOS y Windows*. Esto sin contar con el elevado número de entornos para estaciones de trabajo no móviles. Por ello, con una aplicación web, solo es necesario un cliente http y aún teniendo en cuenta que cada dispositivo tiene un tamaño específico de pantalla que se tiene que estudiar, la adaptación para cada uno de ellos consiste únicamente en cambiar la estructura de la interfaz, nunca saliendo de un mismo entorno o lenguaje.

3.4. Intereses Propios

Es interesante desarrollar un proyecto como éste que puede servir para la vida cotidiana de cualquier persona. Las bondades del sistema pueden ser explicadas al usuario de manera sencilla sin necesidad de facilitar detalles sobre la implementación de la solución. Las dos personas que han llevado a cabo el

trabajo descrito en este documento piensan desde un principio aplicarlo en sus propias casa.

Capítulo 4

Presentación del Documento

Un vez presentado el proyecto de manera general, se describirá cada paso para llevarlo a cabo. Se describirán las ideas iniciales que llevaron a ciertas búsquedas, las pruebas realizadas, los diseños y los pasos seguidos en la implementación definitiva de las diferentes funcionalidades. Para ello se dividirá el documento en 5 grandes partes que pueden ser independientes entre sí pudiendo trabajar con cada una de ellas sin tener en cuenta las demás. Estas son:

- Interfaz web y comunicación con el Servidor.
- Procesamiento de datos en el Servidor.
- Controladores.
- Comunicación entre entidades.
- Plantas controladas.

Parte III

Desarrollo

Capítulo 5

Interfaz Web y Comunicación con el Servidor

Como se dijo anteriormente, una de las ventajas de crear una aplicación web para la solución es que permite acceder a dicha aplicación desde cualquier dispositivo que disponga de un cliente http.De esta manera se consigue independencia respecto a cualquiera de los sistemas operativos disponibles en la actualidad.

Esta interfaz se aloja en la RbP con un Servidor Apache instalado, que dispone de lo necesario para entregar una aplicación web implementada en PHP y con una base de datos en MySQL(15).

5.1. Diseño y Estructura de la Aplicación

La aplicación diseñada, además de ser independiente del sistema operativo utilizado, ha sido desarrollada teniendo en cuenta que existen dos grandes

grupos de dispositivos: los tradicionales que disponen de un teclado y un ratón para interactuar con ellos y los dispositivos móviles donde el manejo se hace gracias a una pantalla táctil. Por ello se diseñaron dos entornos adaptados a cada uno de estos dos tipos de dispositivos.

Para el desarrollo del entorno web se optó por recurrir a 2 frameworks que ofrecen una manera sencilla de conseguir funcionalidades y presentaciones adaptadas a cada uno de los tipos de dispositivos descritos anteriormente. En lo referente a dispositivos móviles se utiliza *JQuery Mobile*, un plugin de la librería JQuery. Para los dispositivos tradicionales, como son los ordenadores se utiliza *Bootstrap*.

Las dos aplicaciones son iguales desde el punto de vista de navegación, las secciones son las mismas y los controles son del mismo tipo. A pesar de estos detalles la disposición de estos no es la misma en las distintas interfaces, ya sea web o móvil. Un buen diseño, el cual debería conllevar un estudio de usabilidad o experiencia de usuario, ha de tener en cuenta la comodidad del usuario para manejar la aplicación. En el caso de un smartphone por ejemplo, hay ciertas zonas de la pantalla donde el dedo pulgar tiene más dificultad para pulsar (Figuras 5.1 y 5.2).

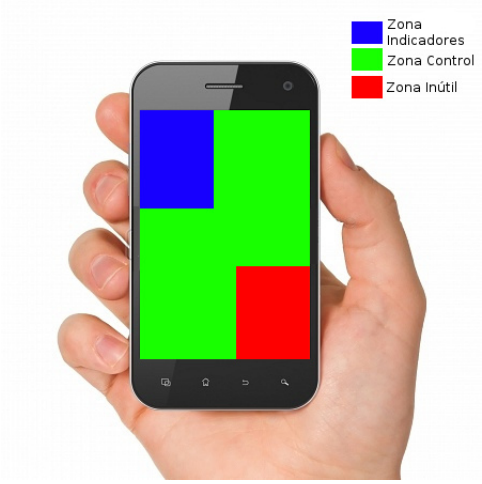


Figura 5.1: *Zonas de Emplazamiento para Dispositivo en Vertical*

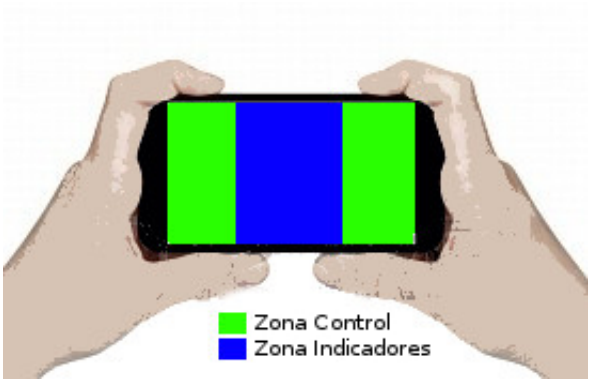


Figura 5.2: *Zonas de Emplazamiento para Dispositivo en Horizontal*

6

Básicamente la aplicación (sea móvil o no) dispone como página principal un plano de la vivienda (Figura 5.3) en la que se puede acceder al panel de control de cada una de las habitaciones pulsando en una zona en particular. Cada sección ofrece los mandos necesarios para administrar las variables controlables. Para acceder a esta interfaz, uno debe iniciar la sesión con nombre de usuario y contraseña. Una vez validado, la aplicación detecta si el dispositivo es móvil o

tradicional, y dependiendo de esto, el servidor entrega una u otra presentación de la aplicación.

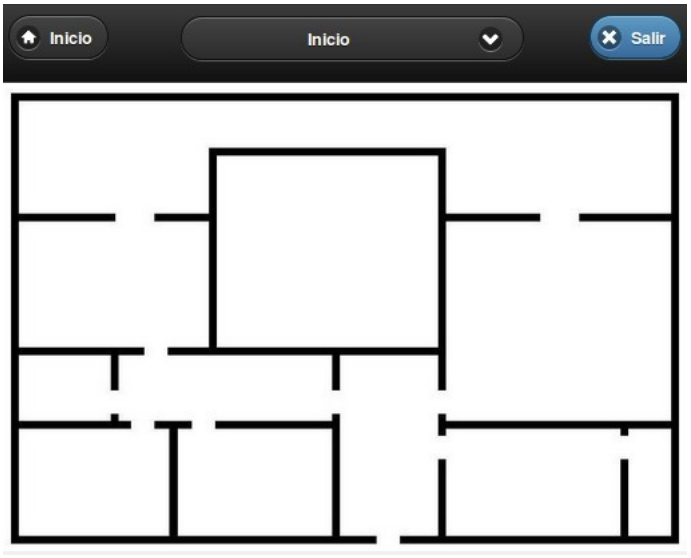


Figura 5.3: *Página Principal de la Aplicación Móvil*

Además de las secciones de control de zona de la vivienda, la aplicación dispone de una interfaz de presentación de datos ambientales guardados en la base de datos del servidor. Esta solución se basa en un formulario que permite seleccionar qué dato ambiental se desea que se represente en una gráfica, en la cual se representan los valores ambientales seleccionados alojados con respecto al tiempo.

5.1.1. Estructura de la Aplicación

Cuando se implementa una aplicación web, como cualquier otro tipo de aplicación, hay que buscar siempre una manera de hacerla *escalable* y fácil de mantener. Por ello, a la hora de desarrollar dicha aplicación, se ha establecido una estructura determinada que respeta estas dos características.

```
DomoWeb/
├── classes/
│   └── Mobile-Detect-2.7.1/ ..... Detector de dispositivo
├── libreria/
│   └── funciones.php ..... Funciones comunes
├── mobile/
│   ├── componentes/ ..... Elementos de control e indicación
│   ├── css/ ..... Hojas de estilo
│   ├── img/ ..... Imágenes
│   ├── js/ ..... Ficheros JavaScript
│   ├── peticionesAjax/ ..... Script de atención a petición Ajax
│   ├── plugins/ ..... Plugins de terceros
│   ├── secciones/ ..... Contenedores de secciones
│   ├── acceder.php ..... Validación de usuario
│   ├── cabecera.php ..... head de la aplicación
│   ├── index.php
│   ├── Inicio de sesión.php ..... Formulario de inicio de sesión
│   └── menu.php ..... Menú de secciones
├── normal/
│   ├── componentes/ ..... Elementos de control e indicación
│   ├── css/ ..... Hojas de estilo
│   ├── img/ ..... Imágenes
│   ├── js/ ..... Ficheros JavaScript
│   ├── peticionesAjax/ ..... Script de atención a petición Ajax
│   ├── plugins/ ..... Plugins de terceros
│   ├── secciones/ ..... Contenedores de secciones
│   ├── acceder.php ..... Validación de usuario
│   ├── cabecera.php ..... head de la aplicación
│   ├── index.php
│   ├── logueo.php ..... Formulario de logueo
│   └── menu.php ..... Menú de secciones
├── config.php ..... Fichero de configuración global
└── index.php
```

Se puede observar que la aplicación móvil y la ordinaria son totalmente independientes entre ellas. Tal vez no sea lo más práctico a la hora de tener que añadir una funcionalidad, pero se ha decidido que debía de ser así porque son lo suficientemente distintas en funcionalidad, diseño, comportamiento y necesidades como para considerarlas aplicaciones diferentes. Por ejemplo, el hecho de tener que detectar eventos de pulsación con el dedo en una pantalla

táctil en vez de un click con el ratón cambia mucho la programación de comportamiento y gestión de dicha detección.

Una vez expuesto el árbol de directorios de las aplicaciones se va a proceder a comentar cómo se ha implementado los distintos elementos de las aplicaciones. La idea básica es modular la aplicación en partes correspondientes a cada uno de las secciones, y dentro de cada una de estas secciones añadir componentes que representan los indicadores, controles, etc...

En el directorio */componentes/* se encuentran todos y cada uno de los elementos que van a aparecer en las secciones. Todos ellos son reutilizables entre secciones, es decir, que el reloj que aparece en la sección "salón" en el mismo que aparece en la sección "cocina". De esta manera, si en un futuro se desease hacer una administración gráfica para añadir y configurar la interfaz web, se podrá hacer sencillamente con un sistema de arrastre de elementos, de la misma manera en la que se arrastran los *widgets* en un entorno *Wordpress* (sistema de gestión de contenidos enfocado a la creación de sitios web periódicamente actualizados).

Cada una de las secciones que aparecen en la aplicación están alojadas en el directorio */secciones/* y son añadidas en el fichero *index.php* dependiendo de la parte del menú que se haya seleccionado. Este último y el *head* de la web son independientes y añadidos también en el fichero *index.php*. Por tanto, prácticamente todas las urls de la aplicación apuntan a este index y es este el que toma las decisiones sobre qué mostrar en cada momento. Por último, existe un fichero de configuraciones generales que contiene, entre otras cosas, los datos de acceso a la base de datos. Del mismo modo, el fichero contiene la información para que, en caso de producirse un cambio que induzca una tarea repetitiva en todo el entorno, dicho cambio se pueda hacer sencillamente desde ese mismo fichero.

El método de *login* o inicio de sesión consiste en un array de nombres y contraseñas definido en el fichero de configuración que es comparado con los datos que el usuario introduce a la hora de logarse. Si dicha información es válida y coincide con alguno de los usuarios autorizados, se crea una variable de sessión que contiene un valor *true* en un índice *válido*. Definir una variable de datos de usuarios no es, a priori, la solución más común, pero resulta la opción más sencilla en caso de necesitar dar de alta nuevas personas en el control de la casa.

Finalmente, esta aplicación dispone de un plugin de reconociento de dispositivos cliente (ya sea entorno web o móvil) que permite diferenciar que disposición de los elementos (mandos necesarios para administrar las variables controlables) es necesaria en la interfaz correspondiente a cada uno.

En el desarrollo de ambas aplicaciones, tanto móvil como ordinaria, se ha optado por utilizar la librería *JQuery* con el fin de aprovechar las ventajas que ofrece a la hora de manejar los documentos HTML.

5.1.2. Aplicación Móvil

En lo referente a la aplicación movil, desde un punto de vista de diseño web, la principal característica para que el usuario encuentre cómoda la navegación en una web es hacer accesibles los elementos de control en zonas donde los dedos tengan facilidad de llegar a pulsar. Ademas los elementos que muestran información deben estar localizados en zonas que no quedan tapadas a la hora de manejar la interfaz (Figuras 5.1 y 5.2). Esta filosofía no es una regla estricta de diseño pero se debe respetar dentro de lo posible.

Teniendo en cuenta estas premisas, se ha realizado una aplicación web pensada para dispositivos móviles utilizando un *diseño responsive* o *diseño adaptable*,

para que la interfaz posicione los elementos de la misma en determinados lugares dependiendo de su utilización horizontal (Figura 5.4) o vertical (Figura 5.5).

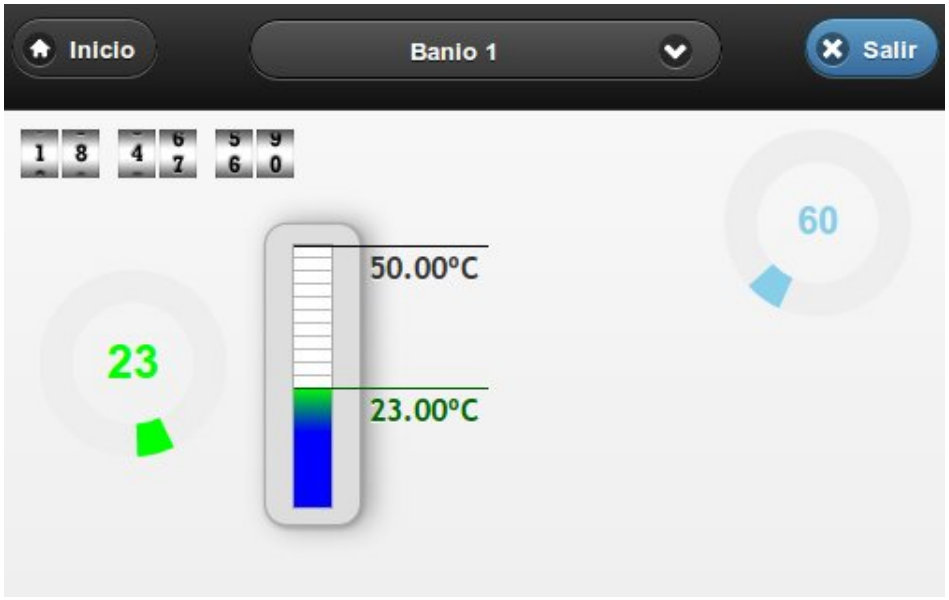


Figura 5.4: *Ejemplo de Pantalla de Control de Variables Ambientales de una Habitación de la Vivienda para un Dispositivo de Pantalla en Horizontal*

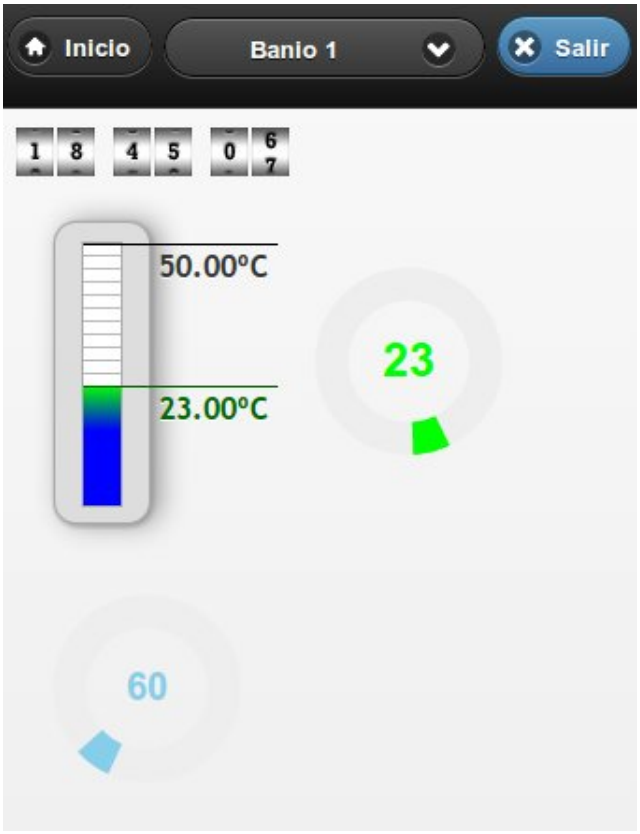


Figura 5.5: *Ejemplo de Pantalla de Control de Variables Ambientales de una Habitación de la Vivienda para un Dispositivo de Pantalla en Vertical*

Para los mandos de control, se ha utilizado un plugin llamado *jQuery Knob*.Este plugin contiene una librería que ofrece un slider circular (también conocido como barra deslizante) idóneo para las pantallas táctiles. Lo único que hay que hacer para generar el mando es crear un elemento *div* (elemento en HTML) con un determinado id (identificador), y por medio de *JQuery* llamar a una función con parámetro de configuración para que el plugin se ocupe de pintar el potenciómetro. También dispone de identificadores de eventos que permiten leer el valor fijado cada vez que el mando es movido por el usuario. De esta manera se puede extraer dicho valor y realizar las acciones pertinentes.

Con el fin de que la aplicación tenga un aspecto adecuado para los dispositivos móviles se ha utilizado *jQuery Mobile*. Se trata de un framework que marca una pautas de maquetación sencillas y se ocupa de generar elementos con aspectos determinados al fijar atributos *class* (referencias que ofrecen la posibilidad de asignar estilos, características o comportamientos específicos). De esta manera, se asignan los estilos oportunos a cada elemento.

5.1.3. Aplicación Ordinaria

Para esta parte de la aplicación web no hay puntos de diseño tan importantes que determinen una posición adecuada para los distintos elementos de la interfaz. Por ello, el posicionamiento de estos se ha fijado bajo criterio propio (Figura 5.6).

En primer lugar, se puede dividir la aplicación base en dos partes: Menú y Contenido.

El primero se encuentra a la derecha de la pantalla, y contiene el acceso a todas las secciones de la aplicación.

El segundo se encuentra a la izquierda de la pantalla y muestra los mandos de control e indicadores de información correspondientes a la sección en la que se encuentra el usuario.

Al contrario de la versión para dispositivos móviles, en la versión ordinaria no es necesario variar la posición de los elementos dependiendo del tamaño o posición de la pantalla. Por tanto, la web se mantiene siempre con la misma presentación independientemente del navegador utilizado.

En este caso, el framework utilizado para los estilos de la aplicación es

Bootstrap (16) el cual es un framework de software libre para diseño de sitios y aplicaciones web. Éste contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de *JavaScript* opcionales adicionales. De la misma manera que con *jQuery Mobile*, se deben aplicar ciertos valores a los atributos *class* de los elementos de la web para que el plugin se ocupe del resto. Para la colocación de cada elemento, *Bootstrap* divide el *site* (página del entorno) en un número variable de columnas de mismo tamaño, por defecto 12 y un número de filas variable. De esta manera se consigue hacer un sistema de rejilla mediante el cúal podremos posicionar los elementos deseados. Cuando se quiere añadir un elemento, como podría ser un título, se asigna un offset que asigna la posición donde se quiere colocar dicho elemento y se asigna un tamaño en función de las columnas que se quiere que ocupe dicho elemento. Así se consigue, entre otras opciones, estructurar los diferentes bloques definidos por *bootstrap*del sitio web a desarrollar. Posteriormente dichos bloques son rellenados por botones, barras deslizantes, menús, etc.

Figura 5.6: *Ejemplo del Bloque de Control de la Aplicación Ordinaria*

5.2. Petición de Datos al Servidor

Hay distintos momentos en los que la aplicación muestra datos que pueden variar mientras la interfaz se está mostrando en el cliente http. De este modo, el usuario puede variar la configuración de la información que está visualizando. Un ejemplo del primer caso es la funcionalidad de mostrar la temperatura ambiental de la zona elegida en tiempo real. Un ejemplo del segundo caso es la sección donde se muestran los datos recogidos en la monitorización de la viviendo en forma de gráfico.

5.2.1. Muestra de Información en Tiempo Real

En esta parte sólo se ha modificado el área que parte desde la interfaz web hasta el Servidor Apache. Así que se fijará una configuración en la que se dará por hecho que el *Raspberry Pi* dispone de la información solicitada por el cliente en el momento de la petición.

Gracias a PHP, las aplicaciones web pueden ser dinámicas. Es decir, que un solo fichero html puede variar su contenido dependiendo de las condiciones en las que se encuentre el entorno. En esta situación, PHP permite utilizar estructuras de control e infinidad de funciones para fijar un determinado contenido y enviarlo al cliente. No obstante, este tipo de dinamización resulta problemática cada vez que un fichero cambia gracias al script PHP. Si el cliente requiere una recarga de la página, dicho cambio produce una petición al servidor y una respuesta que contiene la página completa. Dicho de otra manera, cada vez que PHP cambia el contenido de una pagina y se plasma en el cliente, la totalidad del contenido se reinicia y se produce el mismo efecto que si se estuviese navegando a otra página.

Para una funcionalidad de muestra de datos a tiempo real, este no es un comportamiento adecuado, ya que es una molestia que la interfaz se actualice cada vez que el servidor envía el dato. Pero hay otra alternativa. De alguna manera se necesita pedirle al servidor la información sin que ésta tenga forma de página web a cargar.

La librería JQuery dispone de una función que permite hacer peticiones al servidor en segundo plano, sin que esto se plasme en lo mostrado por el cliente web. Esta función se llama *Ajax* y con esto se consigue modificar el contenido de la web sin que haya recarga.

Aplicado a este proyecto, la solución mas eficaz es pedir al servidor la información necesaria en un periodo de 30 segundos o 1 minuto, y una vez recibida la respuesta, se modificará el contenido con Javascript.

La única información que no depende del sistema y que debe mostrarse a tiempo real es la temperatura ambiente. Este dato se muestra en grados centígrados y en un termómetro virtual con valor máximo de 50 grados.

5.2.2. Muestra de Información de Monitorización

Aunque no se trate de una situación en la que la información se muestre en tiempo real, se ha decidido recurrir a la técnica de desarrollo web *Ajax* también para que la interfaz tenga un aspecto más moderno al no recargar la página cada vez que se valida el formulario de configuración de la gráfica.

Al pulsar el botón del formulario que actualiza la gráfica, se envía la configuración al servidor en una trama *JSON* (formato ligero de Java Script para el intercambio de datos) y este hace las peticiones pertinentes a la base de datos

para sacar los datos acordes con la configuración y los devuelve al cliente donde este los plasma en la gráfica.

El formulario dispone de 4 campos que son modificables:

- Zona Deseada: Se trata de la zona de la vivienda que se desea visualizar la información. Coincide con el controlador sensor que le corresponde. Se representa con un elemento *select* el cual inserta un desplegable de opciones en el formulario y un *option* el cual define una opción dentro de un menú desplegable creado por el elemento *select*, por cada controlador o zona. Dichas zonas se leen de la base de datos antes de ser mostradas.
- Fecha de Inicio y Fecha de Fin: El rango de tiempo que se desea visualizar. Al menos uno de los campos ha de ser completado. En caso de que los dos campos sean rellenados, los resultados mostrados serán los que estén comprendidos entre dichos valores de tiempo. Por o contrario si el campo de inicio es el único fijado, los datos mostrados serán los que estén comprendidos entre dicho valor y el momento actual. Y en caso de que el campo final sea el único fijado, se mostrarán los valores correspondientes al día indicado. Los campos se componen de día, mes, año y hora.
- Variables a Mostrar: Se trata de la variable que se desea ver, solo se puede seleccionar uno, y se elige por medio de un campo con *checkboxes*.

Capítulo 6

Procesamiento de Datos en el Servidor

En esta parte del proyecto existen múltiples posibilidades que pueden ser implementadas. Al poder ejecutar aplicaciones en un dispositivo con un sistema operativo y una capacidad de procesamiento y almacenamiento muy grande, se puede definir cualquier comportamiento cuyas decisiones serán posteriormente enviadas como órdenes sencillas. Para entenderlo, se puede decir que encender una luz a una cierta hora fijada es sencillo, pero tomar las decisiones de a qué hora, con qué frecuencia y dependiendo en base a qué datos, es algo más complicado que requiere un sistema más amplio que un μcontrolador de propósito general. Por ello, todas estas acciones se dejan en manos del servidor.

Estas decisiones pueden ser muy variables, dependiendo de lo que se desee controlar, pero para poder adaptarlas a cada uno de los usuarios, o viviendas, es necesario monitorizar lo que ocurre día a día y actuar en consecuencia en los casos que se repiten con una cierta frecuencia. Si una persona tiene una rutina laboral y sale de casa a una cierta hora y entra a otra, la vivienda debe de poder

adelantarse a estos hechos para controlar las variables ambientales que necesitan tiempo para ser óptimas. Por ejemplo, si en invierno el usuario sale de casa para ir a trabajar, la vivienda debe de ser capaz de apagar la calefacción sin que usted tenga que preocuparse de ello, y lo mismo pasa al volver, la vivienda debe de adelantarse a su llegada para que la casa esté caliente cuando llegue sin haber tenido que tenerla encendida todo el día.

6.1. Monitorización

Esta funcionalidad es muy interesante en el sistema, ya que es una de las bases para la mejora y solución de necesidades que tenga el usuario de la vivienda. Esta monitorización consiste en el almacenamiento de datos recogidos por los sensores con un cierto periodo y las acciones más relevantes que se produzcan tanto en la configuración, como en el día a día de las personas presentes en la casa.

La información se almacena en una Base de Datos gestionada por un servidor MySQL. Se compone de tres tablas, descritas a continuación.

- Tabla *zonas*:
  - *id*: Identificador único de zona, su incremento es automático cuando se añade un dato nuevo y es la clave principal de la tabla. Es de tipo *SmallInt*.
  - *nombre*: Nombre de la zona (por ejemplo "posición persiana salón", "intensidad luz baño 1", ...), es un campo de tipo *VarChar* que permite 50 caracteres.
  - *descripcion*: Descripción de la zona, es un campo de tipo *VarChar* que permite 250 caracteres.

- *idcontrolador*: Identificador del controlador que aporta el dato registrado, coincide con la dirección de red del mismo. Es de tipo *SmallInt*.

- Tabla *variables*:

  - *id*: Identificador único de variable, su incremento es automático cuando se añade un dato nuevo y es la clave principal de la tabla. Es de tipo *SmallInt*.
  - *nombre*: Nombre de la variable, es un campo de tipo *VarChar* que permite 50 caracteres.
  - *descripcion*: Descripción de la variable, es un campo de tipo *VarChar* que permite 250 caracteres.
  - *alias*: Nombre de identificación de la variable, es un campo de tipo *VarChar* que permite 50 caracteres.
  - *unidad*: Nombre de la variable (por ejemplo "grados centígrados", "luxes", ...), es un campo de tipo *VarChar* que permite 50 caracteres.
  - *simbolounidad*: Simbología de la unidad registrada (por ejemplo "ºC", " %", ...), es un campo de tipo *VarChar* que permite 10 caracteres.

- Tabla *registrovars*:

  - *id*: Identificador único de dato registrado, su incremento es automático cuando se añade información nueva y es la clave principal de la tabla. Es de tipo *BigInt*.
  - *idvariable*: Identificador de la variable que representa el dato, coincide con la clave única de la tabla *variables* correspondiente. Es de tipo *SmallInt*.
  - *valor*: Valor del dato registrado, es de tipo *Decimal* de 15 cifras de tamaño, de las cuales 5 componen la parte decimal.

- *fecha*: Fecha de registro del dato, es de tipo *TimeStamp* y su valor por defecto es el valor del tiempo en el momento en el que se registra el dato.
- *idzona*: Identificador de la zona a la que pertenece el valor registrado, coincide con la clave única de la tabla *zonas* correspondiente. Es de tipo *SmallInt*.

En este prototipo, la monitorización consiste en el registro de la temperatura y la intensidad lumínica de la habitación donde se encuentra la placa sensora.

En este escenario entran en juego el sistema *Raspberry Pi* y un controlador sensor. Cada minuto, el primero envía al segundo una petición de dato a recoger por uno de los sensores. Al recibir el destinatario dicho mensaje, éste toma el valor de la variable ambiental midiendo la tensión a la salida del sensor por medio del ADC. Una vez el valor ha sido recogido, lo convierte en el resultado correspondiente a la unidad de la variable medida y envía una respuesta al dispositivo que ha pedido el dato. Al recibir la contestación, el servidor guarda el valor en la base de datos. Acto seguido, se vuelve a repetir la secuencia de acciones para medir otra variable. Esto ocurre tantas veces como variables distintas son capaces de medir los sensores.

Los mensajes enviados en esta funcionalidad tienen un formato común donde se indica lo necesario para que cada dispositivo sepa de qué tipo de mensaje se trata, cuál es el destinatario y, en caso de contener un valor, la representación del mismo.

Esta estructura es la siguiente:

- Mensaje de Petición: *[dirección de destinatario]_[comando de petición]*

- Mensaje de Respuesta: *[dirección origen]_[comando de petición a la que se responde]_[valor]*

Para el mensaje de respuesta, no sería necesaria la dirección de destino, puesto que el dispositivo que ha hecho la petición espera la respuesta de un sensor y un comando determinados. Este aspecto resulta relevante en el momento en el que varios dispositivos hacen la misma petición al mismo tiempo. En este caso, el sensor solo necesitaría enviar la respuesta una sola vez, puesto que ninguno de los dispositivos que preguntan descartaría el mensaje de vuelta.

A continuación se va a describir las pruebas realizadas enumerando las configuraciones de cada dispositivo (Figura 6.1) y la secuencia de acciones realizadas (Figura 6.2).

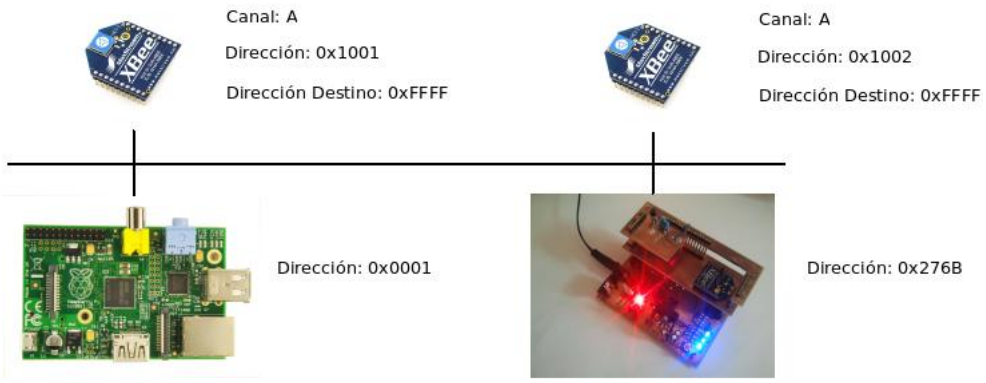


Figura 6.1: *Configuración de Comunicación del Sistema Monitor*

Una vez los valores son recibidos por el sistema *Raspberry Pi*, éste guarda el resultado en la base de datos que tiene instalada y que ha sido descrita anteriormente. Para ello, se ha utilizado una API (interfaz de programación de aplicaciones), llamada sencillamente *MySQL C API*, para comunicarse con el servidor MySQL gracias a una serie de funciones que permiten conectarse y gestionar facilmente una base de datos de dicha tecnología.

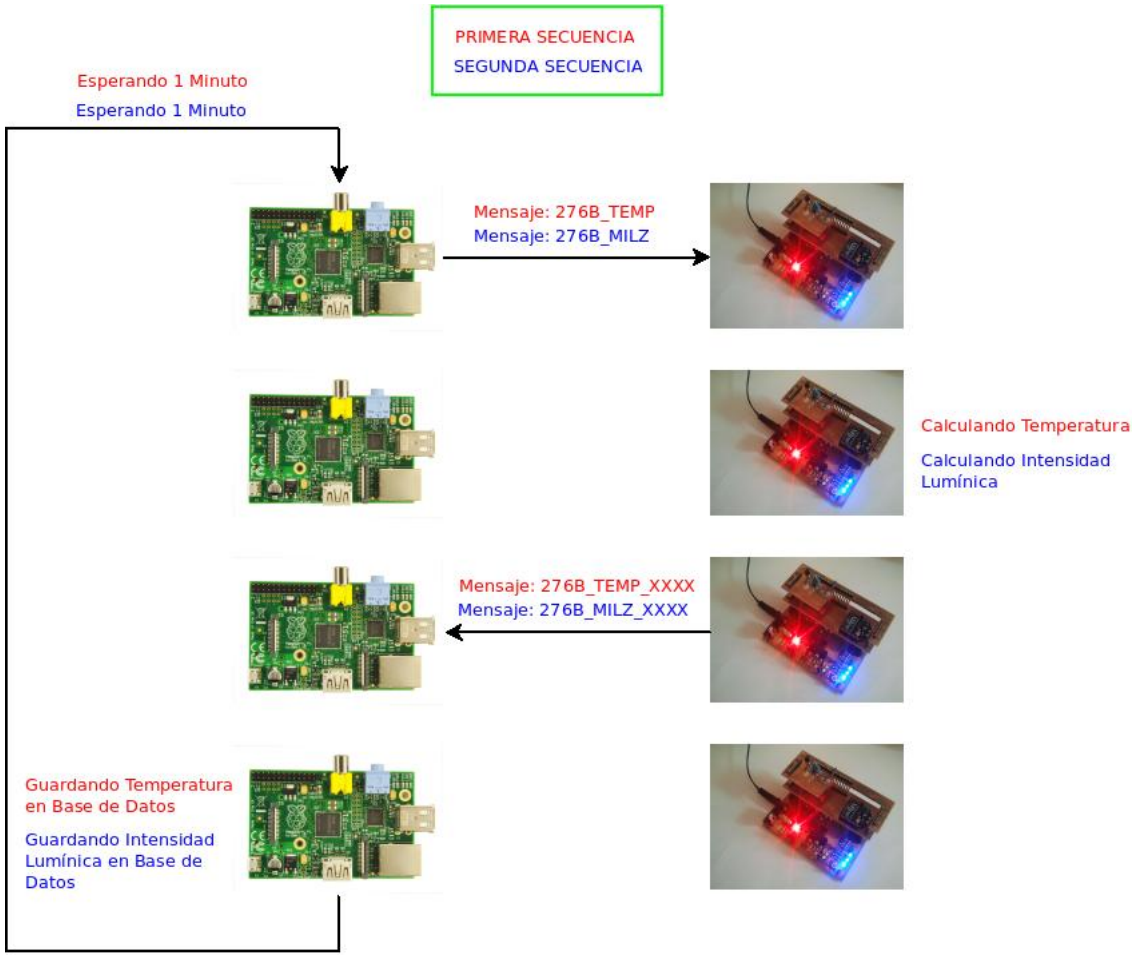


Figura 6.2: *Secuencia de Acciones del Sistema Monitor*

Los datos enviados para registrar el valor medido son el identificador de la zona medida que coincide con la dirección del controlador sensor que envía el dato, el valor de la variable recogida y el identificador de la variable que se ha medido, valor que determina el servidor de la base de datos cuando se da de alta un nuevo tipo de variable. La fecha y hora no son necesarias puesto que este registro de la tabla se define por defecto con el valor *TimeStamp* del momento en el que se guarda el dato.

6.2. Control de Intensidad Lumínica (Tira de Leds)

En esta parte del proyecto se pretende controlar, por medio de la interfaz web, la iluminación de una zona por medio del control de intensidad de una tira de LEDs variando el ciclo de trabajo con el que se conmuta dicha fuente de luz.

En la parte que concierne al servidor, cuando se modifica el valor del control de intensidad lumínica a través del slider circular (o barra deslizante) se genera un dato que es enviado por el dispositivo cliente (por ejemplo un smartphone) a un socket(17) de escucha del servidor y que es recibido por éste en forma de paquete *http*. Acto seguido, el servidor procede a la extracción del mensaje orden y su interpretación de manera que, en función de la orden, se envíe dicho mensaje al controlador correspondiente y éste actúe en consecuencia fijando la intensidad lumínica deseada. Dicho mensaje se envía mediante el módulo de comunicación wireless, añadido a la *Raspberry Pi*, que es recibido por el módulo de comunicación wireless añadido al controlador, el cual lo transmite a través de su *uart* conectada con la *uart* del μcontrolador. Mediante al algorítmo de programacíon que tiene grabado y almacenado en su memoria flash actúa en consecuencia con el mensaje recibido.

Las tareas que realiza la *Raspberry Pi* para cumplir con estas funciones son las indicadas en el siguiente diagrama.

Como se ha explicado en el capítulo Interfaz Web y Comunicación con el Servidor, las peticiones de control o de muestra de información al servidor se realizan por medio de *Ajax*, pero en este caso no se envían al Servidor Apache. Después de un estudio de tiempo de respuesta, se detectó que enviando las peticiones al software servidor, el tiempo de actuación aumentaba ligeramente. A simple vista, tratándose de tiempos del orden de mili segundos, parecía no

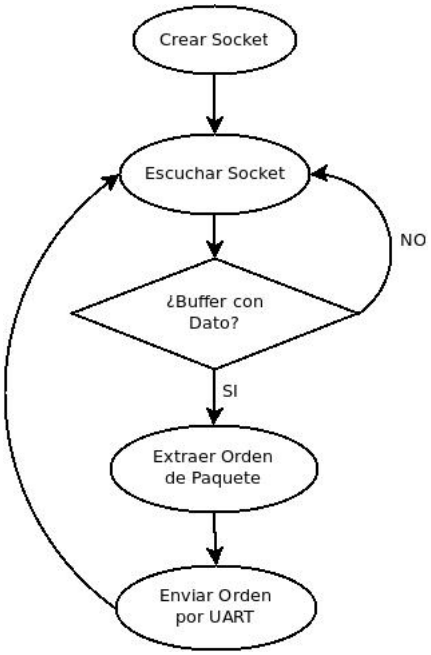


Figura 6.3: *Secuencia de Acciones del Control de Intensidad Lumínica*

tener importancia, pero en este caso, la funcionalidad era crítica, y el número de peticiones era muy alto. Hay que tener en cuenta que la modificación de la intensidad lumínica debe ser en directo, así el usuario puede saber sencillamente que iluminación es la de su gusto. Por ello, el servidor debe cambiar la intensidad al mismo tiempo que el usuario modifica el potenciómetro de la interfaz. Si se tiene un slider (o barra deslizante) que representa la intensidad en porcentaje, de 0 % a 100 % siendo el primero el estado de luz apagada y el segundo el de luz a máxima potencia, un usuario que mueve el control de 0 % a 50 % en medio segundo, significa que se enviarían 50 peticiones en ese tiempo, lo que exige un tiempo de respuesta de 10 milisegundos como máximo. Por ello, la petición se envía directamente a un socket, es decir, una zona virtual designada por la cual dos programas pueden intercambiar información, en un puerto determinado que es escuchado por el software de gestión del sistema del *Raspberry Pi*. Aún así el cuello de botella presente en el sistema es la comunicación entre cliente y servidor por medio del router WIFI. Por tanto aún tomando esa medida, el cambio de

iluminación no es demasiado preciso. Para mitigar ese efecto, se ha fijado el envío de petición de intensidad lumínica en tramos de 5 % de la intensidad máxima. Así, si el usuario modifica de 0 % a 50 % la intensidad, solo se envian 10 mensajes en vez de 50. Esto permite que el tiempo de respuesta máxima del servidor aumente a 50 milisegundos. Debido a que las comunicaciones por WIFI no son muy estables, de vez en cuando se puede tener la sensación de que la luz varía con un ligero parpadeo intermitente.

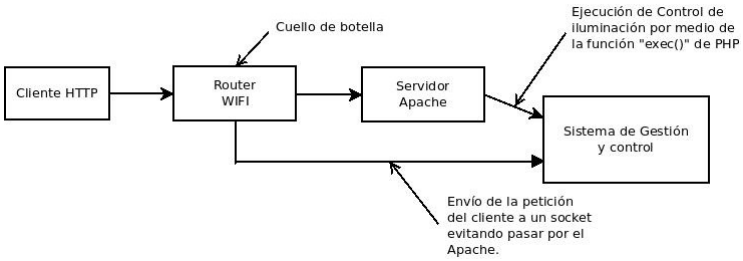


Figura 6.4: *Petición de Cliente por Medio de Socket*

6.3. Envío de Dato de Temperatura Ambiental

Como se describió en el capítulo *Interfaz Web y Comunicación con el Servidor*, hay una funcionalidad para el control de temperatura, en la que se muestra la temperatura ambiental actual. Para ello, se envía al servidor una petición de valor de dicha variable con un periodo determinado, para posteriormente recibirlo y mostrarlo en la interfaz.

De la misma manera que con la intensidad lumínica, el servidor deberá tomar el mensaje del cliente web, pero esta vez enviado al Servidor Apache. Esto debe ser así porque en este caso, la petición produce una respuesta que el cliente debe recibir. Cuando el servidor reciba la petición, este le enviará el mensaje al controlador sensor correspondiente y una vez reciba la respuesta, enviará el valor

al cliente para que este actualize la interfaz.

Capítulo 7

Controladores

7.1. Software de Diseño de PCB's

Con el fin de llevar a cabo el proyecto aqui descrito se ha recurrido al desarrollo de diversas placas provistas de un μcontrolador. Para ello ha sido necesario utilizar diversas herramientas informáticas para facilitar el diseño de la circuitería impresa necesaria para poder desarrollar placas controladoras que informaran al servidor de la situación actual o que actuarán en función de lo que el servidor ordene.

Conocido el manejo de distintos softwares de diseño de PCB's como puede ser Orcad, KiCad, DesingSpark se decidió usar un software llamado EAGLE (Figura 7.1). Las principales ventajas del manejo de Eagle frente a otros sotwares de diseño son:

- La facilidad en su manejo, ya que tanto el editor de esquematicos, layouts y librerias disponen del mismo entorno gráfico.

- La compatibilidad con diversos sistemas operativos, ya que puede trabajar tanto en Windows, Mac y Linux.
- Los escasos requisitos necesarios exigidos por el software para su correcto funcionamiento e instalación.
- Servicio de soporte por expertos de la compañia mediante el foro de su pagina web.

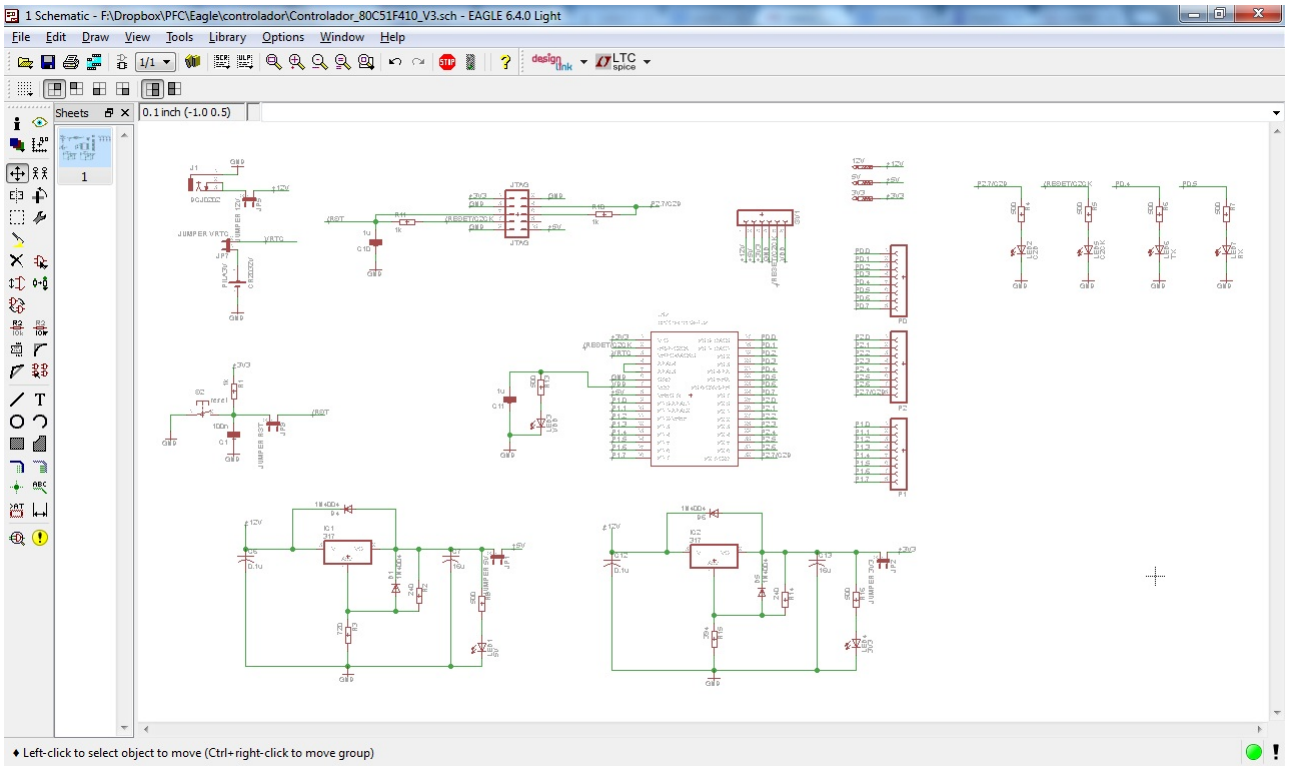


Figura 7.1: *Software de diseño de PCB's Eagle*

Las fuentes para el diseño de nuestra fueron diversos tutoriales conseguidos por internet, el diseño de la ya citada anteriormente de la asignatura ISE y cálculos y desarrollos realizados por cuenta propia como podría ser el diseño de los circuitos reguladores los cuales han sido estudiados en la asignatura Circuitos Electrónicos.

7.2. Controladores

Debido al previo estudio y manejo del μcontrolador 80C51F320 de Silicon Laboratories en la asignatura ISE, se decidió usar un μcontrolador parecido con fin de poder implementar nosotros la placa controladora sin tener que usar una ya diseñada, fabricada y comercializada (tipo Arduino), pero con las facilidades del previo conocimiento de su manejo tanto en concepto de disponibilidad de periféricos, manejo de los registros, programación, etc .

En este caso se decidió usar el modelo de μcontrolador 80C51410 (18) que incluye ciertas mejoras, como puede ser un reloj a tiempo real (Sma Real Time Clock) el cual es un reloj de bajo consumo de 47 bits que permite la cuenta de hasta 137 años con el añadido de que incluye alarmas, 32Kb de memoria interna programable en vez de los 16Kb disponibles de su modelo inferior y un conversor ADC de 12 bits en vez de los 10 bits de los que dispone el modelo inferior. También se escogió dicho μcontrolador con el fin de realizar un estudio previo en su manejo y programación para así ampliar conocimientos a la hora de utilizar diversos μcontrolador en un futuro.

Otro de los motivos por los que se ha elegido dicho μcontolador es por que es capaz de gestionar un sistema multitarea (hasta 16 tareas simultáneas) utilizando el sistema operativo RTX51 Tiny 51 del cual se hablará en el capítulo Software

de Programación y Depuración. Debido a la necesidad de gestionar la recepción de datos (*uart*) por parte de los controladores de manera inesperada, ya que el controlador no es capaz de saber cuando el servidor envía órdenes, se ha tenido que utilizar un sistema de gestión multitarea de manera que éstos sean capaces de recibir datos mientras que ejecutan otras tareas. Gracias a este sistema multitarea se consigue la ventaja frente a los sistemas que manejan una única tarea de, por ejemplo, poder estar ejecutando la tarea de recepción de datos (*uart*) a la vez que se ejecuta la tarea que modifica la iluminación de una habitación mediante la tira de leds.

La placa diseñada funciona como un controlador genérico el cual es válido para la realización no sólo de nuestro proyecto, sino que ésta ha sido diseñada con el fin de poder adaptar PCB's adicionales de manera que se puedan insertar o extraer, dando pie a distintas funcionalidades de nuestro μcontrolador, o en su defecto ampliar el margen de funcionamiento de dicho proyecto.

Cabe destacar que algunos de los diferentes módulos que componen la placa controladora, como pueden ser los reguladores que forman la fuente de alimentación, el circuito de reset, la pila para el SmartClock o conexion al transformador de alimentación externa de 12V han sido aislados del circuito general mediante jumpers con el fin de poder separarlos del μcontrolador. También se han añadido puntos de test donde se podrá verificar el correcto funcionamiento de los módulos de alimentación formados por los reguladores de tensión.

A parte de haberse diseñado y desarrollado la placa controladora, también se han diseñado y desarrollado varias PCB's adicionales y extraíbles como pueden ser:

- De comunicación wireless con un módulo Xbee que utiliza el protocolo

ZigBee

- Sensora, con sensores de temperatura y luz.
- Actuadora, con relés para la activación de luces leds, motores para persianas, etc.

De esta manera se diseñáran y desarrollarán dos módulos controladores por habitación. Uno de ellos será un módulo sensor formado por una placa controladora, una placa de comunicación wireless y una placa sensora, encargado de tomar medidas de luz y temperatura para posteriormente comunicárselo al servidor y al módulo actuador. El otro módulo sera un módulo actuador, el cual estará formado por una placa controladora, una placa de comunicación wireless y una placa actuadora de tal manera que dicho módulo sera el encargado de recibir las ordenes del servidor o del módulo sensor para actuar en consecuencia y bajar/subir persianas, controlar la iluminación por leds, o activar/desactivar la calefacción de la habitación.

La Figura 7.2 muestra el diseño definitivo de la placa controladora

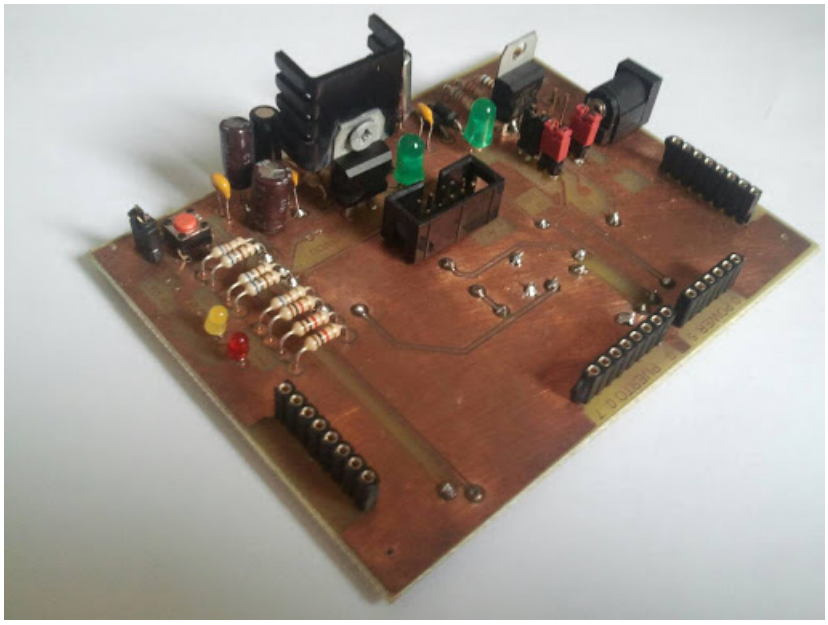


Figura 7.2: *Placa controladora*

7.2.1. Perifericos Digitales Utilizados por el μControlador

El μcontrolador 80C51F410 tiene varios periféricos digitales como pueden ser los puertos I/O, los timers (timer 0, timer 1, timer 2 y timer 3), PCA, UART, SMBus, SPI, Crossbar y CRC. Para la realización este proyecto se han tenido que usar varios perifericos para conseguir los objetivos deseados. Los diferentes perifericos que han sido utilizados son:

- En lo referente a los Puertos I/O se han utilizado los siguientes aquí citados:
  - P0.4 como transmisión (TX) del periférico UART para la comunicación con el módulo Xbee
  - P0.5 como recepción (RX) del periférico UART para la comunicación con el módulo Xbee
  - P1.0 como entrada analógica para el ADC y la conversión de la señal del sensor de temperatura MCP9701A

- P1.1 como entrada analógica para el ADC y la conversión de la señal del sensor de luz SFH5711
- P2.7 como pueto de programación y depuración

- Timer 0 : Usado por el sistema operativo (RTOS) RTX51 Tiny 51 en modo 1 para generar una interrupción periódica con el fin de generar Ticks que ocurren cada 10000 ciclos máquina que luego serán utilizados en funciones del RTOS.

- Timer 2 : Usado en modo autorecarga con el fin de generar una frecuencia de 60 Hz con la que luego modificando su dutty cycle se conseguirá mayor o menor iluminación.

- UART: Usado como periférico de comunicación con el módulo Xbee integrado en una placa adaptadora que nos permita conexionar sus pines de DataIn y DataOut con los pines RX(P0.5) y TX(P0.4) de nuestro μcontrolador para poder gestionar la comunicación wireless. Para ello se configura la *uart* con un *baudrate* determinado y generado por el Timer 1 en modo 2 de autorrecarga. (19)

- Crossbar: Configurado de tal maneras que se aislen los pines P0.4 (TX) y P0.5 (RX) consiguiendo así que dichos pines no se puedan ser utilizados para cualquier otra finalidad que no sea la transmisión y recepción de datos serie.

7.2.2. Perifericos Analógicos Utilizados por el μControlador

El μcontrolador 80C51F410 dispone de varios periféricos analógicos como pueden ser 2 conversores DAC (digital a analógico), ADC (analógico a digital),

AMUX (multiplexador), 2 comparadores de tensión, tensión de referencia, regulador de tensión y un sensor de temperatura.

Para la realización de dicho proyecto se han tenido que usar varios periféricos para conseguir los objetivos deseados. Los diferentes periféricos que se han utilizado han sido:

- ADC: Usado junto con el periférico analógico AMUX (multiplexador) con el fin de convertir las señales generadas por los sensores de luz y de temperatura para poder tratar y transmitir dichas señales y posteriormente reaccionar en función de los valores de dichas señales. Para ello se ha de usar el multiplexador (AMUX) de tal manera que en función del momento y la señal deseada se multiplexe una entrada analógica que será tratada posteriormente por el ADC y así permitir la posibilidad de convertir hasta 27 señales(cada una correspondiente a un pin de un puerto distinto), de los cuales solo se usarán 2 de ellos, uno para la señal del sensor de luz y otra para la señal del sensor de temperatura.
- Tensión de referencia: Utilizada con el fin de tener una referencia a la hora de poder convertir las señales analógicas a digitales ya sea la señal del sensor de luz o del sensor de temperatura.

7.2.3. Módulos que Forman la Placa Controladora

Para que el μcontrolador sea capaz de funcionar se han tenido que diseñar y desarrollar módulos adicionales que le permitan desarrollar sus funciones. A todo el conjunto se le conoce como placa controladora. La placa controladora esta compuesta de diversos módulos los cuales se pueden distinguir por: fuente de alimentación, circuito de reset, módulo de programación y depuración JTAG,

circuito de pila para SmartClock, puertos DIO y módulo de leds indicadores. A continuación se describirán mas detalladamente los distintos módulos de nuestra placa controladora:

- El módulo de la fuente de alimentación esta formado por dos reguladores de tensión modelo LM317(20) (Figura 7.3), los cuales entregan 5V y 3,3V respectivamente habiendo sido configurados anteriormente para poder entregar dichos valores. Con dichas tensiones se podrá proporcionar alimentación al μcontrolador, al módulo de comunicación wireless Xbee y al módulo de programación y depuración JTAG.

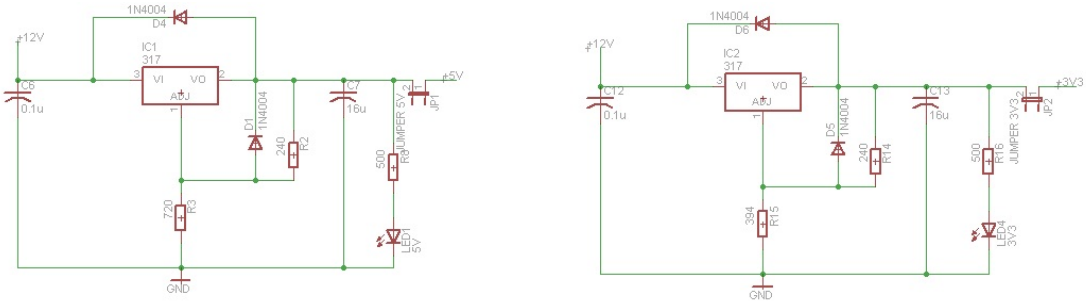


Figura 7.3: *Reguladores de la fuente de alimentación*

- El circuito de reset (Figura 7.4) esta formado por un pulsador y circuitería compuesta por condensadores, resistencia y leds para conseguir un nivel lógico bajo. Este circuito tiene el propósito de entregar al μcontrolador una señal que indique la necesidad de un reinicio de la placa controladora.

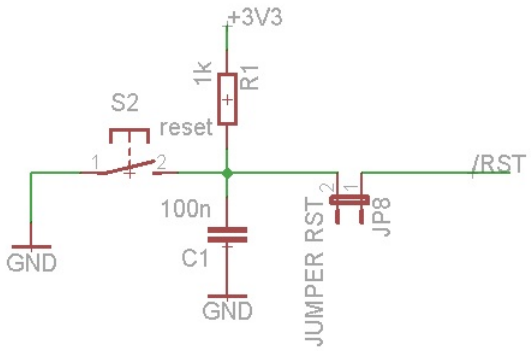


Figura 7.4: *Circuito de Reset*

- Módulo de programacion JTAG compuesto de un conector para cable paralelo de 10 pines (Figura 7.5) más diversos componentes como pueden ser resistencias y leds. A través dicho módulo se podrá programar la memoria flash del μcontrolador y depurar el código de programación desarrollado.

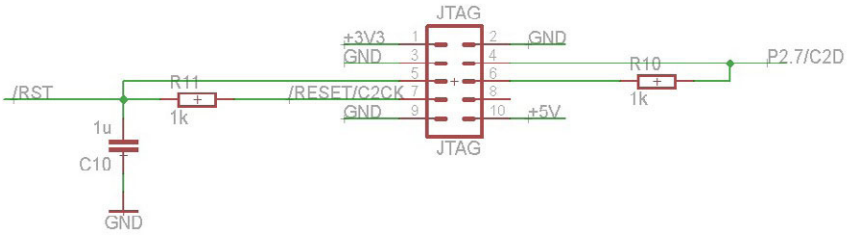


Figura 7.5: *Circuito de programación/depuración JTAG*

- Circuito de pila para SmartColck (Figura 7.6). De esta manera, el reloj no perderá la hora actual aunque se produzca una pérdida de la tensión de alimentación principal.

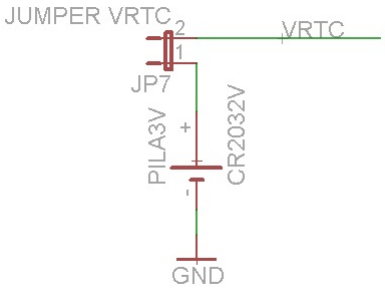


Figura 7.6: *Circuito de Pila para SmartClock*

- Los puertos de entrada/salida también conocidos como puertos DIO o Data Input/Output (Figura 7.7). Compuestos simplemente por conectores planos que permiten acoplar diferentes placas para diversas funcionalidades. Además se incluye un puerto adicional de alimentación independiente de los puertos DIO del μcontrolador donde se dispone de señales como 12V, 5V, 3V3, VDD, Reset y GND que permiten proporcionar diversas señales de alimentación a las distintas placas externas que se podrán conectar a la placa controlador.

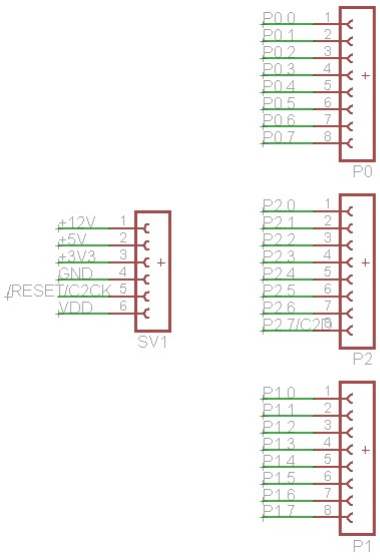


Figura 7.7: *Puertos DIO*

- El módulo de leds indicadores (Figura 7.8). Formado básicamente por sencillos componentes como leds y resistencias mediante los cuales se puede detectar si se transmite o recibe a través de la UART del μcontrolador o si se programa o depura a través del conector JTAG.

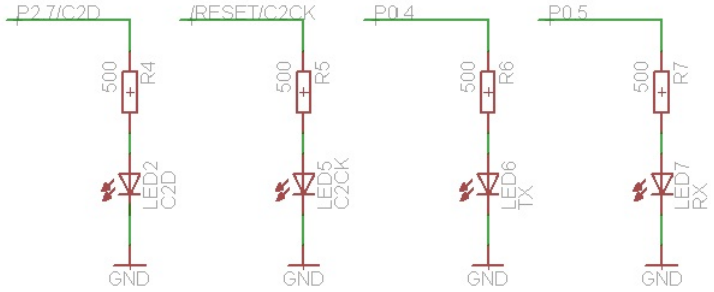


Figura 7.8: *Módulo de Leds Indicadores*

En la Figura B.1 localizada en el Anexo Esquematicos y Layouts de las Placas Desarrolladas se muestra el esquemático completo del circuito controlador.

7.3. Placas Complementarias

Para poder ampliar la funcionalidad de la placa controladora desarrollada fue necesario implementar y diseñar diversas placas complementarias. Dichas placas cumplen un diseño modular con el fin de que conecten unas encima de otras y, a su vez, quedan todas ellas conectadas con la placa controladora a través de sus puertos DIO. De esta manera la placa controladora hace las veces de placa base para las distintas placas complementarias. Como ya se comentó anteriormente es necesario tener dos placas controladoras distintas para cada habitación de la casa. Esto se consigue añadiendo unas u otras placas complementarias a nuestra placa controladora. Para conseguir dicho resultado se añade una placa de comunicación wireless con el módulo Xbee en ambas placas y por encima de dicha placa se añade una placa sensora o una placa actuadora consiguiendo así dos placas con distintas funcionalidades. Como generalidad, dichas placas cuentan todas ellas con un sistema de pines situados en la misma posición con el fin de que se puedan conectar unas con otras consiguiendo así el diseño modular deseado. Las diversas placas que se han tenido que diseñar y desarrollar para crear una placa controladora capaz de intervenir en nuestro sistema final son:

- Placa de comunicación wireless
- Placa sensora
- Placa actuadora

En la Figura 7.9 se muestra el sistema completo de una placa controladora con una placa sensora y una de comunicación wireless

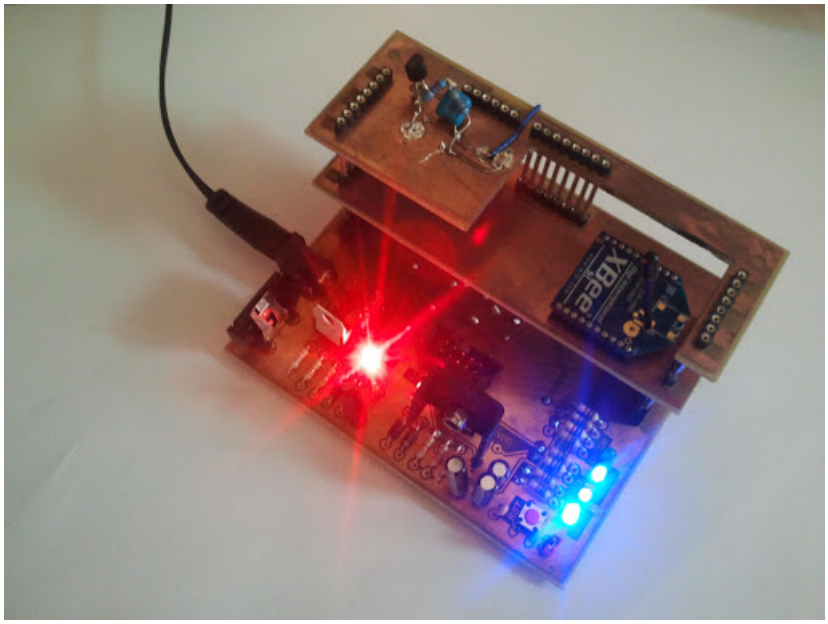


Figura 7.9: *Placa Controladora con Placa Sensora y de Comunicaión Wireless*

7.3.1. Placa de Comunicación Wireless

La placa de comunicación wireless (Figura 7.11) está compuesta por los pines encargados de conexionar dicha placa con la placa controladora además del conexionar éstos con los pines del módulo de comunicación wireless Xbee mediante el cual la placa controladora es capaz de transmitir o recibir mensajes.

Para ello sólo se debe disponer de:

- Una conexión entre el pin de entrada del Xbee (DIN) y el pin de salida TX de la UART del μcontrolador.
- Una conexión entre el pin de salida del Xbee (DOUT) y el pin de entrada RX de la UART del μcontrolador.
- Una conexión a alimentación de tensión a 3,3 V.
- Una conexión a masa.

Xbee: Módulo de Comunicación Wireless

Antes de continuar, se explicarán los aspectos básicos del módulo de comunicación wireless Xbee(21). Éste es un módulo que trabaja por radio frecuencia dentro de la frecuencia ISM a 2,4 GHz mediante el estandar IEEE 802.15.4 teniendo un alcance máximo entre los módulos de 30 metros.

A su vez el Xbee ofrece numerosas posibilidades y complejidades pero en el desarrollo de este proyecto el manejo y configuración tienden a la sencillez ya que se usan las menores conexiones posibles. Una de las ventajas de la utilización del módulo Xbee es su sencilla conexión y comunicación con el μcontrolador a través de la UART sin necesidad de componentes intermedios que transformen los niveles lógicos para la compatibilidad en la comunicación. En la Figura 7.10 se puede observar el método básico de conexión, apreciándose así la sencillez del sistema para conseguir establecer una comunicación entre dispositivos del mismo tipo.

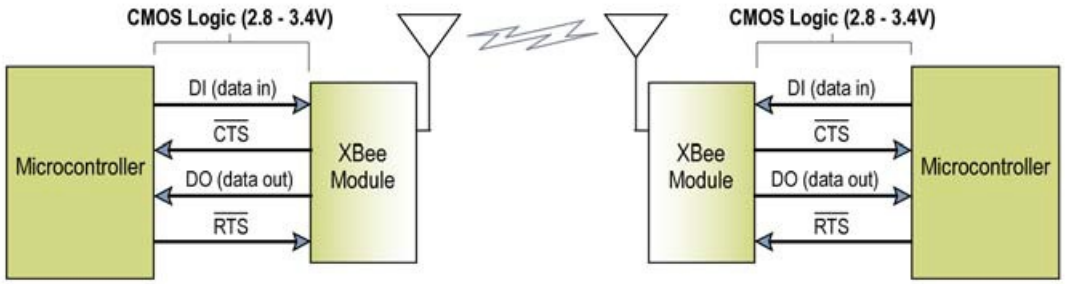


Figura 7.10: *Conexión para la Comunicación entre dos Xbee*

Para que los módulos Xbee de las distintas placas puedan comunicarse entre sí, es necesario cargarles una configuración mediante un software llamado X-CTU (22) en la que se especifiquen los parámetros de configuración como pueden ser el canal por el que van a transmitir los Xbee. Para poder justificar la elección del canal utilizado es necesario rastrear las diferentes redes Wifi que podemos

encontrar dentro de una misma vivienda. Para ello se puede utilizar un comando en la consola de cualquier sistema operativo:

- Netsh wlan show all (Consola de Comandos en Windows)
- Iwlist scan (Consola de Comandos en Linux)

Este comando proporciona la información de las diferentes redes Wifi que se encuentran al alcance indicando estándares, canales, tipo de red en cuestión, cifrado, etc. Atendiendo a estos parámetros se puede tomar la decisión de que canal utilizar para la comunicación de los distintos módulos Xbee. En este caso se ha elegido el Channel C, el cual trabaja en la frecuencia de 2.41 MHz, se aleja de las frecuencias de los extremos del estándar IEEE 802.15.4 y de las frecuencias de los canales de las diferentes redes Wifi detectadas en la vivienda donde se desarrolla dicho proyecto. El resto de parámetros de configuración del módulo Xbee que han sido establecidos son el Baudrate (9600 bps) que ha de coincidir con el Baudrate de la *uart* μcontrolador, su dirección de origen (source adress) y la dirección de destino (destination adress) en caso de requerir una conexión punto a punto, pero en este caso se transmite por Broadcast y se filtra por software dichas direcciones. En la comunicación por Broadcast todos los módulos en la red tienen la misma configuración de direccionamiento la cual será de la siguiente manera:

- DL (Destination Low Address) = 0x0000FFFF
- DH (Destination High Address) = 0x00000000 (valor por defecto)

Los mensajes transmitidos entre los Xbee tienen la siguiente estructura XXXXX_COMANDO_VALOR

Entre los distintos comandos utilizados se pueden encontrar:

- LZFR para modificar el dutty cycle y variar la ilumunación led.
- TEMP para obtener el valor medido por el sensor de temperatura.
- MILZ Para obtener el valor medido por el sensor de luz.

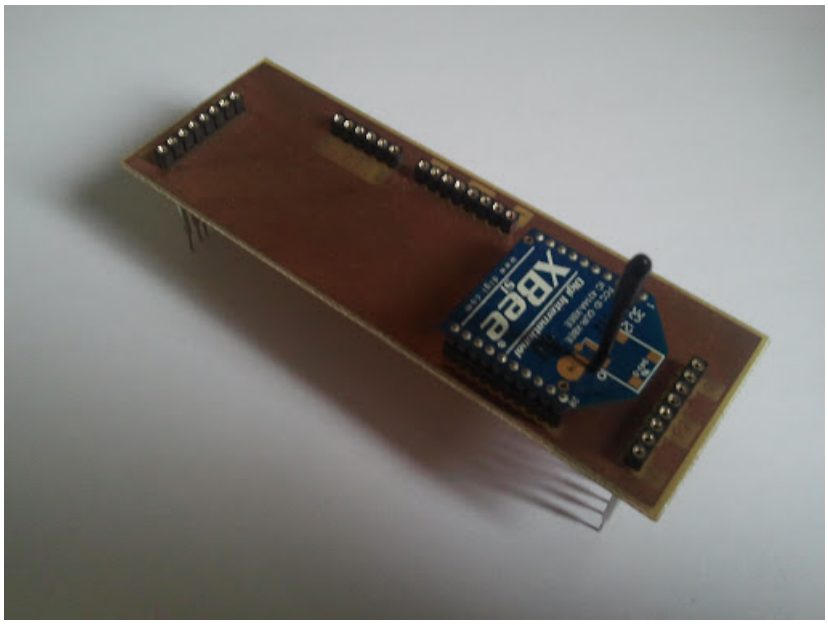


Figura 7.11: *Placa de comunicación wireless con módulo Xbee*

7.3.2. Placa Complementaria Sensora

La placa está compuesta por los pines encargados de conexionar dicha placa con la placa controladora además de un sensor de luz y un sensor de temperatura de los cuáles se obtienen las señales que posteriormente se convierten mediante el ADC del μcontrolador para actuar en consecuencia.

El sensor de luz elegido es el SFH5711(23) de la marca OSRAM, un fotodiodo que proporciona un valor de corriente, el cual equivale al número de luxes (medida de luminiscencia) de la luz de ambiente de la sala donde se encuentra

dicho sensor. Dicho fotodiodo está ajustado de tal manera que permite detectar el mismo cambio en la luminosidad que el ojo humano es capaz de percibir, con una precisión del 1 %. Este aspecto supone una gran ventaja de manejo, en lo que a aspectos domóticos se refiere.

La función de transferencia entregada por dicho sensor se puede obtener mediante la siguiente expresión:

$$Iout = S \times \log(Ev/Eo)$$

Dicho valor de corriente es controlado mediante una resistencia de salida de valor 44 Kohms que nos proporciona unos valores de tension comprendido entre 0V y 2,5V aptos para ser convertidos por nuestro ADC del μcontrolador.

En la gráfica adjunta (Figura 7.12) se puede visualizar dicha función de transferencia.

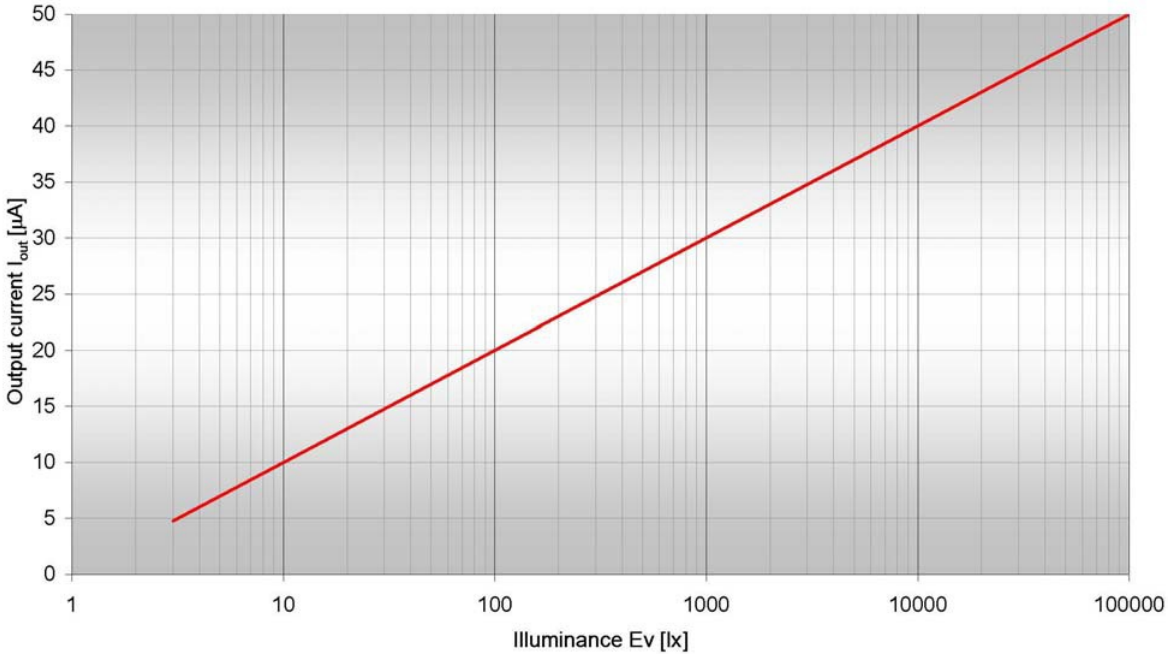


Figura 7.12: *Función de Transeferencia del Sensor de Luz*

Los valores orientativos de iluminación son:

- 50 lux → Sala de una vivienda familia
- 400 lux → Salida o puesta de sol en un día despejado
- 32.000 lux → Luz solar en un día medio (mín.)
- 100.000 lux → Luz solar en un día medio (máx.)

El sensor de temperatura elegido(24) es el MCP9701A(25) de la marca MICROCHIP el cual entrega un valor de tensión equivalente a la temperatura ambiente de la sala donde se encuentra dicho sensor con una precisión de $\pm 2^{o}C$ en un rango comprendido entre $0^{o}C$ a $70^{o}C$.

La función de transferencia entregada por dicho sensor se puede obtener mediante la siguiente expresión:

$Vout = Tc \times Ta + V(0^{o}C)$

Donde Tc es el coeficiente de temperatura, cuyo valor es de 19.5 $mV/^{o}C$.

En la gráfica adjunta(Figura 7.13) se puede visualizar dicha función de transferencia.

Resulta reseñable añadir que se podría haber utilizado el sensor de temperatura integrado en el μcontrolador. Pero con el propósito de obtener una medida de temperatura más fiable se ha utilizado un sensor externo a él, ya que el μcontrolador esta rodeado de diversos componentes que irradian calor, mientras que nuestro sensor de temperatura está más alejado de todas esas fuentes de calor gracias al diseño modular de las PCB's implementado.

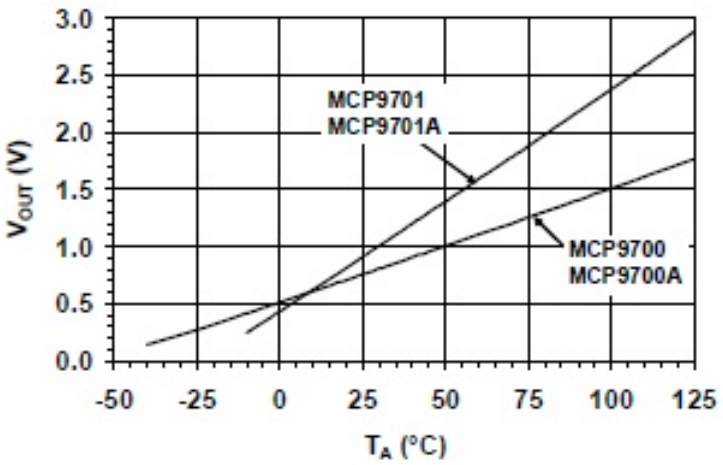


Figura 7.13: *Función de Transeferencia del Sensor de Temperatura*

A continuación se muestra una imagen de la placa sensora (Figura 7.14)

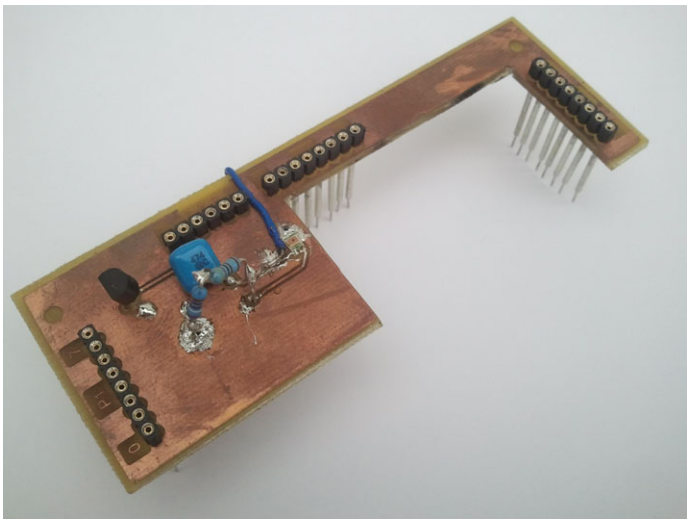


Figura 7.14: *Placa Sensora*

7.3.3. Placa Complementaria Actuadora

Dicha placa está formada por los pines encargados de conexionar dicha placa con la placa controladora además de un transistor de potencia que actúa como

driver para la iluminación led y relés que actuarán sobre motores para persianas y calefacciones eléctricas. Aunque posiblemente la solución más eficiente hubiése consisitido en desarrollar otra placa complementaria se ha decidido realizar dicho desarrollo en una placa protoboard ya que por motivos económicos se decidió no comprar motores para subir/bajar persianas.

7.4. Software de Programación y Depuración

Como ya se comentó anteriormente, uno de los motivos por los que se elegió dicho μcontolador es por que es capaz de gestionar un sistema multitarea, es decir, que tiene la capacidad de ejecutar simultáneamente múltiples tareas. Para la implementación y depuración del código de programación se ha usado el software de Keil μVision (26). Para llevar a cabo dicha implementación y depuración es necesario disponer de un módulo externo USB conocido como Tools Kit Debugger mediante el cual se conecta nuestro PC a la placa controladora. Esta conexión se realiza a través del conector paralelo que ha sido definido como conector JTAG.

Para gestionar dicho sistema multitarea se usa el sistema operativo RTX51 Tiny 51 (27), un pequeño kernel o núcleo a tiempo real(28), que sólo requiere reservar 900 bytes y es ideal para desarrollar aplicaciones que no necesiten el uso de semáforos o de administración de bancos de memoria. Para una gestión óptima de nuestro programa se utiliza una tarea inicial que a su vez crea las diversas tareas que gestionaran nuestro programa y la cual luego se destruye para dar paso a las siguientes tareas que funcionan de manera concurrente.

Las diferentes tareas utilizadas en nuestro programa que se ejecutan de manera concurrente son:

- Tarea inicial: crea el resto de tareas y luego se destruye.
- Tarea principal: encargada de distinguir el tipo de mensaje y actuar en consecuencia en función de si el mensaje se refiere a iluminación, temperatura, etc.
- Tarea de recepción: encargada de la recepción de datos a través de la UART.
- Tarea de envío: encargada del envío de mensajes a través de la UART.
- Tarea leds: encargada de modificar el dutty cycle para el aumento/disminución de la iluminacion led.

Una vez desarrollado el programa en lenguaje C se programa el μcontrolador y se da paso al modo depuración mediante el modo debugger del programa Keil μVision.

Capítulo 8

Comunicación Entre Entidades

Como en cualquier sistema distribuido, debe de haber una comunicación entre dispositivos que se dividen el trabajo. Así cada uno de ellos sabe lo que debe hacer y no trabaja más de lo debido haciendo algo que ya está realizando otro. Por ello en este proyecto se ha decidido acudir a un protocolo de comunicación inalámbrica que permite enviar mensajes cortos.

8.1. Clasificación de los Mensajes

Los mensajes que son enviados en el sistema pueden ser clasificados en tres grupos bien diferenciados:

- Mensajes de orden: Estos mensajes son los que envía el servidor a los controladores de actuadores. Requieren un mensaje de vuelta para saber que han sido recibidos correctamente y su función es básicamente la de fijar una configuración determinada para mantener una variable a un valor fijo.

- Mensajes de petición: Estos mensajes pueden ser enviados por el servidor o por un controlador de variable a cualquier dispositivo. No requieren contestación de control de recibo, puesto que ésta es el propio mensaje de vuelta que completa la comunicación (explicado en el siguiente punto). Su función consiste en pedir el valor actual de la variable interesada, tanto para controlarla como para registrarla.

- Mensaje de contestación a la petición: Estos mensajes pueden ser enviados desde cualquier dispositivo. Sólo debe haber una petición anterior. Requieren una contestación de recibo para asegurar la llegada del mensaje al destino. Sirven para informar del valor de cualquier variable que se pida, como por ejemplo, la hora actual, una configuración que se haya podido perder, la temperatura de una habitación o el estado de cualquiera de los dispositivos que interactúan en ésta.

8.2. Direccionamiento del Sistema

Para poder entender la decisión de implementar un direccionamiento, cabe mencionar que el dispositivo Xbee utilizado en este proyecto es muy sencillo y que se utiliza una red en malla para la comunicación entre ellos. Otra alternativa posible es una red en estrella, pero esto es poco práctico, puesto que el coordinador (en este caso el servidor) tendría que invertir tiempo en redireccionar la información que le llega de un dispositivo destinada a otro. Debido a los inconvenientes que genera una red en estrella se ha elegido la tipología de una red en malla. Usando dicha tipología cualquier dispositivo puede comunicarse directamente con cualquier otro, aún que el Xbee no puede hacer de repetidor para hacer llegar un mensaje que no está destinado a él, por lo que se utiliza el direccionamiento por Broadcast.

El módulo Xbee es un dispositivo que permite una comunicación inalámbrica emulando una comunicación serie punto a punto por medio de la UART. Por ello todo mensaje que se envíe a este dispositivo por su puerto Rx, automáticamente aparecerá en el puerto Tx del otro Xbee destino.

Cada módulo Xbee dispone de un identificador único de cuatro dígitos hexadecimales que lo diferencia de todos los demás, y dispone de un registro donde almacena el identificador del dispositivo con el que establece la comunicación. Este valor de *id destino* puede ser el de otro dispositivo, o *FFFF*, en cuyo caso envía un mensaje por Broadcast. En el caso de utilizar el registro de identificador de dispositivo destino como alternativa, cada vez que se quisiera enviar un mensaje a un elemento del sistema se debería configurar el Xbee del que se dispone utilizando comandos AT enviados por puerto serie. Pero dicha opción parece poco óptima, puesto que se debería gestionar una comunicación serie con el módulo además de la que se utiliza para comunicar los controladores. Por tanto se decidió utilizar una configuración en Broadcast e incluir en las tramas que se enviase la dirección del destinatario. Así un dispositivo que recibe un mensaje testea si la dirección asociada corresponde con la suya y en caso negativo lo desecha. De la misma manera, si el servidor desea enviar un mensaje a todos los dispositivos no haría falta configurar el Xbee. Otra ventaja es que se podrían enviar mensajes a dispositivos selectivos, incluyendo una serie de direcciones destino en la trama enviada.

Una vez explicado el mecanismo utilizado para el direccionamiento, se va a aplicar éste en un escenario teórico de prueba. En este caso, la vivienda dispone de 10 zonas controlables, y cada una de estas tiene un máximo de 4 controladores si se tiene en cuenta que se desea controlar la luz, la temperatura y las persianas (en el caso de haberlas) con un controlador por cada planta. Además, es necesario un controlador más que obtenga los valores ambientales por medio de sensores.

El formato de la dirección que se utiliza es de 4 dígitos hexadecimales, igual que los identificadores de los módulos Xbee, pero en este caso se reparten las direcciones de manera que cada habitación tenga una parte común para todos sus controladores (de la misma manera que se crean subredes en una red TCP/IP utilizando máscaras de subred). Para 10 habitaciones, basta con asignar un solo dígito hexadecimal a cada una de ellas (ya que un dígito en esa base dispone de 16 estados diferentes). Se desecharán el primer y último estado para utilizarlos como direcciones especiales (si se diese el caso).

Por tanto, las direcciones de los controladores podrían ser las siguientes:

1. Salón (*1XXX*):
    - Controlador sensor: *1001*
    - Controlador de luz: *1002*
    - Controlador de calefacción: *1003*
    - Controlador de persiana: *1004*
2. Habitación 1 (*2XXX*):
    - Controlador sensor: *2001*
    - Controlador de luz: *2002*
    - Controlador de calefacción: *2003*
    - Controlador de persiana: *2004*
3. Habitación 2 (*3XXX*):
    - Controlador sensor: *3001*
    - Controlador de luz: *3002*
    - Controlador de calefacción: *3003*

- Controlador de persiana: *3004*

4. Habitación 3 (*4XXX*):
    - Controlador sensor: *4001*
    - Controlador de luz: *4002*
    - Controlador de calefacción: *4003*
    - Controlador de persiana: *4004*
5. Baño 1 (*5XXX*):
    - Controlador sensor: *5001*
    - Controlador de luz: *5002*
    - Controlador de calefacción: *5003*
6. Baño 2 (*6XXX*):
    - Controlador sensor: *6001*
    - Controlador de luz: *6002*
    - Controlador de calefacción: *6003*
    - Controlador de persiana: *6004*
7. Cocina (*7XXX*):
    - Controlador sensor: *7001*
    - Controlador de luz: *7002*
    - Controlador de calefacción: *7003*
    - Controlador de persiana: *7004*
8. Terraza (*8XXX*): Funcionalidades exteriores no contempladas.
    - Controlador sensor: *8001*

9. Pasillo (*9XXX*):

    - Controlador sensor: *9001*
    - Controlador de luz: *9002*
    - Controlador de calefacción: *9003*

10. Recibidor (*AXXX*):

    - Controlador sensor: *A001*
    - Controlador de luz: *A002*
    - Controlador de calefacción: *A003*
    - Controlador de persiana: *A004*

De esta manera se podrá controlar, a nivel de habitación, la vivienda en caso de haber varias dispositivos (persianas, calefacción, luces) relacionados con una variable ambiental, o para recibir *mensajes de contestación a la petición* de todos los controladores enviando únicamente un solo *mensaje de petición*. En este caso se añade a la trama la dirección correspondiente a la habitación seguida de ceros.

8.3. Adaptador Xbee para la Raspberry Pi

Para que la *Raspberry Pi* pueda comunicarse con los controladores, esta tiene que disponer también de un módulo Xbee. Para ello, este módulo se le ha acoplado por medio de la interfaz GPIO de la que dispone y se han conectado los pines necesarios para que pueda utilizarlo.

El puerto GPIO dispone de tres salidas diferentes que tienen como fin alimentar algún circuito que se le conecte: uno de 3.3V que entrega una corriente

de 50mA y otras dos de 5V que puedes entregar más de 200mA dependiendo de la fuente de alimentación con la que se esté alimentando el ordenador. El módulo Xbee se alimenta con 3.3V, pero en este caso se ha utilizado una de las salidas a 5V y se ha integrado un regulador de tensión 5V-3.3V para asegurar la corriente entregada. Los terminales de comunicación se han conectado a la *uart*, que tienen un nivel de 3.3V para el 1 lógico y 0V a nivel bajo. También se ha conectado un pin al terminal de reset de módulo Xbee para tener la posibilidad de reiniciarlo en caso de haber algún problema.

Capítulo 9

Plantas Controladas

En este proyecto se han designado 3 dispositivos a controlar los cuales son: una tira de leds, un motor que sube y baja una persiana, y una calefacción eléctrica. Todas y cada una de estos dispositivos son manejados por diversos controladores o por un único controlador, dependiendo de las dimensiones de la habitación o de la distancia entre los distintos dispositivos.

9.1. Tira de LEDs

El control de la tira de leds es muy sencillo, se trata de un transistor que cambia de estado entre corte y saturación para encender y apagar los leds generando así una frecuencia de conmutación. En esta funcionalidad se puede controlar la intensidad lumínica de la tira, variando el ciclo de trabajo de la señal que conmuta. Cuanto menor es dicho ciclo de trabajo, menor es la intensidad y viceversa. Mediante frecuencias de conmutación elevadas el control de la luminosidad, variando el ciclo de trabajo, es menos apreciable por el ojo

humano. A su vez mediante frecuencias de conmutación bajas como 50 Hz el ojo humano es capaz de apreciar las intermitencias generadas por dicha frecuencia. En función a estas dos premisas se establece una frecuencia de conmutación constante alrededor de 70 Hz mientras que la luz está encendida.

9.2. Motor de Persiana

Para el control de la persiana, se utiliza un motor de corriente continua precedido de un puente en H (Figura 9.1) formado por cuatro relés. Esto permite darle un sentido u otro al motor dependiendo del par de relés que conmuten.

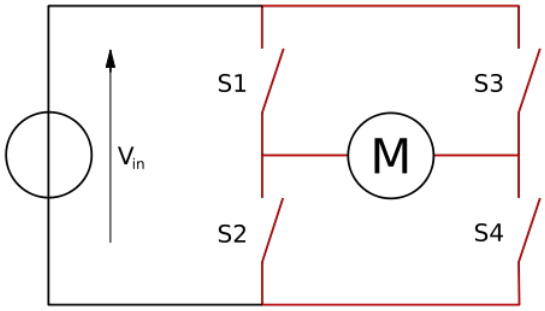


Figura 9.1: *Puente en H para Control de Motor de Corriente Continua*

En la Figura 9.1, se puede ver cómo se forma un puente en H. Para mover el motor en un sentido, se deben poner en conducción los interruptores S1 y S4, y para moverlo en sentido contrario, se deben accionar los interruptores S2 y S3. De esta manera se podrá controlar un motor en dos sentidos y subir y bajar una persiana.

Ahora solo falta dar solución al problema de cómo saber cuándo la persiana está bajada o subida para parar el motor. Gracias a unos pulsadores que se ponen en la base de la persiana y la parte superior de la misma, se puede detectar el momento en el que la persiana está totalmente cerrada o totalmente abierta.

Conectando los pulsadores a entradas del controlador,cuando éstos son pulsados se genera una interrupción que desactiva las señales de control de los relés.

Para la apertura o cierre parcial de la persiana, se pueden utilizar detectores que estén instalados a lo largo de su recorrido en distintos puntos, pero el presupuesto para la realización de este proyecto no es elevado y se pretende que el proyecto sea lo más económico posible a la hora de que un usuario desee la instalación de éste en su vivienda. A parte una de las características de nuestro sistema es la fácil instalación del mismo. Por tanto otra posible solución es controlar la apertura o cierre parcial de las persianas mediante el tiempo de activación. Si se desea desarrollar utilizando esta posibilidad, de alguna manera, el sistema ha de detectar los momentos en los que la persiana puede estar completamente abierta y pasar a estar completamente cerrada y calcular el tiempo que ha tardado en completar esta acción. Así, si se divide el recorrido en secciones de tiempo y se registra en qué posición se encuentra en cada momento se consigue un control sobre la apertura o cierre parcial de la persiana al gusto del usuario.

9.3. Calefacción Eléctrica

Para el control de la calefacción se actúa sobre un relé que permite que la calefacción se encienda cuando sea necesario. El controlador actuador encargado de esta acción debe consultar al controlador sensor de su misma habitación la temperatura cada cierto tiempo, y encender o apagar el electrodoméstico si así lo requiriese la configuración establecida. Como en todo sistema de este tipo, el circuito controlador debe contar con un cierta valor de *Histéresis*, para evitar que la conmutación sea demasiado repetitiva por cambios pequeños de temperatura. Dicha *Histéresis* se ha implementado en el sistema por software, ya que es la

manera más sencilla de probarla y configurarla. Por ello, cuando un usuario configura una temperatura deseada, el controlador toma un margen de 1 o 2 grados centígrados por encima y por debajo del valor fijado como límites de activación o apagado de la calefacción.

Parte IV

Conclusiones

Capítulo 10

Resumen de Prestaciones del Sistema

Como se ha podido comprobar, en este proyecto hay ciertos aspectos que han de tenerse en cuenta a la hora de valorar las decisiones de diseño e implementación. Es bien sabido que existen numerosos diseños y tecnologías para llevar a cabo un sistema como el que se ha presentado en este proyecto, pero cabe decir que este desarrollo tiene ciertas pautas que los alumnos implicados han querido seguir.

10.1. Uso de Raspberry Pi

Este aspecto puede ser el más interesante a nivel didáctico, ya que se trata de un dispositivo que ha creado un nuevo concepto en sistemas de computación, dejando a un lado la gran potencia de procesamiento, y compensándola de alguna manera con sistemas operativos optimizados. También hay que tener en cuenta que la posibilidad de integrar circuitos externos de manera sencilla multiplica las posibilidades de uso del dispositivo. Permite desarrollar aplicaciones como

la descrita en esta memoria de manera muy asequible. Se puede decir que se tiene una placa con μcontrolador con características de ordenador: periféricos de comunicación, I/O de propósito general, alta memoria de almacenamiento (con capacidad para alojar bases de datos muy grandes), alta frecuencia de reloj (cercana a 1GHz), reducido tamaño, reducido coste, reducido consumo de potencia, etc.

Otro aspecto interesante que tiene el RbP es que dispone de un sistema operativo, especialmente optimizado para él, basado en Linux, y más concretamente en Debian. El nombre de esta distribución es *Raspbian Wheezy*, un sistema operativo muy versátil que deja la posibilidad de controlar cada rincón del dispositivo. Dispone también de un gran soporte didáctico en toda la red de internet debido al éxito que tuvo en su lanzamiento y la amplia gama de proyectos llevados a cabo con el.

Al ser uno de los primero que se lanzó de manera oficial, es el más utilizado por las personas que aprecian esta tecnología. También el hecho de ser un proyecto de Hardware Libre que utiliza Software Libre como plataforma es un ventaja a la hora de implementar soluciones gracias al hecho de que se puede manejar cada rincón del sistema.

Para apoyar este nuevo concepto de sistemas de computación, cabe mencionar que desde su salida, Raspberry Pi ha servido como punto de partida para la creación de numerosas alternativas equivalentes, como por ejemplo *Cubietruck*, *Arduino Intel Galileo*, *BeagleBoard*, *Snowball*, *Odroid*, *Adapteva Parallella* y muchas más cuyas características pueden encontrarse facilmente en Internet.

10.2. Fácil Instalación

Esta es probablemente una de las características más importantes que repercute en el bajo coste global del sistema. Es bien sabido que este tipo de proyectos requieren una instalación complicada, la mano de obra es lo más costoso del presupuesto, pero en este caso dicha parte no ocupa mucho tiempo debido a que gran parte del cableado se ahorra gracias a la tecnología wireless utilizada. Los dispositivos pueden colocarse en cualquier lugar donde haya una fuente de energía para alimentarlos y una dispositivo que controlar. De hecho la fuente de energía no tiene por qué ser cercana, ya que este tipo de señales no son sensibles al ruido, al no transportar información.

Dado que la solución implementada cuenta con un sistema operativo y una interfaz Ethernet, la comunicación con el usuario exige cualquier tipo de nodo, ya sea un router WIFI convencional, que permita una comunicación en red TCP/IP.

Al tratarse de un sistema distribuido, el número de controladores puede adaptarse a las necesidades de la vivienda, evitando así que un cableado excesivo. Un controlador puede gestionar la calefacción y la persiana si estos están cercanos. En caso contrario no habría ningún problema en dejar cada función a dispositivos distintos.

10.3. Interfaz Web

Este aspecto es interesante en el coste de desarrollo. Como ya se explicó anteriormente, no es necesario desarrollar una aplicación por cada plataforma existente. Es suficiente tener disponible un cliente web http, aplicación que tiene instalada por defecto cualquier dispositivo hoy en día. Esto facilita la posisbilidad de

mejorar la interfaz por parte de los desarrolladores web que existen en el mundo.

El hecho de ser una aplicación web, permite también reducir el tiempo de desarrollo gracias a las numerosas alternativas que existen para cubrir estas necesidades tales como frameworks, plugins, librerías, etc.

10.4. Bajo Coste

Hoy en día este es un factor clave a tener en cuenta a la hora de desarrollar un proyecto. Los aspectos anteriormente descritos juegan un papel importante en este campo. Si se une el bajo coste del servidor, la fácil instalación del sistema, el sencillo desarrollo de la interfaz y el bajo coste que supone desarrollar una sencilla placa con un μcontrolador dan como resultado una solución muy accesible en lo que a aspectos económicos se refiere.

En nuestro caso, el coste del prototipo, que podría cubrir el control de una habitación ha ascendido a 200 euros aproximadamente. Teniendo en cuenta que solo es necesario un servidor, el cubrir más habitaciones implica un coste mayor. Si se decidiese poner en práctica este proyecto y se decide instalar un sistema como este en una vivienda con cuatro habitaciones (cocina, salon, dormitorio y cuarto de baño), se puede desarrollar con un presupuesto de aproximadamente 600 euros teniendo en cuenta que el material utilizado es cada vez más económico cuantos más componentes se compren en conjunto. Investigando en Internet, hay empresas que ofrecen este servicio por no menos de 2000 euros cubriendo una vivienda de dos habitaciones. Aún habiendo contado el coste de este sistema aproximado, el margen es muy elevado.

Capítulo 11

Ideas de Evolución del Sistema

Hay muchos aspectos que no se han tenido en cuenta en este desarrollo, y probablemente el más importante sea la seguridad, es decir, la capacidad de evitar que una persona ajena a la vivienda pueda acceder a la aplicación. Cabe reseñar que para que esto ocurra, la persona intrusa debe conectarse a la misma red local donde el servidor está situado. Si el router tiene la encriptación adecuada, este factor no debería suponer problema alguno.

Otro servicio que podría ser interesante desarrollar es el control de la vivienda sin necesidad de estar conectado a la red local, sino por medio de una VPN, para poder gestionarla desde cualquier punto del mundo. Esta característica tendría que tener asociado un trabajo de seguridad, ya que esto aumenta las posibilidades de conexiones intrusas.

A parte de las funcionalidades que se ofrecen en el prototipo, un sistema de alarma visual generado en la totalidad de la vivienda, sea para emergencias o para avisos específicos podría hacerse de manera sencilla enviando mensajes, de encendido y apagado de luces, por broadcast a todos los dispositivos

controladores.

Para hacer el sistema mejor, se podría añadir la funcionalidad de detectar presencia humana en cada habitación. De esta manera, se podrían apagar las luces cuando no se necesitasen, ahorrando así energía eléctrica. Esta detección podría hacerse mediante dos métodos: detección de movimiento o sensores de entrada/salida. La primera, por detección de movimiento, la cúal conlleva el inconveniente de que si una persona queda inmóvil el detector dejaría de detectar presencia por lo que apagaría el sistema de iluminación. El otro método es con sensores en las entradas de las habitaciones, el cual realiza un conteo de las personas que entran y salen de la habitación para saber si está vacía u ocupada aún. Este método tiene la incomodidad de tener que apagar la luz manualmente en caso de estar, por ejemplo, en el dormitorio con idea de dormir.

Capítulo 12

Repercusión Tecnológica

A lo largo de este desarrollo, el tema de este proyecto ha salido en diversas conversaciones, y la pregunta que más se daba por parte de los que escuchaban la explicación era ”¿Y qué tiene esto de innovador?”. Este proyecto no es innovador, ya que los sistemas de control son una ciencia que está por delante del proyecto aqui presentado. Pero los participantes en este desarrollo han querido implementar una solución a una necesidad que ya se cubre, en una tecnología de Hardware Libre que ha revolucionado el concepto de los ordenadores versátiles. Lo innovador de este proyecto puede ser que sobre esta base que se presenta, se pueden ampliar infinidad de funcionalidades como anteriormente se han citado como ideas de evolución del sistema. El inicio de las viviendas inteligentes al alcance de cualquiera, sin ser un lujo, como algo normal en la vida cotidiana.

Un aspecto interesante de este sistema es la monitorización, cuyos datos se almacenan en el servidor y pueden ser utilizados para hacer estudios de eficiencia energética. Tener un historial de consumo o eventos que ocurren en una vivienda resulta de gran utilidad para almacenar información a la hora de cubrir necesidades que tal vez no habían sido tenidas en cuenta anteriormente, o

para detectar funcionamientos anómalos de los dispositivos o comportamientos inusuales de las personas de la casa, que puedan significar que el usuario necesita algún tipo de auxilio en caso de accidente o enfermedad.

Desde que se conoce el dispositivo Raspberry Pi, se han visto numerosos proyectos que recurren a sus prestaciones, y hay una característica que es común en la mayoría de las aplicaciones: ninguno utiliza circuitos complementarios de apoyo al primero. Por ejemplo, en el mes de Marzo de 2014 se celebró una conferencia de Hardware libre que formaba parte de una iniciativa de divulgación de este tipo llamada *OpenExpo*(29). En esta charla, uno de los protagonistas era el ordenador del que aquí se habla, y la persona que lo presentaba, dió un ejemplo de control de una vivienda con este dispositivo. El problema que se detectó fué que el Raspberry Pi era el único controlador que participaba en este propósito, parecía que estaba totalmente desaprovechada su capacidad, ya que solo lo utilizaban como conmutador inteligente directamente conectado a los diferenciales de la vivienda. Con esto se quiere decir que se ha tomado este concepto de manera mucho más evolucionada, ya que en el caso del servidor del proyecto que en esta memoria se presenta, gestiona una red de controladores esclavos que puede ser ampliada en cualquier momento. Cada dispositivo es independiente dentro de las órdenes que reciben. Si uno deja de funcionar, el resto del sistema sigue funcionando.

En esta misma conferencia, una empresa presentó un proyecto llamado *Custodium Tracker*. Se trataba de un sistema de localización GPS destinado en un principio a localizar personas con Alzheimer, que envía las coordenadas cada cierto tiempo a un servidor por medio de GPRS. Al final de la presentación propusieron una evolución que nos pareció interesante: Conectar varios dispositivos *Custodium* entre sí por medio de módulos Xbee (como los utilizados en este proyecto) y utilizarlos por ejemplo como sistema de seguimiento de ganado. Si estos dispositivos se conectasen de la misma manera a un Raspberry Pi y se añadiesen

también sensores ambientales, se podrían ampliar las prestaciones del sistema y obtener más información para controlar el estado de los animales, monitorizando cada movimiento o cada lugar por el que estuviesen. (30)

Otra posibilidad, siguiendo en el campo de lo sistemas de control y monitorización, podría ser la aplicación agrícola. El crear un sistema de cuidado de plantas que actuase dependiendo de las condiciones atmosféricas y que registrase cada evento ocurrido para saber en qué estado se encuentra la producción sin necesidad de desplazarse hasta el campo de la plantación.

Como conclusión final, se puede decir que cada vez más se desarrollan dispositivos que actúan de manera conjunta, entes sencillos que colaboran entre sí para crear una funcionalidad mucho más compleja. El caso de los air drones es uno de ellos. En una conferencia publicada en Youtube, cuatro drones que tienen acoplada una única red entre ellos son capaces de atrapar pelotas de golf lanzadas de manera aleatoria por el ponente. Dichos drones están perfectamente sincronizados, para que las pelotas no reboten en la red y caigan al suelo, compensando la fuerza con la que caen.

Parte V

Glosario, Bibliografía y Anexos

Glosario

Baudrate Número de unidades de señal por segundo . 78

BigInt Tipo de dato entero formado por 8 Bytes . 55

Broadcast Forma de transmisión de información donde un nodo emisor envía información a una multitud de nodos receptores de manera simultánea, sin necesidad de reproducir la misma transmisión nodo por nodo . 78, 86, 87

Decimal Tipo de dato. Su definición se realiza indicando 2 parámetros, la longitud total máxima del número (escala) y la longitud máxima de la parte decimal (precisión) . 55

IEEE 802.15.4 Estándar que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos. La actual revisión del estándar se aprobó en 2006. El grupo de trabajo IEEE 802.15 es el responsable de su desarrollo. . 24, 77, 78

JQuery Librería JavaSript que permite facilitar el manejo de documentos HTML, árboles DOM, animaciones y gestión de eventos . 45, 47, 51

MySQL Sistema de gestión de bases de datos relacional, multihilo y multiusuario . 39, 54, 57

PCB Siglas en inglés de Printed Circuit Board. Superficie constituida por pistas de material conductor, las cuales interconectan diversos componentes electrónicos de manera que dicha interconexión conlleve a un circuito final encargado de cumplir ciertos objetivos . 63, 65–67, 81

Servidor Apache Servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh entre otras, que implementa el protocolo HTTP/1.12 y la noción de sitio virtual . 39, 50, 59, 61

SmallInt Tipo de dato entero formado por 2 Bytes. Los rangos de valores posibles son de -32768 a 32767 y de 0 a 65535 para el rango sin signo (unsigned) . 54–56

TimeStamp Tipo de dato que representa el número de segundos transcurridos desde la fecha y hora 00:00:00 del 01/01/1970 UTC y su rango permite representar hasta 03:14:07 del 19/01/2038 UTC . 56, 58

VarChar Tipo de dato de longitud variable. Se compone de caracteres de 1 Byte y su longitud máxima se define al crear la tabla donde se aloja. A diferencia del dato de tipo char, el dato varchar ocupa un número de bytes que corresponde al número de caracteres registrados más 1 Byte adicional que indica el número de caracteres que lo componen. Para ser más específico, un dato de tipo char de longitud fijada en 50 caracteres, siempre ocupará 50 Bytes en el disco duro sea cual sea el número de caracteres escritos . 54, 55

Wordpress Sistema de gestión de contenidos de licencia GPL destinado a la creación de blogs y programado en PHP sobre servidor Apache y gestor de BBDD MySQL . 44

Xbee Módulo creado por Digi International con la marca MaxStream que permite implementar una red de comunicación con estándar IEEE 802.15.4

diseñado para comunicaciones inalámbricas punto a punto y estrella con un baud-rate de hasta 250 Kbps . 24, 66, 68, 69, 71, 75–78, 86–88

Acrónimos

ADC Analog to Digital Converter. 56, 65, 68–70, 79, 80

AMUX Analog Multiplexer. 70

CRC Cyclic Redundancy Check. 68

DAC Digital to Analog Converter. 69

PCA Programmable Counter Array. 68

SoC System on Chip. 31

SPI Serial Peripheral Interface. 68

UART Universal Asynchronous Receiver/Transmitter. 68, 69, 74, 76, 77, 84, 87

VPN Virtual Private Network. 103

Bibliografía

[1] "X10." http://www.eurox10.com/Content/x10information.htm.

[2] "What is knx?." http://www.knx.org/knx-en/knx/association/what-is-knx/index.php.

[3] "Understanding zigbee." http://www.zigbee.org/About/UnderstandingZigBee.aspx.

[4] M. C. B. Hernández, "Sistema domótico orientado a usabilidad. diseño de plataforma de control vocal." https://www.coit.es/pub/ficheros/presumen_orange_b7c65c77.pdf, 05 2007.

[5] H. M. D. y Fernando Sáez Vacas, *Domótica: Un enfoque sociotécnico*. Fundación Rogelio Segovia para el Desarrollo de las Telecomunicaciones Ciudad Universitaria, 2006.

[6] J. M. A. Madrid, "Sistema de control de una vivienda." http://deeea.urv.cat/public/PROPOSTES/pub/pdf/1343pub.pdf, 04 2009.

[7] A. E. d. D. CEDOM, *Cómo ahorrar energía instalando domótica en su vivienda. Gane en confort y seguridad*. AENOR, 2008. http://www.cedom.es/sobre-domotica/publicaciones/guia-como-ahorrar-energia-instalando-domotica-en-su-vivienda-gane-en-confort-y-seguridad/download/guia-del-ahorro-energetico-cedom.

[8] "Domótica y discapacidad. accesibilidad para todos." http://www.discapnet.es/Castellano/areastematicas/Accesibilidad/Accesibilidadenelhogar/Domoticaydiscapacidad/Documents/Guias/Domotica/domdisc_acc.html.

[9] "Raspberry pi." http://www.raspberrypi.org/.

[10] "Documentación del servidor http apache 2.0." http://httpd.apache.org/docs/2.0/es/.

[11] "Manual de php." https://php.net/manual/es/index.php.

[12] "jquery.ajax()." http://api.jquery.com/jquery.ajax/.

[13] "What is jquery." http://jquery.com/.

[14] "What is arduino." http://www.arduino.cc/.

[15] "Mysql: The world's most popular open source database." http://www.mysql.com/.

[16] "Bootstrap getting started." http://getbootstrap.com/getting-started/.

[17] "Sockets tutorial." http://www.linuxhowtos.org/C_C++/socket.htm.

[18] S. Laboratories, "C8051f410 datasheet." http://www.silabs.com/Support%20Documents/TechnicalDocs/C8051F41x.pdf, 11 2008. Rev 1.1.

[19] "Atmel: Code examples for 8051 uart." http://www.atmel.com/tools/CODEEXAMPLESFOR8051UART.aspx.

[20] T. Instrument, "Lm117/lm317a/lm317-n three-terminal adjustable regulator." http://www.ti.com/lit/ds/symlink/lm117.pdf, 01 2014.

[21] D. I. Inc, "Xbee®/xbee-pro® rf modules." https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf, 09 2009.

[22] "Digi xbee examples & guides. configuring xbee radios with x-ctu." http://examples.digi.com/get-started/configuring-xbee-radios-with-x-ctu/.

[23] Osram, "High accuracy ambient light sensor sfh5711 datasheet." http://www.osram-os.com/Graphics/XPic6/00039059_0.pdf/Appnote%20for%20Ambient%20light%20sensor%20SFH%205711.pdf, 08 2006.

[24] "Pic16f877a thermometer with mcp9700a sensor." http://www.micro-examples.com/public/microex-navig/doc/086-mcp9700a-thermometer.html.

[25] Microchip, "Low-power linear active thermistor ics mcp9700 datasheet." http://ww1.microchip.com/downloads/en/DeviceDoc/21942e.pdf, 2009.

[26] A. M. Fernández-Caparrós, "Prontuario del entorno de desarrollo keil uvision para la familia de microcontroladores mcs51." http://www.uco.es/~el1mofer/Docs/IntPerif/Prontuario%20%20(v1.1)%20Keil%20uVision3%20.pdf, 03 2009. Rev 1.1.

[27] "Keil rtx51 tiny user's guide: Theory of operation." http://www.keil.com/support/man/docs/tr51/tr51_theoryop.htm.

[28] "Rtx51 real-time kernel." http://www.keil.com/rtx51/.

[29] "Openexpo: Eventos open source y software libre." http://www.openexpo.es/.

[30] "Custodium en openexpo hardware libre." http://custodiumtracker.com/2014/03/custodium-en-openexpo-hardware-libre/.

Anexos

Anexo A

Algorítmos de Programación

1. Aplicación Gestora del Servidor

```
1  #include <my_global.h>
2  #include <mysql.h>
3  #include <time.h>
4  //Para uso global
5  #include <stdio.h>
6  #include <string.h>
7  #include <stdbool.h>
8  //Para socket y UART
9  #include <stdlib.h>
10 #include <unistd.h>   //Used for UART
11 #include <fcntl.h>    //Used for UART
12 #include <termios.h>    //Used for UART
13 #include <pthread.h>
14 #include <sys/types.h>
15 #include <sys/socket.h>
16 #include <netinet/in.h>
17 //Para sincronizaci n de procesos
18 #include <semaphore.h>
19
20 //Constantes de hilos
21 #define HILO_LEE_SOCKET 0
22 #define HILO_LEE_UART 1
23 #define HILO_MONITOR_VAR 2
24 #define HILO_TIMER 3
25
26 //Constantes de socket
27 #define PUERTO 11000
28 #define NUM_CONEXIONES 3
29
30 //Constantes de BBDD
31 #define BBDD_HOST "localhost"
32 #define BBDD_USER "root"
33 #define BBDD_PASS "euitt"
```

```
34 #define BBDD_NAME "domoweb"
35
36 #define BBDD_ID_TIPOVAR_TEMPERATURA 1
37 #define BBDD_ID_TIPOVAR_INTENSIDADLUZ 2
38 #define BBDD_ID_TIPOVAR_AUTOLUZ_ACTIVADO_USER 3
39 #define BBDD_ID_TIPOVAR_AUTOTEMP_ACTIVADO_USER 4
40 #define BBDD_ID_TIPOVAR_LUZACTION_INTENSIDAD_USER 5
41
42 //Constantes de Mutex
43 #define MUTEX_UART_TX 0
44 #define MUTEX_TIMER_CONTADOR 1
45 #define MUTEX_VARIABLE_MENSAJEGLOBAL 2
46 #define MUTEX_VARIABLE_TIMERACTIVADO 3
47
48 //Constantes de Semaphores
49 #define SEMAPHORE_UART_RX 0
50 #define SEMAPHORE_TIMER_ACTIVAR 1
51
52 //Constantes de mensajes
53 #define PETICION_TEMP "TEMP"
54 #define PETICION_ILUZ "MILZ"
55
56 #define ORDEN_FIJAR_ILUZ "LZFR"
57
58 #define ORDEN_AUTO_TEMPERATURA_APAGAR "AUTTMPOFF"
59 #define ORDEN_AUTO_TEMPERATURA_ENCENDER "AUTTMPON"
60 #define ORDEN_TEMPERATURA_DESEADA_FIJAR "TMPDS"
61 #define ORDEN_AUTO_LUZ_APAGAR "AUTLZOFF"
62 #define ORDEN_AUTO_LUZ_ENCENDER "AUTLZON"
63 #define ORDEN_LUZ_DESEADA_FIJAR "ILZDS"
64
65 #define ORDEN_LUZ_ENCENDER "LZON"
66 #define ORDEN_LUZ_APAGAR "LZOFF"
67 #define ORDEN_CALEFACCION_ENCENDER "CLON"
68 #define ORDEN_CALEFACCION_APAGAR "CLOFF"
```

```
69
70 #define ORDEN_RESET_SISTEMA "RST"
71
72 //Constantes de Direccionamiento
73 #define DIRECCION_SENSOR_SALON "276B"
74 #define DIRECCION_ACTUADOR_SALON "276C"
75 #define DIRECCION_GLOBAL "FFFF"
76
77 //Constantes de Monitorizaci n
78 #define NUM_INTENTOS_CONSULTA 10
79 #define PERIODO_MONITOR 60 //Periodo de monitorizai n en s
80
81 //Constantes de estructura de mensajes
82 #define MAX_NUM_PARTES_MENSAJE 3
83 #define MAX_LONGITUD_MENSAJE 30
84 #define INDICE_MENSAJE 0
85 #define INDICE_DIRECCION 1
86 #define INDICE_ORDEN 2
87 #define INDICE_VALOR 3
88
89 //Constantes de Comandos bash
90 #define REBOOT_SYSTEM "sudo shutdown -r now"
91
92
93 typedef struct{
94   int descriptor_socket;
95   struct sockaddr_in info_maquina;
96   int error;
97 }socket_datos;
98
99 typedef struct{
100   int uart;
101   int socket;
102   MYSQL* bbdd;
103 }manejadores;
```

```
104
105 //Hilos
106 void* leeSocket (void *arg);
107 void* leeUART(void *arg);
108 void* monitorizarVariables (void *arg);
109 void* controltimer (void *arg);
110
111 //Funciones de socket
112 socket_datos crearSocket(unsigned short puerto, int numconexiones);
113
114 //Funciones de UART
115 int abrir_uart();
116 int enviarDato (char datoOut[],int uart);
117
118 //Funciones de BBDD
119 void cerrarBBDD (MYSQL* descriptorBBDD);
120 MYSQL* abrirBBDD (char maquina[],char usuario[],char password[], char
      baseDatos[]);
121 int registrarDatoMonitor(MYSQL* conect, int idvariable, double valor,
      int timestamp, int idzona);
122 int sacarIdZona(MYSQL* conect, int idControlador);
123
124 //Funciones de control de mensaje
125 void sacarDatosMensaje(char buffer[],char datosMensaje[
      MAX_NUM_PARTES_MENSAJE+1][MAX_LONGITUD_MENSAJE]);
126 void fijarMensajesMonitor(void);
127 void sacarMensajeCabecera(char buffer[],char mensaje[]);
128
129
130 pthread_t idhilos[10];
131 pthread_mutex_t mutex[10];
132 sem_t semaforo[10];
133 char mensajeRX[MAX_LONGITUD_MENSAJE];
134
135 char mensajesMonitor[30][MAX_LONGITUD_MENSAJE];
```

```
136
137 char mensajeGlobal[MAX_NUM_PARTES_MENSAJE+1][MAX_LONGITUD_MENSAJE];
138
139 //Contador del TIMER
140 unsigned char timerContador = 250;
141 unsigned char timerActivado = 0;
142
143 //Variables de informaci n de estado de configuraci n del sistema
144 char auto_luz_state = 0;
145 char auto_temp_state = 0;
146 char luz_state = 0;
147
148
149 /***********************
150 *    FUNCION PRINCIPAL    *
151 ***********************/
152
153 void main(void){
154   MYSQL* conector_bbdd=NULL;
155    socket_datos servidor;
156    int descriptor_uart = -1;
157    manejadores hiloDatos;
158    pthread_t idhilos[10];
159    int errorHilo;
160
161    servidor = crearSocket(PUERTO,NUM_CONEXIONES);
162    descriptor_uart = abrir_uart();
163
164    hiloDatos.uart = descriptor_uart;
165    hiloDatos.socket = servidor.descriptor_socket;
166
167    //****************************
168    // Sistemas de Sincronizaci n *
169    //****************************
170
```

```
171   //mutex para TX de la UART
172   if (pthread_mutex_init(&(mutex[MUTEX_UART_TX]), NULL) != 0)
173   {
174     printf("\n[main] Inicializaci n de Mutex %d fallido\n",
      MUTEX_UART_TX);
175   }
176
177   //Sem foro para RX de la UART
178   if (sem_init(&(semaforo[SEMAPHORE_UART_RX]),0,0) != 0)
179   {
180     printf("\n[main] Inicializaci n de Semaphore %d fallido\n",
      SEMAPHORE_UART_RX);
181   }
182
183   //Sem foro para la activaci n y desactivaci n del TIMER
184   if (sem_init(&(semaforo[SEMAPHORE_TIMER_ACTIVAR]),0,0) != 0)
185   {
186     printf("\nmain] Inicializaci n de Semaphore %d fallido\n",
      SEMAPHORE_TIMER_ACTIVAR);
187   }
188
189   //mutex para la variable contador del TIMER
190   if (pthread_mutex_init(&(mutex[MUTEX_TIMER_CONTADOR]), NULL) != 0)
191   {
192     printf("\nmain] Inicializaci n de Mutex %d fallido\n",
      MUTEX_TIMER_CONTADOR);
193   }
194
195   //mutex para la variable mensajeGlobal
196   if (pthread_mutex_init(&(mutex[MUTEX_VARIABLE_MENSAJEGLOBAL]), NULL)
      != 0)
197   {
198     printf("\nmain] Inicializaci n de Mutex %d fallido\n",
      MUTEX_VARIABLE_MENSAJEGLOBAL);
199   }
```

```
200
201   //mutex para la variable indicadora de timer activado
202   if (pthread_mutex_init(&(mutex[MUTEX_VARIABLE_TIMERACTIVADO]), NULL)
       != 0)
203   {
204     printf("\nmain] Inicializaci n de Mutex %d fallido\n",
       MUTEX_VARIABLE_TIMERACTIVADO);
205   }
206
207   //*********************
208   // Generaci n de Hilo *
209   //*********************
210
211   //Generaci n del hilo que se encarga de leer el socket y enviar la
       orden correspondiente por la UART
212   errorHilo = pthread_create(&(idhilos[HILO_LEE_SOCKET]), NULL, &
       leeSocket, &hiloDatos);
213   if(errorHilo != 0)
214     printf("\n[main] Error al crear el hilo %d: [%s]\n",HILO_LEE_SOCKET
       ,strerror(errorHilo));
215   else
216     printf("\n[main]   xito  al crear el hilo %d.\n",HILO_LEE_SOCKET);
217
218   //Generaci n del hilo que se encarga de Monitorizar las variables y
       guardar los datos en la BBDD
219   errorHilo = pthread_create(&(idhilos[HILO_LEE_UART]), NULL, &leeUART,
       &hiloDatos);
220   if(errorHilo != 0)
221     printf("\n[main] Error al crear el hilo %d: [%s]\n",HILO_LEE_UART,
       strerror(errorHilo));
222   else
223     printf("\n[main]   xito  al crear el hilo %d.\n",HILO_LEE_UART);
224
225   //Generaci n del hilo que se encarga de Monitorizar las variables y
       guardar los datos en la BBDD
```

```
226   errorHilo = pthread_create(&(idhilos[HILO_MONITOR_VAR]), NULL, &
      monitorizarVariables, &hiloDatos);
227   if(errorHilo != 0)
228     printf("\n[main] Error al crear el hilo %d: [%s]\n",
      HILO_MONITOR_VAR, strerror(errorHilo));
229   else
230     printf("\n[main]  xito  al crear el hilo %d.\n",HILO_MONITOR_VAR);
231
232   //Generaci n del hilo TIMER
233   errorHilo = pthread_create(&(idhilos[HILO_TIMER]), NULL, &
      controltimer, NULL);
234   if(errorHilo != 0)
235     printf("\n[main] Error al crear el hilo %d: [%s]\n",HILO_TIMER,
      strerror(errorHilo));
236   else
237     printf("\n[main]  xito  al crear el hilo %d.\n",HILO_TIMER);
238
239   while(1){
240     sleep(120);
241   }
242 }
243
244
245 /************************
246 *   HILOS DEL PROGRAMA   *
247 ************************/
248
249 //Hilo que escucha el puerto del socket y recoge el mensaje.
250 void* leeSocket (void *arg){
251   int  descriptor2;
252   struct sockaddr_in cliente;
253   int sockEntTam;
254   int n;
255   char buffer[256];
256   char mensajeSolo[MAX_LONGITUD_MENSAJE];
```

```
257   char mensaje[MAX_NUM_PARTES_MENSAJE+1][MAX_LONGITUD_MENSAJE];
258   //Variables para registro en base de datos
259   double valorVariable=0;
260   int idVariable=0;
261   int timestamp=0;
262   int idZona=0;
263   MYSQL* conector;
264   int idControl;
265   int valorSemaforoTimer=123;
266   unsigned char estadoTimer = 0;
267   char command[50];
268
269   while(1){
270     sockEntTam = sizeof(struct sockaddr_in);
271
272     if((descriptor2 = accept(((manejadores *) arg)->socket, (struct
      sockaddr *) &cliente, &sockEntTam)) == -1){
273       printf("\n[leeSocket] Error en la funcion accept.\n");
274       exit(-1);
275     }
276
277     bzero(buffer, 256);
278     n = read(descriptor2,buffer,255);
279     if(n < 0){
280       printf("\n[leeSocket] Error al leer del socket.\n");
281     }
282
283     sacarMensajeCabecera(buffer,mensajeSolo);
284
285     fprintf(stdout,"\n[leeSocket] Valor mensajeSolo: %s\n",mensajeSolo)
      ;
286     sacarDatosMensaje(mensajeSolo,mensaje);
287     pthread_mutex_lock(&(mutex[MUTEX_VARIABLE_MENSAJEGLOBAL]));
288     memcpy(mensajeGlobal, mensaje, sizeof(mensaje));
289     pthread_mutex_unlock(&(mutex[MUTEX_VARIABLE_MENSAJEGLOBAL]));
```

```
290
291     if(strcmp(mensaje[INDICE_ORDEN],ORDEN_FIJAR_ILUZ)==0){
292         pthread_mutex_lock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
        bloqueado
293         enviarDato(mensaje[INDICE_MENSAJE],((manejadores *) arg)->uart);
294         pthread_mutex_unlock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
        desbloqueado
295
296         pthread_mutex_lock(&(mutex[MUTEX_VARIABLE_TIMERACTIVADO]));
297         timerActivado = 1;
298         pthread_mutex_unlock(&(mutex[MUTEX_VARIABLE_TIMERACTIVADO]));
299
300         sem_getvalue(&(semaforo[SEMAPHORE_TIMER_ACTIVAR]),&
        valorSemaforoTimer);
301
302         if(valorSemaforoTimer<=0){
303             sem_post(&(semaforo[SEMAPHORE_TIMER_ACTIVAR]));
304         }
305
306         pthread_mutex_lock(&(mutex[MUTEX_TIMER_CONTADOR]));
307         timerContador = 0;
308         pthread_mutex_unlock(&(mutex[MUTEX_TIMER_CONTADOR]));
309
310         n = write(descriptor2,"Recibido",8);
311         if (n < 0){
312             printf("\n[leeSocket] ERROR al escribir en el socket\n");
313         }
314     }
315     //ORDENES DE ACTIVACIN Y DESACTIVACIN DE AUTOMATIZACIN DE
        CONTROL DE LA CASA
316     else if(strcmp(mensaje[INDICE_ORDEN],ORDEN_AUTO_TEMPERATURA_APAGAR)
        ==0 || strcmp(mensaje[INDICE_ORDEN],ORDEN_AUTO_TEMPERATURA_ENCENDER
        )==0 || strcmp(mensaje[INDICE_ORDEN],ORDEN_AUTO_LUZ_APAGAR)==0 ||
        strcmp(mensaje[INDICE_ORDEN],ORDEN_AUTO_LUZ_ENCENDER)==0){
```

```
317         pthread_mutex_lock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
        bloqueado
318         enviarDato(mensaje[INDICE_MENSAJE],((manejadores *) arg)->uart);
319         pthread_mutex_unlock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
        desbloqueado
320
321         if(strcmp(mensaje[INDICE_ORDEN],ORDEN_AUTO_TEMPERATURA_APAGAR)
        ==0){
322           idVariable = BBDD_ID_TIPOVAR_AUTOTEMP_ACTIVADO_USER;
323           valorVariable=0;
324         }
325         else if(strcmp(mensaje[INDICE_ORDEN],
        ORDEN_AUTO_TEMPERATURA_ENCENDER)==0){
326           idVariable = BBDD_ID_TIPOVAR_AUTOTEMP_ACTIVADO_USER;
327           valorVariable=1;
328         }
329         else if(strcmp(mensaje[INDICE_ORDEN],ORDEN_AUTO_LUZ_APAGAR)==0){
330           idVariable = BBDD_ID_TIPOVAR_AUTOLUZ_ACTIVADO_USER;
331           valorVariable=0;
332         }
333         else if(strcmp(mensaje[INDICE_ORDEN],ORDEN_AUTO_LUZ_ENCENDER)==0)
        {
334           idVariable = BBDD_ID_TIPOVAR_AUTOLUZ_ACTIVADO_USER;
335           valorVariable=1;
336         }
337
338         idControl=strtol(mensaje[INDICE_DIRECCION],NULL,16);
339         timestamp=(int)time(NULL);
340
341         conector=abrirBBDD(BBDD_HOST,BBDD_USER,BBDD_PASS,BBDD_NAME);
342         idZona=sacarIdZona(conector,idControl);
343         registrarDatoMonitor(conector,idVariable,valorVariable,timestamp,
        idZona);
344         cerrarBBDD(conector);
345
```

```
346         n = write(descriptor2,"Recibido",8);
347         if (n < 0){
348           printf("\n[leeSocket] ERROR al escribir en el socket\n");
349         }
350       }
351       else if(strcmp(mensaje[INDICE_ORDEN],ORDEN_LUZ_ENCENDER)==0)
352       {
353         pthread_mutex_lock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
        bloqueado
354         enviarDato(mensaje[INDICE_MENSAJE],((manejadores *) arg)->uart);
355         pthread_mutex_unlock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
        desbloqueado
356
357         n = write(descriptor2,"Recibido",8);
358         if (n < 0){
359           printf("\n[leeSocket] ERROR al escribir en el socket\n");
360         }
361       }
362       else if(strcmp(mensaje[INDICE_ORDEN],ORDEN_LUZ_APAGAR)==0)
363       {
364         pthread_mutex_lock(&(mutex[MUTEX_VARIABLE_TIMERACTIVADO]));
365         estadoTimer = timerActivado;
366         pthread_mutex_unlock(&(mutex[MUTEX_VARIABLE_TIMERACTIVADO]));
367
368         if(estadoTimer!=0){
369           pthread_mutex_lock(&(mutex[MUTEX_VARIABLE_TIMERACTIVADO]));
370           timerActivado = 0;
371           pthread_mutex_unlock(&(mutex[MUTEX_VARIABLE_TIMERACTIVADO]));
372
373           pthread_mutex_lock(&(mutex[MUTEX_VARIABLE_MENSAJEGLOBAL]));
374           idVariable = BBDD_ID_TIPOVAR_LUZACTION_INTENSIDAD_USER;
375
376           idControl=strtol(mensajeGlobal[INDICE_DIRECCION],NULL,16);
377           valorVariable=strtod(mensajeGlobal[INDICE_VALOR],NULL);
378           pthread_mutex_unlock(&(mutex[MUTEX_VARIABLE_MENSAJEGLOBAL]));
```

```
379             timestamp=(int)time(NULL);
380
381             conector=abrirBBDD(BBDD_HOST,BBDD_USER,BBDD_PASS,BBDD_NAME);
382             idZona=sacarIdZona(conector,idControl);
383             registrarDatoMonitor(conector,idVariable,valorVariable,
    timestamp,idZona);
384             cerrarBBDD(conector);
385         }
386
387         pthread_mutex_lock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
    bloqueado
388         enviarDato(mensaje[INDICE_MENSAJE],((manejadores *) arg)->uart);
389         pthread_mutex_unlock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
    desbloqueado
390
391         idVariable = BBDD_ID_TIPOVAR_LUZACTION_INTENSIDAD_USER;
392         valorVariable = 0;
393
394         idControl=strtol(mensaje[INDICE_DIRECCION],NULL,16);
395         timestamp=(int)time(NULL);
396
397         conector=abrirBBDD(BBDD_HOST,BBDD_USER,BBDD_PASS,BBDD_NAME);
398         idZona=sacarIdZona(conector,idControl);
399         registrarDatoMonitor(conector,idVariable,valorVariable,timestamp,
    idZona);
400         cerrarBBDD(conector);
401
402         n = write(descriptor2,"Recibido",8);
403         if (n < 0){
404             printf("\n[leeSocket] ERROR al escribir en el socket\n");
405         }
406      }
407      else if(strcmp(mensaje[INDICE_ORDEN],
    ORDEN_TEMPERATURA_DESEADA_FIJAR)==0)
408      {
```

```
409         pthread_mutex_lock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
        bloqueado
410         enviarDato(mensaje[INDICE_MENSAJE],((manejadores *) arg)->uart);
411         pthread_mutex_unlock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
        desbloqueado
412
413         n = write(descriptor2,"Recibido",8);
414         if (n < 0){
415           printf("\n[leeSocket] ERROR al escribir en el socket\n");
416         }
417       }
418       else if(strcmp(mensaje[INDICE_ORDEN],ORDEN_RESET_SISTEMA)==0 &&
        strcmp(mensaje[INDICE_DIRECCION],DIRECCION_GLOBAL)==0)
419       {
420         pthread_mutex_lock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
        bloqueado
421         enviarDato(mensaje[INDICE_MENSAJE],((manejadores *) arg)->uart);
422         pthread_mutex_unlock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
        desbloqueado
423
424         n = write(descriptor2,"Recibido",8);
425         if (n < 0){
426           printf("\n[leeSocket] ERROR al escribir en el socket\n");
427         }
428
429         strcpy( command, REBOOT_SYSTEM );
430         system(command);
431       }
432
433       close(descriptor2);
434
435     }
436 }
437
438 //Hilo que lee de la UART
```

```
439 void* leeUART(void *arg){
440   while(1){
441     sem_wait(&(semaforo[SEMAPHORE_UART_RX]));
442     //------ ESPERANDO ALGN BYTE EN RX ------
443     if (((manejadores *) arg)->uart != -1)
444     {
445       // Se leen 255 bytes en el puerto de recepci n
446       unsigned char rx_buffer[256];
447       int rx_length = read(((manejadores *) arg)->uart, (void*)
    rx_buffer, 255);
448       if (rx_length < 0)
449       {
450         //Ocurri  un error
451       }
452       else if (rx_length == 0)
453       {
454         //Ning n dato recibido
455       }
456       else
457       {
458         //Datos recibidos
459         rx_buffer[rx_length] = '\0';
460         strcpy(mensajeRX, rx_buffer);
461         printf("\n[leeUART] %i bytes read : %s\n", rx_length, rx_buffer
    );
462       }
463     }
464     sem_post(&(semaforo[SEMAPHORE_UART_RX]));
465   }
466 }
467
468 //Hilo que pide peri dicamente los datos a monitorizar al controlador
    correspondiente
469 void* monitorizarVariables (void *arg){
470   double valorVariable=0;
```

```
471     int idVariable=0;
472     int timestamp=0;
473     int idZona=0;
474     int i;
475     int indiceMensaje;
476     char mensaje[MAX_NUM_PARTES_MENSAJE+1][MAX_LONGITUD_MENSAJE];
477     char mensajeConsulta[MAX_NUM_PARTES_MENSAJE+1][MAX_LONGITUD_MENSAJE];
478     MYSQL* conector;
479     size_t longOrden=0;
480     int idControl;
481
482     fijarMensajesMonitor();
483
484     while(1){
485       sleep(PERIODO_MONITOR);
486
487       indiceMensaje=0;
488       while(strcmp(mensajesMonitor[indiceMensaje],"FIN")!=0){
489         sacarDatosMensaje(mensajesMonitor[indiceMensaje],mensajeConsulta)
          ;
490
491         pthread_mutex_lock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
          bloqueado
492         enviarDato(mensajesMonitor[indiceMensaje],((manejadores *) arg)->
          uart);
493
494         usleep(500000);
495         sem_post(&(semaforo[SEMAPHORE_UART_RX]));
496         usleep(100000);
497         printf("[monitorizarVariables] Esperando contestaci n...\n");
498         sem_wait(&(semaforo[SEMAPHORE_UART_RX]));
499         fprintf(stdout,"[monitorizarVariables] Contestaci n recbida:-%s
          -\n",mensajeRX);
500
501         if(strlen(mensajeRX)>0)
```

```
502         {
503           sacarDatosMensaje(mensajeRX,mensaje);
504         }
505
506
507         i=0;
508         while((strlen(mensajeRX)==0 || strcmp(mensaje[INDICE_ORDEN],
    mensajeConsulta[INDICE_ORDEN])!=0) && i<NUM_INTENTOS_CONSULTA){
509           enviarDato(mensajesMonitor[indiceMensaje],((manejadores *) arg)
    ->uart);
510           usleep(500000);
511           sem_post(&(semaforo[SEMAPHORE_UART_RX]));
512           usleep(100000);
513           printf("[monitorizarVariables] Esperando contestaci n...\n");
514           sem_wait(&(semaforo[SEMAPHORE_UART_RX]));
515           fprintf(stdout,"[monitorizarVariables] Contestaci n recbida:
    -%s-\n",mensajeRX);
516
517           if(strlen(mensajeRX)>0)
518           {
519             sacarDatosMensaje(mensajeRX,mensaje);
520           }
521           i++;
522         }
523         pthread_mutex_unlock(&(mutex[MUTEX_UART_TX])); //Mutex de UART
    desbloqueado
524
525         if(strlen(mensajeRX)==0){
526           printf("\n[monitorizarVariables] Error al obtener la petici n
    de valor de variable\n");
527         }
528         else{
529           printf("\n[monitorizarVariables] Dato de valor de variable
    obtenido.\n");
```

```
530           printf("\n[monitorizarVariables] Enviando mensaje respuesta a
      procesar...\n");
531           sacarDatosMensaje(mensajeRX,mensaje);
532
533           longOrden=strlen(mensaje[INDICE_ORDEN]);
534           if(strncmp(mensaje[INDICE_ORDEN],PETICION_TEMP,longOrden)==0){
535             idVariable = BBDD_ID_TIPOVAR_TEMPERATURA;
536           }
537           else if(strncmp(mensaje[INDICE_ORDEN],PETICION_ILUZ,longOrden)
      ==0){
538             idVariable = BBDD_ID_TIPOVAR_INTENSIDADLUZ;
539           }
540
541           idControl=strtol(mensaje[INDICE_DIRECCION],NULL,16);
542           valorVariable=strtod(mensaje[INDICE_VALOR],NULL);
543           timestamp=(int)time(NULL);
544
545           conector=abrirBBDD(BBDD_HOST,BBDD_USER,BBDD_PASS,BBDD_NAME);
546           idZona=sacarIdZona(conector,idControl);
547           registrarDatoMonitor(conector,idVariable,valorVariable,
      timestamp,idZona);
548           cerrarBBDD(conector);
549         }
550         indiceMensaje++;
551       }
552
553     }
554 }
555
556 //Hilo que simula un TIMER
557 void* controltimer (void *arg)
558 {
559   int idControl;
560   MYSQL* conector;
561   int idVariable;
```

```
562   double valorVariable;
563   int timestamp;
564   int idZona;
565
566   while(1)
567   {
568     usleep(20000);
569
570     pthread_mutex_lock(&(mutex[MUTEX_TIMER_CONTADOR]));
571     timerContador++;
572
573     if(timerContador >= 250)
574     {
575       timerContador = 0;
576
577       if(timerActivado==1)
578       {
579         pthread_mutex_lock(&(mutex[MUTEX_VARIABLE_MENSAJEGLOBAL]));
580         idVariable = BBDD_ID_TIPOVAR_LUZACTION_INTENSIDAD_USER;
581
582         idControl=strtol(mensajeGlobal[INDICE_DIRECCION],NULL,16);
583         valorVariable=strtod(mensajeGlobal[INDICE_VALOR],NULL);
584         pthread_mutex_unlock(&(mutex[MUTEX_VARIABLE_MENSAJEGLOBAL]));
585         timestamp=(int)time(NULL);
586
587         conector=abrirBBDD(BBDD_HOST,BBDD_USER,BBDD_PASS,BBDD_NAME);
588         idZona=sacarIdZona(conector,idControl);
589         registrarDatoMonitor(conector,idVariable,valorVariable,
    timestamp,idZona);
590         cerrarBBDD(conector);
591
592         pthread_mutex_lock(&(mutex[MUTEX_VARIABLE_TIMERACTIVADO]));
593         timerActivado = 0;
594         pthread_mutex_unlock(&(mutex[MUTEX_VARIABLE_TIMERACTIVADO]));
595       }
```

```
596
597         sem_wait(&(semaforo[SEMAPHORE_TIMER_ACTIVAR]));
598       }
599       pthread_mutex_unlock(&(mutex[MUTEX_TIMER_CONTADOR]));
600     }
601 }
602
603 /*****************************************
604 *    FUNCIONES DE COMUNICACIN CON SOCKET    *
605 *****************************************/
606
607 //Funci n que crea un socket en el puerto definido por el parametro
      puerto.
608 socket_datos crearSocket(unsigned short puerto, int numconexiones){
609   int descriptor1;
610   struct sockaddr_in servidor;
611   socket_datos maquina;
612
613   maquina.error = 0;
614
615   descriptor1 = socket(AF_INET,SOCK_STREAM,0);
616   if(descriptor1 < 0){
617     printf("\n[crearSocket] Error al crear el Socket (socket())\n");
618     maquina.error = 1;
619     return(maquina);
620   }
621
622   bzero((char *) &servidor, sizeof(servidor));
623   servidor.sin_family = AF_INET;
624   servidor.sin_port = htons(puerto);
625   servidor.sin_addr.s_addr = INADDR_ANY;
626
627   if(bind(descriptor1,(struct sockaddr*) &servidor,sizeof(servidor)) ==
        -1){
628     printf("\n[crearSocket] Error al activar el Socket (bind()).\n");
```

```
629       maquina.error=1;
630       return(maquina);
631     }
632
633     if(listen(descriptor1,numconexiones) == -1){
634       printf("\n[crearSocket] Error al establecer tarea de escucha en el
          Socket (listen())\n)");
635       maquina.error = 1;
636       return(maquina);
637     }
638
639     maquina.descriptor_socket = descriptor1;
640     maquina.info_maquina = servidor;
641
642     return(maquina);
643 }
644
645
646 /*************************************
647 *    FUNCIONES DE COMUNICACIN CON UART    *
648 *************************************/
649
650 //Funci n que abre la UART0 (Por el momento parece que ttyAMA0 solo
       dispone el RbP de  l ).
651 int abrir_uart(){
652     int descriptor_uart0 = -1;
653     struct termios opcionesUART;
654
655     descriptor_uart0 = open("/dev/ttyAMA0", O_RDWR | O_NOCTTY | O_NDELAY)
          ;
656     if (descriptor_uart0 == -1)
657     {
658       printf("\n[abrir_uart] Error al abrir la UART, compruebe que no
          est  siendo utilizada por otra aplicaci n.\n");
659       return(-1);
```

```
660     }
661     tcgetattr(descriptor_uart0, &opcionesUART);
662     opcionesUART.c_cflag = B9600 | CS8 | CLOCAL | CREAD;
663     opcionesUART.c_iflag = IGNPAR | ICRNL;
664     opcionesUART.c_oflag = 0;
665     opcionesUART.c_lflag = 0;
666     tcflush(descriptor_uart0, TCIFLUSH);
667     tcsetattr(descriptor_uart0, TCSANOW, &opcionesUART);
668
669     usleep(10000);
670
671     return(descriptor_uart0);
672 }
673
674 //Funci n que env a el dato datoOut[] por la UART
675 int enviarDato (char datoOut[],int uart){
676     unsigned char tx_buffer[20];
677     unsigned char *p_tx_buffer;
678     int respuesta = 2;
679     int i;
680     int numCaractEnviados;
681
682     fprintf(stdout,"\n[enviarDato] Enviando dato-%s-...\n",datoOut);
683
684     p_tx_buffer = &tx_buffer[0];
685
686     i=0;
687     while(datoOut[i] != '\0'){
688       *p_tx_buffer++ = datoOut[i];
689       i++;
690     }
691     *p_tx_buffer++ = '\n';
692     if (uart != -1)
693     {
694       respuesta=1;
```

```
695       numCaractEnviados = write(uart, &tx_buffer[0], (p_tx_buffer - &
          tx_buffer[0]));
696
697       if (numCaractEnviados < 0)
698       {
699         printf("\n[enviarDato] UART TX error\n");
700         respuesta = 0;
701       }
702     }
703     printf("\n[enviarDato] Fin de envio de dato.\n");
704
705     return (respuesta);
706 }
707
708
709 /***************************************
710 *    FUNCIONES DE COMUNICACIN CON BBDD    *
711 ***************************************/
712
713 void cerrarBBDD (MYSQL* descriptorBBDD){
714     printf("\n[cerrarBBDD] Cerrando BBDD...\n");
715     mysql_close(descriptorBBDD);
716     printf("\n[cerrarBBDD] BBDD cerrada\n");
717 }
718
719 MYSQL* abrirBBDD (char maquina[],char usuario[],char password[],char
          baseDatos[]){
720     MYSQL *con;
721
722     printf("\n[abrirBBDD] Abriendo BBDD...\n");
723
724     con = mysql_init(NULL);
725
726     if (con == NULL){
727       fprintf(stderr, "%s\n", mysql_error(con));
```

```
728       return(con);
729     }
730     else{
731       if (mysql_real_connect(con, maquina, usuario, password, baseDatos,
          0, NULL, 0) == NULL){
732         fprintf(stderr, "%s\n", mysql_error(con));
733         mysql_close(con);
734         return(con);
735       }
736       else{
737         printf("\n[abrirBBDD] BBDD abierta\n");
738         return(con);
739       }
740     }
741 }
742
743 int registrarDatoMonitor(MYSQL* conect, int idvariable, double valor,
      int timestamp, int idzona){
744   char query[255];
745
746   printf("\n[registrarDatoMonitor] Registrando dato en BBDD...\n");
747
748   snprintf(query,sizeof(query),"INSERT INTO registrovars (idvariable,
       valor,idzona) VALUES(%d,%lf,%d);",idvariable,valor,idzona);
749
750   fprintf(stdout,"\n[registrarDatoMonitor] Query: %s\n",query);
751
752   if (mysql_query(conect, query))
753   {
754     fprintf(stderr, "\n[registrarDatoMonitor] %s\n", mysql_error(conect
       ));
755     mysql_close(conect);
756     return(0);
757   }
758   else
```

```
759   {
760     printf("\n[registrarDatoMonitor] Dato registrado en BBDD\n");
761     return(1);
762   }
763 }
764 int sacarIdZona(MYSQL* conect,int idControlador){
765   char query[100];
766   MYSQL_RES *result;
767   int num_fields=0;
768   MYSQL_ROW row;
769   int i;
770   int idZona;
771   int resultadoQuery;
772
773   sprintf(query,"SELECT id FROM zonas WHERE idcontrolador=%d;",
        idControlador);
774
775   printf("\n[sacarIdZona] Query: %s\n",query);
776
777   resultadoQuery = mysql_query(conect, query);
778
779   if (resultadoQuery != 0)
780   {
781     printf("\n[sacarIdZona] Error al consultar con Query (Error: %d)\n"
        ,resultadoQuery);
782   }
783
784   result = mysql_store_result(conect);
785
786   if (result == NULL)
787   {
788     printf("\n[sacarIdZona] Error al sacar los datos de la consulta\n")
        ;
789   }
790
```

```
791   num_fields = mysql_num_fields(result);
792
793   while ((row = mysql_fetch_row(result)))
794   {
795     for(i = 0; i < num_fields; i++)
796     {
797       idZona=atoi(row[i]);
798     }
799   }
800
801   mysql_free_result(result);
802   return(idZona);
803 }
804
805
806 /*************************************
807 *    FUNCIONES DE CONTROL DE MENSAJES    *
808 *************************************/
809
810 void sacarDatosMensaje(char buffer[],char datosMensaje[
      MAX_NUM_PARTES_MENSAJE+1][MAX_LONGITUD_MENSAJE]){
811   int contadorBuffer=0;
812   int contadorMensaje=0;
813   int contadorPartes=0;
814   int parteMensaje=1;
815
816   fprintf(stdout,"\n[sacarDatosMensaje] Procesando datos de mensaje-%s
       -...\n",buffer);
817
818   while(buffer[contadorBuffer] != ' ' && buffer[contadorBuffer] != 10
      && buffer[contadorBuffer] != 13){
819     datosMensaje[INDICE_MENSAJE][contadorMensaje] = buffer[
      contadorBuffer];
820
821     if(buffer[contadorBuffer]!='_'){
```

```
822         datosMensaje[parteMensaje][contadorPartes] = buffer[
        contadorBuffer];
823         contadorPartes++;
824       }
825       else{
826         datosMensaje[parteMensaje][contadorPartes] = '\0';
827         contadorPartes=0;
828         parteMensaje++;
829       }
830       contadorBuffer++;
831       contadorMensaje++;
832     }
833     datosMensaje[parteMensaje][contadorPartes] = '\0';
834     datosMensaje[INDICE_MENSAJE][contadorMensaje] = '\r';
835     contadorMensaje++;
836     datosMensaje[INDICE_MENSAJE][contadorMensaje] = '\0';
837
838     printf("\n\n\tMensaje: %s",datosMensaje[INDICE_MENSAJE]);
839     printf("\n\tDireccion: %s",datosMensaje[INDICE_DIRECCION]);
840     printf("\n\tValor: %s",datosMensaje[INDICE_VALOR]);
841     printf("\n\tOrden: %s\n\n",datosMensaje[INDICE_ORDEN]);
842
843     printf("\n[sacarDatosMensaje] Datos de mensaje procesados\n");
844 }
845
846 void sacarMensajeCabecera(char buffer[],char mensaje[]){
847     int contadorBuffer=14;
848     int contador=0;
849
850     while(buffer[contadorBuffer]!=' '){
851       mensaje[contador]=buffer[contadorBuffer];
852
853       contadorBuffer++;
854       contador++;
855     }
```

```
856     mensaje[contador]='\r';
857     contador++;
858     mensaje[contador]='\0';
859 }
860
861 void fijarMensajesMonitor(void){
862     sprintf(mensajesMonitor[0],"%s_%s\r",DIRECCION_SENSOR_SALON,
          PETICION_TEMP);
863     sprintf(mensajesMonitor[1],"%s_%s\r",DIRECCION_SENSOR_SALON,
          PETICION_ILUZ);
864
865     strcpy(mensajesMonitor[2],"FIN\0");
866 }
```

/

2. Código del Controlador Sensor

```
1  #include <STDLIB.h>
2  #include <STRING.h>
3  #include <RTX51TNY.h>
4  #include <math.h>
5  #include "c8051f410.h"
6
7  void setup_micro(void);
8  unsigned int unirDatos(char datoAlto, char datoBajo);
9  float sacarTension (char dato[]);
10 void procesarMensaje();
11
12 #define TIMER2_INT 5
13 #define ADC_INT 10
14
15 #define INICIO 0
16 #define PRINCIPAL 1
17 #define ENVIA 2
18 #define RECIBE 3
19 #define CONTROLAR_VARIABLES 4
20
21 #define VREF_TEMP 3.365
22 #define VREF_LIGHT 2.48
23 #define VREF_ADC 2.48
24 #define NUMBITS_ADC 12
25
26 #define ORDEN_RESET "RST"
27
28 #define ORDEN_LUZ_ENCENDER "LZON"
29 #define ORDEN_LUZ_APAGAR "LZOFF"
30 #define ORDEN_FIJAR_INTENSIDAD_LUZ "LZFR"
31 #define ORDEN_CALEFACCION_ENCENDER "CLON"
32 #define ORDEN_CALEFACCION_APAGAR "CLOFF"
33
```

```
34 #define ORDEN_PEDIR_INTENSIDAD_LUZ "MILZ"
35 #define ORDEN_PEDIR_VALOR_TEMPERATURA "TEMP"
36
37 #define ORDEN_AUTO_TEMPERATURA_APAGAR "AUTTMPOFF"
38 #define ORDEN_AUTO_TEMPERATURA_ENCENDER "AUTTMPON"
39 #define ORDEN_TEMPERATURA_DESEADA_FIJAR "TMPDS"
40 #define ORDEN_AUTO_LUZ_APAGAR "AUTLZOFF"
41 #define ORDEN_AUTO_LUZ_ENCENDER "AUTLZON"
42 #define ORDEN_LUZ_DESEADA_FIJAR "ILZDS"
43
44 #define INDICADOR_LUZ_ENCENDIDA "INDLZON"
45 #define INDICADOR_LUZ_APAGADA "INDLZOFF"
46 #define INDICADOR_CALEFACCION_ENCENDIDA "INDCLON"
47 #define INDICADOR_CALEFACCION_APAGADA "INDCLOFF"
48
49 #define DIRECCION 0
50 #define ORDEN 1
51 #define VALOR 2
52 #define TOTALCAMPOS 3
53 #define MAXCHARSDATOSMENSAJES 20
54 #define MAXCHARSMENSAJE 30
55
56 #define DIRECCION_CONTROLADOR "276B"
57 #define DIRECCION_CONTROLADOR_LUZ "276C"
58 #define DIRECCION_CONTROLADOR_TEMPERATURA "276C"
59 #define DIRECCION_GLOBAL "FFFF"
60
61 #define TEMP_SENSOR_COEF 0.0195
62 #define TEMP_SENSOR_CERO 0.4
63
64 #define SENSIBILIDAD_SENS_LUZ 0.00001
65 #define RESISTENCIA_SENS_LUZ 44000
66 #define SALIDA_BASE 1
67
68 #define PERIODO_TIMER2_INTERRUPT 0.025
```

```
69 #define PERIODO_MIRAR_VARIABLES 1 //En Segundos
70 #define MAX_CONTADOR_INTERRUPS_TIMER2 (PERIODO_MIRAR_VARIABLES/
       PERIODO_TIMER2_INTERRUPT)
71
72 #define HISTERESIS_LUZ 50
73 #define HISTERESIS_TEMP 1
74
75 xdata char comando[TOTALCAMPOS][MAXCHARSDATOSMENSAJES];
76 xdata char mensaje[MAXCHARSMENSAJE];
77 char dutty_cycle=0,dutty_cycle2=0;
78 int overflow_T2=0;
79 xdata char datoADC[2];
80
81 xdata float iluminacionDeseada = 10;
82 xdata char iluminacionDeseada_encendido = 0;
83 xdata float temperaturaDeseada = 22;
84 xdata char temperaturaDeseada_encendido = 0;
85
86 char adc_ocupado = 0;
87 char tarea_origen_peticion_adc = 0;
88
89 xdata char luz_encendida = 0;
90 xdata char calefaccion_encendida = 0;
91
92
93
94 /*Funcion de setup y ceracion de tareas*/
95 void inicio (void)_task_ INICIO
96 {
97   setup_micro();
98   os_create_task(ENVIA);
99   os_create_task(RECIBE);
100  os_create_task(PRINCIPAL);
101  os_create_task(CONTROLAR_VARIABLES);
102  os_delete_task(INICIO);
```

```
103 }
104
105
106
107 /*****************************************/
108 /*************TAREA PRINCIPAL*************/
109 /*****************************************/
110
111 void principal(void)_task_ PRINCIPAL
112 {
113   xdata float tension;
114   xdata float t_ambiente;
115   xdata char temperatura[MAXCHARSDATOSMENSAJES];
116   xdata float ilum_ambiente;
117   xdata char iluminica[MAXCHARSDATOSMENSAJES];
118
119   while(1)
120   {
121     os_wait(K_SIG,0,0);
122     procesarMensaje();                    //Esperamos recibir un evento
      para comenzar a enviar
123
124     if(strcmp(comando[DIRECCION],DIRECCION_CONTROLADOR)==0 || strcmp(
      comando[DIRECCION],DIRECCION_GLOBAL)==0)
125     {
126       if(strcmp(comando[ORDEN],ORDEN_PEDIR_VALOR_TEMPERATURA)==0)
127       {
128         while(adc_ocupado != 0);
129         adc_ocupado = 1;
130         tarea_origen_peticion_adc = PRINCIPAL;
131
132         ADC0MX&=0xE0;
133         ADC0MX|=0x08;
134         ADC0CN|=0x10;
135
```

```
136         os_wait(K_SIG,0,0);
137
138         tension = sacarTension (datoADC);
139         t_ambiente = (tension - TEMP_SENSOR_CERO)/TEMP_SENSOR_COEF;
140         sprintf(temperatura, "%f", t_ambiente);
141         sprintf(mensaje,"%s_%s_%s\r",comando[DIRECCION],comando[ORDEN],
    temperatura);
142         os_send_signal(ENVIA);
143         os_wait(K_SIG,0,0);
144
145         adc_ocupado = 0;
146       }
147       else if(strcmp(comando[ORDEN],ORDEN_PEDIR_INTENSIDAD_LUZ)==0)
148       {
149         while(adc_ocupado != 0);
150         adc_ocupado = 1;
151         tarea_origen_peticion_adc = PRINCIPAL;
152
153         ADC0MX&=0xE0;
154         ADC0MX|=0x09;
155         ADC0CN|=0x10;
156
157         os_wait(K_SIG,0,0);
158
159         tension = sacarTension (datoADC);
160         ilum_ambiente = pow(10,(tension/(SENSIBILIDAD_SENS_LUZ*
    RESISTENCIA_SENS_LUZ)))*SALIDA_BASE;
161         sprintf(iluminica, "%f", ilum_ambiente);
162         sprintf(mensaje,"%s_%s_%s\r",comando[DIRECCION],comando[ORDEN],
    iluminica);
163         os_send_signal(ENVIA);
164         os_wait(K_SIG,0,0);
165
166         adc_ocupado = 0;
167       }
```

```
168         else if(strcmp(comando[ORDEN],ORDEN_AUTO_TEMPERATURA_ENCENDER)==0
        || strcmp(comando[ORDEN],ORDEN_AUTO_LUZ_ENCENDER)==0)
169         {
170           if(strcmp(comando[ORDEN],ORDEN_AUTO_TEMPERATURA_ENCENDER)==0)
171           {
172             temperaturaDeseada_encendido = 1;
173           }
174           if(strcmp(comando[ORDEN],ORDEN_AUTO_LUZ_ENCENDER)==0)
175           {
176             iluminacionDeseada_encendido = 1;
177           }
178           TR2 = 1;
179         }
180         else if(strcmp(comando[ORDEN],ORDEN_AUTO_TEMPERATURA_APAGAR)==0)
181         {
182           temperaturaDeseada_encendido = 0;
183         }
184         else if(strcmp(comando[ORDEN],ORDEN_AUTO_LUZ_APAGAR)==0)
185         {
186           iluminacionDeseada_encendido = 0;
187         }
188         else if(strcmp(comando[ORDEN],ORDEN_TEMPERATURA_DESEADA_FIJAR)
      ==0)
189         {
190           temperaturaDeseada = atof(comando[VALOR]);
191         }
192         else if(strcmp(comando[ORDEN],ORDEN_LUZ_DESEADA_FIJAR)==0)
193         {
194           iluminacionDeseada = atof(comando[VALOR]);
195         }
196         else if(strcmp(comando[ORDEN],INDICADOR_LUZ_ENCENDIDA)==0)
197         {
198           luz_encendida = 1;
199         }
200         else if(strcmp(comando[ORDEN],INDICADOR_LUZ_APAGADA)==0)
```

```
201         {
202           luz_encendida = 0;
203         }
204         else if(strcmp(comando[ORDEN],INDICADOR_CALEFACCION_ENCENDIDA)
    ==0)
205         {
206           calefaccion_encendida = 1;
207         }
208         else if(strcmp(comando[ORDEN],INDICADOR_CALEFACCION_APAGADA)==0)
209         {
210           calefaccion_encendida = 0;
211         }
212         else if(strcmp(comando[ORDEN],ORDEN_RESET)==0)
213         {
214           RSTSRC |= 0x10;
215         }
216       }
217
218       os_switch_task ();                    //Permiso para cambiar de Tarea
219     }
220 }
221
222
223
224
225
226 /*****************************************/
227 /************ENVIO Y RECEPCION************/
228 /*****************************************/
229
230
231 /**************FUNCION DE ENVIO***********/
232 void envia(void)_task_ ENVIA
233 {
234   int i=0;
```

```
235   while(1)
236   {
237     os_wait(K_SIG,0,0);          //Esperamos recibir  un evento para
      comenzar a enviar
238     for(i=0;i<=strlen(mensaje);i++)      //Calculamos cuantos caracteres
       tenemos que enviar
239     {
240       SBUF0=mensaje[i];         //Sacamos el dato por SBUF0
241       os_wait(K_SIG,0,0);         //Esperamos a que la interrupcion nos
      avise de que se ha enviado el caracter
242     }
243     os_send_signal(tarea_origen_peticion_adc);         //Signal tarea
      PRINCIPAL
244     os_switch_task();          //Permitimos cambio tarea
245   }
246 }
247
248
249
250 /***********FUNCION DE RECEPCION*********/
251 void recibe(void)_task_ RECIBE
252 {
253   int x=0;
254   while(1)
255   {
256     os_wait(K_SIG,0,0);
257     if((SBUF0>=32)&&(SBUF0<=95))
258     {
259       while(SBUF0!='\r')              //Leemos mientras que el caracter
      sea distinto del CR(retorno de carro)
260       {
261         mensaje[x]=SBUF0;             //Almacenamos el caracter en una
      variable
262         x++;
```

```
263           os_wait(K_SIG,0,0);              //Esperamos a que la interrupcion
      nos avise de que se ha recibido un caracter
264         }
265         mensaje[x]='\0';
266         os_send_signal(PRINCIPAL);
267
268         x=0;                        //Reseteamos la variable
269         os_switch_task();                //Permitimos cambio tarea
270       }
271     }
272 }
273
274
275 /*****************************************/
276 /**************INTERRUPCIONES**************/
277 /*****************************************/
278
279 /**********INTERRUPCION UART**************/
280 void uart(void) interrupt INTERRUPT_UART0
281 {
282   if (RI0)
283   {
284     RI0=0;
285     os_clear_signal(PRINCIPAL);
286     isr_send_signal(RECIBE);
287   }
288
289   if(TI0)
290   {
291     TI0=0;
292     isr_send_signal(ENVIA);
293   }
294 }
295
296
```

```
297 /*************INTERRUPCION TIMER2*****************/
298 void timer2(void) interrupt TIMER2_INT
299 {
300   TF2H = 0;                 // Clear the interrupt request
301
302   overflow_T2++;        // Aumentamos contador de overflow
303   if(overflow_T2==MAX_CONTADOR_INTERRUPS_TIMER2)
304   {
305     os_send_signal(CONTROLAR_VARIABLES);
306     overflow_T2=0;      //Si llegamos a 100 reseteamos el contador
307   }
308 }
309
310 /*************INTERRUPCION ADC*****************/
311 void adc_complete(void) interrupt ADC_INT
312 {
313   datoADC[1] = ADC0L;
314   datoADC[0] = ADC0H;
315   AD0INT = 0;
316   os_send_signal(tarea_origen_peticion_adc);
317 }
318
319 /*****************************************/
320 /********** Tareas Adicionales *************/
321 /*****************************************/
322
323 void controlar_variables(void)_task_ CONTROLAR_VARIABLES
324 {
325   xdata float tension;
326   xdata float t_ambiente;
327   xdata float ilum_ambiente;
328
329   while(1)
330   {
331     os_wait(K_SIG,0,0);
```

```
332
333     if(temperaturaDeseada_encendido == 0 &&
        iluminacionDeseada_encendido == 0)
334     {
335       TR2 = 0;
336     }
337     else
338     {
339       if(temperaturaDeseada_encendido == 1){
340         //LECTURA DE LA TEMPERATURA
341         while(adc_ocupado != 0);
342         adc_ocupado = 1;
343         tarea_origen_peticion_adc = CONTROLAR_VARIABLES;
344
345         ADC0MX&=0xE0;
346         ADC0MX|=0x08;
347         ADC0CN|=0x10;
348
349         os_wait(K_SIG,0,0);
350
351         tension = sacarTension (datoADC);
352         t_ambiente = (tension - TEMP_SENSOR_CERO)/TEMP_SENSOR_COEF;
353
354         if(t_ambiente < temperaturaDeseada - HISTERESIS_TEMP &&
        calefaccion_encendida == 0){
355           sprintf(mensaje,"%s_%s\r",DIRECCION_CONTROLADOR_TEMPERATURA,
        ORDEN_CALEFACCION_ENCENDER);
356           os_send_signal(ENVIA);
357           os_wait(K_SIG,0,0);
358         }
359         else if(t_ambiente > temperaturaDeseada + HISTERESIS_TEMP &&
        calefaccion_encendida == 1)
360         {
361           sprintf(mensaje,"%s_%s\r",DIRECCION_CONTROLADOR_TEMPERATURA,
        ORDEN_CALEFACCION_APAGAR);
```

```
362             os_send_signal(ENVIA);
363             os_wait(K_SIG,0,0);
364           }
365
366           adc_ocupado = 0;
367         }
368
369         if(iluminacionDeseada_encendido == 1){
370           //LECTURA DE LA INTENSIDAD LUMNICA
371           while(adc_ocupado != 0);
372           adc_ocupado = 1;
373           tarea_origen_peticion_adc = CONTROLAR_VARIABLES;
374
375           ADC0MX&=0xE0;
376           ADC0MX|=0x09;
377           ADC0CN|=0x10;
378
379           os_wait(K_SIG,0,0);
380
381           tension = sacarTension (datoADC);
382           ilum_ambiente = pow(10,(tension/(SENSIBILIDAD_SENS_LUZ*
      RESISTENCIA_SENS_LUZ)))*SALIDA_BASE;
383
384           if(ilum_ambiente < iluminacionDeseada - HISTERESIS_LUZ &&
      luz_encendida == 0){
385             sprintf(mensaje,"%s_%s\r",DIRECCION_CONTROLADOR_LUZ,
      ORDEN_LUZ_ENCENDER);
386             os_send_signal(ENVIA);
387             os_wait(K_SIG,0,0);
388           }
389           else if(ilum_ambiente > iluminacionDeseada + HISTERESIS_LUZ &&
      luz_encendida == 1)
390           {
391             sprintf(mensaje,"%s_%s\r",DIRECCION_CONTROLADOR_LUZ,
      ORDEN_LUZ_APAGAR);
```

```
392             os_send_signal(ENVIA);
393             os_wait(K_SIG,0,0);
394           }
395
396           adc_ocupado = 0;
397         }
398       }
399
400       os_switch_task ();
401     }
402 }
403
404
405 /*****************************************/
406 /***********SFR'S MICRO 80C51F410***********/
407 /*****************************************/
408
409
410 void setup_micro(void)
411 {
412   //CLKSEL=0x00;
413   OSCICN|=0x87;    //Oscilador interno con divisor de frecuencia a 1
414   EA=0;          //Interrupciones desactivadas
415   XBR0=0x01;       //TX0 y RX0 de la UART configurados en los pines P0.4
        y P0.5
416   XBR1=0x40;       //Crossbar activado
417   P1MDIN=0x00;     //Configura P1.0 y P1.1 como entradas analogicas
418   P1|=0x03;       //escribimos un '1' en el Port Latch
419   P0MDOUT=0xFF;    //Puerto 0 est  en modo salida Push-Pull
420   P1SKIP=0XFF;     //Saltamos los pines en el crossbar para ADC
421
422   //UART
423   SCON0&=0x7F;     //UART modo 8bits
424     SCON0|=0x10;     //UART con habilitacion de recepcion
425   ES0=1;          //Habilitaci n de la interrupci n de la UART
```

```
426   PS0=1;          //Prioridad de la interrupci n de la UART fijada en
        baja prioridad
427
428   //TIMER 1 UART
429   TH1=0x96;       //Valor de recarga del timer para CLK=24,5MHZ y
        baudrate de 9600
430   TCON=0x45;      //Interrupciones de los Timers 1 y 0 configuradas por
         detecci n de flanco.
431   TMOD&=0x0F;     //Inicializamos la parte del registro correspondiente
         al Timer 1
432   TMOD=0x20;      //Timer 1 8bits modo autorrecarga
433   CKCON&=0xF4;    //Timer 1 configurado con SYSCLK/12, Timer 0
        configurado con SYSCLK, Timer 2 configurado con SYSCLK, Timer 3
        configurado con SYSCLK
434   TR1=1;          //Timer 1 arrancado
435   P0=0x00;
436   P2=0x01;
437
438   //PCA Y PWM
439   PCA0MD=0x02;    //deshabilita watchdog y frecuencia PCA = Timer 0
        Overflow
440   //P0SKIP=0X01;     //configuramos la salida del PWM CEX0 en el P0.0
441   PCA0CPM0=0x42;    //PWM 8bits
442
443
444   //ADC
445   ADC0TK&=0xF0;   //Inicializados los bits de modo de tracking
446   ADC0TK|=0x07;   //Modo post tracking con un tiempo de 16 ciclos de
        reloj
447   ADC0CF&=0xF8;
448   ADC0CN&=0xFC;
449
450   REF0CN|=0x08;   //Vdd Usada como tensi n de referencia
451   EIE1|=0x08;     //Habilitada la interrupci n de conversi n
        completada del ADC0
```

```
452   ADC0CN|=0x80;    //Activado el ADC y listo para usarse
453
454   //TIMER2
455   TMR2CN=0x00;     //Inicio del registro de control del timer 2
456   T2SPLIT=0;       //Timer 2 trabaja como timer de 16 bits con
        autorecarga
457   TF2LEN=0;      //Desactivada la interrupci n del overflow del timer 2
458   CKCON&=0xEF;       //Frecuencia de incremento del Timer 2 fijado por
        los 2 siguientes par mentros
459   TMR2CN&=0xFE;    //Frecuencis del timer 2 fijada en SYSCLK/12
460
461
462   TMR2RLH=0x38;    //Con esta recarga, la interrupci n salta cada 25 ms
463   TMR2RLL=0x9D;
464   ET2 = 1;              // Enable Timer 2 Interrupts
465
466
467   ET0=1;
468   EA=1;         //habilitacion interrupciones globales
469 }
470
471 unsigned int unirDatos(char datoAlto, char datoBajo)
472 {
473   int result = 0;
474
475   result = 0x00FF & datoAlto;
476   result = result << 8;
477   result = result + (0x00FF & datoBajo);
478
479   return result;
480 }
481
482 float sacarTension (char dato[])
483 {
484   int temp_d;
```

```
485    float tension;
486
487    temp_d=unirDatos(dato[0],dato[1]);
488    tension = (VREF_ADC/pow(2,NUMBITS_ADC))*temp_d;
489
490    return(tension);
491 }
492
493 void procesarMensaje()
494 {
495    int x=0;
496    int y=0;
497    int parteMensaje=0;
498
499    while(mensaje[x]!='\0')
500    {
501       if(mensaje[x]=='_')
502       {
503          comando[parteMensaje][y]='\0';
504          y=0;
505          parteMensaje++;
506       }
507       else
508       {
509          comando[parteMensaje][y]=mensaje[x];
510          y++;
511       }
512       x++;
513    }
514    comando[parteMensaje][y]='\0';
515 }
```

3. Código del Controlador Actuador

```
516 #include <STDLIB.h>
517 #include <STRING.h>
518 #include <RTX51TNY.h>
519 #include <math.h>
520 #include "c8051f410.h"
521
522 void setup_micro(void);
523 unsigned int unirDatos(char datoAlto, char datoBajo);
524 float sacarTension (char dato[]);
525 void procesarMensaje();
526
527 #define TIMER2_INT 5
528
529 #define INICIO 0
530 #define PRINCIPAL 1
531 #define ENVIA 2
532 #define RECIBE 3
533 #define LEDS 4
534
535 #define ORDEN_RESET "RST"
536
537 #define ORDEN_LUZ_ENCENDER "LZON"
538 #define ORDEN_LUZ_APAGAR "LZOFF"
539 #define ORDEN_FIJAR_INTENSIDAD_LUZ "LZFR"
540 #define ORDEN_CALEFACCION_ENCENDER "CLON"
541 #define ORDEN_CALEFACCION_APAGAR "CLOFF"
542
543 #define INDICADOR_LUZ_ENCENDIDA "INDLZON"
544 #define INDICADOR_LUZ_APAGADA "INDLZOFF"
545 #define INDICADOR_CALEFACCION_ENCENDIDA "INDCLON"
546 #define INDICADOR_CALEFACCION_APAGADA "INDCLOFF"
547
548 #define FREC_SYSCLK 24500000
```

```
549 #define FREC_LEDS 70
550 #define VALOR_MAX_OVERFLOW_T2_LOW 200
551 #define DUTTY_CYCLE_INICIAL_LUZ_ENCENDER 0
552
553 #define DIRECCION 0
554 #define ORDEN 1
555 #define VALOR 2
556 #define TOTALCAMPOS 3
557 #define MAXCHARSDATOSMENSAJES 20
558 #define MAXCHARSMENSAJE 30
559
560 #define DIRECCION_CONTROLADOR "276C"
561 #define DIRECCION_CONTROLADOR_SENSOR "276B"
562 #define DIRECCION_GLOBAL "FFFF"
563
564 xdata char comando[TOTALCAMPOS][MAXCHARSDATOSMENSAJES];
565 xdata char mensaje[MAXCHARSMENSAJE];
566 unsigned char dutty_cycle=0, dutty_cycle_mem=50, dutty_cycle_objetivo
        =0;
567 unsigned int overflow_T2_high=0;
568 unsigned char overflow_T2_low=0;
569 xdata char luz_encendida = 0;
570
571
572 /*Funcion de setup y ceracion de tareas*/
573 void inicio (void)_task_ INICIO
574 {
575   setup_micro();
576   os_create_task(ENVIA);
577   os_create_task(RECIBE);
578   os_create_task(PRINCIPAL);
579   os_create_task(LEDS);
580   os_delete_task(INICIO);
581 }
582
```

```
583
584
585 /*****************************************/
586 /************TAREA PRINCIPAL**************/
587 /*****************************************/
588
589 void principal(void)_task_ PRINCIPAL
590 {
591   while(1)
592   {
593     os_wait(K_SIG,0,0);
594     procesarMensaje();                    //Esperamos recibir un evento
      para comenzar a enviar
595
596     if(strcmp(comando[DIRECCION],DIRECCION_CONTROLADOR)==0 || strcmp(
      comando[DIRECCION],DIRECCION_GLOBAL)==0)
597     {
598
599       if(strcmp(comando[ORDEN],ORDEN_FIJAR_INTENSIDAD_LUZ)==0)
600       {
601         if(luz_encendida==1)
602         {
603           dutty_cycle_objetivo=atoi(comando[VALOR]);
604           dutty_cycle_mem=dutty_cycle_objetivo;
605           TR2 = 1;
606           TF2LEN=1;
607           luz_encendida=1;
608           /*strcat(comando,"\r");          //A adimos \r al comando para
      reenviarlo de vuelta
609           os_send_signal(ENVIA);*/
610         }
611       }
612       else if(strcmp(comando[ORDEN],ORDEN_LUZ_ENCENDER)==0)
613       {
614         if(luz_encendida == 0)
```

```
615             {
616               overflow_T2_high=0;
617               overflow_T2_low=0;
618               dutty_cycle=DUTTY_CYCLE_INICIAL_LUZ_ENCENDER;
619               dutty_cycle_objetivo=dutty_cycle_mem;
620               TR2 = 1;
621               TF2LEN=1;
622               luz_encendida=1;
623
624               sprintf(mensaje,"%s_%s\r",DIRECCION_CONTROLADOR_SENSOR,
    INDICADOR_LUZ_ENCENDIDA);
625               os_send_signal(ENVIA);
626             }
627         }
628         else if(strcmp(comando[ORDEN],ORDEN_LUZ_APAGAR)==0)
629         {
630           //overflow_T2_high=0;
631           //overflow_T2_low=0;
632           dutty_cycle_objetivo=0;
633           luz_encendida=0;
634           TR2 = 1;
635           TF2LEN=1;
636
637           sprintf(mensaje,"%s_%s\r",DIRECCION_CONTROLADOR_SENSOR,
    INDICADOR_LUZ_APAGADA);
638           os_send_signal(ENVIA);
639         }
640         else if(strcmp(comando[ORDEN],ORDEN_CALEFACCION_ENCENDER)==0)
641         {
642           P1|=0x80;
643
644           sprintf(mensaje,"%s_%s\r",DIRECCION_CONTROLADOR_SENSOR,
    INDICADOR_CALEFACCION_ENCENDIDA);
645           os_send_signal(ENVIA);
646         }
```

```
647         else if(strcmp(comando[ORDEN],ORDEN_CALEFACCION_APAGAR)==0)
648         {
649           P1&=0x7F;
650
651           sprintf(mensaje,"%s_%s\r",DIRECCION_CONTROLADOR_SENSOR,
      INDICADOR_CALEFACCION_APAGADA);
652           os_send_signal(ENVIA);
653         }
654         else if(strcmp(comando[ORDEN],ORDEN_RESET)==0)
655         {
656           RSTSRC |= 0x10;
657         }
658       }
659
660       os_switch_task ();                    //Permiso para cambiar de Tarea
661     }
662 }
663
664
665
666
667
668 /*****************************************/
669 /************ENVIO Y RECEPCION************/
670 /*****************************************/
671
672
673 /**************FUNCION DE ENVIO************/
674 void envia(void)_task_ ENVIA
675 {
676   int i=0;
677   while(1)
678   {
679     os_wait(K_SIG,0,0);             //Esperamos recibir  un evento para
      comenzar a enviar
```

```
680       for(i=0;i<=strlen(mensaje);i++)      //Calculamos cuantos caracteres
          tenemos que enviar
681       {
682         SBUF0=mensaje[i];          //Sacamos el dato por SBUF0
683         os_wait(K_SIG,0,0);          //Esperamos a que la interrupcion nos
          avise de que se ha enviado el caracter
684       }
685       os_switch_task();            //Permitimos cambio tarea
686     }
687 }
688
689
690
691 /***********FUNCION DE RECEPCION**********/
692 void recibe(void)_task_ RECIBE
693 {
694   int x=0;
695   while(1)
696   {
697     os_wait(K_SIG,0,0);
698     if((SBUF0>=32)&&(SBUF0<=95))
699     {
700       while(SBUF0!='\r')               //Leemos mientras que el caracter
        sea distinto del CR(retorno de carro)
701       {
702         mensaje[x]=SBUF0;              //Almacenamos el caracter en una
        variable
703         x++;
704         os_wait(K_SIG,0,0);              //Esperamos a que la interrupcion
         nos avise de que se ha recibido un caracter
705       }
706       mensaje[x]='\0';
707       os_send_signal(PRINCIPAL);
708
709       x=0;                        //Reseteamos la variable
```

```
710         os_switch_task();             //Permitimos cambio tarea
711       }
712     }
713 }
714
715
716 /*****************************************/
717 /*************INTERRUPCIONES**************/
718 /*****************************************/
719
720 /**********INTERRUPCION UART*************/
721 void uart(void) interrupt INTERRUPT_UART0
722 {
723   if (RI0)
724   {
725     RI0=0;
726     os_clear_signal(PRINCIPAL);
727     isr_send_signal(RECIBE);
728   }
729
730   if(TI0)
731   {
732     TI0=0;
733     isr_send_signal(ENVIA);
734   }
735 }
736
737
738 /***********INTERRUPCION TIMER2***********/
739 void timer2(void) interrupt TIMER2_INT
740 {
741   if(TF2L == 1)
742   {
743     TF2L = 0;
744     overflow_T2_low++;       // Aumentamos contador de overflow
```

```
745
746       if(overflow_T2_low>=VALOR_MAX_OVERFLOW_T2_LOW)
747       {
748         overflow_T2_low=0;      //Si llegamos a 200 reseteamos el contador
749       }
750
751       if(dutty_cycle < 100)
752       {
753         if(overflow_T2_low<VALOR_MAX_OVERFLOW_T2_LOW-(dutty_cycle*(
      VALOR_MAX_OVERFLOW_T2_LOW/100)))        //En funcion del dutty cycle
      encenderemos o apagaremos el puerto 1
754         {
755           P2&=0xFB;
756         }
757         else
758         {
759           P2|=0x04;
760         }
761       }
762       else
763       {
764         P2|=0x04;
765       }
766     }
767     else if(TF2H == 1)
768     {
769       TF2H = 0;
770       overflow_T2_high++;
771       if(overflow_T2_high >=100)
772       {
773         if(dutty_cycle_objetivo>dutty_cycle)
774         {
775           overflow_T2_high=0;
776           dutty_cycle++;
777         }
```

```
778         else if(dutty_cycle_objetivo<dutty_cycle)
779         {
780           overflow_T2_high=0;
781           dutty_cycle--;
782         }
783         else
784         {
785           TR2=0;
786           overflow_T2_high=0;
787           if(luz_encendida == 0)
788           {
789             TF2LEN=0;
790           }
791         }
792       }
793     }
794 }
795
796 /*****************************************/
797 /***************FUNCIONES*****************/
798 /*****************************************/
799
800 /*****************************************/
801 /**********SFR'S MICRO 80C51F410**********/
802 /*****************************************/
803
804
805 void setup_micro(void)
806 {
807   xdata unsigned long aux1, aux2, autoload_T2L, valorOverflow,
        fecuencia_leds, frecuencia_sysclk;
808
809   OSCICN|=0x87;    //Oscilador interno con divisor de frecuencia a 1
810   EA=0;          //Interrupciones desactivadas
```

```
811   XBR0=0x01;        //TX0 y RX0 de la UART configurados en los pines P0.4
        y P0.5
812
813   P0MDOUT=0xFF;     //Puerto 0 est  en modo salida Push-Pull
814
815   P2MDOUT=0xFF;
816   P2=0x00;          //Inicializamos el puerto 2 con todo a nivel bajo
817
818   XBR1=0x40;        //Crossbar activado
819
820   //UART
821   SCON0&=0x7F;      //UART modo 8bits
822     SCON0|=0x10;      //UART con habilitacion de recepcion
823   ES0=1;            //Habilitaci n de la interrupci n de la UART
824   PS0=1;            //Prioridad de la interrupci n de la UART fijada en
        baja prioridad
825
826   //TIMER 1 UART
827   TH1=0x96;         //Valor de recarga del timer para CLK=24,5MHZ y
        baudrate de 9600
828   TCON=0x45;        //Interrupciones de los Timers 1 y 0 configuradas por
        detecci n de flanco.
829   TMOD&=0x0F;       //Inicializamos la parte del registro correspondiente
        al Timer 1
830   TMOD=0x20;        //Timer 1 8bits modo autorrecarga
831   CKCON&=0xE4;      //Timer 1 configurado con SYSCLK/12, Timer 0
        configurado con SYSCLK, Timer 2 configurado con SYSCLK/12 para los
        2 timers de 8 bits, Timer 3 configurado con SYSCLK
832   TR1=1;            //Timer 1 arrancado
833   P0=0x00;
834   P1=0x00;
835
836   //PCA Y PWM
837   PCA0MD=0x02;      //deshabilita watchdog y frecuencia PCA = Timer 0
        Overflow
```

```
838   PCA0CPM0=0x42;      //PWM 8 bits
839
840   //TIMER2
841   valorOverflow = VALOR_MAX_OVERFLOW_T2_LOW;
842   fecuencia_leds = FREC_LEDS;
843   frecuencia_sysclk = FREC_SYSCLK;
844
845   aux1 = (valorOverflow * 12 * fecuencia_leds);
846   aux2 = (frecuencia_sysclk / aux1);
847   autoload_T2L = (255-aux2);
848
849   TMR2CN=0x00;
850   T2SPLIT=1;        //Timer 2 como 2 timers de 8 bits
851   TMR2RLH=0x00;
852   TMR2RLL=(char) autoload_T2L; //0x92;
853   IP|=0x20;              //Prioridad de la interrupci n del TIMER 2 en
        alta prioridad
854   ET2 = 1;                          // Enable Timer 2 Interrupts
855
856   ET0=1;
857   EA=1;         //habilitacion interrupciones globales
858 }
859
860 void procesarMensaje()
861 {
862   int x=0;
863   int y=0;
864   int parteMensaje=0;
865
866   while(mensaje[x]!='\0')
867   {
868     if(mensaje[x]=='_')
869     {
870       comando[parteMensaje][y]='\0';
871       y=0;
```

```
872             parteMensaje++;
873         }
874         else
875         {
876             comando[parteMensaje][y]=mensaje[x];
877             y++;
878         }
879         x++;
880     }
881     comando[parteMensaje][y]='\0';
882 }
```

Anexo B

Esquematicos y Layouts de las Placas Desarrolladas

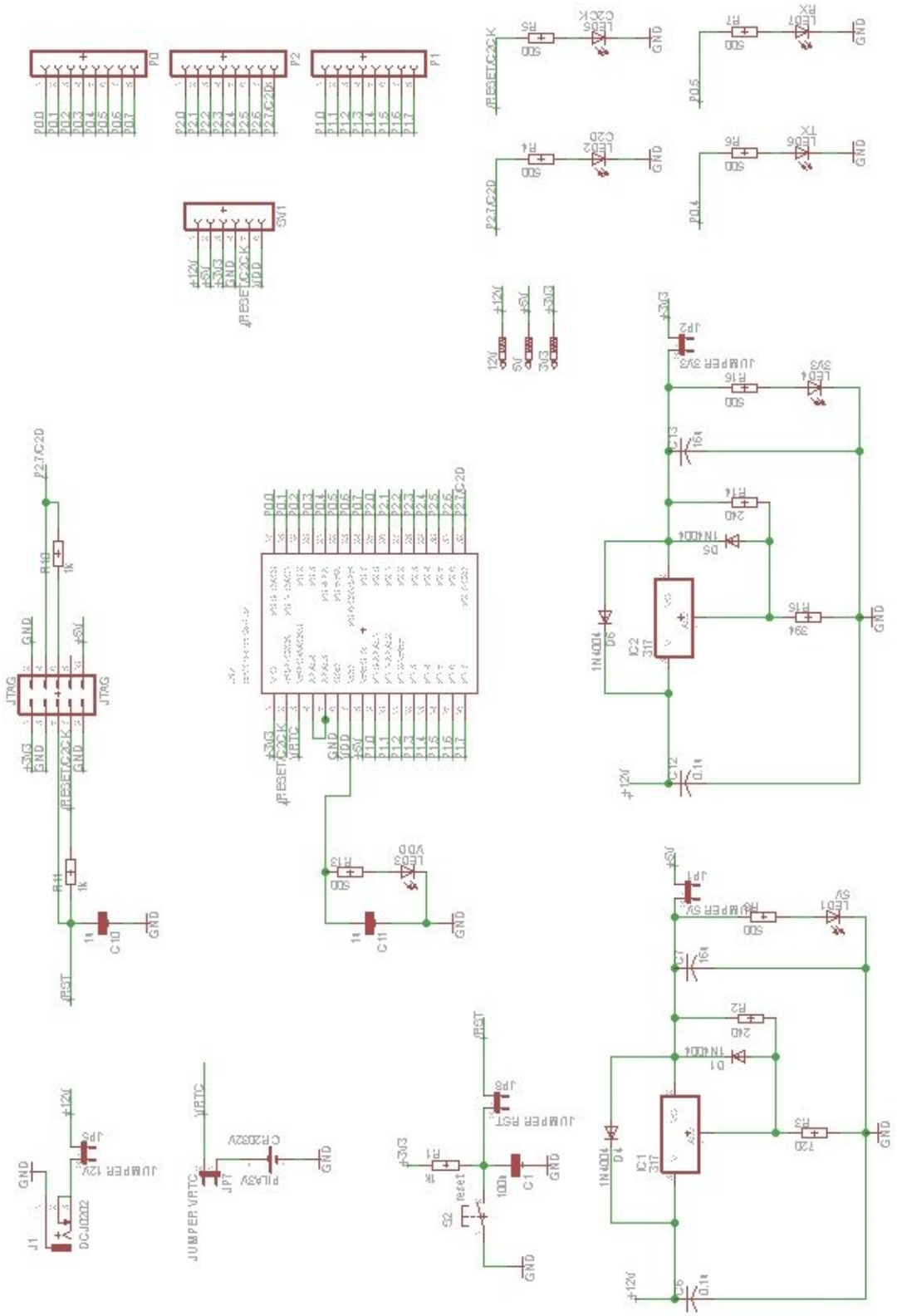


Figura B.1: *Esquemático del Circuito Completo de la placa Controladora*

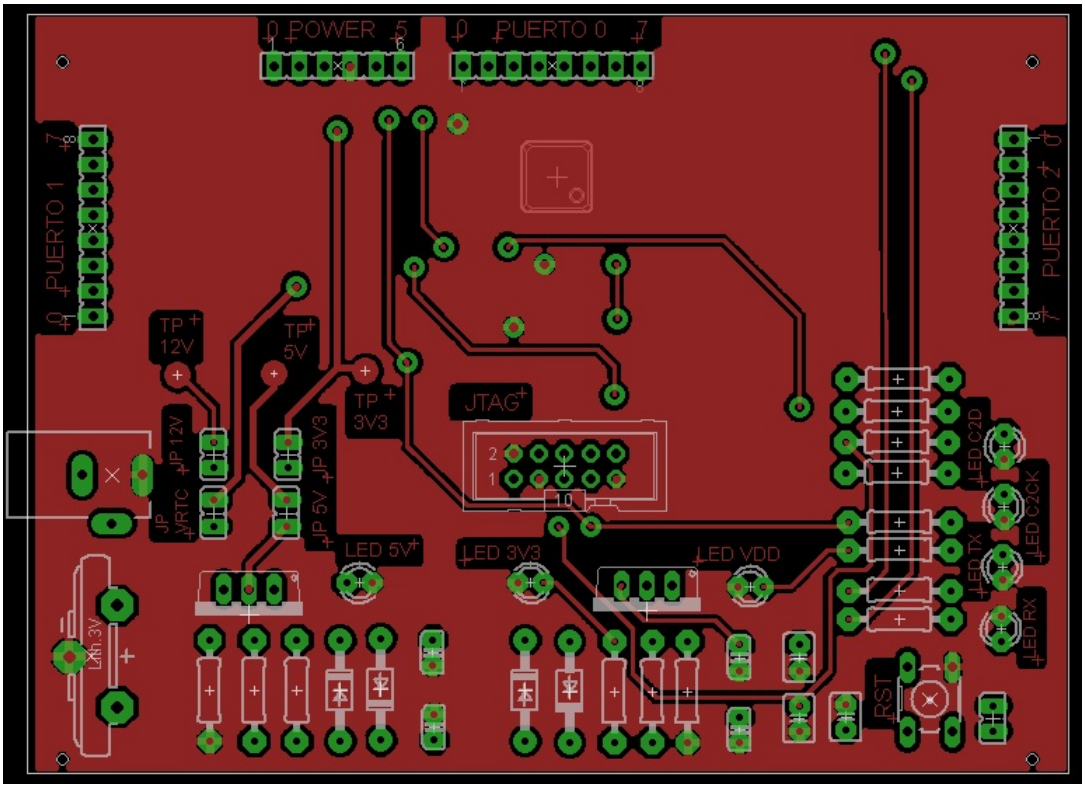


Figura B.2: *Layout Top del Circuito Completo de la placa Controladora*

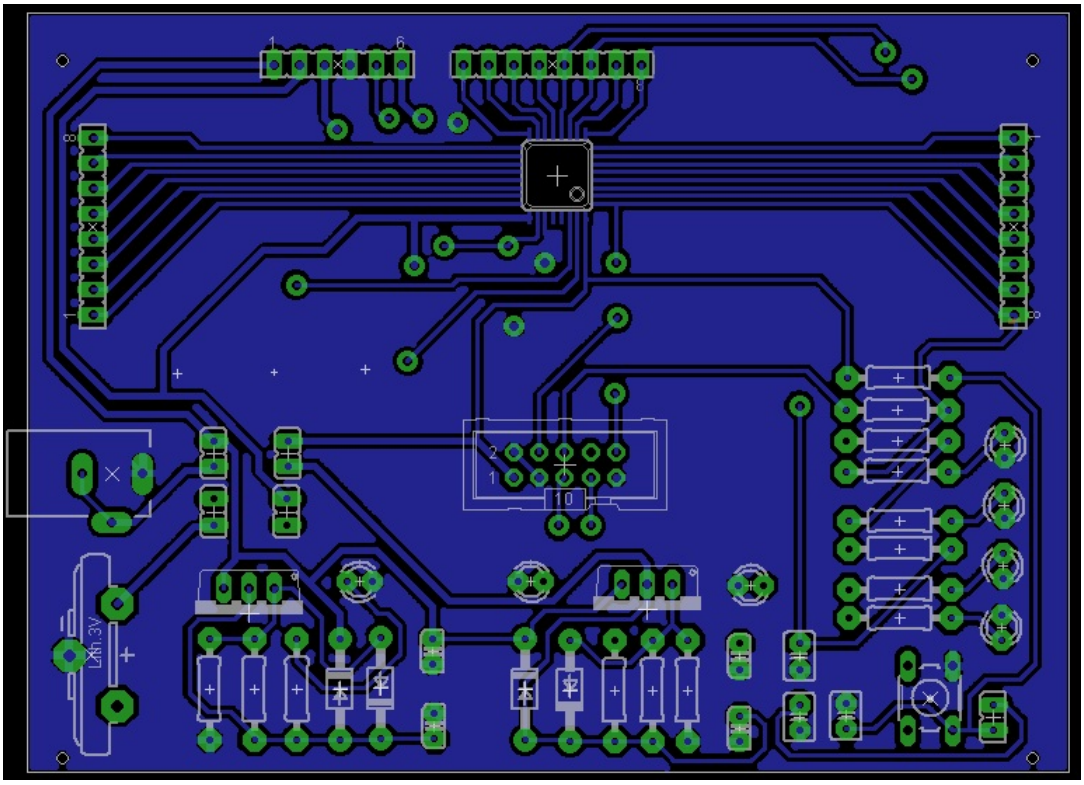


Figura B.3: *Layout Bottom del Circuito Completo de la placa Controladora*

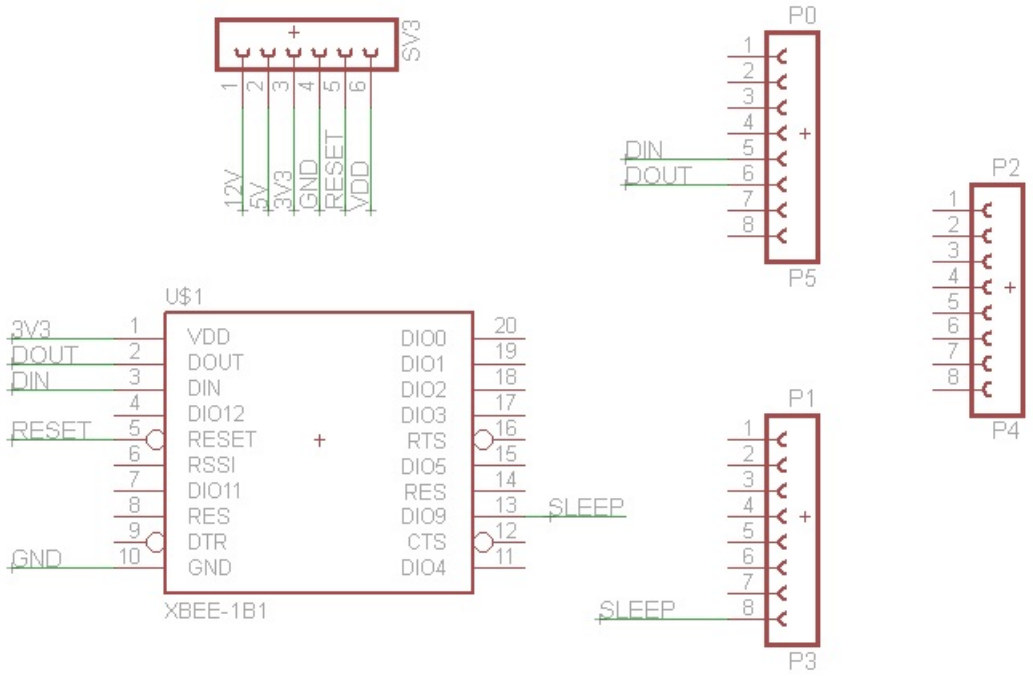


Figura B.4: *Esquemático de la Placa Adaptadora Xbee*

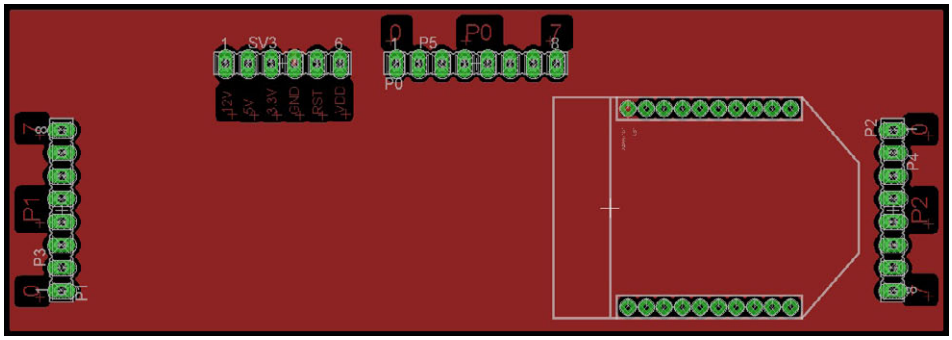


Figura B.5: *Layout Top de la Placa Adaptadora Xbee*

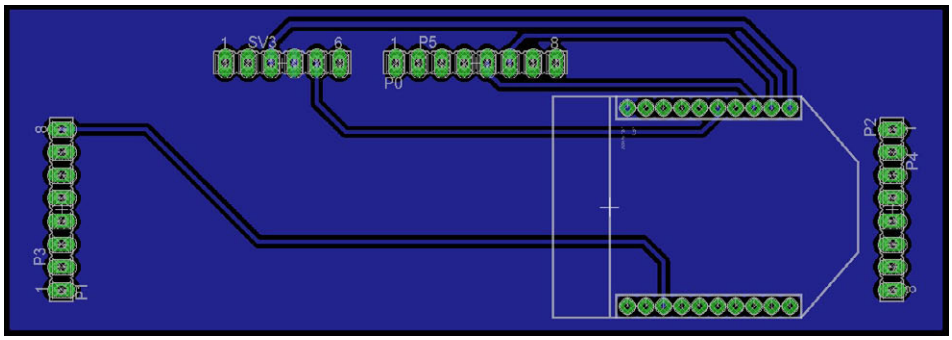


Figura B.6: *Layout Bottom de la Placa Adaptadora Xbee*

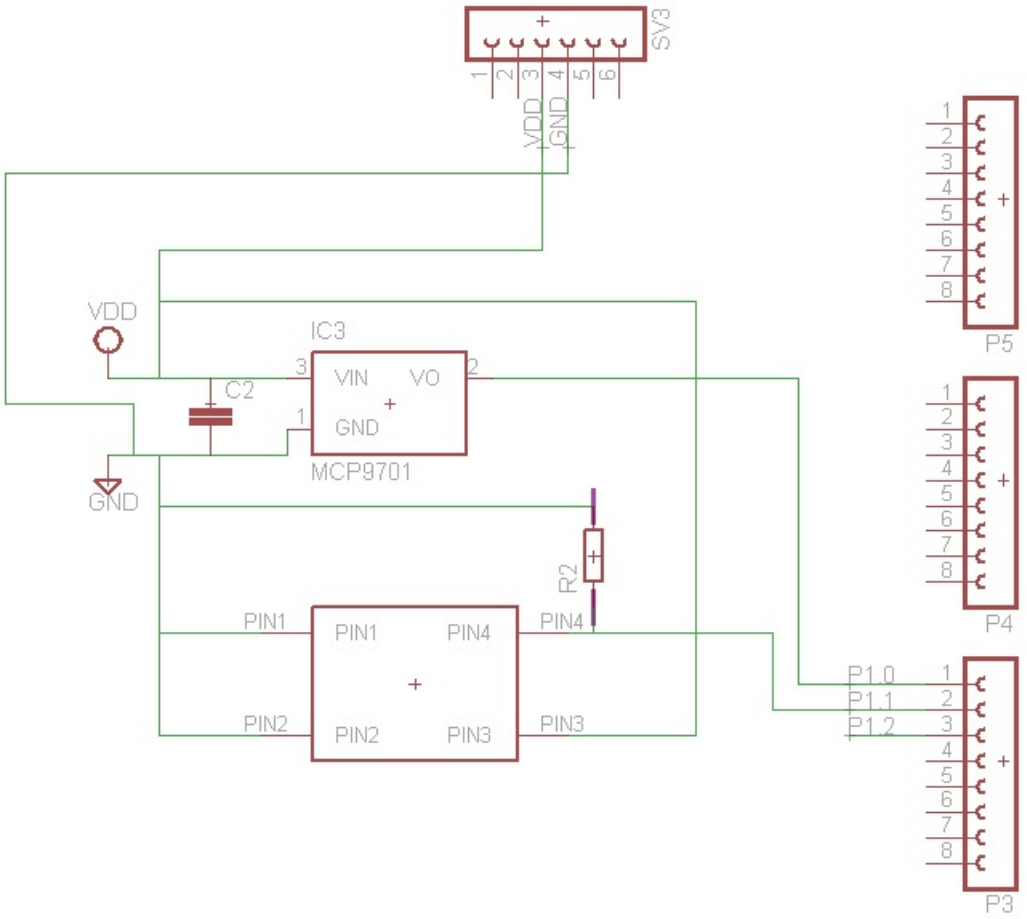


Figura B.7: *Esquemático de la Placa Complementaria Sensora*

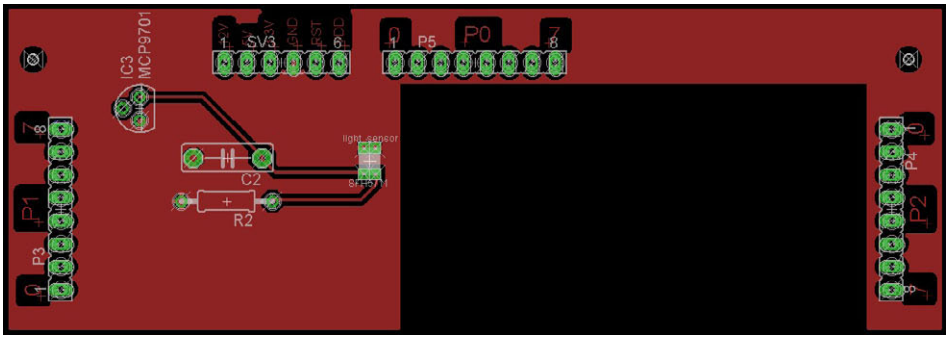


Figura B.8: *Layout Top de la Placa Complementaria Sensora*

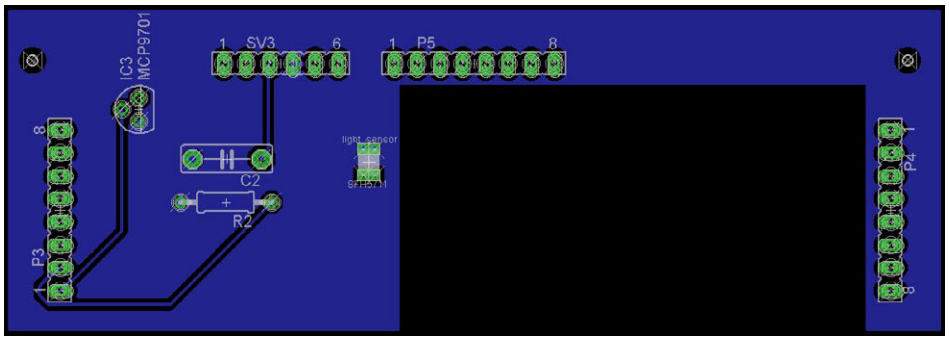


Figura B.9: *Layout Bottom de la Placa Complementaria Sensora*

Anexo C

Datasheets de Componentes Electrónicos

C8051F410/1/2/3

2.0 V, 32/16 kB Flash, smaRTClock, 12-bit ADC

Analog Peripherals
- **12-Bit ADC**
  - ±1 LSB INL; no missing codes
  - Programmable throughput up to 200 ksps
  - Up to 24 external inputs
  - Data dependent windowed interrupt generator
  - Built-in temperature sensor (±3 °C)
- **Two 12-Bit Current Mode DACs**
- **Two Comparators**
  - Programmable hysteresis and response time
  - Configurable as wake-up or reset source
- **POR/Brownout Detector**
- **Voltage Reference—1.5, 2.2 V (programmable)**

On-Chip Debug
- On-chip debug circuitry facilitates full-speed, non-intrusive in-system debug (No emulator required)
- Provides breakpoints, single stepping
- Inspect/modify memory and registers
- Complete development kit

Supply Voltage 2.0 to 5.25 V
- Built-in LDO regulator: 2.1 or 2.5 V

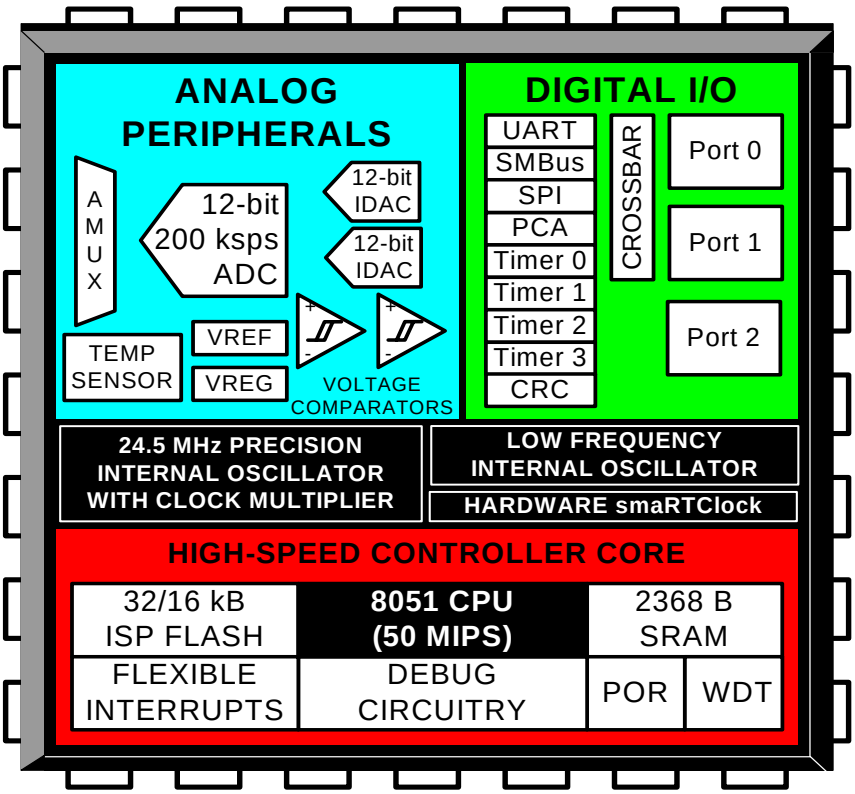
High Speed 8051 µC Core
- Pipelined instruction architecture; executes 70% of instructions in 1 or 2 system clocks
- Up to **50 MIPS** throughput with 50 MHz system clock
- Expanded interrupt handler

Memory
- 2304 bytes internal data RAM (256 + 2048)
- 32/16 kB Flash; In-system programmable in 512 byte sectors
- 64 bytes battery-backed RAM (smaRTClock)

Digital Peripherals
- 24 port I/O; push-pull or open-drain, up to 5.25 V tolerance
- Hardware SMBus™ (I2C™ Compatible), SPI™, and UART serial ports available concurrently
- Four general purpose 16-bit counter/timers
- Programmable 16-bit counter/timer array with six capture/compare modules, WDT
- Hardware smaRTClock operates down to 1 V with 64 bytes battery-backed RAM and backup voltage regulator

Clock Sources
- Internal oscillators: 24.5 MHz 2% accuracy supports UART operation; clock multiplier up to 50 MHz
- External oscillator: Crystal, RC, C, or Clock (1 or 2 pin modes)
- smaRTClock oscillator: 32 kHz Crystal or self-resonant oscillator
- Can switch between clock sources on-the-fly

32-PIN LQFP or 28-PIN 5x5 QFN
Temperature Range: –40 to +85 °C



1. XBee/XBee-PRO OEM RF Modules

The XBee and XBee-PRO OEM RF Modules were engineered to meet IEEE 802.15.4 standards and support the unique needs of low-cost, low-power wireless sensor networks. The modules require minimal power and provide reliable delivery of data between devices.

The modules operate within the ISM 2.4 GHz frequency band and are pin-for-pin compatible with each other.

1.1. Key Features

Long Range Data Integrity

XBee

- Indoor/Urban: up to 100' (30 m)
- Outdoor line-of-sight: up to 300' (100 m)
- Transmit Power: 1 mW (0 dBm)
- Receiver Sensitivity: -92 dBm

XBee-PRO

- Indoor/Urban: up to 300' (100 m)
- Outdoor line-of-sight: up to 1 mile (1500 m)
- Transmit Power: 100 mW (20 dBm) EIRP
- Receiver Sensitivity: -100 dBm

RF Data Rate: 250,000 bps

Advanced Networking & Security

Retries and Acknowledgements

DSSS (Direct Sequence Spread Spectrum)

Each direct sequence channels has over 65,000 unique network addresses available

Source/Destination Addressing

Unicast & Broadcast Communications

Point-to-point, point-to-multipoint and peer-to-peer topologies supported

Coordinator/End Device operations

Low Power

XBee

- TX Current: 45 mA (@3.3 V)
- RX Current: 50 mA (@3.3 V)
- Power-down Current: < 10 µA

XBee-PRO

- TX Current: 215 mA (@3.3 V)
- RX Current: 55 mA (@3.3 V)
- Power-down Current: < 10 µA

ADC and I/O line support

Analog-to-digital conversion, Digital I/O

I/O Line Passing

Easy-to-Use

No configuration necessary for out-of box RF communications

Free X-CTU Software (Testing and configuration software)

AT and API Command Modes for configuring module parameters

Extensive command set

Small form factor

Free & Unlimited RF-XPert Support

1.1.1. Worldwide Acceptance

FCC Approval (USA) Refer to Appendix A [p57] for FCC Requirements. Systems that contain XBee/XBee-PRO RF Modules inherit MaxStream Certifications.

ISM (Industrial, Scientific & Medical) **2.4 GHz frequency band**

Manufactured under **ISO 9001:2000** registered standards

XBee/XBee-PRO RF Modules are optimized for use in the **United States**, **Canada**, **Australia, Israel and Europe**. Contact MaxStream for complete list of government agency approvals.

1.2. Specifications

Table 1-01. Specifications of the XBee/XBee-PRO OEM RF Modules

Specification	XBee	XBee-PRO
Performance		
Indoor/Urban Range	up to 100 ft. (30 m)	Up to 300' (100 m)
Outdoor RF line-of-sight Range	up to 300 ft. (100 m)	Up to 1 mile (1500 m)
Transmit Power Output (software selectable)	1mW (0 dBm)	60 mW (18 dBm) conducted, 100 mW (20 dBm) EIRP*
RF Data Rate	250,000 bps	250,000 bps
Serial Interface Data Rate (software selectable)	1200 - 115200 bps (non-standard baud rates also supported)	1200 - 115200 bps (non-standard baud rates also supported)
Receiver Sensitivity	-92 dBm (1% packet error rate)	-100 dBm (1% packet error rate)
Power Requirements		
Supply Voltage	2.8 – 3.4 V	2.8 – 3.4 V
Transmit Current (typical)	45mA (@ 3.3 V)	If PL=0 (10dBm): 137mA(@3.3V), 139mA(@3.0V) PL=1 (12dBm): 155mA (@3.3V), 153mA(@3.0V) PL=2 (14dBm): 170mA (@3.3V), 171mA(@3.0V) PL=3 (16dBm): 188mA (@3.3V), 195mA(@3.0V) PL=4 (18dBm): 215mA (@3.3V), 227mA(@3.0V)
Idle / Receive Current (typical)	50mA (@ 3.3 V)	55mA (@ 3.3 V)
Power-down Current	< 10 µA	< 10 µA
General		
Operating Frequency	ISM 2.4 GHz	ISM 2.4 GHz
Dimensions	0.960" x 1.087" (2.438cm x 2.761cm)	0.960" x 1.297" (2.438cm x 3.294cm)
Operating Temperature	-40 to 85º C (industrial)	-40 to 85º C (industrial)
Antenna Options	Integrated Whip, Chip or U.FL Connector	Integrated Whip, Chip or U.FL Connector
Networking & Security		
Supported Network Topologies	Point-to-point, Point-to-multipoint & Peer-to-peer	
Number of Channels (software selectable)	16 Direct Sequence Channels	12 Direct Sequence Channels
Addressing Options	PAN ID, Channel and Addresses	PAN ID, Channel and Addresses
Agency Approvals		
United States (FCC Part 15.247)	OUR-XBEE	OUR-XBEEPRO
Industry Canada (IC)	4214A XBEE	4214A XBEEPRO
Europe (CE)	ETSI	ETSI (Max. 10 dBm transmit power output)*
Japan	n/a	005NYCA0378 (Max. 10 dBm transmit power output)**

* When operating in Europe: XBee-PRO RF Modules must be configured to operate at a maximum transmit power output level of 10 dBm. The power output level is set using the PL command. The PL parameter must equal "0" (10 dBm).

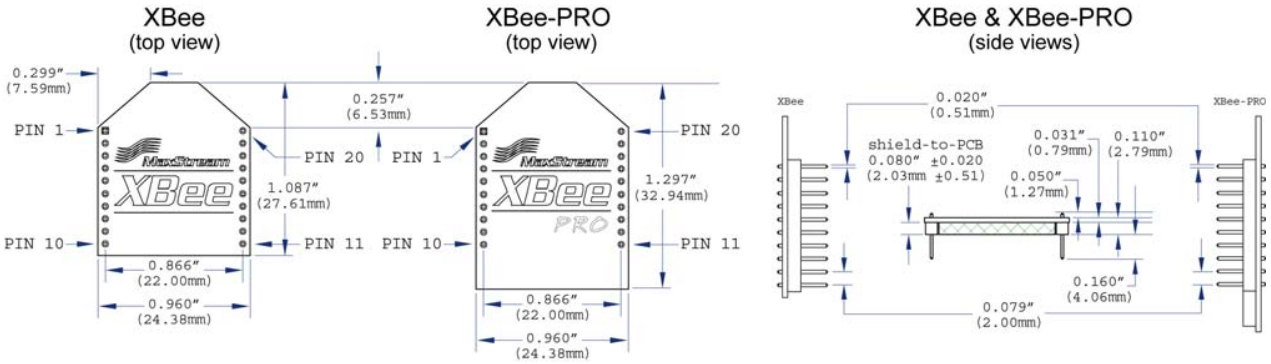
Additionally, European regulations stipulate an EIRP power maximum of 12.86 dBm (19 mW) for the XBee-PRO and 12.11 dBm for the XBee when integrating high-gain antennas.

** When operating in Japan: Transmit power output is limited to 10 dBm. A special part number is required when ordering modules approved for use in Japan. Contact MaxStream for more information [call 1-801-765-9885 or send e-mails to sales@maxstream.net].

Antenna Options: The ranges specified are typical when using the integrated Whip (1.5 dBi) and Dipole (2.1 dBi) antennas. The Chip antenna option provides advantages in its form factor; however, it typically yields shorter range than the Whip and Dipole antenna options when transmitting outdoors. For more information, refer to the "XBee Antenna" application note located on MaxStream's web site (http://www.maxstream.net/support/knowledgebase/article.php?kb=153).

1.3. Mechanical Drawings

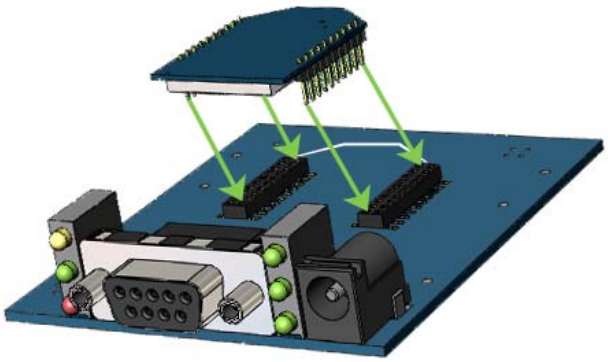
Figure 1-01. Mechanical drawings of the XBee/XBee-PRO OEM RF Modules (antenna options not shown)
The XBee and XBee-PRO RF Modules are pin-for-pin compatible.



1.4. Mounting Considerations

The XBee/XBee-PRO RF Module was designed to mount into a receptacle (socket) and therefore does not require any soldering when mounting it to a board. The XBee Development Kits contain RS-232 and USB interface boards which use two 20-pin receptacles to receive modules.

Figure 1-02. XBee Module Mounting to an RS-232 Interface Board.



The receptacles used on MaxStream development boards are manufactured by Century Interconnect. Several other manufacturers provide comparable mounting solutions; however, MaxStream currently uses the following receptacles:

- Through-hole single-row receptacles -
  Samtec P/N: MMS-110-01-L-SV (or equivalent)
- Surface-mount double-row receptacles -
  Century Interconnect P/N: CPRMSL20-D-0-1 (or equivalent)
- Surface-mount single-row receptacles -
  Samtec P/N: SMM-110-02-SM-S

MaxStream also recommends printing an outline of the module on the board to indicate the orientation the module should be mounted.

1.5. Pin Signals

Figure 1-03. XBee/XBee-PRO RF Module Pin Numbers
(top sides shown - shields on bottom)

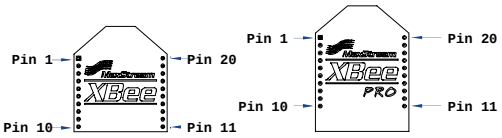


Table 1-02. Pin Assignments for the XBee and XBee-PRO Modules
(Low-asserted signals are distinguished with a horizontal line above signal name.)

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / CONFIG	Input	UART Data In
4	DO8*	Output	Digital Output 8
5	RESET	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	PWM1	Output	PWM Output 1
8	[reserved]	-	Do not connect
9	DTR / SLEEP_RQ / DI8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog Input 4 or Digital I/O 4
12	CTS / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / SLEEP	Output	Module Status Indicator
14	VREF	Input	Voltage Reference for A/D Inputs
15	Associate / AD5 / DIO5	Either	Associated Indicator, Analog Input 5 or Digital I/O 5
16	RTS / AD6 / DIO6	Either	Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

* Function is not supported at the time of this release

Design Notes:

- Minimum connections: VCC, GND, DOUT & DIN
- Minimum connections for updating firmware: VCC, GND, DIN, DOUT, RTS & DTR
- Signal Direction is specified with respect to the module
- Module includes a 50k Ω pull-up resistor attached to RESET
- Several of the input pull-ups can be configured using the PR command
- Unused pins should be left disconnected

1.6. Electrical Characteristics

Table 1-03. DC Characteristics (VCC = 2.8 - 3.4 VDC)

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
V_{IL}	Input Low Voltage	All Digital Inputs	-	-	0.35 * VCC	V
V_{IH}	Input High Voltage	All Digital Inputs	0.7 * VCC	-	-	V
V_{OL}	Output Low Voltage	I_{OL} = 2 mA, VCC >= 2.7 V	-	-	0.5	V
V_{OH}	Output High Voltage	I_{OH} = -2 mA, VCC >= 2.7 V	VCC - 0.5	-	-	V
II_{IN}	Input Leakage Current	V_{IN} = VCC or GND, all inputs, per pin	-	0.025	1	µA
II_{OZ}	High Impedance Leakage Current	V_{IN} = VCC or GND, all I/O High-Z, per pin	-	0.025	1	µA
TX	Transmit Current	VCC = 3.3 V	-	45 (XBee) / 215 (PRO)	-	mA
RX	Receive Current	VCC = 3.3 V	-	50 (XBee) / 55 (PRO)	-	mA
PWR-DWN	Power-down Current	SM parameter = 1	-	< 10	-	µA

Table 1-04. ADC Characteristics (Operating)

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
V_{REFH}	VREF - Analog-to-Digital converter reference range		2.08	-	V_{DDAD}	V
I_{REF}	VREF - Reference Supply Current	Enabled	-	200	-	µA
		Disabled or Sleep Mode	-	< 0.01	0.02	µA
V_{INDC}	Analog Input Voltage[1]		V_{SSAD} - 0.3	-	V_{DDAD} + 0.3	V

1. Maximum electrical operating range, not valid conversion range.

Table 1-05. ADC Timing/Performance Characteristics[1]

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
R_{AS}	Source Impedance at Input[2]		-	-	10	kΩ
V_{AIN}	Analog Input Voltage[3]		V_{REFL}		V_{REFH}	V
RES	Ideal Resolution (1 LSB)[4]	2.08V ≤ V_{DDAD} ≤ 3.6V	2.031	-	3.516	mV
DNL	Differential Non-linearity[5]		-	±0.5	±1.0	LSB
INL	Integral Non-linearity[6]		-	±0.5	±1.0	LSB
E_{ZS}	Zero-scale Error[7]		-	±0.4	±1.0	LSB
F_{FS}	Full-scale Error[8]		-	±0.4	±1.0	LSB
E_{IL}	Input Leakage Error[9]		-	±0.05	±5.0	LSB
E_{TU}	Total Unadjusted Error[10]		-	±1.1	±2.5	LSB

1. All ACCURACY numbers are based on processor and system being in WAIT state (very little activity and no IO switching) and that adequate low-pass filtering is present on analog input pins (filter with 0.01 µF to 0.1 µF capacitor between analog input and VREFL). Failure to observe these guidelines may result in system or microcontroller noise causing accuracy errors which will vary based on board layout and the type and magnitude of the activity.
Data transmission and reception during data conversion may cause some degradation of these specifications, depending on the number and timing of packets. It is advisable to test the ADCs in your installation if best accuracy is required.
2. R_{AS} is the real portion of the impedance of the network driving the analog input pin. Values greater than this amount may not fully charge the input circuitry of the ATD resulting in accuracy error.
3. Analog input must be between V_{REFL} and V_{REFH} for valid conversion. Values greater than V_{REFH} will convert to $3FF.
4. The resolution is the ideal step size or 1LSB = (V_{REFH}–V_{REFL})/1024
5. Differential non-linearity is the difference between the current code width and the ideal code width (1LSB). The current code width is the difference in the transition voltages to and from the current code.
6. Integral non-linearity is the difference between the transition voltage to the current code and the adjusted ideal transition voltage for the current code. The adjusted ideal transition voltage is (Current Code–1/2)*(1/((V_{REFH}+E_{FS})–(V_{REFL}+E_{ZS}))).
7. Zero-scale error is the difference between the transition to the first valid code and the ideal transition to that code. The Ideal transition voltage to a given code is (Code–1/2)*(1/(V_{REFH}–V_{REFL})).
8. Full-scale error is the difference between the transition to the last valid code and the ideal transition to that code. The ideal transition voltage to a given code is (Code–1/2)*(1/(V_{REFH}–V_{REFL})).
9. Input leakage error is error due to input leakage across the real portion of the impedance of the network driving the analog pin. Reducing the impedance of the network reduces this error.
10. Total unadjusted error is the difference between the transition voltage to the current code and the ideal straight-line transfer function. This measure of error includes inherent quantization error (1/2LSB) and circuit error (differential, integral, zero-scale, and full-scale) error. The specified value of E_{TU} assumes zero E_{IL} (no leakage or zero real source impedance).

High accuracy Ambient Light Sensor SFH 5711
Application Note

Abstract

This application note describes the technical details as well as the operation of the ambient light sensor SFH 5711.

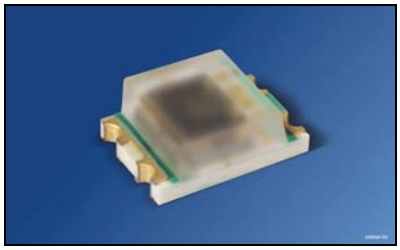


Figure 1: Ambient light sensor SFH 5711

Introduction

The ambient light sensor SFH 5711 is a photo detector with the following features:

- perfect V-λ characteristics[1]
- opto hybrid with logarithmic current output
- low temperature coefficient
- high accuracy over wide illumination range
- (2.8 x 2.2 x 1.1)mm SMT package
- automotive qualified

[1] V-λ characteristics describes the spectral sensitivity of the human eye.

The SFH 5711 consists of a photodiode which is used for the light detection and an IC with the following functions: amplification of the photodiode output signal, logarithmic converter and temperature correction.

This application note describes the technical details of the sensor. For more detailed information about ambient light sensing and its applications, please refer to the OSRAM OS general application note on ambient light sensors.

Spectral sensitivity of the SFH 5711

Ambient light sensors are used wherever the settings of a system[2] need to be adjusted to the ambient light conditions as perceived by humans. They are designed to detect brightness in the same way as human eyes do. To achieve this, the sensor needs to have a spectral sensitivity which is similar to that of human eyes. Figure 2 shows the spectral sensitivity of a standard silicon (Si) photo sensor, the SFH 5711 and the human eye (V-λ curve).

[2] Such as backlighting of mobile devices or instrument panels in cars

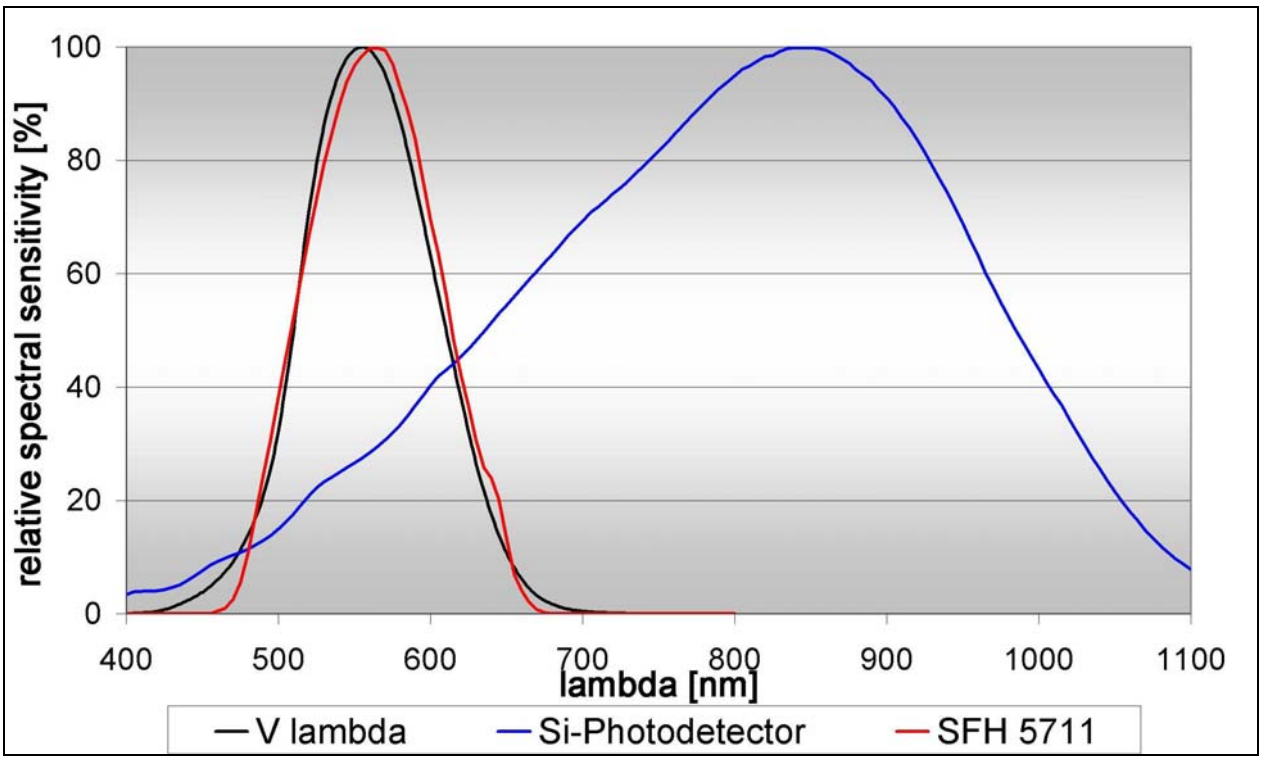


Figure 2: Relative spectral sensitivity of a standard Si-detector and the SFH 5711 compared to the human eye (V-λ).

From figure 2 one can see that a standard Si-detector has its maximum sensitivity in the IR range, which is invisible to human eyes. Lamps however do emit light in this "invisible range", which then leads a standard Si detector to "see" high brightness, whereas in fact it is not bright to human eyes. This match with the human eye characteristics is the most important parameter for the performance of an ambient light sensor.

The effect can be seen in Figure 3**.** It shows the detector signals for different lamp types at the same brightness level. To the human eye, all these light sources appear equally bright.

All signals in Figure 3 are normalized to standard light A (2856K), which is a general point of reference for ambient light sensors. Figure 3 illustrates the brightness measurement deviations of the different detectors compared to the human eye. A light bulb, for instance emits a high portion of IR light, which is fully detected by the standard Si-detector, but not seen by the human eye. Fluorescent lamps, on the other hand, do not emit much IR light. Hence the signals yielded by the standard Si-detectors are much higher for light bulbs than they are for fluorescent lamps, even though both lamps appear equally bright to the human eye. The deviation of the brightness measurement for the different light sources can directly be derived from figure 3. Compared to the human eye, the standard Si-detector signal is 3% too high in case of a light bulb and over 90% too low for a fluorescent lamp. The respective values for the SFH 5711 are ~1% only. When designing an ambient light sensor application, all possible light sources have to be taken into account.

MCP9700/9700A
MCP9701/9701A

Low-Power Linear Active Thermistor™ ICs

Features

- Tiny Analog Temperature Sensor
- Available Packages:
  - SC70-5, SOT-23-5, TO-92-3
- Wide Temperature Measurement Range:
  - -40°C to +125°C (Extended Temperature)
  - -40°C to +150°C (High Temperature) **(MCP9700/9700A)**
- Accuracy:
  - ±2°C (max.), 0°C to +70°C (**MCP9700A/9701A)**
  - ±4°C (max.), 0°C to +70°C (**MCP9700/9701**)
- Optimized for Analog-to-Digital Converters (ADCs):
  - 10.0 mV/°C (typical) **MCP9700/9700A**
  - 19.5 mV/°C (typical) **MCP9701/9701A**
- Wide Operating Voltage Range:
  - V_{DD} = 2.3V to 5.5V **MCP9700/9700A**
  - V_{DD} = 3.1V to 5.5V **MCP9701/9701A**
- Low Operating Current: 6 µA (typical)
- Optimized to Drive Large Capacitive Loads

Typical Applications

- Hard Disk Drives and Other PC Peripherals
- Entertainment Systems
- Home Appliance
- Office Equipment
- Battery Packs and Portable Equipment
- General Purpose Temperature Monitoring

Description

The MCP9700/9700A and MCP9701/9701A family of Linear Active Thermistor™ Intergrated Circuit (IC) is an analog temperature sensor that converts temperature to analog voltage. It's a low-cost, low-power sensor with an accuracy of ±2°C from 0°C to +70°C (MCP9700A/9701A) ±4°C from 0°C to +70°C (MCP9700/9701) while consuming 6 µA (typical) of operating current.

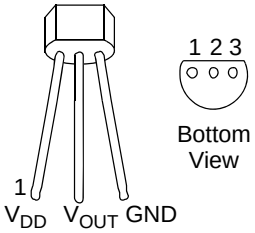
Unlike resistive sensors (such as thermistors), the Linear Active Thermistor IC does not require an additional signal-conditioning circuit. Therefore, the biasing circuit development overhead for thermistor solutions can be avoided by implementing this low-cost device. The voltage output pin (V_{OUT}) can be directly connected to the ADC input of a microcontroller. The MCP9700/9700A and MCP9701/9701A temperature coefficients are scaled to provide a 1°C/bit resolution for an 8-bit ADC with a reference voltage of 2.5V and 5V, respectively.

The MCP9700/9700A and MCP9701/9701A provide a low-cost solution for applications that require measurement of a relative change of temperature. When measuring relative change in temperature from +25°C, an accuracy of ±1°C (typical) can be realized from 0°C to +70°C. This accuracy can also be achieved by applying system calibration at +25°C.

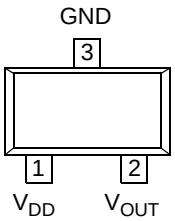
In addition, this family is immune to the effects of parasitic capacitance and can drive large capacitive loads. This provides Printed Circuit Board (PCB) layout design flexibility by enabling the device to be remotely located from the microcontroller. Adding some capacitance at the output also helps the output transient response by reducing overshoots or undershoots. However, capacitive load is not required for sensor output stability.

Package Type

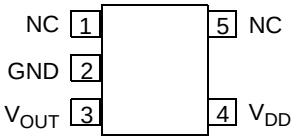
3-Pin TO-92
MCP9700/9701
Only



3-Pin SOT-23
MCP9700/9700A
MCP9701/9701A



5-Pin SC70
MCP9700/9700A
MCP9701/9701A



4.0 APPLICATIONS INFORMATION

The Linear Active Thermistor™ IC uses an internal diode to measure temperature. The diode electrical characteristics have a temperature coefficient that provides a change in voltage based on the relative ambient temperature from -40°C to 150°C. The change in voltage is scaled to a temperature coefficient of 10.0 mV/°C (typical) for the MCP9700/9700A and 19.5 mV/°C (typical) for the MCP9701/9701A. The output voltage at 0°C is also scaled to 500 mV (typical) and 400 mV (typical) for the MCP9700/9700A and MCP9701/9701A, respectively. This linear scale is described in the first-order transfer function shown in Equation 4-1 and Figure 2-16.

EQUATION 4-1: SENSOR TRANSFER FUNCTION

$$V_{OUT} = T_C \bullet T_A + V_{0°C}$$

Where:

T_A = Ambient Temperature

V_{OUT} = Sensor Output Voltage

$V_{0°C}$ = Sensor Output Voltage at 0°C (See DC Electrical Characteristics table)

T_C = Temperature Coefficient (See DC Electrical Characteristics table)

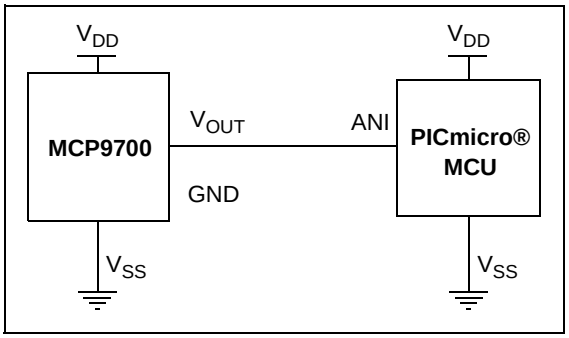


FIGURE 4-1: *Typical Application Circuit.*

4.1 Improving Accuracy

The MCP9700/9700A and MCP9701/9701A accuracy can be improved by performing a system calibration at a specific temperature. For example, calibrating the system at +25°C ambient improves the measurement accuracy to a ±0.5°C (typical) from 0°C to +70°C, as shown in Figure 4-2. Therefore, when measuring relative temperature change, this family measures temperature with higher accuracy.

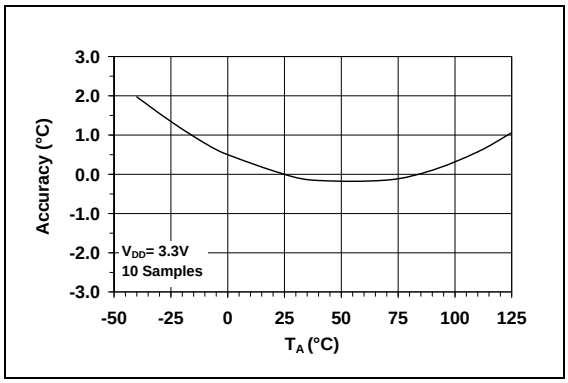


FIGURE 4-2: *Relative Accuracy to +25°C vs. Temperature.*

The change in accuracy from the calibration temperature is due to the output non-linearity from the first-order equation, as specified in Equation 4-2. The accuracy can be further improved by compensating for the output non-linearity.

For higher accuracy using a sensor compensation technique, refer to AN1001 *"IC Temperature Sensor Accuracy Compensation with a PICmicro® Microcontroller"* (DS01001). The application note shows that if the MCP9700 is compensated in addition to room temperature calibration, the sensor accuracy can be improved to ±0.5°C (typical) accuracy over the operating temperature (Figure 4-3).

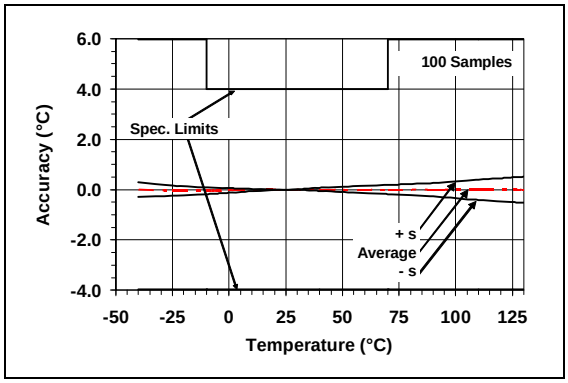


FIGURE 4-3: *MCP9700/9700A Calibrated Sensor Accuracy.*

The compensation technique provides a linear temperature reading. A firmware look-up table can be generated to compensate for the sensor error.

Anexo D

Manual de Usuario

Manual de Usuario

Índice general

1. Introducción

En este documento se va a explicar el manejo de la aplicación que presentamos. Se trata de una interfaz para el control de una vivienda equipada con servicio de domotización. Para acceder a la aplicación, solo es necesario un cliente *http* (como por ejemplo *Firefox*, *Chrome*, *Opera*, *Safari*, ...) y acceder al servidor escribiendo en el campo de dirección *http://<ip_del_servidor>/weberry/DomoWeb/* donde la ip del servidor es la asignada en la red local en la que se encuentra (como por ejemplo 192.168.1.137). A partir de aquí es donde se va a explicar cada uno de los pasos a seguir para utilizar el servicio *DomoWeb*.

2. Dos Aplicaciones para dos Tipos de Dispositivo

Esta aplicación puede manejarse desde un ordenador corriente o desde un dispositivo móvil, como puede ser un smartphone o una tablet. Como son dos tipos de dispositivo distintos, la aplicación varía según el que se utilice para acceder a ella. Esto es debido a que las maneras de interactuar con cada uno de estos dispositivos no es la misma, además del tamaño de las pantallas de las que disponen. Aún así, las dos versiones son muy parecidas y sencillas de manejar.

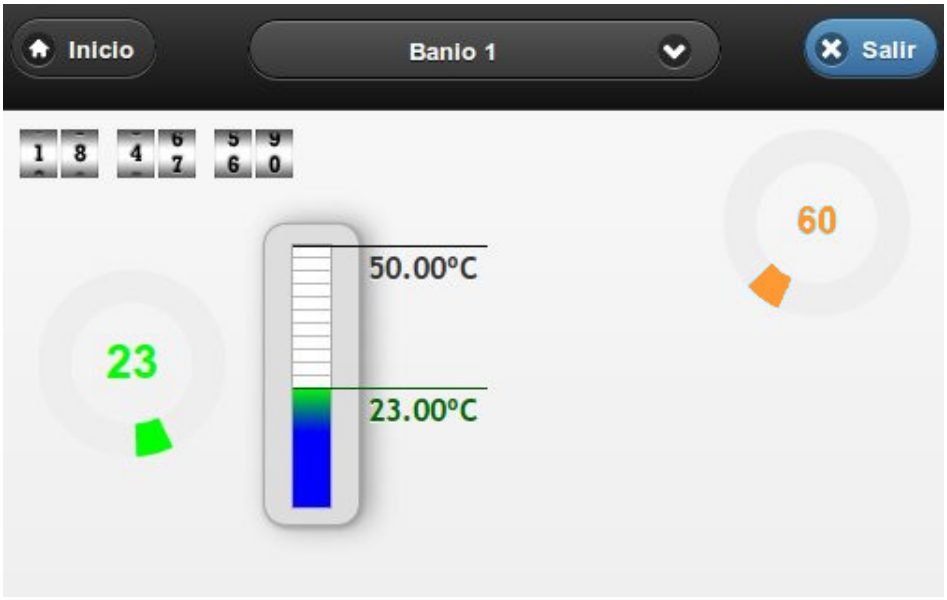


Figura 1: Interfaz para Dispositivos Móviles

Figura 2: Interfaz para Ordenadores Corrientes

3. Acceso a la Aplicación

En las dos aplicaciones el acceso es el mismo. Cuando se accede a esta, aparece un formulario para identificarse como usuario autorizado del control del servicio. Los datos que deben introducir son un nombre de usuario y una contraseña asociada. Una vez rellenado el formulario, se debe pulsar el botón *Acceder* para visualizar los paneles de control.

Figura 3: Sección de Login para Acceder a la Aplicación

4. Página Principal de la Aplicación

Una vez validado el usuario, se puede ver la aplicación. Esta se compone de 3 partes diferenciadas sea cual sea el dispositivo utilizado. Este formato es común a todas las secciones:

1. Indicador de habitacióin o sección.
2. Menú.
3. Panel de control con indicadores y controles.

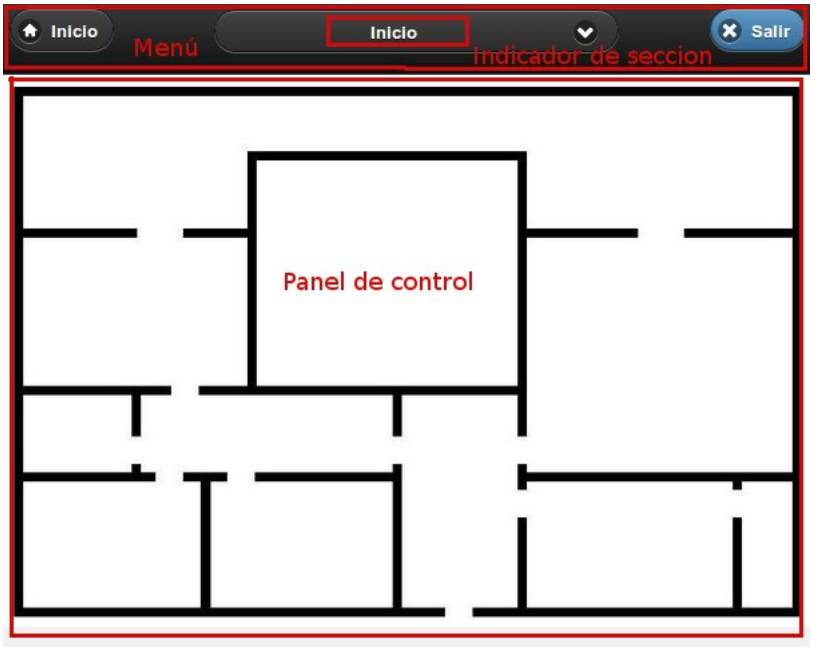


Figura 4: Partes de la Interfaz para Dispositivos Móviles

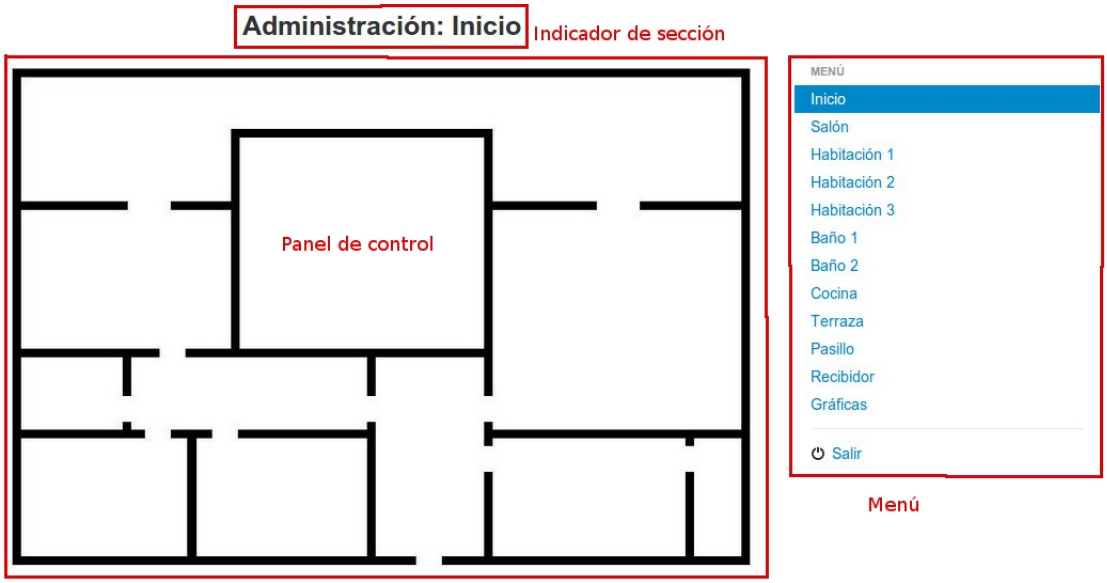


Figura 5: Partes de la Interfaz para Ordenadores Corrientes

Como se puede ver, lo primero que se ve al acceder a la aplicación en un plano de la vivienda donde se puede seleccionar ua habitación pulsando o haciendo click en ella. Además de esta posibilidad, se puede seleccionar la zona a partir del menú. En el caso de la aplicación móvil, este aparecerá si se pulsa el indicador de la habitación o sección. Se deplegará una lista de opciones al reaizar esta acción.

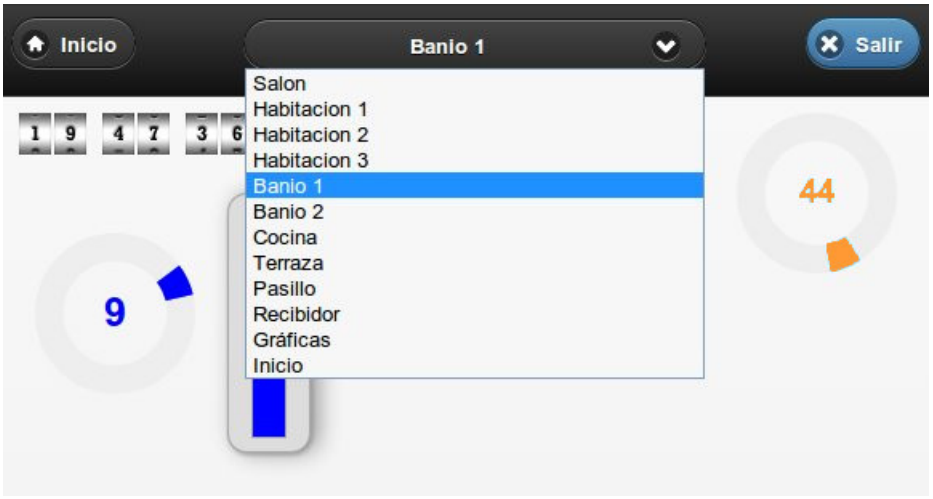


Figura 6: Despliegue del Menú en la Aplicación Móvil

5. Control de Habitaciones

Cada sección que corresponde a una habitación dispone de los controles necesarios para manejar los servicios que se ofrecen. Un mando que maneja un servicio determinado es igual en todas las habitaciones que dispongan de dicho servicio.

5.1. Termómetro y Control de Temperatura

Este componente se ocupa de informar de la temperatura actual de la zona seleccionada. Mide hasta 50ºC y se actualiza cada minuto. Junto al termómetro, hay un control de temperatura deseada. Para fijarla simplemente se debe modificar el valor deslizando el cursor. Para activar el termostato y así conseguir que el sistema mantenga la temperratura deseada se debe pulsar el control en la zona de indicación numérica. El color gris en el control indica que el termostato está apagado, un color distinto (azul, verde o rojo) indica que este está encendido.

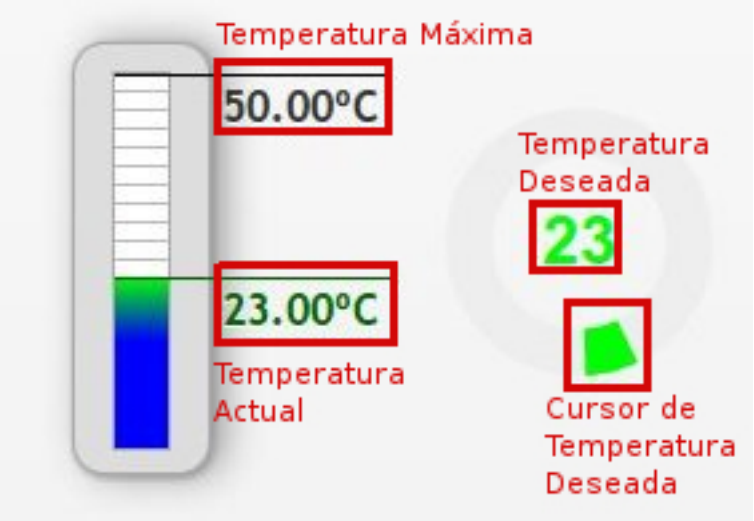


Figura 7: Termómetro y Control de Temperatura

5.2. Control de Intensidad Lumínica

Con este componente es posible controlar la intensidad de la luz de una zona determinada. Al igual que con el control de temperatura deseada, simplemente se debe deslizar el cursor a la intensidad deseada. El cambio se realiza en tiempo real. La intensidad se indica en porcentaje con respecto a la intensidad

máxima. Para encender o apagar la luz se debe pulsar el controlador en la zona de indicación numérica. Un color gris indica que la luz está apagada, un color naranja indica que esta está encendida y por tanto se puede regular su intensidad.

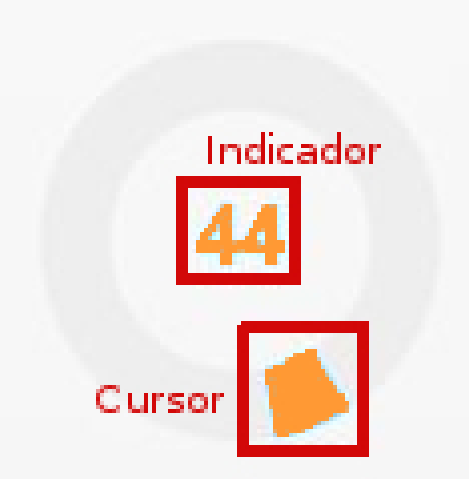


Figura 8: Control de Intensidad Lumínica

5.3. Reloj

Este componente únicamente indica la hora actual. Para ello toma la hora del sistema que se está utilizando. Es decir que si se desea cambiarla, será necesario modificarla en el dispositivo que se esté usando.

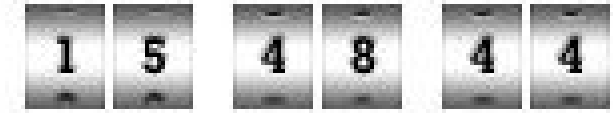


Figura 9: Reloj de la Aplicación

6. Representación de Datos Monitorizados

Este sistema, además de permitir controlar las variables de una vivienda, monitoriza ciertos estados ambientales y los almacena en el servidor para poder visualizarlos. Para ello, se debe acceder a la sección *Gráficas* de la aplicación normal. Este servicio no está disponible para dispositivos móviles, por lo tanto, si se desea visualizar estos datos, es necesario acceder con un ordenador normal.

Figura 10: Sección de Gráficas de la Aplicación

6.1. Gráfica

La gráfica es una curva que relaciona el valor de la variable seleccionada (eje de las ordenadas) a lo largo del tiempo (eje de abcisas). Cuando se accede a la sección de gráficas, esta muestra el valor de la intensidad lumínica en el día actual.

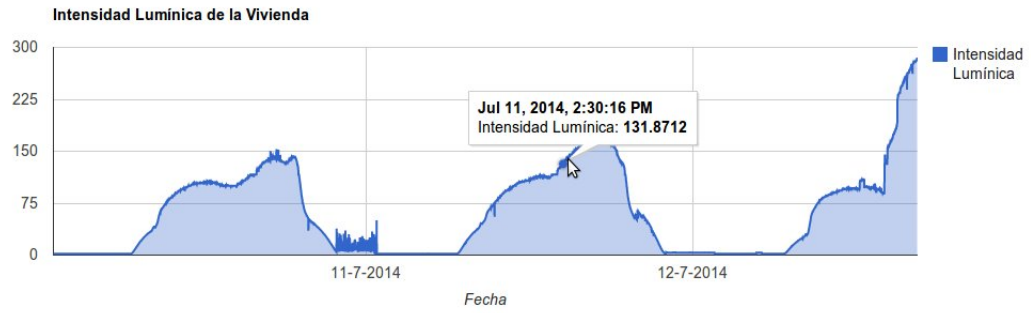


Figura 11: Gráfica Representando la Intensidad Lumínica a los Largo de Tres Días

Cada inicio de nuevo día, se indica la fecha en el eje X y si se desa ver el detalle en un momento determinado, solo se debe colocar el puntero encima de la zona de interés de la curva.

6.2. Formulario de Petición de Datos

Este formulario permite visualizar datos determinados al gusto del usuario. Lo campos son los siguientes:

1. *Selector de Zona*: Se trata de un menú desplegable que contiene las zonas monitorizadas de la vivienda. Es obligatorio fijar un valor para poder visualizar una petición de datos.

2. *Selector de Fechas*: Dispone de dos campos que representan la fecha inicial y la fecha final del periodo que se desea visualizar. Hay tres modalidades de franja posibles:

    - *Desde una Fecha Hasta Otra Fecha*: Para esta modalidad se deben rellenar los dos campo de fecha con los valores deseados como fecha inicial y

fecha final.

- *Desde una Fecha Hasta la Actualidad*: Para esta modalidad se debe rellenar el campo de fecha inicial con el valor deseado de inicio y dejar el de fecha final sin valor.

- *Un Día Determinado*: Para esta modalidad se debe rellena únicamente el campo de fecha final con el valor del día que se desea visualizar.

3. *Selector de Variable*: Es un menú con valores que se pueden seleccionar. Sólo se puede fijar uno a la vez de entre las variables que se están monitorizando en una zona.

4. *Botón de Petición*: El botón "Visualizar Datos.es el que se debe pulsar para obtener los datos deseados.

6.3. Panel de Opciones

En esta parte podemos encontrar la solución a un mal funcionamiento del sistema. Una manera sencilla de solventar este problema es reiniciando todo el sistema. Para ello la sección de opciones ofrece un sencillo botón que solo con pulsarlo reinicia tanto el servidor como todos los controladores.

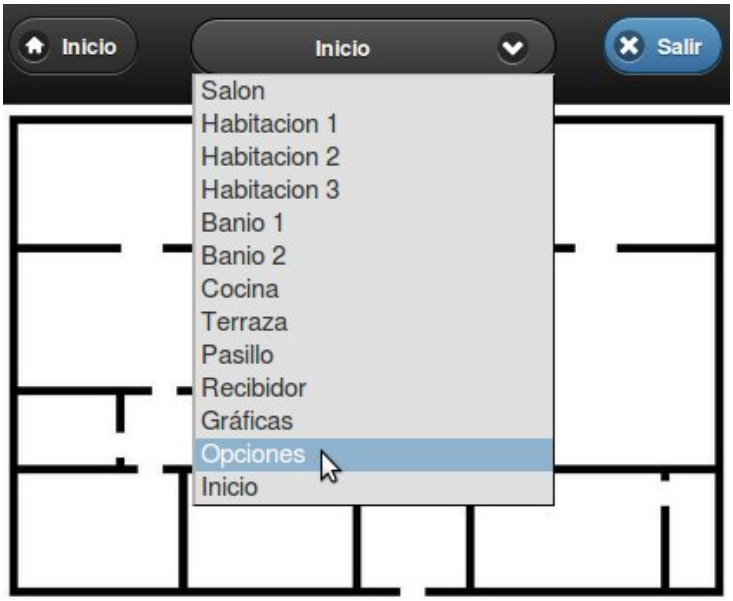


Figura 12: Posición de las Opciones en el Menú

Figura 13: Sección de Opciones