2010

Realización de sistema domótico con microcontroladores de bajo coste (AVR) y módulos RF verificando el estándar 802.15.4

(II)

ÍNDICE GENERAL

Escuela Universitaria Ingeniería Técnica Industrial ZARAGOZA

Francisco Pérez Pellicena

HOJA DE IDENTIFICACIÓN DE DATOS

DATOS DEL PROYECTO *REALIZACIÓN DE SISTEMA DOMÓTICO CON MICROCONTROLADORES DE BAJO COSTE (AVR) Y MÓDULOS RF, VERIFICANDO EL ESTÁNDAR 802.15.4*
DIRECTORA DEL PROYECTO *ARÁNZAZU OTÍN ACÍN* *RAZÓN SOCIAL: C\MARÍA DE LUNA 1, EDIFICIO ADA BYRON, 50015 ZARAGOZA*
AUTOR DEL PROYECTO *FRANCISCO PÉREZ PELLICENA* *C\COIMBRA 2 2ºC 50008 ZARAGOZA*
FECHA Y FIRMA *EN ZARAGOZA A 31 DE AGOSTO DE 2010*

MEMORIA

ANEXOS

PLANOS

PLIEGO.DE.CONDICIONES

MEDICIONES

PRESUPUESTO

MANUAL DE USUARIO

2010

Realización de sistema domótico con microcontroladores de bajo coste (AVR) y módulos RF verificando el estándar 802.15.4

(II)

MEMORIA

Escuela Universitaria Ingeniería Técnica Industrial ZARAGOZA

Directora: Aránzazu Otín Acín

Autor: Francisco Pérez Pellicena

HOJA DE IDENTIFICACIÓN DE DATOS

DATOS DEL PROYECTO
REALIZACIÓN DE SISTEMA DOMÓTICO CON MICROCONTROLADORES DE BAJO COSTE (AVR) Y MÓDULOS RF, VERIFICANDO EL ESTÁNDAR 802.15.4
DIRECTORA DEL PROYECTO
Aránzazu Otín Acín *Razón social: C\María de Luna 1, Edificio Ada Byron, 50015 Zaragoza*
AUTOR DEL PROYECTO
Francisco Pérez Pellicena *C\Coimbra 2 2ºC 50008 Zaragoza*
FECHA Y FIRMA
En Zaragoza a 31 de Agosto de 2010

Agradecimientos

En primer lugar quiero agradecer a mi familia el apoyo que me ha prestado y la comprensión que han tenido, no sólo estos últimos meses sino desde el primer momento que vi la luz en este mundo. Sin ellos nada habría sido posible. Especialmente a mi madre, porque el presente proyecto me ha robado mucho tiempo de estar a su lado.

En segundo lugar, a todas las personas que de una forma u otra, han contribuido a mi desarrollo personal y profesional en la Escuela Universitaria de Ingeniería Técnica Industrial de Zaragoza, porque su labor es de un valor incalculable.

Mi especial agradecimiento a Aránzazu Otín Acín como directora de este proyecto, por sus consejos y su paciencia.

A mi compañero de proyecto David Pilarcés Collado por su esfuerzo para que esta empresa haya salido adelante.

A Diego Antolín Cañada por su apoyo personal y profesional.

ÍNDICE

1. - OBJETO

El objeto del presente proyecto es la realización de un prototipo funcional sencillo, que permita la medición de ciertos parámetros físicos haciendo uso de los pertinentes transductores o sensores.

Las mediciones realizadas de forma aislada, se propagarán a través de una red inalámbrica hasta llegar a un sistema central, el cual procesará las medidas y las almacenará para su posterior tratamiento.

La principal idea a aplicar es el hacer accesible esa información, siempre convenientemente tratada, a internet, de tal forma que pueda ser consultada desde cualquier terminal con acceso y en cualquier momento.

Además de realizar mediciones se ofrece la posibilidad de realizar ciertas tareas de control sencillas, como activar y desactivar dispositivos, regular sistemas de iluminación...etc. , y aunque el prototipo no incluirá todas las opciones posibles, cabe destacar que las tareas de control, también podrán ser llevadas a cabo desde cualquier terminal con acceso a internet, lo que dotará de una alta disponibilidad al conjunto.

Las aplicaciones son inmensas, desde la realización de sistemas de monitorización ambiental en ciudades o espacios naturales, hasta sistemas complejos con un gran número de elementos de red y con complejas opciones de control, como podría ser la domotización de un edificio.

En este proyecto se va a intentar que la generalidad sea una cuestión primordial, bajo la idea de que el sistema pueda adaptarse tanto a pequeñas como a grandes implementaciones. Esto solo se puede conseguir abstrayendo los elementos comunes a los posibles problemas que queremos resolver e ideando soluciones eficientes a dichos problemas de base.

No obstante, se va a pretender desarrollar un prototipo de un sistema reducido, como podría ser una solución para un sistema domótico, más o menos general.

Este proyecto se realiza en conjunto con David Pilarcés Collado, quien se encarga principalmente del desarrollo del hardware.

El trabajo elaborado por el presente proyectante está centrado en la programación tanto del software de los sistemas hardware (motes) que interactúan con sensores y actuadores, como en el software de aplicación, sin dejar de lado las implicaciones que el diseño del hardware tienen sobre el software y participando activamente en el propio diseño del hardware.

Los objetivos concretos de la parte que se presenta son los siguientes:

- Programación de los motes de la red inalámbrica que permita la gestión de auto asociación a alto nivel y el envío de información estructurada.
- Gestión de la información que circula por la red, almacenando la que es relevante para el funcionamiento y para los usuarios.
- Creación de una infraestructura de servidor web, para permitir el acceso a la información almacenada.
- En general, establecer las bases para un sistema sensorial que permita la adaptación al cambio con el menor coste.

2.-ALCANCE

Una vez marcados los objetivos tal y como se describen en el apartado anterior, y dado el gran abanico de posibilidades, se va a centrar el proyecto en el desarrollo de un prototipo funcional de lo que podría ser un sistema de medición y control de parámetros que se pueden encontrar en una vivienda habitual.

2.1.- HARDWARE

Se van a contemplar como parámetros de medición los siguientes:

1. Temperatura ambiental
2. Iluminación
3. Humedad relativa
4. Presencia

Para ello se dispondrán los dispositivos pertinentes que más en profundidad se detallan en apartados siguientes.

Como se ha mencionado, también aparece el término control, y aunque no se trata de un control estricto, si que va a ser suficiente para el propósito de este proyecto. En este sentido, se va a realizar un control de iluminación de una bombilla típica mediante la realización hardware de un dimmer, controlado por software.

Los sistemas autónomos o motes de medición y control, están formados por un núcleo microcontrolador que gestiona la adquisición de medidas y las acciones de control, así como el sistema de comunicación.

En cuanto al sistema de comunicación, como ya se ha mencionado se va a disponer un transceptor inalámbrico de coste moderado y consumo reducido, que permita la transmisión y recepción de información desde cada mote a un sistema central, donde se van a almacenar y tratar toda la información que llegue, formando una red con una determinada topología.

Uno de los principales objetivos es la flexibilidad del sistema, de tal forma que la red formada por los motes sea capaz de gestionar la configuración de los motes, permitiendo añadir nuevos en lo que podría ser un sistema plug&play de alto nivel.

2.2- SOFTWARE

En cuanto al software, cabe diferenciar tres actuaciones. Una primera centrada en el desarrollo de las aplicaciones que van instaladas en los motes, donde se trabaja a bajo nivel con el principal elemento, el microcontrolador, usando el lenguaje de programación C.

Los principales objetivos son el desarrollo de un software que sea eficiente energéticamente y que permita la flexibilidad de autogestión de la red. En cuanto al concepto de “software eficiente energéticamente” cabe decir que está basado en el aprovechamiento de los recursos que ofrece el microcontrolador en términos de bajo consumo, como son los modos de sueño y el control de activación de bloques independientes. Ambos recursos se van a tratar de explotar para minimizar el consumo.

Una segunda actuación del desarrollo de software está pensada como aplicación principal que gestiona el sistema autónomo, es decir la red de motes, dentro de lo que sería una instalación física. Esta aplicación debe funcionar de forma autónoma e ininterrumpida, independientemente de que exista o no una conexión a internet, por lo que las dependencias deben minimizarse. Esto puede entrar en contraposición al objetivo de accesibilidad, pero nada más lejos de la realidad, lo que interesa es que el sistema de motes funcione bajo cualquier circunstancia.

La gestión de la red implica la capacidad de autogestión de la misma a alto nivel, con la posibilidad de añadir motes en caliente y sin previa configuración en el sistema informático, la detección de tramas específicas de datos, el envío de tramas de control a los motes que realizan acciones y la coordinación de toda esta información con el sistema central, donde se almacenará, lo que implica que debe ser capaz de acceder remotamente a través de internet a un sistema de base de datos externo.

De igual forma que es capaz de gestionar las tramas que circulan por la red, debe permitir la ejecución de acciones de control, como puede ser regular la iluminación de una bombilla, desde un terminal externo conectado a internet. Para lograrlo, hay que permitir el acceso al sistema instalado de forma remota, así un cliente remoto podrá realizar las acciones de control, por lo que la instalación puede considerarse conceptualmente como un servidor remoto, que ofrece servicios de control de dispositivos, entre otros.

La tercera actuación está centrada en el sistema central donde se recibe la información, se almacena y se pone a disposición de los usuarios. Se va a enfatizar en el concepto de servicio, de tal forma que el conjunto del proyecto es en sí mismo un servicio, el cual es contratado por determinados clientes. Esto afecta directamente al desarrollo del sistema central, donde se van a considerar diferentes perfiles de acceso a la aplicación, permitiendo un perfil de administración para tareas restringidas y al menos un perfil de cliente, con propósitos de monitorización y control de la instalación. Básicamente esto se consigue empleando un servidor de aplicaciones en el cual, se ejecuta una aplicación que es capaz de gestionar la información de cada perfil adecuadamente, así como de delimitar el acceso a las zonas restringidas.

Evidentemente es necesario un mecanismo de persistencia para almacenar la información que es relevante tanto para la gestión de usuarios y perfiles como para la configuración de la red, asegurando el correcto funcionamiento de la misma.

3.-ANTECEDENTES

3.1.- INTRODUCCIÓN

Etimológicamente, domótica proviene de la unión de *domus* (del latín: *casa*) y robota (del checo: *esclavo*) y se define como "*la integración de la tecnología en el diseño inteligente de un recinto*"

El concepto Domótica, se asocia con un conjunto de sistemas capaces de automatizar una vivienda, aportando así servicios de: gestión energética, seguridad, bienestar y comunicación; e integrados por medio de redes interiores\exteriores de comunicación, cableadas o inalámbricas (wireless).

Estos servicios se crean a partir de las nuevas tecnologías de la información (TIC), las cuales han sido aplicadas y utilizadas en las viviendas siempre, contribuyendo a cambiar las relaciones familiares y las estructuras de las ciudades. La penetración e inserción de las TIC en la sociedad y el territorio tiene sus raíces en el reciente proceso de convergencia tecnológica, facilitado en buena medida por la estandarización de una de las unidades básicas con que hoy se mide la información y su flujo: los bits.

Las TIC son una especie de disciplina emergente en la que conjuntamente están implicados arquitectos, ingenieros eléctricos, electrónicos y civiles, programadores de sistemas y diseñadores.

Esta forma de trabajar implica varias arquitecturas, principalmente existen:

- *Arquitectura Centralizada*: en donde un controlador centralizado recibe información de múltiples sensores y, una vez procesada, genera las órdenes oportunas para los actuadores.
- *Arquitectura Distribuida*: en este caso, no existe la figura del controlador centralizado, sino que toda la inteligencia del sistema está distribuida por todos los módulos sean sensores o actuadores. Suele ser típico de los sistemas de cableado en bus.
- *Arquitectura Descentralizada*: dispone de varios pequeños dispositivos capaces de adquirir y procesar la información de múltiples sensores y transmitiéndolos al resto de dispositivos distribuidos por la vivienda. (Se conoce como Arquitectura Mixta).

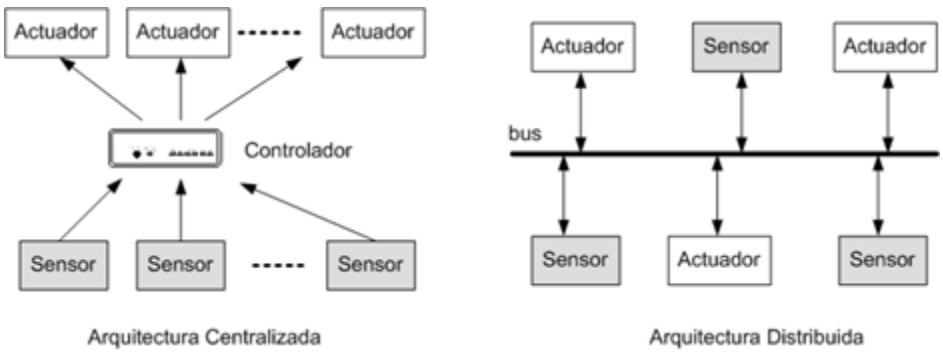


FIGURA 1: ARQUITECTURAS PARA SISTÉMAS DOMÓTICOS

En cualquier caso, saber qué es un:

- *Actuador*: es un dispositivo de salida de datos, capaz de recibir una orden del controlador y realizar una acción (on\off, subida\bajada, apertura\cierre, etc.)
- *Controlador*: es un dispositivo central que gestiona el sistema, habitualmente un microprocesador o microcontrolador dependiendo del diseño utilizado; del que dependen los interfaces de usuario: teclado, monitor, etc.
- *Sensor*: es un dispositivo que genera un evento, siendo éste procesado por el controlador. Los sensores son de: temperatura, humedad, viento, humo, escape de agua o gas, etc.

Para llevar a cabo este tipo de arquitecturas es necesario conocer el tipo de red a utilizar en la instalación, según la infraestructura necesaria, existen tres tipos:

- *Cableada*: todos los sensores y actuadores, están cableados a la central, la cual es el controlador principal de todo el sistema. Esta tiene normalmente una batería de respaldo, para en caso de fallo del suministro eléctrico, poder alimentar a todos sus sensores y actuadores y así seguir funcionando normalmente durante unas horas.
- *Inalámbrica*: en este caso usan sensores inalámbricos alimentados por pilas o baterías y transmiten vía radio la información de los eventos a la central, la cual está alimentada por la red eléctrica y tiene sus baterías de respaldo.
- *Mixta*: combinan el cableado con el inalámbrico. Sus principales prestaciones o funciones son: una mayor seguridad, la automatización y el telecontrol de los electrodomésticos y otros dispositivos, el acceso a los nuevos sistemas de telecomunicaciones y la superior disponibilidad de ocio y entretenimiento en casa.

En todos los casos existe una fuerte tendencia a hacer más cómoda y versátil la estancia en el lugar de vivienda, al igual que se espera tener una mayor capacidad de gestión y monitoreo, tanto de los electrodomésticos como de los servicios públicos, donde se destacan aspectos como el consumo, el gasto y el ahorro energético.

Algunos de los principales protocolos o lenguajes informáticos de comunicación entre el usuario y los artefactos domóticos, y de ellos entre sí, que están disponibles son: X10, CEBus, Bacnet, TCP/IP, Konnex, Lonworks, SCP, HAVi, Jini, UpnP, HAPI, etc...

Con todo lo comentado hasta ahora, lo que se pretende a la hora de llevar a cabo una instalación domótica es:

- Incrementar el confort, produciendo bienestar y comodidades que anteriormente no existían o simplemente reduciendo sus tiempos, esfuerzos o costes.
- Automatización del control de luces, persianas, ventanas, cortinas, enchufes, riego, climatización…
- Optimización en la gestión de consumos: energía eléctrica, gas, recursos hídricos. Uso de energías renovables: Energía solar, Energía geotérmica, Energía eólica.
- Ubicuidad en el control tanto externo como interno, control remoto desde Internet, PC, mandos inalámbricos (p.ej. PDA con WiFi).
- Facilidad de uso, entorno gráfico intuitivo.
- Gestión del ocio.
- Alarmas. Vigilancia anti-incendios. Temperatura. Detección de fugas de gas o agua. Y su correspondiente gestión: corte de suministros, posibilidad de visualización remota de la vivienda.

- Control de accesos. Control de servicios para emular la presencia de gente durante las ausencias prolongadas.

Resulta imposible precisar una fecha concreta para el nacimiento de la domótica, ya que no se trata de un hecho puntual, sino de todo un proceso evolutivo que comenzó con las redes de control de los edificios inteligentes y se ha ido adaptando a las necesidades propias de la vivienda.

Si hemos de destacar una fecha en concreto, esta sería el año 1.978 con la salida al mercado del sistema X-10, considerado el primer sistema domótico propiamente dicho.

Actualmente, los nuevos edificios tienen que cumplir con el protocolo de Kyoto, el cual regula la utilización de materiales contaminantes y por tanto perjudiciales para el medio ambiente, esto supone un diseño especial en la elección de componentes y materiales a emplear y en las diferentes técnicas a emplear para su desarrollo y posterior instalación.

3.2.-EVOLUCIÓN HISTÓRICA

La domótica, como ya hemos citado anteriormente, es un proceso evolutivo en el cual se han unido diversas disciplinas de la arquitectura, ingeniería...

Dicho proceso evolutivo comenzó con los primeros reguladores allá por la antigua Grecia, entonces eran reguladores que funcionaban con agua o elementos mecánicos como: “El reloj de agua” de Ktesibios, “La alarma” o “Clepsydra” de Platón, “La lámpara” de Philon, etc. Con el tiempo aparecieron los sistemas realimentados, dejando constancia en la biblioteca de Alejandría con los textos: “Autómata” y “Pneumática”. De los sistemas realimentados cabe destacar el “Dispensador automático de vino”, “El odómetro”, “La aelopila” o “Primera Turbina” ambos de Herón.

Todos estos sistemas son hasta entonces para aplicaciones lúdicas, y se van mejorando hasta llegar a la Edad Media, época en la cual toman un carácter adaptado a los problemas cotidianos, encontramos por ejemplo: “ El sistema de orientación de las aspas de los molinos”.

Hacia el siglo XVII nacen los primeros sistemas de regulación de temperatura, aplicándose a los hornos y a la incubadora de Drebel. A partir de aquí se da una época conocida como la Revolución Industrial, su principal aportación fue el de la primera regulación automática, con la máquina de vapor de Watt, en la que un dispositivo gobernaba la velocidad de la máquina actuando sobre una válvula que regulaba el paso del vapor.

A lo largo del siglo XIX se descubren: las ondas electromagnéticas por parte de Maxwell basándose en las teorías y estudios de Faraday, ondas de radio por parte de Hertz probando así la teoría de ondas electromagnéticas de Maxwell, la energía de la luz... También aparecen muchos nombres, Boole (cálculo binario), Babbage (la primera persona en concebir la idea de lo que hoy llamaríamos un ordenador). A final del siglo se patenta lo que hoy conocemos como televisión por parte de Nipkow, mejorándose posteriormente con los tubos de rayos catódicos de Thompson, por otro lado Marconi inventa la radio o por lo menos se le atribuye a él ya que se basa en los estudios de Oersted, Faraday, Hertz, Tesla, Edison, Popov…

Más adelante a principios del siglo XX, Siemens y Halske fabrican el primer tubo de vacío utilizado como amplificador de baja frecuencia en los aparatos de radio. Lorentz se basa en la energía de la luz para enun-

ciar el principio de la relatividad, normalmente atribuida en solitario a Einstein. Carrier asentó las bases del aire acondicionado.

En los años 30 aparecen los primeros termostatos con contactos de mercurio.

En 1.936 se prohíben, por razones sanitarias, y aparecen los termostatos por contacto fijos.

A finales de esta década crece el interés por la calefacción con combustibles líquidos, por lo que se crea la necesidad de transformadores especiales de encendido y termostatos de calderas. En 1.936 el alemán Konrad Zuse fabricó la primera calculadora mecánica basada en el concepto del cálculo binario. En 1937 Shannon utiliza el Cálculo binario para crear una máquina: "Modelo K" la cual consiste en un computador construido con relés.

En 1.940 Landis & Gyr crea un departamento independiente para fomentar y controlar el desarrollo y la comercialización de los equipos de regulación para aplicaciones de calefacción y aire acondicionado. En 1.944 se construye el primer ordenador (MARK I), fruto de la colaboración entre la universidad de Harvard y la empresa IBM. Estaba construido únicamente con piezas electromecánicas y usaba el sistema decimal.

En 1.946 nacen los ordenadores de 1ª generación con el ENIAC. Construido con válvulas de vacío y relés. Bajo el concepto de arquitectura de Von Neumann, en el que también contribuyeron los científicos: Eckert y Mauchly.

Hay que destacar la importancia de la arquitectura de Von Neumann porque con ella están concebidos todas las máquinas, microcomputadores, microcontroladores de nuestros días

Los laboratorios Bell presentan el transistor en el año 1.948 pero no se comercializa hasta 1.952 por la fuerte competencia de los tubos de vacío. En 1.959 IBM, construye los primeros ordenadores transistorizados, también llamados de segunda generación. Texas Instruments desarrolla la tecnología de circuitos integrados, que disminuirá drásticamente el tamaño y precio de los equipos.

En 1.968 Intel crea el microprocesador, componente revolucionario, pudiéndose utilizar para las más diversas funciones de control.

Desde las primeras computadoras han nacido por lo tanto lo que hoy en día conocemos como ciencias de la información o Informática. Desarrollando lenguajes de programación y sistemas operativos para poder automatizar y controlar los computadores.

Internet aparece por primera vez en 1969 cuando tres universidades estadounidenses crean una red por medio de ARPANET, mejorándose en los años 80 por trabajadores del CERN, consiguiendo no sólo conectar los ordenadores si no compartiendo los archivos entre ellos.

Todas estas contribuciones en el campo de las matemáticas, física, química y demás ramas científicas son las que han permitido unir todos los elementos de los que podemos disponer hoy en día en nuestro hogar.

3.3.-EVOLUCIÓN EN LA DOMÓTICA

A principios de los setenta se implantan los sistemas de control de las instalaciones domóticas, los motivos del enorme éxito de estos sistemas fueron varios:

- Las posibilidades aportadas por el enorme avance de la electrónica, que continuamente aumentaba sus prestaciones, a un precio cada vez más bajo.
- La necesidad manifestada por proyectistas y usuarios de disponer de equipos más fiables y con menos costes de mantenimiento.
- La cada vez más compleja técnica de instalaciones obliga a la realización de secuencias de control más complicadas. Los edificios cada vez mayores y más sofisticados que los arquitectos iban construyendo, obligaban a centralizar y a tratar cada vez con mayor cantidad de información y más rápido, lo que se hacía imposible con técnicas que no fueran electrónicas.
- Las crisis energéticas de los años 70 obligó a un consumo de la energía lo más racional posible, lo que exigía una precisión en los equipos de control que sólo la electrónica podía dar.
- Cambios en la legislación de los países desarrollados, que obligan a un mayor control del gasto energético y protección medioambiental.
- En un principio sólo se hacían cosas básicas como la gestión integral de calefacción y aire acondicionado, que hasta entonces se hacía de forma aislada. Posteriormente se ha ido sofisticando continuamente hasta llegar a una integración total de la gestión.
- Tradicionalmente, desde los años 80, se dice que un edificio inteligente es aquel que descansa sobre los cuatro pilares siguientes:

AUTOMATIZACIÓN DE FUNCIONES:

Con lo que se pretende proporcionar la mayor autonomía funcional posible al edificio. Es decir, el edificio debe ser capaz de controlar, por sí mismo, todas las instalaciones que pueda incorporar para que se satisfagan las necesidades de los usuarios. Estas funciones son tales como seguridad, control de presencia, climatización, depuración de aire, abastecimiento de agua, ascensores, iluminación , sistemas anti-incendios, apertura automática de puertas, ventanas, toldos, persianas, etc.

AUTOMATIZACIÓN DE ACTIVIDADES:

Entendiéndose con ello que el edificio debería incorporar la infraestructura necesaria para dar soporte a la actividad que se vaya a desarrollar en el mismo. Así, debería incorporar todo tipo de tomas de alimentación, de transmisión de datos, audio, vídeo y de control, así como los dispositivos que sean necesarios para cualquier trabajador (fax, modem, etc...), teniendo en cuenta incluso el diseño ergonómico, desde su puesto físico de trabajo, hasta el sistema operativo de los sistemas de procesado que se vaya a utilizar. Todo ello unido e integrado por medio de redes locales y globales.

TELECOMUNICACIONES AVANZADAS:

Entendiendo como tales aquellas que permiten transmitir cualquier tipo de información multimedia (audio, vídeo, datos, señales de control, etc...) de la que se pueda hacer uso en el

edificio. Para conseguir esto hacen falta, como mínimo, una serie de sistemas: Cableado estructurado, control de servicios técnicos y seguridad, televisión en circuito cerrado, telefonía interior, intercomunicación, megafonía y otros servicios de valor añadido. Hay que hacer hincapié en el hecho de que el sistema de telecomunicaciones de un edificio inteligente deber ser integral (es decir, debe permitir que por el mismo terminal se acceda a cualquier punto tanto interior como exterior al propio edificio) y ampliable a exigencias futuras, sin necesidad de realizar ningún tipo de modificación en la estructura del cableado. Para conseguir todo esto son necesarias una homogenización del cableado del edificio en un solo sistema, flexibilidad máxima que posibilite añadir y modificar servicios sin cambiar el cableado y, finalmente, una estructuración y dimensionamiento del sistema para que se puedan aprovechar al máximo las ventajas que ofrezcan las futuras autopistas de la información.

FLEXIBILIDAD AL CAMBIO:

Éste es un aspecto fundamental en la "inteligencia" del edificio. Se trata de garantizar que el edificio sea capaz de satisfacer las necesidades de cualquier conjunto de usuarios diferentes que pueda albergar a lo largo de su vida útil.

Por ejemplo, cambiar una planta estructurada como una oficina abierta, tipo paisaje, a una serie de oficinas cerradas independientes, debe poderse hacer sin necesidad de modificar estas infraestructuras, ni de realizar ningún tipo de obras de albañilería.

Por supuesto, los puntos anteriores deben estar completamente integrados, formando parte de un ente común, en el cual se manifieste la "inteligencia" del edificio.

3.4.-LUZ

La luz (del latín *lux, lucis*) es una onda electromagnética capaz de ser percibida por el ojo humano y cuya frecuencia determina su color.

En términos generales, el espectro electromagnético abarca, según un orden creciente de frecuencia:

- las de radio
- las microondas
- los rayos infrarrojos
- la luz visible
- la radiación ultravioleta
- los rayos X
- los rayos gamma.

ESPECTRO VISIBLE

La luz visible forma parte de una estrecha franja que va desde longitudes de onda de 380 nm (violeta) hasta los 780 nm (rojo). Los colores del espectro se ordenan como en el arco iris, formando el llamado espectro visible.

Frecuencia y longitud de onda se relacionan por la expresión:

$$c = f\ \lambda$$

donde *c* es la velocidad de la luz en el vacío, frecuencia *f* ó *v*, y longitud de onda *λ*.

Existen dos tipos de objetos visibles: aquellos que por sí mismos emiten luz y los que la reflejan. El color de estos depende del espectro de la luz que incide y de la absorción del objeto, la cual determina qué ondas son reflejadas.

La luz blanca se produce cuando todas las longitudes de onda del espectro visible están presentes en proporciones e intensidades iguales.

Las ondas que tienen menor frecuencia que la luz (por ejemplo las de radio), tienen mayor longitud de onda, por eso rodean los objetos sin interaccionar con ellos, gracias a esto tenemos cobertura en el móvil aunque estemos dentro de casa. Las ondas de mayor frecuencia que la luz tienen una longitud de onda tan pequeña que atraviesan la materia, por ejemplo los rayos X atraviesan algunos materiales como la carne humana, aunque no los huesos.

Es sólo en la franja del espectro que va desde el violeta hasta el rojo donde las ondas electromagnéticas interaccionan (se reflejan o absorben) con la materia y nos permiten ver los objetos, sus formas, su posición, y dentro de esta franja del espectro podemos determinar qué frecuencia o conjunto de frecuencias refleja o emite cada objeto, es decir, el color que tiene.

3.5.-TEMPERATURA

La temperatura es un parámetro termodinámico del estado de un sistema que caracteriza el calor, o transferencia de energía.

Concretamente, dado un sistema en él se pueda expresar como suma de energías cinéticas de todas las partículas, y suma de energías potenciales de partículas tomadas por pares (es decir, H=T+V donde $V = \Sigma_{i<j} V(r_{ij})$), entonces tendremos que se cumple $3/2\ N\ K_B T = 1/n * \Sigma_{i<n} 1/2\ m_i v_i^2$. Siendo K_B la constante de Boltzmann.

Para medir la temperatura se utiliza el termómetro.

Multitud de propiedades fisicoquímicas de los materiales o las sustancias varían en función de la temperatura a la que se encuentren, como por ejemplo su estado (gaseoso, líquido, sólido, plasma...), la densidad, la solubilidad, la presión de vapor o la conductividad eléctrica. Así mismo es uno de los factores que influyen en la velocidad a la que tienen lugar las reacciones químicas.

En el Sistema Internacional de Unidades, la unidad de temperatura es el kelvin. Sin embargo, está muy generalizado el uso de otras escalas de temperatura, concretamente la escala Celsius (o centígrada), y, en los países anglosajones, la escala Fahrenheit. Una diferencia de temperatura de un kelvin equivale a una diferencia de un grado centígrado.

La temperatura adecuada para estar cómodos es un poco compleja de medir, ya que el calor recibido no sólo puede venir del aire que nos rodea, sino también de la radiación de objetos como las paredes o un sofá

al que le ha dado el sol. Para tener una idea más aproximada de la sensación se puede tomar la temperatura de varias formas.

TEMPERATURA SECA:

Se llama “Temperatura seca del aire de un entorno”, o más sencillamente, *temperatura seca*, a la del aire, prescindiendo de la radiación calorífica de los objetos que rodean ese ambiente concreto y de los efectos de la humedad relativa y de la velocidad del aire.

Se puede obtener con el termómetro de mercurio, cuyo bulbo, reflectante y de color blanco brillante, se supone razonablemente que no absorbe la radiación.

TEMPERATURA RADIANTE

La temperatura radiante tiene en cuenta el calor emitido por radiación de los elementos del entorno.

Se toma con un termómetro de bulbo, que tiene el depósito de mercurio encerrado en una esfera o *bulbo* metálico de color negro, para asemejarlo lo más posible a un cuerpo negro y absorba la máxima radiación. Para anular en lo posible el efecto de la temperatura del aire, el bulbo negro se aísla mediante otro bulbo en el que se ha hecho al vacío.

Las medidas se pueden tomar bajo el sol o a la sombra. En el primer caso tendrá en cuenta la radiación solar y dará una temperatura bastante más elevada.

También sirve para dar una idea de la sensación térmica.

La temperatura de bulbo negro hace una función parecida, dando la combinación de la temperatura radiante y la ambiental

TEMPERATURA HÚMEDA

Temperatura de bulbo húmedo o *Temperatura húmeda* es la temperatura que da un termómetro a la sombra con el bulbo envuelto en una mecha de algodón húmedo bajo una corriente de aire.

La corriente de aire se produce mediante un pequeño ventilador o poniendo el termómetro en un molinete y haciéndolo girar.

Al evaporarse el agua, absorbe calor, rebajando la temperatura, efecto que reflejará el termómetro. Cuanto menor sea la humedad relativa ambiente, más rápidamente se evapora el agua que empapa el paño.

Se utiliza para dar una idea de la sensación térmica o en los psicrómetros para calcular la humedad relativa.

UNIDADES DE TEMPERATURA

Kelvin (SI)
Grados Celsius (unidades habituales)
Grados Fahrenheit (unidades anglosajonas)
Grados Rankine (no convencional)
Grados Réaumur (no convencional)

3.6.-HUMEDAD

Se denomina humedad ambiental a la cantidad de vapor de agua presente en el aire. Se puede expresar de forma absoluta mediante la humedad absoluta, o de forma relativa mediante la humedad relativa o grado de humedad:

- La humedad absoluta es la cantidad de vapor de agua presente en el aire, se expresa en gramos de agua por kilogramos de aire seco (g/kg), gramos de agua por unidad de volumen (g/m^3) o como presión de vapor (Pa o KPa o mmHg). A mayor temperatura, mayor cantidad de vapor de agua permite acumular el aire.

- La humedad relativa es la humedad que contiene una masa de aire, en relación con la máxima humedad absoluta que podría admitir sin producirse condensación, conservando las mismas condiciones de temperatura y presión atmosférica. Esta es la forma más habitual de expresar la humedad ambiental. Se expresa en tanto por ciento.

3.7.- PRESENCIA

El sensor de proximidad es un transductor que detecta objetos o señales que se encuentran cerca del elemento sensor.

Existen varios tipos de sensores de proximidad según el principio físico que utilizan. Los más comunes son los interruptores de posición, los detectores capacitivos, los inductivos y los fotoeléctricos, como por ejemplo el de infrarrojos.

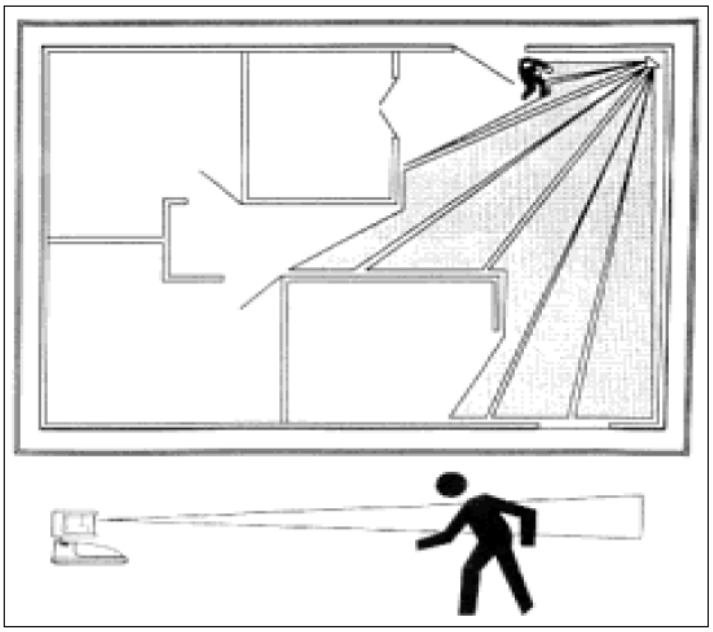


FIGURA 2: DETECCIÓN DE MOVIMIENTO MEDIANTE SENSORES PIR

En este caso se utilizarán sensores PIR (sensores de proximidad de infrarrojos) como se puede observar en la figura.

3.8.-ACTUADOR DIMMER

Los dimmer son dispositivos usados para regular el voltaje de una o varias lámparas. Así, es posible variar la intensidad de la luz, siempre y cuando las propiedades de la luminaria lo permitan.

Para controlar esa intensidad lumínica, se varía el ángulo de disparo, para ello antes habrá que tener una referencia de ese ángulo, con lo que será necesario detectar el cruce por cero de la red eléctrica.

4. - NORMAS Y REFERENCIAS

4.1.-DISPOSICIONES LEGALES Y NORMAS APLICADAS

A continuación se cita la normativa aplicada a la realización de este proyecto en materia eléctrica y de fabricación de las PCB:

- Norma UNE 20 524 75(1). Técnica de los circuitos impresos. Parámetros fundamentales.
- Norma UNE 20 524 77(2). Técnica de los circuitos impresos. Terminología.
- Norma UNE 20 621 80. Circuitos impresos. Métodos de ensayo.
- Norma UNE 20 621 84. Circuitos impresos. Diseño y utilización de placas impresas.
- Norma UNE 20 621 85. Circuitos impresos. Especificaciones para placas impresas de simple y doble cara con agujeros metalizados.
- Norma UNE 20 622 81. Código de símbolos para agujeros de circuito impreso.
- MI BT 031. Hace referencia a las condiciones de instalación y hace alusión a que durante el funcionamiento de los aparatos no se deben producir perturbaciones en la red.
- Directiva de Baja Tensión (BT 73/23/CEE) relativo a todo aquel material electrotécnico utilizado a una tensión menor de 1000V.
- Instrucción técnica complementaria para baja tensión. ITC-BT-36. Instalaciones a muy baja tensión.
- Instrucción técnica complementaria para baja tensión ITC-BT-51. Instalaciones de sistemas de automatización, gestión técnica de la energía y seguridad para viviendas y edificios.

4.2.-BIBLIOGRAFÍA

- Programación de aplicaciones
  - Core Java vol.2, Prentice Hall, Cay Horstmann, Gary Cornell
  - The Java Language Specification Third Edition, Gosling et al., Addison Wesley
  - Effective Java Second Edition, Josua Bloch, Addison Wesley
  - Programación concurrente en Java 2ª Edición, Doug Lea, Addison Wesley
- Programación en C
  - C Primer Plus,Stephen Prata, Sams Publishing
- Microcontrolador Atmel
  - C programming for Microcontrollers featuring Atmel Butterfly
- Redes 802.15.4
  - Wi-Fi, Bluetooth, ZigBee and Wi-Max , H. Labiod, H. Afifi,c. De Santis, Springer
  - ZigBee wireless networks and transceivers, Shahin Farahani PhD, Newnes
  - Sensor networks and configuration, Nitaigour P. Mahalik,Springer
- Drools
  - Jboss Drools Business Rules, Paul Browne, Packt Publishing
  - Drool Jboss Rules 5.0 Developers guide, Michael Bali, Packt Publishing

- Electrónica
  - Circuitos electrónicos. Discretos e integrados. Donald L. Schilling, Charles Belove. McGraw-Hill.
  - Sensores y acondicionadores de señal, Ramón Pallás Areny, Marcombo

4.3.- OTRAS REFERENCIAS

- Páginas web
  - www.digikey.com
  - www.avrfreaks.net
  - www.oracle.com
  - www.mysql.com
  - www.jboss.org
  - www.developer.yahoo.com
  - www.w3.org
  - www.atmel.com
  - www.arduino.cc
  - www.digi.com
  - www.zigbee.org
  - www.sparkfun.com
  - www.freescale.com
  - www.microchip.com
  - www.maxim-ic.com
  - www.nxp.com
  - www.zilog.com
  - www.humirel.com
  - www.sensirion.com
- Application notes y datasheets
  - AVR 133: Long delay generation using the AVR microcontroller
  - XBee/XBee Pro 802.15.4 OEM RF Modules v.1.x.E.x
  - Atmega 48/88/168/328 datasheet
  - UM10204 I2C bus specification and user manual
  - TU0113 Performing signal integrity analises, Altium
  - Atmel white paper, Innovative techniques for extremely low power consumption with 8 bit microcontroller.
  - AVR 042: AVR hardware desing considerations
  - ADG719 datasheet, Analog Devices
  - FT232R datasheet FTDI Chip
  - TPS79333 datasheet, Texas Instruments
  - TPS61131 datasheet, Texas Instruments
  - MAX 1555 datasheet, Maxim
- Referencias y manuals
  - MySql 5.0 Reference guide

5. - DEFINICIONES Y ABREVIATURAS

5.1. - DEFINICIONES

5.1.1.- Accesibilidad: Propiedad de una aplicación de facilitar la información que produce a quien la consume.

5.1.2.- Alias: Pseudónimo, palabra que identifica a un elemento.

5.1.3.-Base de datos: Sistema de almacenamiento de información basado en tablas y relaciones entre tablas.

5.1.4.-Clase: Representa una entidad modelada en lenguaje de programación orientado a objetos.

5.1.5.-Clase abstracta: Clase con métodos no implementados cuya funcionalidad depende de quién implemente dicha clase abstracta.

5.1.6.-Coordinador: Elemento de una red de sensores que está encargado de crear la red y gestionar la información que circula por ella.

5.1.7.-Data center: Espacio físico donde se albergan los servidores web, que dispone de mecanismos de control de temperatura y humedad, así como sistemas de seguridad, energía y red redundantes.

5.1.8.-Disponibilidad: Capacidad de una aplicación web de ser accedida en cualquier momento, independientemente de las condiciones.

5.1.9.-Escalabilidad: Capacidad de una aplicación web de incrementar su uso y volumen de datos.

5.1.10.-Interfaz: En programación orientada a objetos, es un contrato, una definición de cabeceras de métodos que quienes hereden de esa interfaz, deberán implementar.

5.1.11.-Listener: En programación orientada a objetos, es una clase especializada en escuchar eventos, como por ejemplo, una pulsación de ratón o de teclado.

5.1.12.-Mote: Elemento de una red inalámbrica de bajo coste, que está formado por un transceptor de red y un microcontrolador que lo gestiona.

5.1.13.-Multiplataforma: Capacidad de una aplicación de funcionar de la misma forma en diferentes sistemas.

5.1.14.-Objeto: Es una instancia de una clase, que ocupa un espacio en memoria y es manejable.

5.1.15.-Plug and Play: Capacidad de un bloque software o hardware de añadirse o eliminarse de un sistema completo, sin afectar al funcionamiento de dicho sistema.

5.1.16.-Proxy: Entidad intermedia entre dos elementos que se comunican y que generalmente se encarga de gestionar funcionalidades de bajo nivel.

5.1.17.-Sistema embebido: Aquel sistema electrónico con recursos limitados, que forma parte de un producto final, cuyo cometido está restringido a una operación en concreto.

5.1.18.-Toolchain: Conjunto de herramientas informáticas que se utilizan para la compilación, programación y depurado de aplicaciones.

5.1.19.-Topología: Disposición física de elementos, con unas determinadas propiedades.

5.1.20.-Trama de datos: Conjunto de información enviada a través de un sistema de comunicación ya sea cableado o inalámbrico, que tiene una determinada estructura, determinada por el protocolo empleado.

5.2.- ABREVIATURAS

5.2.1.- A/D: Conversor analógico a digital

5.2.2.- AES: Advanced encryption standard

5.2.3.- API: Application Programming Interface

5.2.4.-CISC: Complex instruction set computing

5.2.5.-CRC: Comprobación de redundancia cíclica

5.2.6.-CSMA: Carrier sense multiple access

5.2.7.-DIP: Dual in line package

5.2.8.-FFD: Full function device

5.2.9.-IEEE: Institute of Electrical and Electronics engineers

5.2.10.-ISP: Internet service provider

5.2.11.-JNI: Java native interface

5.2.12.-JVM: Java virtual machine

5.2.13.-LQI: Link quality interface

5.2.14.-LR-WPAN: Low rate wireless personal area network

5.2.15.-MAC: Media access control

5.2.16.-OSI: Open system interconection

5.2.17.-P2P: peer to peer

5.2.18.-PAN: Personal area network

5.2.19.-PWM: Pulse width modulation

5.2.20.-QFN: Quad flat no leads package

5.2.21.-QLP: Quad lead package

5.2.22.-RFD: Reduced function device

5.2.23.-RISC: Reduced instruction set computing

5.2.24.-RSSI: Received signal strength indication

5.2.25.-SH: Serial high

5.2.26.-SL: Serial low

5.2.27.-SMD: Surface mount device

5.2.28.-SPI: Serial peripheral interface

5.2.29.-THD: Through hole device

5.2.30.-TIC: Tecnología de la información y la comunicación

5.2.31.-UART: Universal asynchronous receiver transmitter

5.2.32.-USB: Universal serial bus

5.2.33.-WLAN: Wireless local area network

5.2.34.-XML: Extensible markup language

6. - REQUISITOS DE DISEÑO

6.1. - ARQUITECTURA DEL SISTEMA

Desde un punto de vista general, la arquitectura, es decir, cómo se reparten las funciones principales en bloques y cómo se interconectan estos bloques entre sí, es la parte fundamental del análisis del sistema.

Dados los requisitos estructurales enunciados en el apartado 2.-ALCANCE y que a continuación se resumen, se puede plantear un diseño lo suficientemente flexible y escalable:

- Gestión de la red de motes (PAN) centralizada y ortogonal frente a comportamientos anómalos del proveedor de ISP.
- Acceso desde cualquier terminal con conectividad a internet.
- Comunicación bidireccional entre los sistemas locales y el sistema principal.
- Persistencia robusta y estructurada de datos

6.2. - SISTEMA LOCAL

En cada una de las actuaciones físicas o instalaciones que se pueden realizar, es necesaria la existencia de un sistema central que sea capaz de gestionar la red de dispositivos a alto nivel, es decir, la información que transporta.

Y no solo esto, también debe ser capaz de establecer una comunicación bidireccional con el exterior, entendiendo exterior como cualquier usuario del sistema que quiere recibir información de lo que está ocurriendo, como actuar directamente sobre el sistema.

Esto establece en principio los criterios para la selección de la tecnología a emplear:

- Por un lado, al tener como primer objetivo la gestión de la red, debe acceder a la misma, de tal forma que deberá formar parte de ella y además con una relevancia importante. Este acceso debe ser físico, es decir, en sistema en el cual va a correr el software, debe disponer de un soporte para acceso a la red, lo que se puede conseguir instalando un dispositivo transceptor en dicho sistema.
- Una vez que hemos determinado que el sistema debe acceder físicamente a la red, y por lo tanto dispondrá de un transceptor de red, el software que ejecuta debe poder acceder a dicho transceptor, lo que implica que deberá ser posible crear o usar las librerías pertinentes para tal fin.
- Por otra parte, una vez sea posible acceder físicamente a la red, y obtener la información que por ella circula, hay que poder estructurar esa información, de tal forma que sea manejable para la programación y para el almacenamiento de la misma. Esto se consigue usando lenguajes de programación que permitan la creación de estructuras de datos adecuadas a la necesidad del problema.

- Una vez ya tenemos la información de la red y la hemos estructurado de forma adecuada, hay que hacerla accesible a los interesados en la misma. Para esto el sistema debe permitir acceso a internet para poder enviar dicha información al sistema servidor y de ahí ponerla a disposición de los usuarios. Esta es la primera parte de la comunicación bidireccional, del sistema instalado al sistema servidor.

- La segunda parte de la comunicación bidireccional, del sistema central a los sistemas instalados, es un poco más compleja, pues implica la identificación de cada uno de los sistemas instalados, usando una comunicación punto a punto. Para esto, la infraestructura que se utilice debe permitir una comunicación P2P cliente-servidor.

6.3.- SISTEMA EMBEBIDO

Siguiendo lo dispuesto hasta el momento en el planteamiento del presente proyecto, para el diseño y programación del sistema embebido que conforma la red de motes surgen necesidades que ya se han planteado con anterioridad y que se han de trasladar y resolver en el mundo embebido. También surgen necesidades específicas como puede ser el tratamiento de los sensores, la gestión de eventos asíncronos…etc. La siguiente lista presenta los requisitos que se plantean en ambos ámbitos con carácter general:

- El software embebido debe permitir gestionar las tramas de datos en el mismo sentido que el sistema local, generando tramas de asociación y sensado.
- El software embebido debe realizar una gestión eficiente de la energía consumida por los motes, promoviendo modos de bajo consumo.
- El diseño de los motes debe ser flexible para permitir múltiples aplicaciones sin tener que modificarlos sustancialmente.
- El hardware debe ser eficiente energéticamente, pues pueden darse aplicaciones en las que la alimentación dependa de baterías.

6.4. - SISTEMA SERVIDOR

Los requisitos principales que se plantean son los relacionados con aplicaciones web, en los que podemos destacar, seguridad, disponibilidad y escalabilidad.

La seguridad se puede plantear a distintos niveles, pero en un nivel básico o inicial, se debe implementar un mecanismo de autenticación para los usuarios. La autenticación consiste en que un usuario introduce unas credenciales, que generalmente son un usuario y una contraseña, que se validan contra un mecanismo de gestión de usuarios, ya sea LDAP, base de datos, fichero de texto…etc.

La disponibilidad indica el grado de respuesta que tiene una aplicación web corriendo en un servidor de aplicaciones frente a eventos anómalos. Supongamos que ubicamos nuestro servidor de aplicaciones en una zona donde se producen frecuentes terremotos y como consecuencia cortes en el suministro eléctrico y telefónico. La disponibilidad de nuestro servidor para ser accedido tiene una probabilidad muy baja de ser aceptable. Sin embargo, ubicándolo en un Data Center con unas condiciones ambientales controladas y con

sistemas redundantes de alimentación y conectividad, tendremos una alta probabilidad de que la disponibilidad sea muy buena.

El término escalabilidad es una medida de cómo es capaz de crecer el sistema. Los sistemas por lo general tienden a crecer y además lo hacen en varias dimensiones. Puede crecer el número de usuarios que acceden, el volumen de datos que se maneja o la funcionalidad de la aplicación entre otros factores. La escalabilidad mide como de preparado está el sistema frente a estas ampliaciones. Algunas de las características de cambio se pueden solucionar empleando más sistemas hardware como memoria dinámica o sistemas de almacenamiento masivo, pero otras dependen directamente de la arquitectura de la aplicación y de cómo se ha programado. Una mala programación puede hacer que la escalabilidad tenga un límite muy bajo y recorte la efectividad de los elementos hardware.

7.-ANÁLISIS DE SOLUCIONES

7.1- ARQUITECTURA DEL SISTEMA

La figura 3, representa un posible esquema que sería capaz de aunar las principales características estructurales.

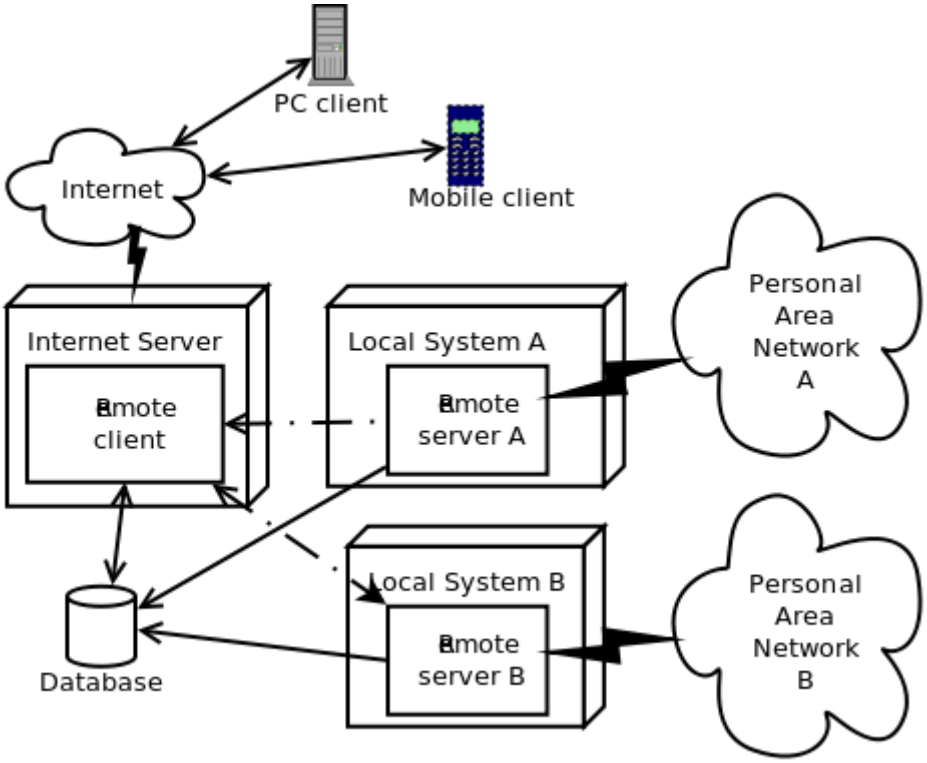


FIGURA 3: ARQUITECTURA GENERAL DEL SISTEMA

El sistema se puede particionar en cuatro regiones bien diferenciadas:

1. La red de motes o Personal Area Networks (PAN)
2. Los sistemas locales que gestionan las redes de motes
3. El sistema de persistencia, que almacena la información de configuración y los datos relevantes.
4. El sistema central que hace accesible la información al exterior y gestiona el acceso.

7.2.- RED DE MOTES

Uno de los objetivos del presente proyecto es el empleo de un sistema de elementos coordinados, de medida y actuación, trabajando dentro de una red inalámbrica.

En el párrafo anterior aparece el término *coordinación* que es fundamental en el desarrollo que se realiza a continuación, porque el objetivo no es simplemente enviar información de un punto a otro, sino que además esa información tenga un formato estructurado, que se realice un control del tráfico de red o que se permita añadir nuevos elementos a la red en caliente, entre otros objetivos.

Además hay que tener en cuenta ciertos requisitos en cuando a rango o distancia de transmisión, ya que puede existir la necesidad de alcanzar distancias de decenas de metros en espacios no abiertos.

La velocidad de transmisión no es un factor determinante en nuestro caso y de hecho se ha de emplear un baudrate bajo para minimizar el coste energético. No se necesita velocidad extrema, se necesita bajo consumo.

Esto que hoy parece tan común, con el incremento del uso de las tecnologías WIFI o comunicaciones móviles de banda ancha, desde el punto de vista del coste monetario y del coste energético, se convierte en un verdadero reto.

Para formar una red inalámbrica con las características anteriormente mencionadas, con carácter genérico, son necesarios un conjunto de elementos en el nivel físico como:

- Coordinadores o router de red, con capacidad para formar y gestionar la red.
- Elementos asociados a la red con capacidad para enviar y recibir información.
- Una topología definida con capacidad de auto asociación/des asociación de elementos.

En el nivel lógico, la tecnología elegida debería proporcionar solución a los siguientes problemas:

- Existencia de una API para el manejo y programación de la red.
- Estructuración de la información en tramas bien definidas según la API.
- Control del tráfico, control de colisiones...etc.
- Posibilidad de encriptación de dichas tramas.

Esto es una visión genérica pero suficiente para tomar una decisión informada.

7.2.1.- PROTOCOLOS DE COMUNICACIÓN

El protocolo de comunicación es lo que fundamenta cómo se realiza la comunicación, cuales son los mecanismos de envío y recepción, cómo se gestionan los envíos fallidos, cómo se controla la congestión del tráfico, cuales son los niveles del modelo OSI implementados...etc., por lo que define indefectiblemente cómo funciona el sistema.

Antes de realizar la implementación, se va a realizar un pequeño análisis de varios protocolos que pueden encajar dentro de las necesidades planteadas para la red inalámbrica, llegando finalmente a la justificación de la decisión adoptada.

7.2.1.1- WLAN

Una posibilidad podría ser el uso de la tecnología WLAN, que extiende Ethernet al campo inalámbrico, con ventajas e inconvenientes.

Como ventajas se podría afirmar que ofrece un formato de trama estructurado, implementa un control de colisiones basado en CSMA/CD, establece dos tipos de elementos de red, los DCE(dispositivos de control de comunicaciones) que gestionan el tráfico de red y los DTE(dispositivos terminales de red) que son quienes generan y finalmente reciben las tramas.

Como inconvenientes se podría destacar, el elevado consumo de los transceptores inalámbricos, que está entre 100 mA y hasta 400 mA, la complejidad inherente al modelo OSI implementado sobre los mismos, hace relativamente compleja la programación y su uso en microcontroladores. También los encapsulados empleados son de alta complejidad para soldadura manual, aunque se podría pensar en usar una solución on-board, donde nos ahorraría el tener que soldar el encapsulado SMD a costa de un mayor coste.

La velocidad de transmisión va variando conforme avanzan las especificaciones WLAN, aunque por ahora la mejor relación coste-prestaciones está sobre los 54Mb/s, en la última revisión del modelo, se especifican velocidades sobre los 400Mb/s. Esto que parece que siempre es bueno, en nuestro caso es innecesario, pues a mayor velocidad, mayor coste energético.

7.2.1.2.- BLUETOOTH

Otra posibilidad podría ser el empleo de la tecnología bluetooth, que está pensada para la transmisión serie de información punto a punto entre dispositivos que dispongan de transceptores con esta tecnología. Cabe destacar que la velocidad de transmisión está entre 1Mb/s y 24Mb/s, rangos bastante elevados teniendo en cuenta nuestros propósitos.

Bluetooth permite la creación de pequeñas redes de hasta 7 elementos, donde existe un coordinador o maestro de red y una serie de dispositivos asociados, en lo que se conoce como *piconet*. Existe también la posibilidad de enlazar piconets entre sí por medio de gateways o dispositivos que actúan de enlace, en lo que se conoce como *scatternet.*

El alcance máximo viene determinado por la potencia de transmisión y en concreto se especifican tres potencias para transceptores bluetooth:

- 1mW, con un alcance de hasta 1metro.
- 10mW, con un alcance hasta 10 metros.
- 100mW con un alcance de hasta 100 metros.

A la hora de formar la red, siempre hay un dispositivo que hace de maestro, permitiendo a los demás interesados en formar parte de la red, convertirse en esclavos. Ahora bien, los roles de los elementos pueden cambiar dinámicamente por lo que un esclavo puede convertirse en maestro para realizar alguna tarea que requiera de este nuevo rol.

Bluetooth permite varios modos de sueño para dispositivos dentro de una misma piconet, todos ellos controlados por el maestro de la red.

- Hold mode

  Un dispositivo bluetooth entra en modo de mantenimiento cuando ha de permanecer un largo periodo sin enviar información.

- Sniff mode

  Cuando un dispositivo entra en modo escucha, permanece activo en un modo de bajo consumo, escuchando tramas concretas que envía el maestro, de tal forma que alterna periodos de sueño, con periodos de escucha.

- Park mode

  En este modo, un dispositvo queda excluido para la transmisión de información dentro de una piconet, entrando en un modo de sueño cíclico. En este modo de sueño cíclico, durante los periodos activos, escucha el medio para detectar las tramas de guía o beacons. Si un beacon contiene la dirección del dispositivo en este estado, cambia a un estado activo y se reincorpora a la red de nuevo.

La transmisión de información está basada en tramas o paquetes de datos con el siguiente formato:

- Código de acceso: 72 Bytes que identifican la piconet y al dispositivo dentro de la piconet.
- Cabecera de control: 54Bytes
- Datos: hasta unos 2Kb

Dentro de la red, los paquetes se direccionan usando direcciones de 48bits, aunque generalmente se emplean *alias* para los dispositivos.

Una de las cuestiones relevantes de bluetooth es el control de acceso, es decir, a qué dispositivos permitimos que hagan solicitudes de servicios de la red. Esta restricción es totalmente lógica pues en una piconet, se ponen a disposición de todos los elementos, información y servicios de acceso restringido. Para ello, se diseñó un mecanismo denominado *pairing,* que se inicia manualmente al activar el enlace inalámbrico. El proceso de pairing se inicia automáticamente con la primera solicitud de un dispositivo que no está pareado y por lo tanto solo existirá intercambio de información cuando dos dispositivos están pareados. Se emplean dos métodos para controlar el acceso:

- Legacy pairing o introducción de un código que ambos dispositivos conocen.
- Secure simple pairing(SSP) que usa una técnica simplificada de encriptación con clave pública.

Una vez analizado, podemos concretar lo siguiente al respecto de la posibilidad de utilización en el presente proyecto:

- Las distancias de transmisión que establece de 1 y 10 metros son relativamente bajas y el rango de los 100 metros se consigue a costa de un consumo elevado (100mW).
- La capacidad para formar una red poblada de elementos es reducida (piconet) a pesar de que permite la asociación de varias redes entre sí, la complejidad no sería razonable.
- Las velocidades de transmisión son aceptables aunque quizá elevadas para nuestro propósito.

7.2.1.3.- IEEE 802.15.4

El protocolo IEEE 802.15.4 especifica subcapa MAC y capa física para redes inalámbricas de baja velocidad y bajo coste energético o LR-WPAN. Este protocolo está asociado al más conocido ZigBee, que especifica dos capas más por encima de 802.15.4, como son, capa de red y capa de aplicación.

Este protocolo defina básicamente tres tipos de nodos:

- Coordinador PAN

  Es el principal coordinador de la red, el cual identifica su PAN a la cual otros nodos se pueden asociar. Adicionalmente proporciona mecanismos de sincronización para los nodos de la red mediante la transmisión de una guía que contiene la identificación de la PAN y otra información relevante.

- Coordinador o router

  Tiene las mismas funcionalidades que el coordinador PAN excepto que no puede formar un red por sí mismo. Un coordinador está asociado a un coordinador PAN y proporciona el mecanismo de sincronismo a nivel local a los nodos que tiene a su alcance.

- Nodo

  Es un elemento de red que no tiene ninguna funcionalidad de coordinación.

En el estándar 802.15.4, los dos primeros tipos de nodos son denominados como FFD (full function device), los cuales implementan todas las funcionalidades del protocolo para asegurar sincronismo y gestión de la red. El tercer elemento es denominado RFD (reduced function device) el cual opera con una mínima implementación del estándar.

Una red bajo este protocolo, debe al menos incluir un dispositivo FFD actuando como coordinador PAN. Una vez se ha formado la red, esta debe ser mantenida por su coordinador PAN mediante el envío de guías de sincronismo, gestionando la asociación y disociación de nodos.

Se establecen tres posibles topologías:

- Estrella

  Un único nodo actúa como coordinador PAN. La topología es centralizada, esto es, un nodo de la red que quiere comunicarse con otro nodo de la misma red, debe enviar la información al coordinador PAN el cual la reenviará al destino. En cuanto al consumo de energía, el estándar recomienda que el coordinador PAN debiera estar alimentado mediante un suministro estable y no agotable a corto plazo, mientras que para los demás nodos, permite el uso de sistemas de alimentación autónomos como baterías. Esta topología presenta problemas de escalabilidad en cuanto aumenta en número de nodos de la red, ya que las direcciones están limitadas y se pueden producir degradación si existen múltiples nodos que quieren transmitir al mismo tiempo.

- Punto a punto

  Un único nodo actúa como coordinador PAN para formar la red, pero en este caso la topología es descentralizada, por lo que cada nodo puede comunicarse directamente con otro dentro de su rango. Esto ofrece cierta flexibilidad pero añade complejidad a la hora de añadir la conectividad punto

a punto a todos los nodos de la red. Esta topología es más escalable y ofrece mayores oportunidades para el ahorro energético, pues la comunicación no está centrada en un único nodo.

- Agrupación de árboles

  Un coordinador es nombrado como el coordinador PAN de toda la red. No obstante, cualquier nodo puede actuar como coordinador y proporcionar sincronismo a otros nodos. El estándar no define como construir el árbol, solo indica que es posible y que debe ser inicializado en capas superiores. Para sistemas complejos, es posible formar una red de múltiples árboles vecinos.

Desde el punto de vista físico, 802.15.4 ofrece tres bandas operacionales: 2.4Ghz, 915Mhz y 868Mhz. Hay un único canal en la banda de los 868 Mhz, 10 canales en la de 915 Mhz y hasta 16 canales en la banda de 2.4 Ghz.

Los baudrates que se manejan para la banda de 2.4 Ghz pueden alcanzar los 250Kbps, 40Kbps para la banda de los 915 Mhz y 20 Kbps para los 868 Mhz.

Como características que ofrece la especificación 802.15.4 en capa física se pueden destacar las siguientes:

- Activación y desactivación del transceptor de radio

  El transceptor de radio puede trabajar en tres estados: transmitiendo, recibiendo o durmiendo. Cuando la capa MAC realiza una solicitud, la radio es activada o desactivada según la necesidad.

- Detección de energía de recepción

  Es una estimación de la potencia de la señal recibida.

- Indicación de la calidad del enlace

  Caracteriza la relación fuerza/calidad de la señal recibida en un enlace.

- Selección del canal

  En cuanto al mecanismo de acceso al medio, sin entrar en detalles cuantitativos, podemos decir que emplea una técnica denominada CSMA-CA (carrier sense multiple access collision avoidance) lo que traducido viene a ser, acceso múltiple con detección de portadora, evitando colisiones de paquetes.

  A grandes rasgos el funcionamiento es el siguiente: un transceptor cuando tiene información para transmitir, lo que hace en primera instancia es escuchar el medio para detectar si alguien está transmitiendo. En caso de que determine que hay alguien transmitiendo, espera un tiempo aleatorio dentro de un rango de valores, y realiza el mismo proceso de detección. Este proceso se repite hasta que encuentra el medio desocupado y realiza una transmisión corta o solicitud de envío, para activar al receptor y ver si está disponible para recibir. De esta forma, las estaciones cercanas detectarán esa transmisión corta y no transmitirán. Si el receptor está disponible para recibir, envía una confirmación al transmisor para que inicie la transmisión efectiva.

7.1.2.4.- LA SOLUCIÓN A LA RED DE MOTES

Una vez analizadas las opciones más razonables para implementar la red de dispositivos inalámbricos que confirmarán el sistema de medida y control, parece que lo que más adecuado por sus características, es el empleo de la tecnología IEEE 802.15.4, con la siguiente relación de justificaciones:

- El consumo energético es bajo en relación a la tecnología WLAN.
- El coste monetario de los transceptores 802.15.4 es menor que los WLAN y bluetooth.
- Realiza una gestión de la transmisión que evita colisiones en el mismo sentido que la tecnología WLAN detecta las colisiones.
- Permite varias topologías, lo que da una flexibilidad que Bluetooth no permite.
- Las distancias que se alcanzan son razonables sin un incremento exagerado del consumo, como ocurre con Bluetooth o WLAN.
- Permite un estado de sueño para los transceptores, de forma independiente, en contraposición a Bluetooth, que es el maestro quién establece los criterios de sueño.

Habiendo determinado que posiblemente IEEE 802.15.4 sea la mejor solución, ahora queda por elegir alguno de los dispositivos que existen en el mercado. A continuación se describen algunos de los transceptores comerciales que encajarían en nuestro diseño.

7.2.2.- TRANSCEPTORES

7.2.2.1.- ST MICROELECTRONICS SN260

Es un controlador que implementa las capas física y MAC para el estándar 802.15.4, y además es compatible con ZigBee, para la banda de los 2.4 Ghz.

Permite el control por un host externo o microcontrolador a través de conexiones SPI o UART y dispone de periféricos y memoria integrados para minimizar el uso de componentes externos.

Proporciona tres modos de funcionamiento:

- Idle o ausente: Modo de funcionamiento normal donde el transceptor permanece activo a la espera de recibir o enviar información.
- Deep sleep: Modo de sueño en el que se desactivan la mayoría de bloques integrados en el transceptor, dejando activas las funciones críticas. El regulador interno se desactiva y todas las salidas se congelan en el estado en el que estaban justo antes de entrar en este modo. Para sacar al dispositivo de este estado a un modo activo, se puede lograr mediante un evento temporizado o una señal externa. El consumo se reduce entorno al microamperio.
- Power down: Funciona idénticamente al modo deep sleep pero en este caso el bloque del timer permanece desactivado de tal forma que la única posibilidad de cambiar a un estado activo, es mediante una señal externa. El consumo se reduce entorno al microamperio.

Dispone de un bloque acelerador para encriptación AES así como funciones de capa MAC implementadas en hardware para cumplir con requisitos de timing estricto.

El transceptor, internamente se alimenta a 1.8 V, y el sistema de alimentación externo recomendado debe proporcionar entre 2.1V y 3.6 V.

El consumo en condiciones estándar de 1.8V de alimentación interna, a una temperatura de 25ºC y a potencia máxima, está en 36mA tanto en transmisión como en recepción.

El encapsulado es un QFN de 40 pines de 6 x 6 mm.

7.2.2.2.- TEXAS INSTRUMENTS CC2420

Se trata de un transceptor que implementa capa física en la banda de 2.4 Ghz con soporte para capa MAC.

Implementa un bloque SPI para el control y comunicación con microcontrolador externo.

Dispone de un regulador integrado al que hay que proveerle de una tensión externa entre 2.1 y 3.6 V.

Implementa múltiples funciones de capa MAC en hardware tales como, generador de preámbulo automático, generación y comprobación del checksum, RSSI, LQI y encriptación AES.

El consumo tanto en transmisión como en recepción está por debajo de los 20 mA.

Dispone de tres modos de operación:

- Regulador de tensión off: En este estado se puede considerar que el transceptor está completamente apagado con un consumo de 0.02 uA.
- Power down: Con un consumo de 20 uA, donde el oscilador interno está deshabilitado como también las colas de transmisión y recepción y la memoria interna.
- Idle: Con un consumo de unos 430 uA.

El encapsulado es un QLP de 48 pines de 7 x 7 mm.

7.2.2.3.- DIGI XBEE

Se trata de un bundle de componentes que en conjunto implementan la especificación IEEE 802.15.4 con el añadido de un sistema conversor A/D para funcionamiento autónomo.

El formato físico es atractivo para prototipos ya que el encapsulado no requiere de soldadura de precisión.

En principio se presenta en las tres bandas de funcionamiento: 868 Mhz, 915 Mhz y 2.4 Ghz, con lo que cubre todas las necesidades.

Además se presentan en varias potencias de transmisión: 1 mW, 2 mw, 50 mW, 60mW y 100 mW, lo que permite ajustar con precisión el producto a la necesidad.

En cuanto a la interfaz con controladores externos, únicamente dispone de una UART para transmisión serie, con un modem interno que permite configurar los parámetros de funcionamiento mediante comandos AT.

Dispone de dos formatos de transmisión de información:

- Modo transparente: Se envía la información por RF tal y como llega a la UART.
- Modo API: Envía la información de forma estructurada en paquetes de datos con comprobación CRC.

Dispone de varios modos de bajo consumo configurables bien por hardware o firmware:

- Hibernate: Se activa al poner a un nivel alto el terminal Sleep_RQ, así que es activado por hardware. Al activar el terminal, el módulo termina de transmitir y entra en un modo de bajo consumo, en torno a 10 uA y no responde hasta que no se desactiva el terminal.
- Doze: Funciona igual que el modo hibernate pero tiene un tiempo de wake up menor y un consumo mayor, sobre los 50 uA.
- Cyclic: Se configura únicamente en el firmware y permite que el transceptor duerma periódicamente, con un ciclo de sueño fijado por un parámetro configurable.

El rango de alimentación está entre 2.8 V y 3.4 V y el consumo en general está sobre los 45 mA en transmisión y 50 mA en recepción, en condiciones de alimentación de 3.3V y para el caso de transceptor de 1mW.

7.2.2.4.- ELECCIÓN DEL TRANSCEPTOR

Una vez vistas las características de algunos modelos comerciales de transceptores IEEE 802.15.4, se ha decidido que para los propósitos de prototipado, la solución más sencilla desde el punto de vista de montaje físico es el empleo de los módulos XBee, ya que no requieren de soldadura de precisión, como en el resto de casos analizados, cuyos encapsulados SMD, dificultarían notablemente el montaje de los prototipos.

En cuanto a la interfaz que ofrece es más que suficiente para trabajar con microcontroladores sencillos, ya que la gran mayoría disponen te uno o varios bloques UART.

La flexibilidad que ofrecen en cuanto a la gama de potencias de transmisión y la posibilidad de mezclar transceptores de diferentes potencias en una misma red, puede ser útil a la hora de resolver problemas en casos en los que las distancias o las atenuaciones sean mayores de lo previsto.

El interfaz de programación es relativamente sencillo y en el caso API, las tramas de datos están bien definidas a alto nivel y únicamente se requiere la elaboración del software necesario para generar y reconocer las tramas especificadas, por lo que no existe la necesidad de trabajar con un stack complejo para IEEE.

Los modos de sueño que ofrece son suficientes para la mayoría de aplicaciones y aunque el consumo es sensiblemente mayor que el resto, se compensa con la facilidad de montaje y programación, que en términos de coste de proyecto son ventajosos.

7.2.3. - MICROCONTROLADORES

Dentro del desarrollo de los motes, el otro elemento principal, junto al transceptor wireless, es el microcontrolador. Éste ejecuta el programa que hace que un mote funcione y esto incluye generar las tramas de datos para envío, parsear las tramas que se reciben, realizar tareas temporizadas, realizar conversiones A/D o responder ante eventos externos, entre otras posibles tareas.

Existen microcontroladores con arquitecturas Von-Neumann y con arquitectura Harvard, con un elevado número de instrucciones (CISC) o con un conjunto pequeño (RISC), con anchos de palabra de 8 bits hasta 32 bits y con una memoria interna de centenas de bytes hasta centenas de Kilobytes, por lo que la gama es amplia, y debemos restringir el conjunto a la necesidad concreta.

Para la aplicación en concreto, los aspectos a alto nivel más a tener en cuenta son los siguientes:

- Curva de aprendizaje “rápida”.
- Disponibilidad de herramientas para desarrollo (compilación, programación y depurado).
- Disponibilidad de librerías testadas y posibilidad de crear nuevas.

En un nivel inferior, hay que contemplar las necesidades de los motes, que desde un punto de vista genérico pueden ser las siguientes:

- Conversor A/D integrado, para minimizar coste, con una resolución adecuada.
- Sistemas de temporización o timers, para realizar tareas periódicas.
- Bloque de comunicaciones serie, para poder emplear el transceptor 802.15.4 elegido.
- Bajo consumo de potencia y alimentación a tensiones adecuadas y concordantes con el transceptor inalámbrico, para evitar usar dos sistemas de alimentación.
- Disponibilidad de encapsulados DIP de fácil montaje para prototipado.

A todo esto y con carácter personal se van a plantear algunos requisitos que, aunque a priori no están dentro de los requisitos técnicos, se verá que son una valiosa ayuda:

- Herramientas de desarrollo open source y open hardware, lo que evita tener que pagar licencias en elementos como compilador, entorno de desarrollo software o dispositivo de programación para el microcontrolador, además de proveer de librerías testadas por comunidades de desarrolladores de todo el mundo con un buen soporte en internet.
- Herramientas multiplataforma, para permitir al ingeniero, desarrollador...etc, moverse libremente en el entorno que más cómodo se sienta.

Dados los requisitos de interfaz con el transceptor wireless, el microcontrolador debe incluir al menos un bloque UART para la transmisión de información.

En cuanto a rangos de alimentación, como ya se ha mencionado, sería bueno que el microcontrolador tuviera un rango similar o que menos incluyera el que ofrece en transceptor, que está en torno a los 3 V.

En cuanto a la velocidad del clock no existen requisitos especiales.

7.2.3.1.- ATMEL

Atmel fabrica una familia de microcontroladores denominada AVR, que van desde microcontroladores de 8 hasta 32 bits, pasando por los recientes XMEGA de 16 bits.

Atmel, al igual que Microchip, basa el diseño de sus dispositivos en una arquitectura Harvard, con memoria y buses separados para instrucciones y datos, para maximizar el paralelismo mediante etapas de pre-fetch.

El conjunto de instrucciones, aunque todavía se encaja en RISC, tiene más instrucciones, en torno a 130, que su competidor Microchip. No obstante, la mayoría de ellas solo requieren de un ciclo de reloj para ejecutarse, mientras que el ciclo de ejecución en un microcontrolador PIC típico, es de cuatro ciclos.

Una característica importante es que los dispositivos de una misma familia, son intercambiables en el sentido en que el software no cambia si cambia el microcontrolador, siempre que no se cambie de familia.

Los periféricos integrados más comunes son los siguientes:

- Bloques de comunicaciones I2C, SPI y USART.
- Conversor A/D de 10 bits por aproximaciones sucesivas, con varias entradas multiplexadas.
- Bloques de contaje (capture), temporización (timers) y PWM.
- Entradas de interrupción externa específicas y generales.

Disponen de seis modos de bajo consumo, lo que supone una gran flexibilidad a la hora de elegir un modo, en función de la necesidad concreta. En estos modos, el consumo se reduce hasta la décima de microamperio, dependiendo de la tensión de alimentación.

En cuanto al entorno de desarrollo, ATMEL ofrece la herramienta AVR Studio para los microcontroladores de 8 bits y AVR32 Studio para la gama de 32 bits. Esta herramienta es gratuita y permite desarrollar aplicaciones tanto en ensamblador como en lenguaje C, programar usando un programador externo e incluso depurar si se dispone de un debugger. Tiene la desventaja de que solo funciona bajo Windows.

No obstante existe una buena cantidad de recursos libres que hacen muy fácil el desarrollo para los AVR. Así podemos indicar que existe un toolchain para AVR totalmente libre, con versiones para Windows (winavr) y para Linux (avr-gcc-toolchain), mediante el cual se puede compilar código C y C++, se puede transferir el programa al microcontrolador e incluso permite depurar.

Atmel y otros fabricantes independientes, ofrecen placas de evaluación y desarrollo a precios muy competitivos, lo que minimiza todavía más el coste de desarrollo.

7.2.3.2.- MICROCHIP

Microchip fabrica una familia de microcontroladores denominada PIC, que dispone de una importante gama de dispositivos con diferentes prestaciones, que pueden ser empleados en muy diversas aplicaciones.

Todos los microcontroladores PIC están unidos por un denominador común, la arquitectura Harvard modificada, y el juego de instrucciones RISC.

En la arquitectura Harvard del PIC, el ancho de palabra para los datos es de 1 byte mientras que el ancho de palabra para instrucciones es de 14 bits, lo que permite albergar todas las instrucciones en una sola pala-

bra, al contrario que en la arquitectura Vonn Neuman, donde pueden existir instrucciones que ocupan dos o más palabras.

El conjunto de instrucciones es bastante reducido y oscilan entre 35 instrucciones para los PIC de gama baja y 70 instrucciones para los de gama alta, lo que facilita el aprendizaje.

Los PIC integran los bloques periféricos más comunes como:

- Módulos PWM con resolución de hasta 10 bits.
- Bloques contadores y comparadores, con una resolución de hasta 16 bits.
- Conversor A/D integrado de 10 bits y varias entradas multiplexadas.
- Comunicaciones medianto I2C, SPI y USART.
- Comparadores analógicos programables.
- Mútiples entradas para interrupciones externas.

Por lo general, disponen de varios modos de bajo consumo, en los que un oscilador interno a 32Khz hace de clock del sistema. Mientras se encuentra en alguno de estos modos, el consumo indicado por el fabricante está en torno a la decena de microamperios, dependiendo de varios factores como la tensión de alimentación y la temperatura.

En cuanto al entorno de desarrollo, Microchip ofrece MPLAB, que integra un compilador para ensamblador con la posibilidad de programar los binarios y de similar, no en tiempo real, el código escrito. Este entorno funciona bajo Windows y al tratarse de un freeware, dispone de funcionalidad reducida, en tanto en cuanto, para los microcontroladores de alta gama hay que adquirir una versión completa de pago.

Existe una alternativa libre llamada SDCC (Small Device C Compiler), que entre otros, soporta compilación de código C para PIC16 y PIC18.

En cuanto a dispositivos programadores o placas de entrenamiento, hay una gran disponibilidad en el mercado y los precios rondan los 100€-200€.

7.2.3.3.- FREESCALE

Freescale ofrece tres familias para microcontroladores de 8 bits, HC08, HCS08 y RS08. En cuanto a arquitectura se desmarca del resto usando un sistema Von Neumann, donde los datos y las instrucciones se ubican en el mismo sistema de memoria y comparten el mismo bus.

Así mismo usa un juego de instrucciones CISC amplio, que unido a los varios modos de direccionamiento permite realizar complejas operaciones.

De la misma forma que los anteriores, ofrece similares características en cuanto a periféricos en modelos equivalentes, aunque se puede destacar que se ofrecen modelos con características especiales, que han sido fabricados bajo demanda y se han incluido en el catálogo como productos habituales.

Los encapsulados que ofrecen no son demasiado adecuados para el desarrollo de prototipos cuando los recursos son limitados, ya que todos tienen formato SMD.

En cuanto a la herramienta de desarrollo, Freescale ofrece el conocido CodeWarrior en distintas versiones según el microcontrolador sobre el que se va a desarrollar y con distintas licencias. Tiene la ventaja de que

en una única aplicación integra todo lo necesario para el desarrollo y programación, además de ofrecer una utilidad de depuración muy potente.

Están disponibles versiones que funcionan sobre varios sistemas operativos.

7.2.3.4.- ELECCIÓN DE MICROCONTROLADOR

A priori elegir un microcontrolador que se ajuste a la perfección, de tal forma que se aprovechen al máximo los recursos que ofrece, para cubrir las necesidades de la aplicación es una tarea casi imposible, pues siempre nos vemos forzados a elegir un dispositivo que exceda en recursos, dando así un margen para cambios de última hora y posibles mejoras a futuro.

En el presente proyecto, el objetivo es construir un prototipo sencillo, por lo que se van a descartar todos aquellos dispositivos que no dispongan de encapsulados insertables, pues la complejidad de fabricación es innecesaria para aportar lo mismo que las versiones THD. Tras este razonamiento quedarían automáticamente descartados los microcontroladores de la familia Freescale.

En segundo lugar, debemos tener en cuenta las características de los dispositivos en cuanto a periféricos integrados, consumo, facilidad de aprendizaje...etc. En este ámbito las familias de Atmel y Microchip se encuentran bastante equilibradas, ya que ambas ofrecen un conjunto RISC de instrucciones, permiten varios modos de bajo consumo y los periféricos son similares. Se desmarca por delante Microchip en cuanto a la documentación disponible, que es muy abundante en contenido y ejemplos prácticos, mientras que Atmel delega más en foros y personas anónimas para promocionar y ofrecer soporte en webs como www.avrfreaks.net.

Otro punto a tener en cuenta y ciertamente importante, es el coste de desarrollo, es decir, que herramientas hardware y software son necesarias para realizar la aplicación completa. Tanto Microchip como Atmel ofrecen herramientas de programación propias de forma gratuita, MPLAB y AVR Studio cuyas características son similares: compilan asm y c, permiten programación y depurado.

No obstante ambas funcionan bajo sistemas Windows, lo que obligaría a tener una licencia de sistema operativo. Con el objetivo de minimizar los costes todavía más, se va a usar como estación de desarrollo un PC corriendo un sistema operativo Linux, de tal forma que las herramientas anteriores no nos sirven.

Microchip tiene algunas alternativas libres en internet pero no parecen realmente fiables, pues la mayoría no están actualizadas como YaPIDE, un entorno de desarrollo para PIC o GPUtils, un toolchain igualmente para PIC.

Sin embargo Atmel tiene un buen soporte de la comunidad de software libre y ello deriva en la existencia de un buen toolchain Avr-gcc-toolchain, que incluye herramientas de compilación (avr-gcc), programación (avr-dude) y librería standard para programación en lenguaje C (avr-libc), actualizado a los nuevos dispositivos y con abundante documentación.

Con todo esto, creemos que es motivo suficiente para elegir Atmel como proveedor de microcontroladores para el presente proyecto.

7.3. - SISTEMA LOCAL

En la figura 3, en la parte del sistema local se puede apreciar que existe una conexión directa con la red de motes o PAN, lo que permitiría el acceso físico a la red y por lo tanto la gestión a alto nivel del tráfico de datos que por ella circula.

Dentro del sistema local, vemos que se está ejecutando un programa que realiza principalmente tres funciones:

- Gestiona la red de motes o PAN

  Desde el punto de vista de la red PAN, se encarga de la gestión de la información que por ella circula, de tal forma que funcionalmente es el router de dicha red ya que toda la información pasa por este punto.

- Tiene la capacidad de acceso remoto al mecanismo de persistencia.

  Toda la información relevante que circula por la red es almacenada, pero este almacenamiento no puede ser local, pues violaría una de las premisas, la accesibilidad de la información desde el exterior. Por ello, cada uno de los sistemas locales, tiene acceso directo al mecanismo de persistencia, que puede y debe estar alojado en una ubicación segura y diferente a la del sistema local. Cabe destacar que es necesaria una conexión activa a internet, pues es la única forma de lograr este requisito a un coste razonable.

- Realiza la función de servidor punto a punto, con el servidor de internet.

  Este es uno de los puntos clave que permite que la red de motes sea controlada desde cualquier terminal, es decir, que los motes con funciones de control, sean accesibles desde el exterior. Analicemos esto con un poco más de detalle. Si por ejemplo, queremos que un usuario en una ubicación A, sea capaz de controlar el sistema de iluminación, situado en una ubicación B, tiene que ser capaz de acceder justo hasta el mote o motes que se encargan de ello. Este es un camino largo y con varios obstáculos. Uno de los principales requisitos para lograr esto es que tanto el usuario como nuestro sistema deben ambos, estar accesibles a internet. Esto es relativamente sencillo. Ahora bien, si pensáramos en acortar el camino, lo más directo sería que el cliente accediera directamente al sistema local, de tal forma que el sistema local sería realmente un servidor de internet en sí mismo. Esto no es aconsejable por los siguientes motivos entre otros:

    - El sistema local debería entonces disponer de una IP pública fija, lo que implica un coste adicional en el ISP.
    - El sistema hardware que alberga el sistema local, debe ser capaz de correr un servidor web.
    - Este servidor web, debe permitir características avanzadas como autenticación, para evitar que un usuario no autorizado acceda al sistema.

  Únicamente con los motivos anteriores, ya se hace desaconsejable esta solución, que a priori podría parecer interesante.

  Una verdadera solución sin coste añadido y que permite el acceso desde el exterior para realizar tareas de control, podría pasar por el uso de un servidor remoto. Un servidor remoto actúa como un servidor de internet en cuanto a la accesibilidad, pero no atiende a peticiones web tradicionales, sino que usa un mecanismo de comunicación basado en el paso de objetos desde el cliente al servidor y viceversa.

Con toda esta información ya se puede desarrollar una solución más concreta en cuanto al uso de tecnologías y la implementación software necesaria para lograr los objetivos funcionales.

7.3.1.- LA SOLUCIÓN ARQUITECTÓNICA Y SU IMPLEMENTACIÓN

La siguiente figura muestra un esquema de las capas que forman el sistema local desde el punto de vista de aplicación.

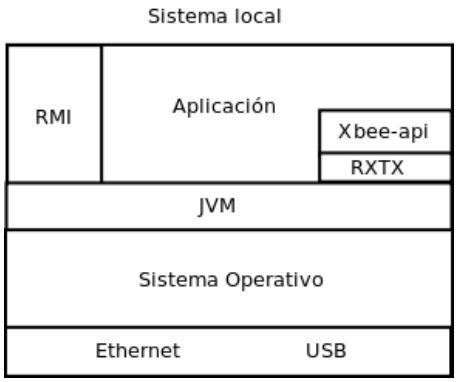


FIGURA 4: ARQUITECTURA DE CAPAS DE LA APLICACIÓN

Como se puede observar en la figura 4, en el nivel más bajo se encuentra una capa física formada por dispositivos hardware para comunicación. En el presente proyecto se van a emplear principalmente comunicación serie a través de USB y comunicación con la red Internet.

Gestionando el hardware, se encuentra el sistema operativo y las utilidades que implementa para los dispositivos de comunicación anteriormente mencionados. En el presente proyecto, se emplea un sistema operativo Linux.

Funcionando sobre la base del sistema operativo está la máquina virtual de Java (Java Virtual Machine), que hace uso de los recursos del sistema operativo y ejecuta los bytecodes de la aplicación final.

Como ya se ha mencionado, para poder programar el acceso a los puertos USB y con ello interactuar con el XBee, se emplea la librería XBee-api que está implementada sobre RXTX, que emplea las librerías nativas del sistema operativo desde Java, usando JNI (Java Native Interface). Los detalles de implementación se encuentran en el apartado 7.4.2.1.- Coordinador del presente documento y en el apartado 3.1.1.- Comunicación serie con Xbee, del documento Anexos.

Por otro lado, para hacer accesible el sistema local al exterior, se emplea una tecnología cliente-servidor intermedia dentro de la JVM, denominada RMI. El esquema de implementación del servidor remoto se encuentra en el apartado 3.1.3.- Arquitectura del servidor remoto, del documento Anexos.

Finalmente, la aplicación, que se está ejecutando sobre la JVM, hace uso de las librerías de alto nivel XBee-api y RMI desde un punto de vista estructural. Esta aplicación se encarga de gestionar las tramas de datos y el manteniemiento de la red, permitiendo el reinicio remoto de un mote concreto como la configuración automática de los mismos, a alto nivel. Los detalles de implementación se encuentran en los apartados 3.1.2.- Gestión de tramas de red, 3.1.4.- Autoconfiguración a alto nivel y 3.1.5.- Reasociación, en el documento Anexos.

7.4. - SISTEMA EMBEBIDO

Otro de los grandes bloques dentro del presente proyecto es el formado por la red de motes en donde podemos distinguir dos partes, una formada por el hardware, es decir, microcontroladores, transceptores 802.15.4, sensores...etc. y otra formada por el software de los motes que gestiona su funcionamiento.

Debe quedar claro que esto es una mera distinción organizativa pues cada parte se apoya y condiciona la otra comportándose como si de solo una se tratara. Es lo que se conoce como HW/SW Co-design y en cuyo planteamiento viene a decir que el desarrollo de un sistema electrónico programable, deben llevarse en paralelo tanto el diseño del hardware como la programación del software y esa filosofía es la que se ha pretendido seguir.

7.4.1.- SOFTWARE

Para el desarrollo del software de los motes, se ha empleado el lenguaje de programación C, que es el estándar para sistemas embebidos.

En general el programa de los motes tiene una estructura común, funcionalmente relacionada con el proceso de auto asociación y otra parte específica de tratamiento de sensores o de actuadores según el tipo de mote.

En los siguientes apartados se desarrolla tanto la parte común como las individuales de cada mote que forma el prototipo.

7.4.1.1.- LIBRERÍAS

Una de las primeras partes que es común al software de todos los motes son las librerías que se han generado y en concreto dos, una para encapsular el funcionamiento básico del XBee y otra para la generación de las tramas de datos.

Los detalles de la implementación de la librería para el transceptor XBee se encuentran en el apartado 3.2.1.- Librería XBee del documento Anexos.

Los detalles de la implementación de la librería para la generación de las tramas de datos se encuentran en el apartado 3.2.2.- Librería para gestión de información

7.4.1.2.- AUTOGESTIÓN

La otra parte común del software de los microcontroladores es la que atañe al proceso de asociación.

Esta parte implica la interacción con el transceptor XBee, pues es quién realmente se asocia a la red.

Esta interacción como ya se ha mencionado anteriormente, se realiza por medio de las USART que integran tanto XBee como el microcontrolador, convenientemente sincronizadas al mismo baudrate.

Lo que se pretende solucionar es cómo llevamos la auto asociación que la red de transceptores XBee realiza internamente y de forma transparente, es decir sin intervención externa, a los motes como entidades autónomas que integran además de los transceptores, microcontrolador, sensores..etc. Para ello, lo primero es entender como un coordinador XBee realiza el proceso de creación de una PAN y cómo se une un end device a ella.

Los dispositivos XBee disponen de un firmware actualizable, el cual define un conjunto de atributos configurables para customizar el funcionamiento del transceptor. Dichos atributos definen por ejemplo la velocidad de transmisión de la USART, el canal, el identificador de PAN, modos de bajo consumo o potencia de transmisión, entre otros.

En cuanto a la configuración de la PAN hay varios parámetros que definen el comportamiento en el momento de la asociación, tanto por el lado del coordinador como de los end devices, de tal forma que se permite mayor o menor flexibilidad.

Los detalles de la implementación de la librería para la generación de las tramas de datos se encuentran en el apartado 3.2.3.-Autogestión

7.4.1.3.- PROGRAMA PRINCIPAL

Toda aplicación para microcontroladores tiene un punto de inicio que suele ser el programa principal. Generalmente se trata de un bucle infinito, que se ejecuta siempre donde se encuentran las instrucciones o las llamadas a funciones necesarias.

Esa aproximación tradicional es válida siempre y cuando los requisitos de consumo no sean un problema, pero en nuestro caso, un punto importante es minimizar el consumo energético y para ello hay que hacer uso de modos de sueño del microcontrolador.

Por lo tanto, el modelo de bucle infinito tradicional no es válido y hay que adoptar una aproximación diferente. No obstante, el microcontrolador debe permanentemente estar ejecutando código o debe estar preparado para hacerlo, lo que implica que el programa principal no puede terminar.

La solución más sencilla es emplear los modos de sueño dentro de un bucle infinito, de tal forma que el microcontrolador está siempre en un modo de bajo consumo hasta que una interrupción lo despierta, ejecuta la rutina asociada y vuelve al modo de bajo consumo.

Esta aproximación es ideal para nuestra aplicación, pues generalmente podremos implementar todas las funcionalidades como interrupciones, minimizando el consumo con respecto a la aproximación tradicional.

Desde este punto de vista, se ha generado un programa principal estándar para los motes implementados y que puede ser empleado en muchas otras aplicaciones similares.

El fragmento de programa empleado se encuentra en el apartado 3.2.4.- Programa principal del documento Anexos.

7.4.1.4.- MOTE AMBIENTAL

Este mote se ha diseñado para albergar tres sensores analógicos que proporcionan medidas de las propiedades ambientales de temperatura, iluminación y humedad relativa.

El diseño hardware del mote base se explica en detalle en el apartado 7.4.2.2, así como los cálculos relacionados se reflejan en el apartado 2.2.- Mote base del documento Anexos.

En cuanto a la autogestión del mote, sigue las reglas explicadas 7.4.1.2.

Las mediciones realizadas se envían periódicamente al coordinador y se almacenan en la base de datos principal. De esta forma se disponen de los datos para poder realizar históricos y posibles análisis con el objetivo de mejorar el uso de los recursos energéticos del hogar, aunque esta segunda parte no se ha implementado.

Uno de los requisitos que se han impuesto es tratar de reducir el consumo de este tipo de motes y para ello se ha hecho uso de los modos de bajo consumo del microcontrolador principal, en la medida de lo posible, pues existen ciertas restricciones que hacen difícil compatibilizar un mínimo consumo con alta funcionalidad del mote.

En este sentido se plantean las siguientes situaciones:

- El mote realiza medidas periódicas, lo que da pie al uso de modos de sueño en el microcontrolador.
- Esta periodicidad se puede conseguir mediante eventos temporizados, generados por elementos externos o usando los contadores integrados.
- El mote puede recibir interrupciones desde el XBee para resetear el mote.

Sería deseable disponer al mote en el modo de más bajo consumo posible, pero esto discrepa con la capacidad del mismo de recibir información RF de forma asíncrona.

Lo que se ha intentado es llegar a un compromiso entre funcionalidad, coste económico y coste energético, y para ello:

- El modo de bajo consumo del microcontrolador debe permitir que despierte con interrupciones de USART.
- El evento periódico que lanza la tarea de medición se implementa usando los timer integrados en lugar de usar componentes externos, lo que implica menor coste económico y energético.
- Los bloques funcionales integrados en el microcontrolador que no se usan, se desactivan para minimizar el coste energético.

En primer lugar hay que establecer la periodicidad de la actividad del mote, es decir, cual es el ciclo de trabajo. En principio se ha planteado que el espacio temporal entre las medidas proporcionadas sea no inferior a 5 segundos, con propósitos de prueba, pues en un caso real podría ser muy superior dependiendo de los parámetros involucrados.

Como ya se ha mencionado, se van a usar los timer de los que dispone el microcontrolador para tratar de generar interrupciones temporizadas con un retraso no inferior a 5 segundos. Para ello es necesario lo siguiente: configuración de los timer para conseguir estos retrasos y generación del código de interrupción asociado.

Los cálculos relacionados con la temporización de la toma de medidas se encuentran detallados en el apartado 3.2.5.- Mote ambiental del documento Anexos.

La secuencia de la rutina es la siguiente:

- Inicia el conversor A/D
- Para los timer para que no generen ruido interno.
- Toma las muestras y calcula el promedio para cada sensor.
- Genera la trama XBee .
- Envía la trama
- Arranca de nuevo los timer
- Desactiva el conversor A/D para ahorrar energía.

7.4.1.5.- MOTE DE DETECCIÓN

Este mote está diseñado para detectar movimiento en un rango de unos 7 metros máximo y con un barrido de unos 30 º. Cada vez que un elemento se mueve alrededor del mote, el sensor genera un pulso que despierta al microcontrolador, que a su vez envía una trama al sistema local indicando que algo se está moviendo.

El diseño hardware del mote base se explica en detalle en el apartado 7.4.2.2, así como los cálculos relacionados se reflejan en el apartado 2.2.- Mote base del documento Anexos.

En cuanto a la autogestión del mote, sigue las reglas explicadas 7.4.1.2.

Hay un aspecto que se ha tenido que solucionar o al menos mitigar, que es el hecho de que el sensor proporciona un tren de pulsos repetitivo cuando detecta movimiento continuo, de tal forma que interrumpe de forma continua al microcontrolador, con lo que se pierde el bajo consumo y se satura la red con un tráfico innecesario. Como se ha mantenido el hardware que se diseñó en un primer momento sin tener en cuenta esta situación, se ha tenido que evitar este problema, introduciendo un retraso en el código de la interrupción, de tal forma que solo se puede interrumpir al micro cada 5 segundos.
De esta forma, no se satura la red aunque no se consigue reducir el consumo, pero no hay otra solución si no se cambia el diseño de la placa del sensor PIR.

La implementación de las interrupciones que genera el sensor se encuentra detallada en el apartado 3.2.6.- Mote de detección del documento Anexos.

7.4.1.6.- MOTE DIMMER

Este mote está diseñado para controlar la iluminación, en nuestro caso de una bombilla convencional, mediante la variación del ángulo de disparo de un tiristor. Para ello y como se explica en el punto 7.4.3.4, se ha diseñado una PCB que realiza dos funciones:

- Detección de paso por cero de la red, para sincronizar el disparo al ángulo deseado.
- Disparo del tiristor mediante triac optoacoplado, controlado por microcontrolador.

El diseño hardware del mote base se explica en detalle en el apartado 7.4.2.2, así como los cálculos relacionados se reflejan en el apartado 2.2.- Mote base del documento Anexos.

En cuanto a la autogestión del mote, sigue las reglas explicadas 7.4.1.2.

Antes de mostrar la implementación hay que ver cómo funciona el sistema en general, pues se mezclan dos actividades diferentes: detección de paso por cero y disparo del tiristor con un ángulo controlado.

Fundamentalmente el elemento más importante es la de la red monofásica que se utilizará para alimentar al dispositivo lumínico y como referencia de paso por cero.

Al pasar por cero la red, se inicia un mecanismo para contar tiempo o ángulo de la tensión de red, de tal forma que cuando alcanza el valor de disparo, se activa el elemento lumínico, hasta que la red vuelve a pasar de nuevo por cero y se repite el proceso. El control se realizará por lo tanto en ambos semiciclos de la tensión de red.

El siguiente diagrama de tiempos muestra cómo es el proceso:

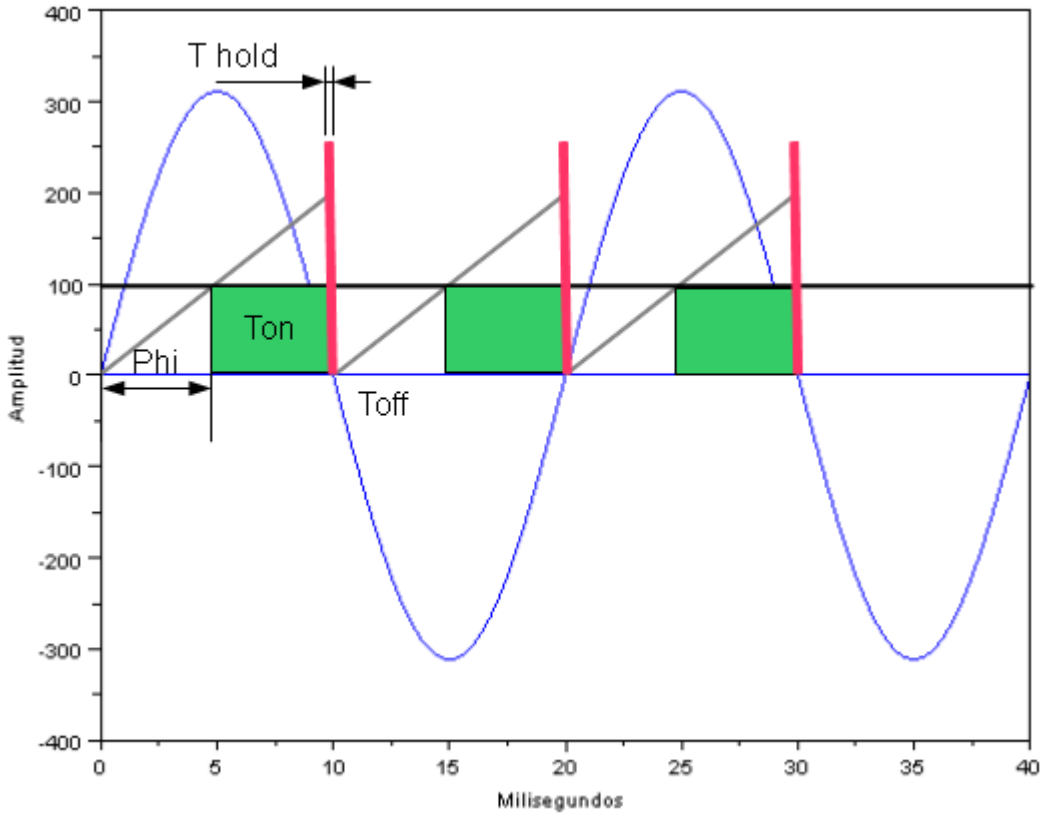


FIGURA 5

Se trata por lo tanto de una vez que la red pasa por cero, contar el tiempo necesario hasta el ángulo deseado y para ello hay que trasladar el problema resuelto en diseño analógico, a implementación en microcontrolador.

Por un lado tenemos un detector de paso por cero, que nos va a servir como referencia para contar tiempo. El dispositivo hardware proporciona una señal cuadrada que va en sincronismo con la red tal y como se indica en la figura.

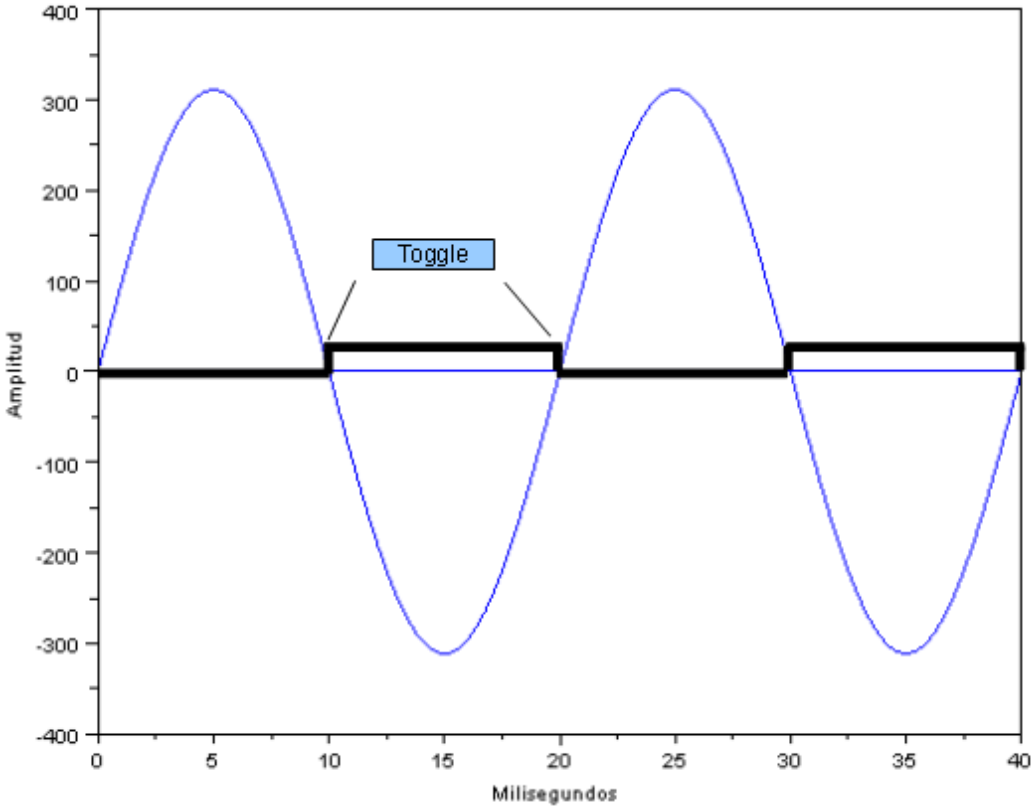


FIGURA 6

Por lo tanto en los flancos de subida y bajada de dicha señal, se producirán los sincronismos que pueden ser tratados como interrupciones. Esto tiene una ventaja, y es que se consigue una buena precisión en los disparos, pues una interrupción tarda aproximadamente 1 uS en entrar en la rutina asociada. La señal cuadrada se aplicará a un pin dedicado al tratamiento de interrupciones de tal forma que su prioridad sea elevada para no perder el sincronismo cuando acontecen otras interrupciones, retrasando las de menor prioridad. Como se ha analizado, esta interrupción se ha de generar tanto con flancos de subida como de bajada y por lo tanto se ha de configurar para disparar con el cambio (toggle).

La implementación de la rutina que gestiona los pasos por cero de la red y la rampa de contaje del ángulo de disparo, se encuentran en el apartado 3.2.7.- Mote dimmer, del documento Anexos.

7.4.2. – HARDWARE

7.4.2.1.- COORDINADOR

El elemento principal de la red 802.15.4 es el transceptor que actúa como coordinador, siendo el dispositivo encargado de crear la PAN, seleccionando un PAN ID y un canal libres. Como ya se ha mencionado anteriormente, éste debe ir conectado directamente al sistema local de tal forma que exista una vía de comunicación serie entre ambos.

En un compromiso entre coste, sencillez y fiabilidad, se ha empleado el mecanismo más sencillo de conexión directa entre un dispositivo USART y un puerto USB, un cable USB de cinco terminales y un adaptador Serie-USB. Se ha decidido emplear un conector Mini-USB A en la PCB, para minimizar el volumen.

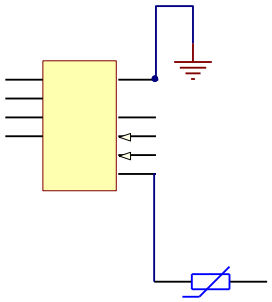


FIGURA 7

Inicialmente, la corriente máxima que puede suministrar un puerto USB está limitada sobre los 500mA, aunque esto no es un problema, pues el consumo máximo del transceptor XBee que hará de coordinador está en torno a 50 mA. No obstante a esto habrá que sumar el consumo correspondiente al propio adaptador Serie-USB, así como el de los dispositivos indicadores instalados.

Como dispositivo adaptador Serie-USB, se ha empleado el conocido FT232RL fabricado por FTDI Chip, que proporciona una interfaz sencilla y drivers para los sistemas operativos más relevantes.

Así mismo dispone de una versión con encapsulado SSOP de 28 pines, lo que facilita el proceso de soldadura manual.

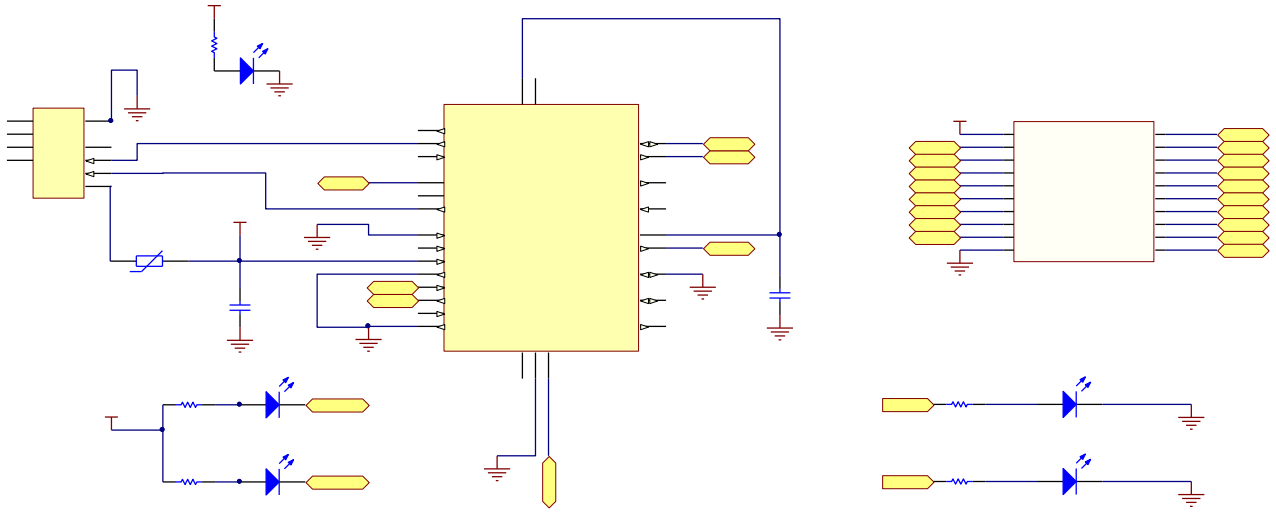


FIGURA 8

Cabe destacar que en esta versión de la PCB para el coordinador, se han añadido indicadores luminosos tal y como se puede apreciar en el fragmento de esquema anterior. Así pues, encontramos diodos led para indicación de alimentación, para transmisión y recepción, indicación de RSSI e indicador de asociación.

Información ampliada y cálculos, se encuentra en el apartado 2.1.2.- Comunicación, del documento Anexos.

La alimentación del FT232RL es proporcionada por el propio puerto USB a través de uno de los cinco terminales del cable y para la alimentación del transceptor XBee se ha empleado un regulador lineal fijo, con una salida de 3.3 voltios. El dispositivo empleado es el TPS79333 de Texas instruments, que se trata de un regulador lineal de muy bajo drop-out(unos 112mV a 200mA), una corriente de polarización de unos 120 uA y una salida en corriente de 200mA máximo, más que suficiente para alimentar los XBee de la serie PRO.

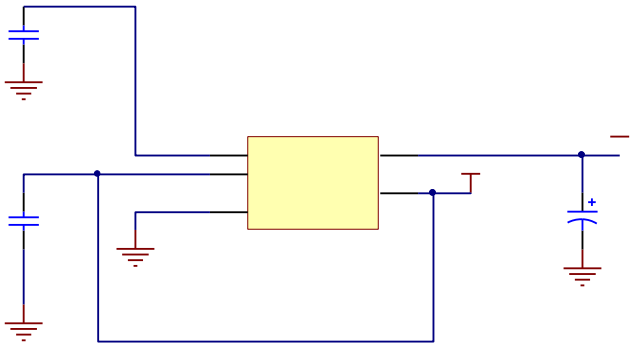


FIGURA 9

Información ampliada y cálculos, se encuentra en el apartado 2.1.1.- Alimentación, del documento Anexos.

7.4.2.2. - MOTE BASE

El mote base es la estructura electrónica que da el soporte necesario en cuanto a procesamiento y comunicación y además es la estructura mecánica que permite la ubicación de las PCB funcionales. Principalmente consta de los siguientes bloques:

- Bloque de potencia: formado por los sistemas de alimentación y por el sistema de carga de las mismas.
- Bloque de procesamiento: formado por el microcontrolador y los elementos pasivos necesarios para su funcionamiento.
- Bloque de comunicación wireless: formado por el transceptor XBee e indicadores funcionales.
- Bloque de comunicación USB: formado por un adaptador Serie-USB, empleado para la programación del microcontrolador.

Se ha desarrollado una nueva versión del mote base que mejora el diseño anterior, ofreciendo mejores características y funcionalidades. Los principales puntos de mejora son los siguientes:

- Se ha sustituido el regulador lineal que alimentaba a todo el sistema por un sepic/flyback con una eficiencia superior al 90% y que proporciona 3.3V configurados mediante resistencias y 1100 mA.
- Se ha añadido indicador de asociación para el transceptor XBee, que indica si el mote se ha asociado correctamente a la red.
- Se ha añadido un bloque de cuatro interruptores que permiten configurar el transceptor XBee a través del microcontrolador.
- Se ha incorporado un mecanismo para el reset automático del mote, que lo habilita para reprogramar el firmware inalámbricamente, usando el transceptor XBee.
- Se ha añadido un interruptor para desconectar la batería sin tener que desenchufarla del mote.

BLOQUE DE POTENCIA

El sistema de alimentación del mote está formado por un SEPIC/Flyback, el TPS61131 de Texas Instruments. Este dispositivo proporciona dos salidas para alimentación, una a partir de un Buck/Boost y otra a partir de un regulador líneas LDO. Como se ha indicado una de las mejoras ha consistido en mejorar la eficiencia del sistema de alimentación y por ello se ha empleado la configuración Buck/Boost que indica la figura.

La tensión necesaria para el sistema son 3.3V, que van tanto al microcontrolador y al transceptor XBee como a los sensores y actuadores que van instalados en las PCB funcionales. Para ello, se ha configurado al TPS61131 con el par de resistencias R8 y R9 con los valores indicados en el esquema, obteniendo una salida en Vout de 3.3V.

Información ampliada y cálculos, se encuentra en el apartado 2.2.1.- Alimentación, del documento Anexos.

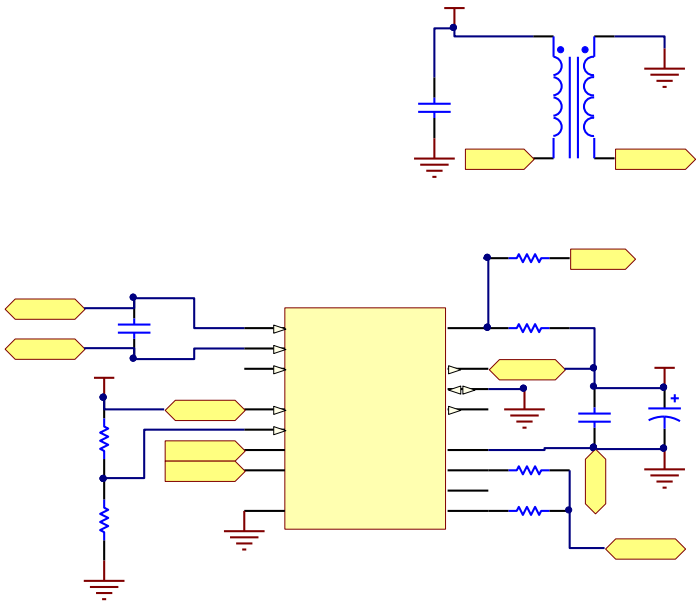


FIGURA 10

Por otro lado está el sistema encargado de cargar las baterías, formado por el MAX 1555 de Maxim, que se trata de un cargador de baterías de ión-litio (L+) de una celda, que puede realizar el proceso de carga desde un terminal USB o una fuente de alimentación. En la implementación se ha usado un conector de barril para una fuente de alimentación externa. Incorpora un indicador de carga, que se activa cuando la batería está en proceso de carga, indicándose el hecho por medio de un diodo led activo.

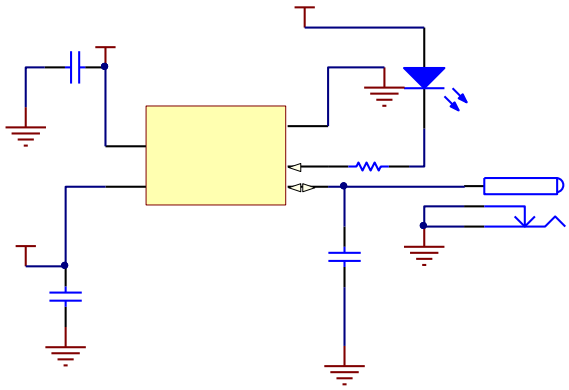


FIGURA 11

BLOQUE DE PROCESAMIENTO

Este bloque está formado por el microcontrolador y los elementos pasivos necesarios para el funcionamiento básico.

Los microcontroladores AVR tienen la ventaja de que apenas necesitan componentes externos para funcionar y en este caso se han empleado un cristal de cuarzo que resuena 8Mhz, con dos condensadores para el circuito interno del oscilador, las conexiones de alimentación y masa.

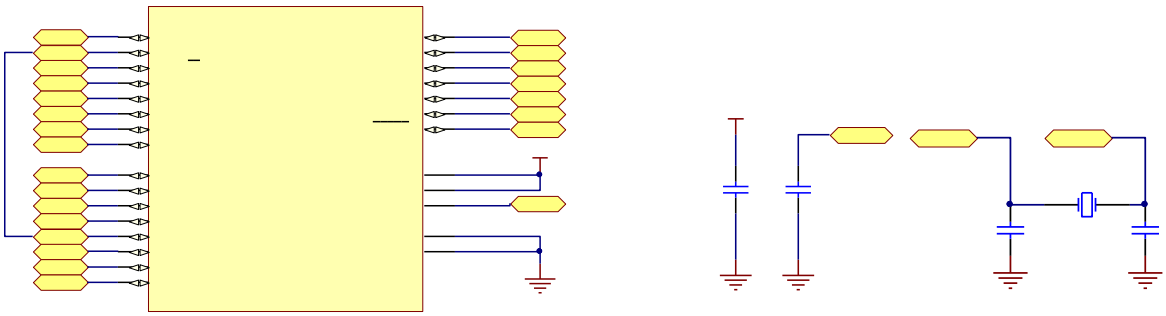


FIGURA 12

Adicionalmente y siendo necesario, se ha incluido un sistema de reset manual, formado por el típico circuito de reset con pulsador, resistencia y condensador.

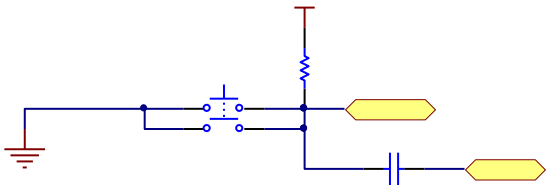


FIGURA 13

Para contemplar la funcionalidad de programación remota del firmware del microcontrolador, es necesario un mecanismo para que el microcontrolador pueda auto resetearse, activando alguno de sus terminales. Para esto se ha empleado un switch CMOS, el ADG719 de Analog Devices, que funcionalmente es un multiplexor de dos entradas y una salida controlado digitalmente. Es usado para seleccionar la fuente de reset, es decir, por defecto o en condiciones normales, el reset viene dado por interruptor manual pero cuando cambia el estado del terminal de control a nivel lógico 1, el reset está conectado a masa, por lo que se resetea automáticamente. Como terminal de control se ha empleado el terminal ICP del microcontrolador.

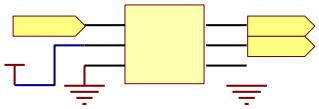


FIGURA 14

BLOQUE DE COMUNICACIÓN 802.15.4

Este bloque lo conforma el transceptor XBee con la configuración que muestra la figura.

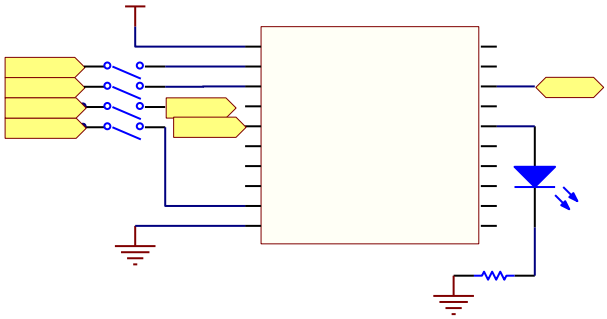


FIGURA 15

Se ha incorporado un led indicador de asociación y un bloque de cuatro interruptores, de los que dos estarán siempre en una posición de paso, RX y TX, pues son los necesarios para realizar la transmisión de datos, los otros dos irán in función de los requisitos del mote concreto. Concretamente se han configurado los terminales SLEEP y SLEEP/RQ para controlar manualmente el estado de activación del transceptor XBee. Habitualmente este dispositivo funcionará en un modo de sueño cíclico, en el que cada cierto tiempo configurado en el firmware, se despertará para ver si tiene información que enviar o recibir, volviendo al estado de sueño posteriormente. Pero puede ocurrir el caso en el que el mote sea totalmente autónomo, es decir, que solo envíe información, no la reciba. En ese caso, se puede realizar un control del consumo del XBee de forma manual, configurando los terminales INT0 y SCK del microcontrolador por medio del software, para hacer que prácticamente consuma la corriente de polarización.

Información ampliada y cálculos, se encuentra en el apartado 2.1.1.- Alimentación, del documento Anexos.

BLOQUE DE COMUNICACIÓN USB

Al igual que el coordinador, el mote base dispone de conectividad USB por medio del adaptador Serie-USB FTDI232RL.

En este caso se puede usar para realizar la programación del mote a modo de herramienta de desarrollo, ya que el microcontrolador incorpora un bootloader que hace innecesario el uso de un programador dedicado.

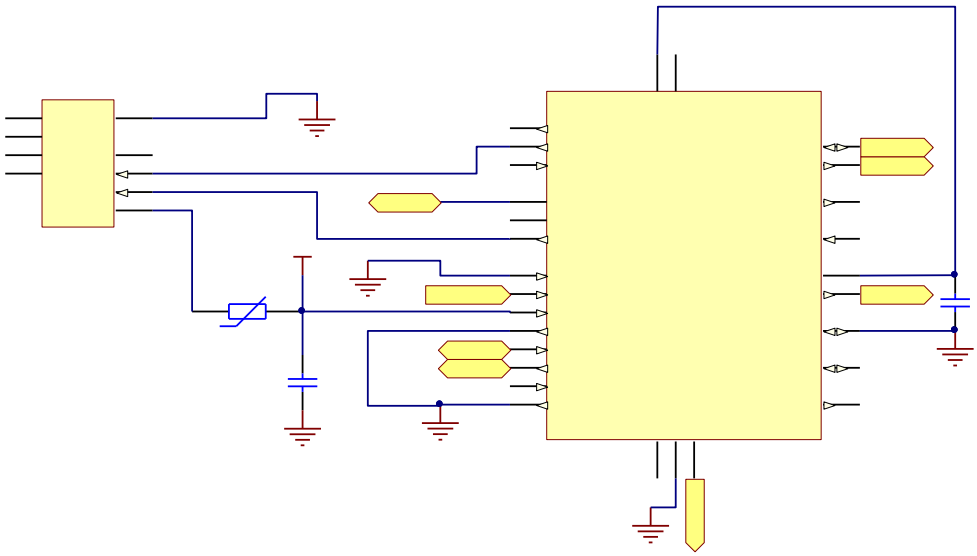


FIGURA 16

Información ampliada y cálculos, se encuentra en el apartado 2.1.1.- Alimentación, del documento Anexos

7.5. - SISTEMA SERVIDOR

Hasta ahora ya tenemos solucionados los problemas de la red de motes y de los sistemas locales pero queda el tercer gran bloque, que permite almacenar la información importante y hacerla accesible a los usuarios. Este bloque se denominará sistema servidor e integra dos elementos, un servidor de aplicaciones y un servidor de base de datos. Por lo general y como norma, estos servidores nunca se van a encontrar físicamente en una instalación local por motivos de seguridad, mantenimiento y accesibilidad. Este sistema en conjunto permite que los usuarios de los sistemas locales instalados, puedan acceder a información sobre el estado de dichos sistemas así como realizar sencillas tareas de control desde una conexión a internet.

7.5.1.- SERVIDOR DE APLICACIONES

Uno de los elementos principales del sistema servidor es el servidor de aplicaciones.

Un servidor de aplicaciones permite la ejecución de un programa específicamente diseñado para trabajar en entorno de internet y por lo tanto accesible desde un navegador web. La función principal de un servidor de aplicaciones es la de recibir peticiones http en los formatos que establece la norma http://www.w3.org/Protocols/rfc2616/rfc2616.html y responder al solicitante con la información requerida lo más rápidamente posible.

Esto que parece sencillo se convierte en un verdadero problema cuando tenemos miles de usuarios haciendo peticiones cada pocos milisegundos y se ha de responder sin perder efectividad. Afortunadamente, en la actualidad existen numerosos servidores de aplicación en los que los aspectos de concurrencia están muy depurados y el rendimiento aumenta conforme aumentan las peticiones.

Una de las tareas es pues elegir un servidor de aplicaciones que encaje dentro de los objetivos del presente proyecto en cuanto a coste, filosofía open source..etc.

El estándar open source para servidores web es Apache Server, de la Apache Software Foundation, que tiene una larga tradición como servidor HTTP desde pequeños sistemas hasta grandes servidores. Tiene una arquitectura basada en plugins, lo que lo hace muy modular, activando solo las que sean necesarias y así por ejemplo dispone de plugins para SSL,autenticación, cachè, filtros o LDAP.

Ahora bien un servidor web está pensado para realizar las tareas de gestión a bajo nivel de las peticiones HTTP y las respuestas asociadas, no para formar un sistema completo por sí mismo. Teniendo en cuenta que la aplicación del lado del servidor va a ser desarrollada usando la tecnología JAVA para aplicaciones web, necesitamos algo más.

La especificación JAVA para la web está formada por dos elementos principales, Servlets y JSP. Un servlet puede ser entendido como un elemento de servidor que tiene asociada una dirección web y que está encargado de responder las peticiones HTTP sobre esa dirección.

Teniendo en cuenta que un servidor puede recibir miles de peticiones simultáneas sobre una misma dirección, la especificación indica que existirá una instancia de un servlet concreto por cada petición, eliminando los problemas derivados de la concurrencia.

Por otro lado tenemos la tecnología JSP, que básicamente se emplea para la creación de páginas web dinámicas. Una página HTML dinámica es creada a partir de cierta información generada en el momento de recibir la petición, como puede ser un listado de motes en nuestro caso.

Dispone de elementos muy útiles para usar objetos Java que son pasados a las JSP como iteradores de listas, utilidades de formateo de datos, bloques de control…etc.

Con esto ya tenemos la infraestructura necesaria para generar el contenido dinámicamente, pero queda resolver cómo se trasladan las peticiones HTTP del servidor de aplicaciones hasta un Servlet en concreto. Para esto es necesario y así lo indica la especificación web de JAVA, lo que se conoce como un contenedor de Servlets, es decir, una aplicación que está por encima del servidor HTTP y que es capaz de ejecutar los Servlet cada vez que procesa una petición.

Nuevamente la opción más robusta, open source y de los desarrolladores de Apache, se denomina Apache Tomcat, que es el contenedor de Servlet por excelencia. Existen varias aplicaciones privativas similares a Tomcat pero su coste lo hace inviable para nuestro propósito.

7.5.2.- SERVIDOR DE BASE DE DATOS

La otra pieza fundamental para que funcione el conjunto, es la que da soporte al almacenamiento de información. Sin un mecanismo para almacenar datos como, qué coordinador está conectado en cada sistema local, que motes pertenecen a una red y qué roles juega cada mote en función de los dispositivos que tiene instalados, sencillamente nada podría funcionar.

Por lo tanto se trata de un sistema vital para el conjunto y que además debe ser eficiente y robusto para no provocar cuellos de botella en el almacenamiento de los datos.

Existe un conjunto amplio de servidores de base de datos y los podemos catalogar en dos tipos:

- Relacionales: La información se estructura como relaciones entre entidades.
- Orientados a objetos: La información se guarda en forma de objetos, tal y como se representan en un lenguaje orientado a objetos.

Tradicionalmente, los sistemas más usados son las bases de datos relacionales por la flexibilidad y las herramientas que ofrecen. Es muy importante que ofrezca un bueno conjunto de herramientas, sobre todo las relacionadas con la seguridad y la recuperación de datos ante fallos y eso es a día de hoy una lacra de las bases de datos orientadas a objetos.

Las bases de datos orientadas a objetos tienen algunas ventajas, precisamente para los programadores orientados a objetos:

- No es necesaria una capa objeto-relacional para realizar la persistencia.
- Pueden considerarse más rápidas pues las relaciones son punteros, no filas y columnas.

Pero los motivos anteriores no son suficientes para plantearse el uso de una base de datos no relacional, así que siguiente una aproximación conservadora, se empleará una base de datos relacional.

Siguiendo la filosofía de minimizar el coste maximizando las prestaciones de las aplicaciones elegidas, a la hora de elegir una base de datos relacional, open source, con un gran rendimiento y unas buenas herramientas, no hay duda que MySQL es el estándar que siguen las demás.

MySQL es una base de datos open source, que hasta hace unos meses pertenecía a MySQL AB y ha sido comprada recientemente por Oracle. Su rendimiento es espectacular siempre y cuando nos mantengamos por debajo de la media docena de gigabytes de datos, cuando el rendimiento empieza a caer. Dispone de herramientas de administración y para desarrolladores en las que se incluyen, editor gráfico de tablas y administración, herramientas de backup y replicación, balanceo de carga...etc.

Existen otras bases de datos open source como PostgreSQL, que poco a poco va ganando peso de mercado o la variante MariaDB, que es una rama de MySQL, que emplea un nuevo motor de tablas.

7.5.3.- APLICACIÓN

Como conclusión a todo lo anterior y sirviendo de base al presente apartado, se va a desarrollar la aplicación del lado del servidor en lenguaje JAVA, usando la especificación web de Servlets y JSP, que va a estar corriendo sobre el contenedor de Servlets Apache Tomcat. Así mismo, el motor de datos elegido el MySQL, de tal forma que la información se persistirá en forma de tablas y relaciones entre ellas.

La aplicación queda estructurada principalmente en cuatro partes: modelo de dominio, mecanismos de persistencia, gestión de solicitudes web y presentación.

Se ha seguido una arquitectura de tres capas MVC (Model View Controller), típica de las aplicaciones web empresariales, de tal forma que se reparten las responsabilidades de forma coherente dentro de la aplicación.

7.5.3.1.- REQUISITOS

Lo que se pretende con el desarrollo de una aplicación web son dos objetivos, por un lado ofrecer un sistema a través del cual, los usuarios de los sistemas domóticos, dispongan de un lugar en el que puedan comprobar el estado de su vivienda e incluso puedan realizar tareas de control sencillas. Por otro lado se pretende crear una infraestructura empresarial para gestionar toda la información relativa a instalaciones domóticas.

Ambos objetivos son ambiciosos y tienen características bien diferenciadas en la práctica.

Para permitir que un usuario acceda a un servicio, previamente hay que tenerlo registrado y proveerle de un mecanismo de autenticación. Además, para que pueda realizar tareas de control de dispositivos remotamente, se han de crear las interfaces web adecuadas y un mecanismo para hacer llegar las órdenes hasta el sistema local.

El acceso que tiene un usuario de una instalación domótica es limitado en el sentido en que no puede por ejemplo, añadir un nuevo mote o modificar uno existente o no puede tampoco cambiar las propiedades de la red, tareas que se ceden a un nivel superior de privilegios.

En este sentido se puede enlazar con el segundo objetivo, el de crear una infraestructura empresarial que permita gestionar todo lo relativo a las instalaciones domóticas, usuarios, dispositivos...etc.

Por lo tanto parece claro que desde una perspectiva de privilegios de acceso, se ha de crear una jerarquía para los distintos perfiles de usuarios que usarán la aplicación.

7.5.3.2.- MODELO ORIENTADO A OBJETOS

En términos informáticos se denomina modelo a las estructuras de datos que son necesarias desde un punto de vista lógico, para modelar el problema que se pretende resolver. Desde la perspectiva de orientación a objetos, existe una relación directa entre las entidades físicas y los objetos. También los conceptos abstractos tienen un mapeo a objetos, aunque quizá es menos trivial.

En el presente proyecto aparecen entidades como mote o sensor que han de ser modeladas como clases y que además tienen una entidad relacional a nivel de base de datos. Además se han de contemplar las relaciones entre las entidades, de tal forma que esas relaciones deben seguir unas reglas de normalización cuando se mapean en base de datos, en lo que se conoce como Forma Normal de Boyce Codd, en teoría de bases de datos.

En este apartado vamos a dar algunas pinceladas importantes sobre el diseño del modelo y la interacción en base de datos que tiene, considerando los elementos principales que maneja la aplicación.

Según todo lo explicado hasta este punto, se pueden extraer algunos conceptos que se modelan como entidades:

- **Usuario**: Persona que accede a la aplicación usando ciertas credenciales, tiene acceso a sus instalaciones domóticas...etc.
- **Privilegios de usuario**: Hasta dónde puede llegar el usuario en el uso de la aplicación, puede realizar tareas administrativas o únicamente de consulta.
- **Red**: Conjunto de elementos que forman un sistema de comunicación distribuido. Se identifica por una clave única, usa un canal y transmite a una determinada velocidad.
- **Mote**: Elemento de red que dispone de una dirección de red, un número de serie y una configuración de sensores y actuadores.
- **Sensor**: Dispositivo que es capaz de medir una magnitud física y dar como respuesta una medida continua o discreta, en forma eléctrica. Tiene unos márgenes de sensado y una característica.
- **Actuador**: Dispositivo que es capaz de controlar o ser controlado por medio de una acción eléctrica, neumática...etc. Tiene unos márgenes de actuación y una característica.

Adicionalmente se han de considerar las acciones o hechos relevantes como eventos, que se usarán para tener un registro de lo que ocurre y las situaciones anómalas como alertas, que servirán para crear un sistema de alertas para cada mote que disponga de sensores. La parte de alertas merece una consideración especial y se estudiará más adelante.

Así pues, consideraremos dos entidades más:

- **Evento**: Hecho relevante en el sistema identificado por una acción y un instante de tiempo.
- **Alarma**: Estado del sistema al que se llega tras ocurrir un evento anómalo.

Estas son las entidades más relevantes aunque no las únicas, pues aparecerán otras de menor calado pero que también son importantes en la medida en que dan soporte a las anteriores

Los modelos de datos empleados aparecen en el apartado 3.3.1.- Modelado orientado a objetos, en el documento Anexos.

7.5.3.3- MODELO RELACIONAL

El modelo relacional pretende llevar al nivel de base de datos el modelo orientado a objetos que se ha desarrollado en el anterior punto. Por lo tanto, los atributos de las clases se mapearán como columnas de las tablas y las relaciones se identificarán como claves foráneas.

El diagrama entidad-relación que se desprende del mapeo objeto-relacional, así como las sentencias en lenguaje SQL que generan la estructura de la base de datos, se encuentra en el apartado 3.3.2.- Modelo relacional

7.5.3.4.- CONTROLADOR

Dentro del modelo MVC, la parte que se encarga de procesar las peticiones HTTP, ejecutar los programas necesarios y generar las respuestas, es el controlador.

Como ya se ha mencionado, la capa del controlador está formada por Servlets que gestionan las solicitudes, usando los objetos de modelo que se han presentado en el apartado anterior y lógica de negocio.

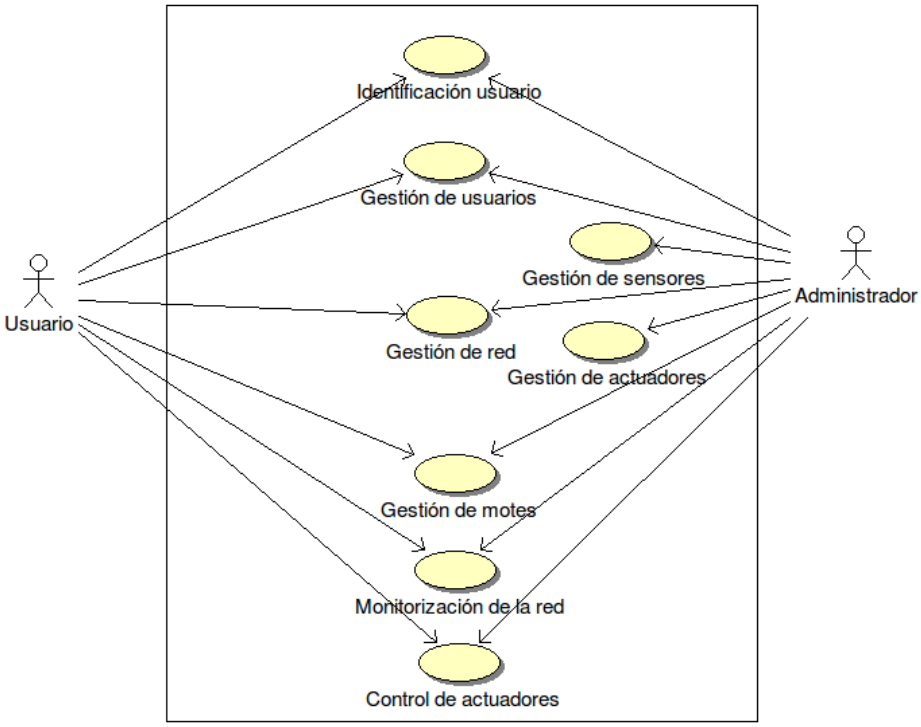
En esta sección no se va a presentar el código salvo en los casos que sea necesario, ya que representa una implementación concreta entre muchas posibles, pero si se van a indicar, en forma de diagramas, todas las posibilidades que ofrece la aplicación.

Los diagramas que a continuación se presentan, se denominan en la jerga de UML como casos de uso, es decir, qué acciones están permitidas y quién las puede realizar, así como la relación entre los casos de uso.

En principio se han planteado tres roles o perfiles de acceso con diferentes niveles de acceso a la aplicación:

- Usuario
  Dispone de un acceso limitado a la aplicación, de tal forma que no puede realizar tareas de configuración que afecten de forma directa al funcionamiento de una red.
- Gestor de red
  El nivel de acceso permite a este rol efectuar todas las tareas de gestión de una red, incluso las que afectan al funcionamiento.
- Administrador
  Dispone de acceso a las funcionalidades del gestor de red pero sobre cualquier red que gestione el sistema. Además permite la gestión de elementos de configuración interna de la aplicación.

La siguiente figura presenta los principales casos de uso de la aplicación.



1. Identificación de usuario

   Todos los usuarios que acceden a la aplicación disponen de credenciales para autenticarse. Para ello, se ha configurado un mecanismo de autenticación basado en el contenedor de servlets Tomcat denominado JDBCRealm.

   Con este sistema, se configura una base de datos y las tablas que contienen la información de autenticación y automáticamente, efectúa la operación una vez que el usuario ha introducido sus credenciales.

   La configuración se almacena en un archivo XML que es cargado al iniciar el servidor Tomcat.

CONTEXT.XML

```
<Context debug="0" reloadable="true">

<Realm className="es.automation.home.util.HomeRealm"
        driverName="com.mysql.jdbc.Driver"
        connectionURL="jdbc:mysql://localhost:3306/@@DB_NAME@@?
                user=@@DB_USER@@&amp;password=@@DB_PASSWORD@@"
        userTable="users" userNameCol="user_name" userCredCol="password"
        userRoleTable="user_roles" roleNameCol="role_id" userInRoleNameCol="user_id"/>


        <Resource name="jdbc/homeAutomation"
                auth="Container"
                type="javax.sql.DataSource"
                username="@@DB_USER@@"
                password="@@DB_PASSWORD@@"
                driverClassName="com.mysql.jdbc.Driver"
                url="jdbc:mysql://localhost:3306/@@DB_NAME@@"
                maxActive="100"
                maxIdle="30"/>

        <Environment name="databaseVendor" value="mysql" type="java.lang.String" />

</Context>
```

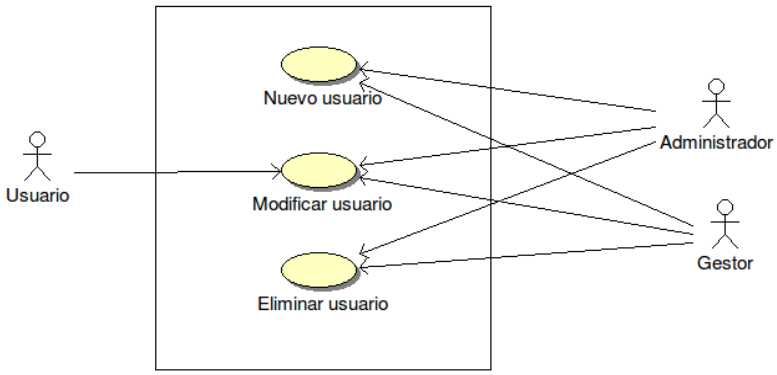
Adicionalmente hay que configurar la aplicación para que emplee este mecanismo que ofrece Tomcat y para ello, en el descriptor de despliegue hay que añadir lo siguiente:

```
<!-- Configuración de la autenticación-->
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/WEB-INF/jsp/login.jsp</form-login-page>
    <form-error-page>/WEB-INF/jsp/login.jsp?error=true</form-error-page>
  </form-login-config>
</login-config>
```

Básicamente indica que la autenticación se realiza mediente formulario, en el que el usuario introduce sus credenciales.

2. Gestión de usuarios

Este caso de uso se puede dividir en varios casos como indica el diagrama.



La creación de nuevos usuarios está permitida únicamente al Administrador y al Gestor de red, de tal forma que el Administrador puede crear usuarios en cualquier red mientras que el Gestor solo puede crear usuarios en las redes que tiene asignadas.

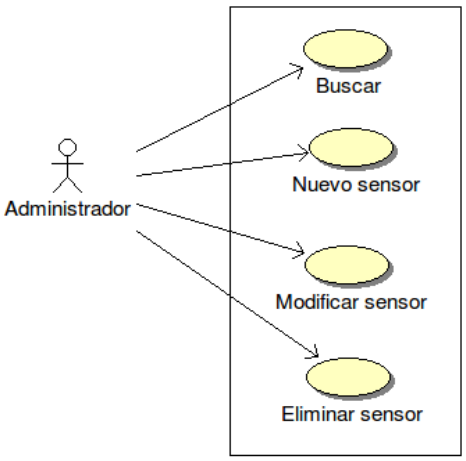
Así mismo la modificación de usuarios se ha de entender como la modificación de los datos de los usuarios de una red, o bien como la modificación de los propios datos del usuario que accede. En este sentido, un Usuario únicamente puede modificar sus datos y tanto el Administrador como el Gestor de red, pueden modificar sus propios datos y los de los usuarios que tienen en sus zonas de control.

Finalmente la eliminación de usuarios únicamente se permite al Administrador y al Gestor de red bajo las mismas condiciones anteriores.

3. Gestión de sensores

Este caso de uso está reservado al Administrador, pues trata de gestionar los sensores que admite el sistema, de los que depende la instalación física y el buen funcionamiento del sistema de autoconfiguración.

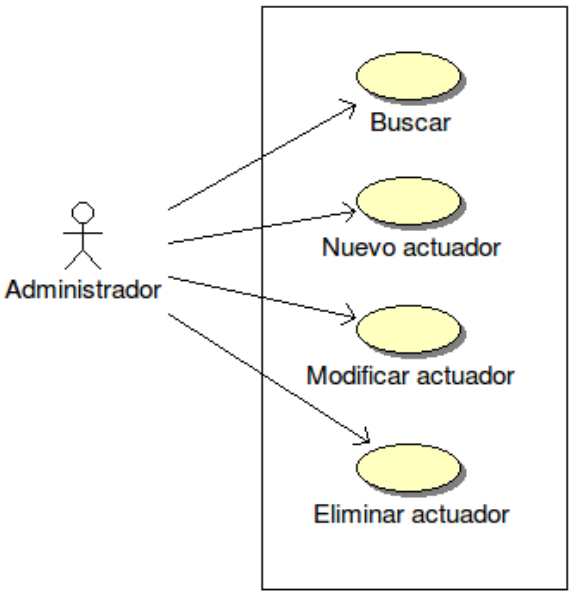
Podemos dividir este caso de uso tal y como aparece en el diagrama.



4. Gestión de actuadores

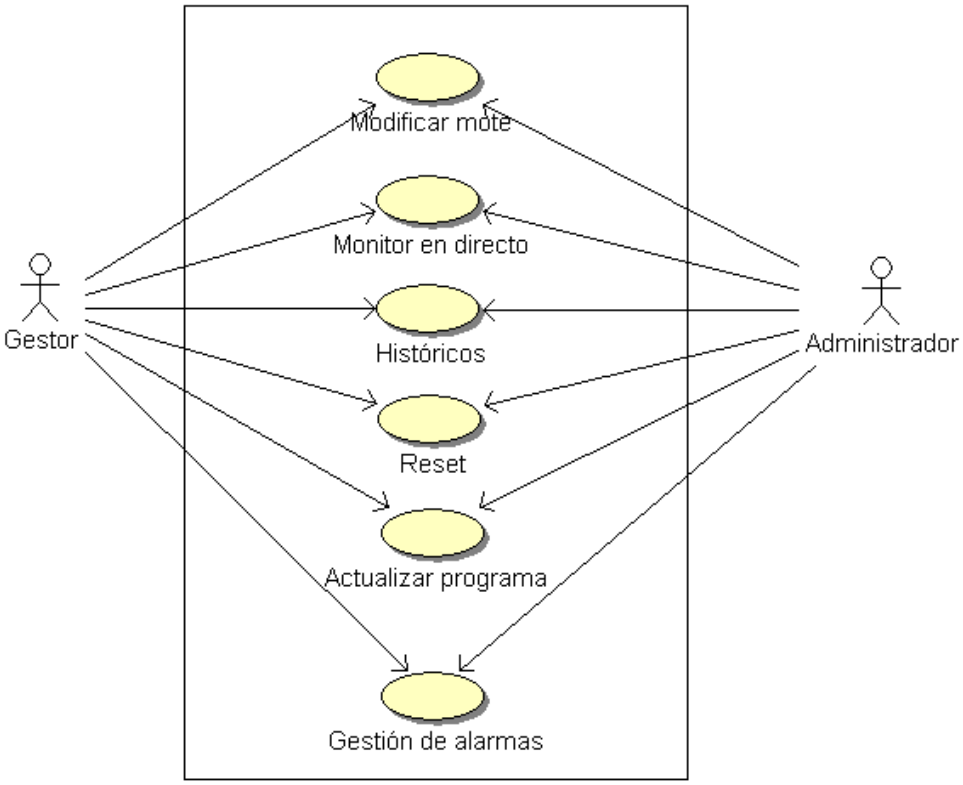
   Este caso de uso está reservado al Administrador, pues trata de gestionar los actuadores que admite el sistema, de los que depende la instalación física y el buen funcionamiento del sistema de autoconfiguración.

   Podemos dividir este caso de uso tal y como aparece en el diagrama.



5. Gestión de motes

El apartado de gestión de los motes está reservado para el Administrador y el Gestor de red, pues permite realizar operaciones relevantes sobre la red instalada.



Por un lado permite la edición de algunos de los datos de los motes, concretamente los que no afectan a su funcionamiento y configuración, pues esto se realiza de forma automática.

Se dispone de una opción para resetear la conectividad del mote, sin tener que efectuar un reset manual.

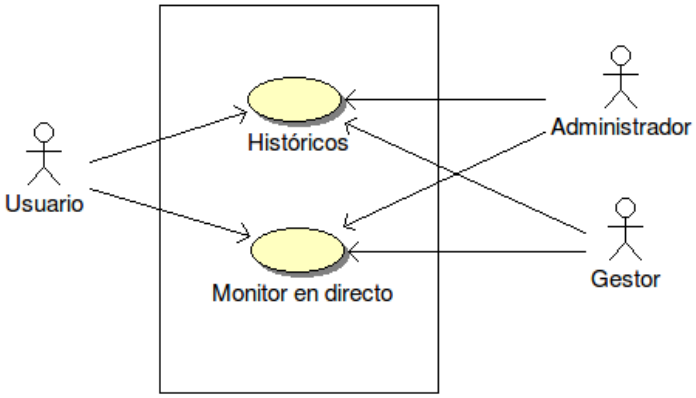
También se permite actualizar o modificar el programa que ejecuta el microcontrolador remotamente, sin tener que moverlo de la ubicación final.

La gestión de alarmas, permite crear alarmas para los sensores instalados en los motes que disponen de dichos elementos.

6. Monitorización de la red

La monitorización de la red es un caso de uso permitido a todos los usuarios de la aplicación, pues no afecta al funcionamiento sino que es una mera consulta del estado de la red.

Se puede dividir en dos partes bien diferenciadas.



Por un lado está el caso de uso Históricos, que permite obtener información de los sensores agrupada por espacios temporales, ofreciendo gráficos en formato histograma.

Por otro lado está el caso de uso del Monitor en directo, de tal forma que se obtiene información de los sensores en el momento, cada breves intervalos de tiempo.

Ambos aspectos se detallan en el apartado 7.5.4.5, monitorización de sensores.

7. Control de actuadores

El case de uso Control de actuadores, permite a los usuarios de la aplicación realizar sencillas tareas de control, de acuerdo con la configuración de los motes de la red. Está habilitada para todos los perfiles de usuarios de aplicación.

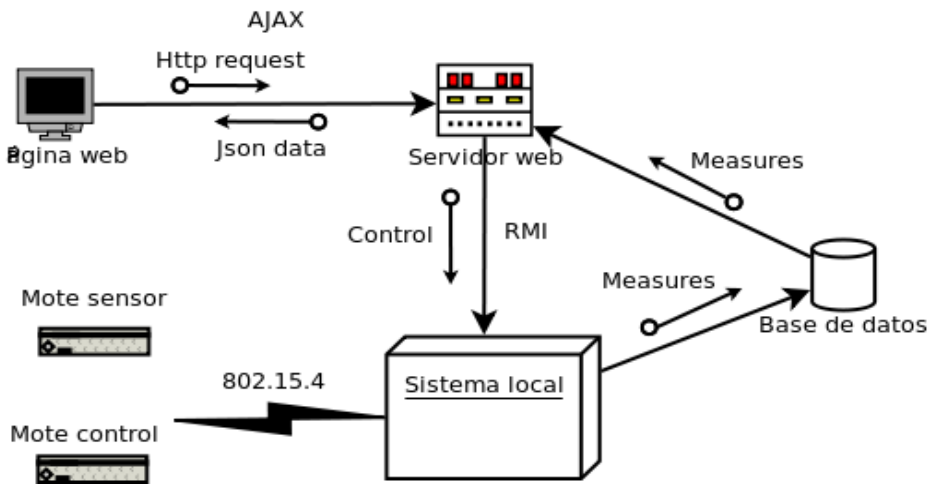
Este aspecto se detalla en el apartado 7.5.4.5, control de actuadores.

7.5.3.5.- VISTA

Lo que el usuario ve y con lo que puede interactuar es lo que se denomina vista y en el caso de una aplicación web, son las páginas que se muestran en un navegador. Están directamente relacionadas con los casos de uso antes presentados ya que precisamente esto representa lo que los usuarios pueden hacer.

No tiene interés alguno el presentar fragmentos de código JSP y remitimos al lector a visitar las páginas del manual.

No obstante cabe destacar dos de las funcionalidades implementadas, la posibilidad de acceder remotamente desde una página web hasta los motes de control instalados en un sistema local, para enviar órdenes y la monitorización de baja latencia para ver el estado de los sensores. Para ver como se ha realizado este apartado, primero hay que ver el camino que recorre la información desde la página web que el usuario ve en el navegador y viceversa, hasta el mote.



1. Monitorización de sensores

   Este apartado pretende mostrar una serie de gráficas temporales de las medidas de los sensores instalados en un mote concreto. Además se ha pretendido que no sea algo estático, sino que dinámicamente se actualicen los datos mostrados con nuevas medidas, pero sin la interacción constante de usuario, es decir, sin refrescar manualmente la página web.

   Para conseguir esto inicialmente el usuario accede a una página en la que aparecen una serie de gráficas, cada una relacionada con los sensores del mote que ha seleccionado. Dentro de la página se ha programado un script que periódicamente realiza peticiones automáticas a una dirección web concreta, dentro de nuestra aplicación, de tal forma que le devuelve un conjunto de medidas que son visualizadas en la página.

   Ciertos detalles relevantes de implementación se encuentran en el apartado 3.3.3.1.- Monitorización de sensores, del documento Anexos.

2. Histórico de datos

   A parte de la funcionalidad que permite monitorizar en directo a los sensores de la red, se ha implementado una opción para obtener históricos de los datos proporcionados por los sensores. Esto se puede conseguir ya que las medidas de dichos sensores se almacenan en una base de datos. Esta funcionalidad está limitada a la agrupación de datos temporalmente pero se puede extender la funcionalidad para hacer estimaciones de consumo estacional y plantear estrategias de minimización del mismo.

   Ciertos detalles relevantes de implementación se encuentran en el apartado 3.3.3.2.- Histórico de datos, del documento Anexos.

3. Control de actuadores

   Para realizar el control de los motes que disponen de actuadores, el flujo de datos va únicamente desde la página web hacia el servidor y finalmente al mote. No por esto es más sencillo que la monitorización, pues desde el servidor web hay que acceder hasta el servidor local y de ahí a la red de motes.

En un primer paso, el usuario interactúa con la página web estableciendo unas nuevas condiciones para el controlador que ha elegido, que pueden ser cambiar el ángulo de disparo de la iluminación de una habitación o subir una persiana. Estos cambios se envían en background hasta el servidor web que genera la información necesaria a partir de los parámetros que llegan, contacta con el servidor remoto y envía los datos de la trama generada.

Ciertos detalles relevantes de implementación se encuentran en el apartado 3.3.3.3.- Control de actuadores, del documento Anexos.

El caso que se ha implementado en el prototipo es el de control de la iluminación de una bombilla mediante la variación del ángulo de disparo de un tiristor, tal y como se detalla en las secciones 7.4.2.6 y 7.4.3.4.

7.5.3.6.- REGLAS

Como se indicó en 7.5.4.4, la aplicación permite la creación de alarmas asociadas a los sensores instalados en los motes, entendiendo por alarma, una condición fuera de lo marcado como normal.

En este sentido cabe recordar que cada vez que un mote sensor envía datos al coordinador, se comprueba la existencia de alarmas y si las hay, las ejecuta para comprobar si alguna de ellas se cumple.

Hay que distinguir entre lo que es una regla y la ejecución de reglas.

Una regla es una condición o conjunto de condiciones a las que se les aplica determinados valores de entrada.

La ejecución de una o varias reglas es el proceso por el que a dichas reglas se les aplica un conjunto de valores de entrada, produciendo una salida que determina si se cumplen o no.

Desde el punto de vista del usuario de la aplicación, lo que hace es crear las reglas a las que se les aplicarán posteriormente los vectores de entrada con los datos provenientes de los motes sensores, de tal forma que queda por implementar la parte que ejecuta las reglas.

En la primera versión del presente proyecto, se creó un motor de ejecución de reglas sencillo, el cual realizaba la comprobación de las reglas creadas con los datos recibidos en una sola pasada de las reglas, es decir si la sensibilización de una regla daba como resultado la sensibilización de una segunda, esta segunda no se llegaba a ejecutar.

Como mejora al sistema anteriormente implementado, se ha hecho uso de un auténtico motor de reglas empleando la librería open source, Drools, proporcionada por Jboss.

Se trata de un motor de creación, ejecución y gestión de reglas basado en un algoritmo RETE.

Un sistema basado en reglas consta de los siguientes elementos:

- Motor de inferencia: coordina la información procedente del resto de módulos.
- Base de conocimientos: contiene las reglas utilizadas para representar el conocimiento disponible en un determinado dominio.
- Base de hechos: contiene los hecho iniciales como los que se van generando.
- Interfaz de usuario: a través del cual el usuario puede interaccionar con el sistema.

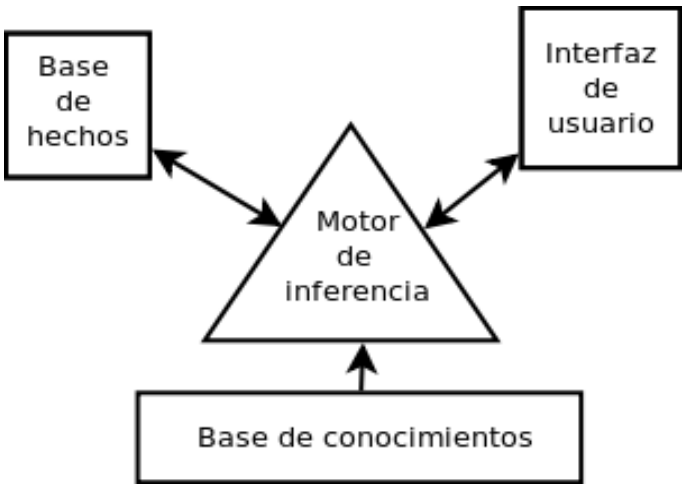


FIGURA 17: ESTRUCTURA DE UN MOTOR DE INFERENCIA

Una regla por lo general consta de dos partes: antecedente o parte izquierda y consecuente o parte derecha. En el antecedente figuran las condiciones que se han de cumplir para que la regla sea sensibilizada. En el consecuente, aparecerán las conclusiones o acciones a desarrollar cuando la regla sea ejecutada.

El proceso de inferencia en un sistema basado en reglas consiste en establecer la verdad de determinadas conclusiones a partir de la información que se tiene en la base de conocimiento y en la base de hechos. Es el motor de inferencia el encargado de llevar a cabo dicho proceso, que por lo general puede realizarse de dos formas:

- Encadenamiento hacia adelante

  Se basa en ejecutar aquellas reglas cuyo antecedente es cierto a partir de la base de hechos que hay en el sistema. Por otra parte la introducción de nueva información a partir del consecuente de cada regla ejecutada, permitirá la ejecución de otras reglas.

- Encadenamiento hacia atrás

  Se basa en seleccionar aquellas reglas cuyo consecuente permita demostrar cierta condición, si ésta no puede ser demostrada a partir de la base de afirmaciones. A su vez, las condiciones que figuran en los antecedentes de las reglas seleccionadas volverán a convertirse en nuevos subobjetivos de forma recursiva hasta que se encuentra la información buscada en la base de hechos.

Drools emplea un algoritmo mejorado del encadenamiento hacia atrás denominado RETE.

La idea que constituye la base de este algoritmo es que , en lugar de buscar qué reglas satisfacen los hechos existentes en cada momento, son los nuevos hechos generados en cada ciclo los que buscan o determinan qué nuevas reglas se seleccionan para una posible ejecución.

8.- CONCLUSIONES

El sistema final queda definido por el siguiente esquema en el que se especifica la solución empleada en cada uno de los bloques:

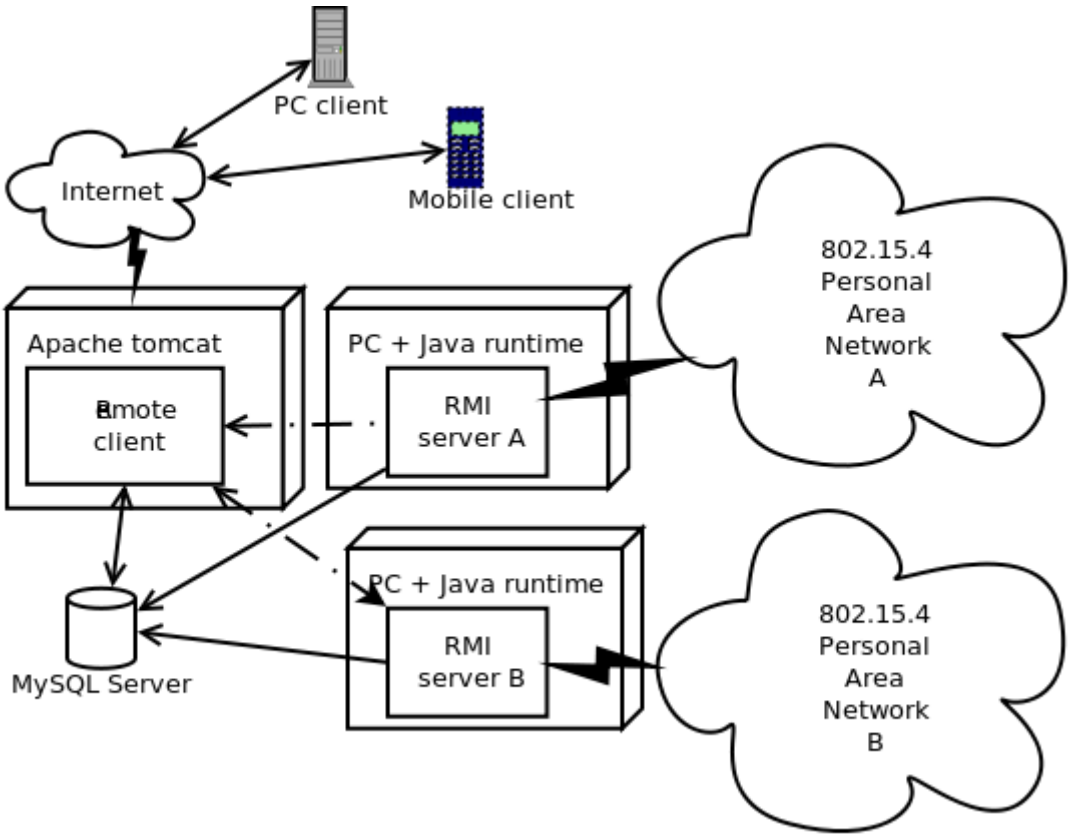


FIGURA 18: ARQUITECTURA IMPLEMENTADA

- Red domótica
  La red domótica se ha implementado usando el estándar 802.15.4, que es un protocolo para redes inalámbricas de baja tasa de datos y bajo consumo, de acuerdo con los requerimientos del proyecto. Como transceptor inalámbrico se ha empleado XBee de Digi, y aunque en la práctica no es el mejor que se puede encontrar, para los propósitos de prototipado es el adecuado.
- Motes
  Los motes o elementos de red a alto nivel, están formados por un microcontrolador Atmel AVR 168P/328P, un transceptor XBee como elementos principales.
  Disponen de una estructura física modular que permite el acoplamiento de diversos módulos para sensado o control.
  Su programación está pensada para bajo consumo lo que también los hace adecuados para aplicaciones en redes sensoriales. Además se han implementado opciones importantes como la autoconfiguración de la red o el reseteo de la misma, al nivel de los motes.
- Sistema local
  El sistema local está encargado de gestionar a alto nivel la información que circula por la red de motes.

Esta gestión incluye el manejo de tramas de autoconfiguración, de sensado y control así como la persistencia de forma remota de la información relevante y la detección de alarmas.
Se ha implementado mediante un servicio RMI en Java, lo que permite la ejecución de tareas de control desde el exterior.

- Sistema servidor
  El sistema servidor es el clásico Apache Tomcat que engloba las tecnologías de servidor de aplicaciones y contenedor de servlets.
  Implementa todas las funcionalidades que están accesibles para los usuarios de los sistemas domóticos y para los administradores o gestores de la aplicación global.
  Tanto Apache Http Server como Apache Tomcat, son tecnologías Open Source y no tienen coste de implantación en cuanto a licencias
- Base de datos
  El sistema de base de datos empleado es el conocido MySQL, que es una solución robusta, probada, con herramientas de administración y soporte de la comunidad open source y que además no tiene coste de licencia.

Como conclusiones más personales, podía destacar las siguientes que a continuación se indican.

Se ha empleado el transceptor XBee para la realización de la red 802.15.4, principalmente por el encapsulado DIP de fácil prototipado y por la abstracción que ofrece en términos de programación, ya que limita el proceso de comunicación al envío y recepción de tramas estructuradas vía RS232. En ocasiones el comportamiento no es el que parece lógico y la documentación no profundiza en ningún aspecto, ni hardware ni software, lo que dificulta sobremanera el proceso de desarrollo. El coste es elevado respecto a otros transceptores 802.15.4 pero tiene la ventaja de que es intercambiable con los modelos de mayor potencia de transmisión. Para una supuesta producción a gran escala, debería seleccionarse algún otro transceptor de los existentes en el mercado que dan soporte para 802.15.4 y ZigBee, con menor consumo, con un stack de programación que permita controlar los aspectos de bajo nivel y con un encapsulado que permita reducir las dimensiones del mote.

El microcontrolador elegido es bastante adecuado a la aplicación, dadas sus características técnicas y la facilidad de programación. No obstante, se debería emplear un modelo con encapsulado TQFP o QFN, que reduce sensiblemente el área ocupada. El proceso de desarrollo se ha hecho sin contar con un depurador de código y aunque no es estrictamente necesario, lo ha ralentizado.

Se ha implementado que los motes dispongan de varios mecanismos de alimentación empleando baterías, terminal USB y conector de barril a fuente de alimentación. Esto es innecesario. En el caso de una aplicación domótica como la planteada, una mejor solución teniendo en cuenta el ámbito, habría sido la incorporación de un sistema típico de fuente de alimentación, con mini transformador para PCB en conjunción con un convertidor buck. La mayoría de los motes pueden tomar la alimentación de la red monofásica.

En cuanto a los sistemas locales o instalaciones, se ha optado por una implementación de un sistema cliente servidor (RMI) para el control y monitorización de cada una. Esto, para una instalación domótica no es estrictamente necesario, ya que generalmente el usuario va a interactuar con el sistema cuando se encuentre en él. No cabe duda de que es un valor añadido pero quizá la complejidad introducida sea innecesaria.

Así mismo el sistema local se ha implementado sobre un ordenador personal, con las ventajas y desventajas que ofrece. En primer lugar la propensión a errores y los reinicios que puede sufrir un PC, influirían no-

tablemente en el funcionamiento estable del sistema. El PC no se podría parar, ya que el coordinador al está conectado directamente a uno de los puertos USB y si el coordinador desaparece, la red PAN deja de existir.

Sería una mejor implementación, y se plantea como mejora, el emplear un Single Board Computer (SBC) o un sistema electrónico que disponga de un microprocesador relativamente potente, de 32 bits y a 500 MHz por ejemplo, que disponga de conectividad Ethernet o WiFi. Si no se desea modificar la programación del sistema local, es necesario el empleo de un microprocesador Intel ya que el sistema de invocación remota RMI, tiene una implementación estable sobre esta arquitectura.

Así mismo, sería conveniente incluir un sistema de visualización táctil o un control (mando) avanzado, comunicándose por ejemplo, empleando el propio 802.15.4, con cada uno de los motes.

En cuanto al sistema servidor, la solución es bastante correcta aunque la interfaz de usuario se puede mejorar mucho todavía para hacerla realmente usable, principalmente evitando el empleo de palabras técnicas como *mote* y en lugar de mostrar elementos tabulares, dar la posibilidad de incluir un mapa de la casa, sobre el que se disponen los dispositivos instalados.

En líneas generales, la idea de llevar 802.15.4 al campo de la domótica puede suponer grandes ventajas para el desarrollo de una red de dispositivos que interoperan de forma estándar, siempre y cuando se estandaricen capas superiores a las que implementa 802.15.4.

2010

Realización de sistema domótico con microcontroladores de bajo coste (AVR) y módulos RF verificando el estándar 802.15.4

(II)

ANEXOS

Escuela Universitaria Ingeniería Técnica Industrial ZARAGOZA

Directora: Aránzazu Otín Acín

Autor: Francisco Pérez Pellicena

HOJA DE IDENTIFICACIÓN DE DATOS

DATOS DEL PROYECTO
REALIZACIÓN DE SISTEMA DOMÓTICO CON MICROCONTROLADORES DE BAJO COSTE (AVR) Y MÓDULOS RF, VERIFICANDO EL ESTÁNDAR 802.15.4
DIRECTORA DEL PROYECTO
Aránzazu Otín Acín *Razón social: C\María de Luna 1, Edificio Ada Byron, 50015 Zaragoza*
AUTOR DEL PROYECTO
Francisco Pérez Pellicena *C\Coimbra 2 2ºC 50008 Zaragoza*
FECHA Y FIRMA
En Zaragoza a 31 de Agosto de 2010

ÍNDICE

1. DOCUMENTACIÓN DE PARTIDA

El establecimiento de los requisitos viene dado por las hojas de características establecidas por los proyectantes en el apartado 6, "Requisitos de diseño", del documento "Memoria". Además de tener en cuenta la normativa aplicable, reflejada en el apartado 4,"Normas y referencias" del documento "Memoria" y de los puntos 4.1. "Disposiciones legales y normas aplicadas" y 4.2. "Legislación a aplicar" del documento "Pliego", que puede establecer criterios concretos para la realización de determinadas partes. Se considera también documentación de partida, toda aquella relacionada con los datasheet, especificaciones y recomendaciones de los diferentes fabricantes. Esta documentación estará incluida en el apartado 4 del presente documento.

2. CÁLCULOS

2.1. COORDINADOR

2.1.1. ALIMENTACIÓN

El módulo del coordinador está formado por dos bloques funcionales, un bloque de potencia o alimentación y un bloque de comunicaciones, tal y como se puede apreciar en el apartado 2.1 del documento "Planos".

La siguiente figura muestra el bloque de alimentación, cuyo elemento principal es un regulador lineal de salida fija.

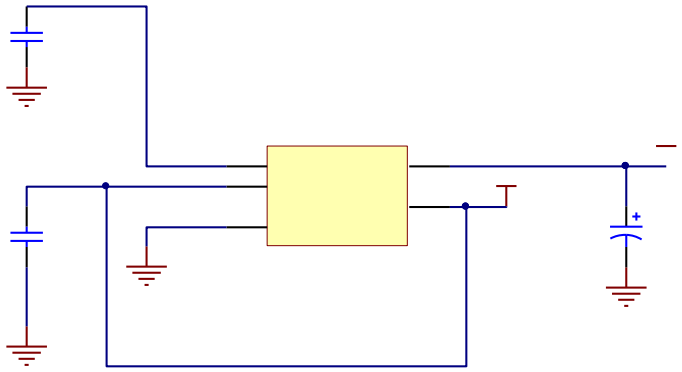


FIGURA 1: CONFIGURACIÓN DEL REGULADOR LINEAL TPS79333

El transceptor XBee dispone de un rango de tensiones de alimentación comprendidos entre 2.8 v y 3.4 v, por lo que la tensión proporcionada por el regulador debe pertenecer a ese rango.

El consumo que establece el transceptor XBee está en función del modelo, pues recordemos que DIGI proporciona dispositivos con varias potencias de transmisión, y por lo tanto de más consumo. El transceptor que se ha empleado es un Serie 1 de 1mW, que se trata de la serie de menor potencia. El consumo de este transceptor es de 45 mA en transmisión y de 50 mA en recepción, según indica el datasheet del fabricante. En la práctica se pueden apreciar picos de corriente que casi llegan a doblar las cifras proporcionadas, hecho que se va a tener en cuenta para dimensionar el regulador.

La tensión de entrada al regulador, está proporcionada por un terminal que cumple la especificación USB 2.0 y tiene un valor de 5 voltios.

Para conseguir la mayor potencia de trabajo, se alimenta al transceptor XBee al mayor valor posible que esté dentro de los valores típicos que proporcionan los reguladores comerciales. Así encontramos que el valor más próximo para alimentar al dispositivo, sin comprometer el coste, es de 3.3 voltios.

Lo anterior supone que se ha decidido emplear un regulador de salida fija, lo que minimiza el empleo de componentes externos para fijar la salida, redundando en un menor coste, una mayor fiabilidad y una mejor precisión en la tensión proporcionada.

Dada la tensión proporcionada por el terminal USB, que es de 5 voltios y la salida del regulador de 3.3 voltios, queda un margen de 1.7 voltios, por lo que se ha de emplear un regulador de bajo dropout.

Se ha elegido el regulador lineal TPS79333DBVR, que proporciona una tensión de salida no regulable de 3.3V y una corriente máxima de 200mA.

La potencia que disipa en el peor de los casos, suponiendo una corriente media de 100mA, aunque ya sabemos que va a ser aproximadamente la mitad:

$$P_{max} = (V_{in} - V_{out}) * I_{max} = (5 - 3.3) * 0.1 = 170\ mW$$

Suponiendo que el módulo del coordinador va a estar generalmente a una temperatura ambiente, en torno a los 25ºC, el fabricante indica en el apartado "Dissipation ratings table" que es capaz de soportar hasta 390 mW sin emplear disipador. Dado que este valor es bastante superior al que se ha calculado, no es necesario el empleo de disipador térmico.

Los condensadores externos necesarios son los indicados por el fabricante en el apartado "External capacitor requirements".

2.1.2. COMUNICACIÓN

El segundo bloque está formado por la circuitería necesaria para realizar una transmisión de datos efectiva entre el transceptor XBee y el ordenador al que está conectado. Por lo tanto se hace necesario el empleo de un adaptador Serie-USB.

Se ha empleado el conocido FT232R del fabricante FTDI, y la siguiente figura lo ilustra.

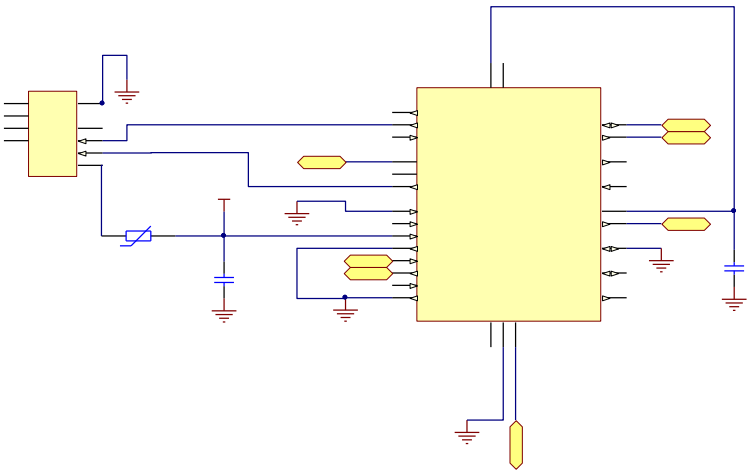


FIGURA 2: CONFIGURACIÓN DEL ADAPTADOR FT232R

Se va a emplear el dispositivo FT232R alimentado desde el mismo puerto USB, por lo que para este propósito no se requiere ningún dispositivo adicional.

Se ha incluido un fusible reseteable PTC que abre el circuito de alimentación en caso de que se superen los 500mA que especifica el estándar USB como límite de corriente máxima que puede proporcionar.

Se ha seguido el apartado "7.1.- USB to RS232 converter" que recomienda el fabricante, para configurar una interfaz sencilla entre el terminal USB del ordenador, con la USART del transceptor XBee.

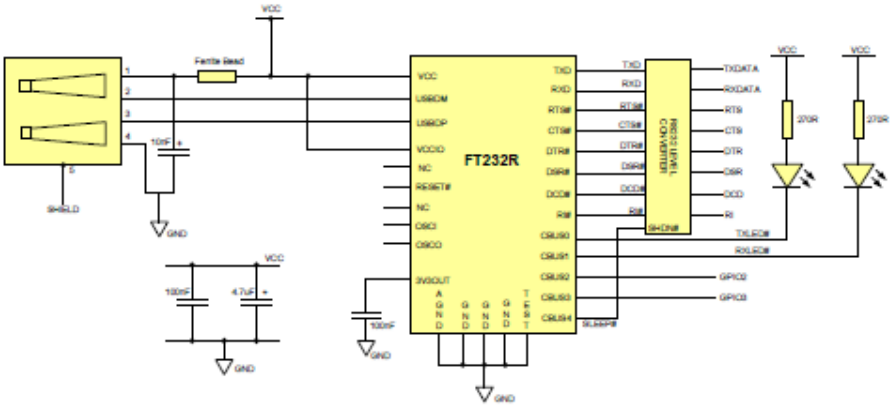


FIGURA 3: APLICACIÓN TÍPICA USB-SERIAL DEL FT323R

En este sentido, se han añadido dos diodos led indicadores para transmisión y recepción, con sus respectivas resistencias limitadoras de corriente, según especifica el fabricante en el apartado "7.5.- Led interface".

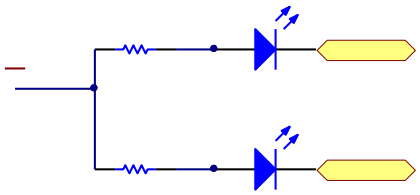


FIGURA 4: LEDS INDICADORES PARA EMISIÓN Y RECEPCIÓN

Se ha aprovechado la salida del regulador de 3.3 voltios y empleando unos diodos led SMD con una tensión umbral de 2 voltios, se ajusta la resistencia limitadora de corriente como sigue:

$$R_{led} = \frac{(Vccio - Vled_f)}{I_{led}}$$

Teniendo en cuenta que el fabricante de los diodos led indica una corriente máxima de 20 mA, la resistencia mínima que hay que colocar:

$$R_{led}min = \frac{(3.3 - 2.03)V}{20\ mA} = 63.5\ Ohm$$

Estableciendo la corriente por el diodo a 4mA para conseguir una luminosidad aceptable sin ocasionar un consumo notable, la resistencia necesaria:

$$R_{led} = \frac{(3.3 - 2.03)\ V}{4\ \ mA} \approx 330\ Ohm$$

De la misma forma, se ha empleado la configuración anterior para indicar que el circuito está alimentado.

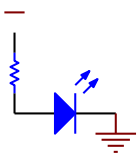


FIGURA 5: LED DE POWER

En este caso es necesario que la alimentación provenga directamente del terminal USB, por lo que la tensión tiene un valor de 5 voltios. Los cálculos para la resistencia limitadora son los siguientes, bajo las mismas condiciones del diodo led:

$$R_{led}min = \frac{(5 - 2.03)V}{20\ mA} 135\ Ohm$$

Por otro lado el transceptor XBee también dispone de dos indicadores led, dispuestos de la misma forma que en el caso del FT232R:

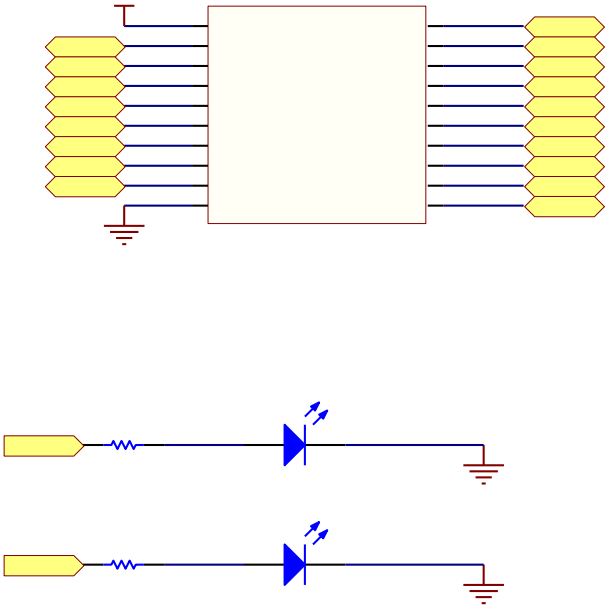


FIGURA 6: TRANSCEPTOR XBEE Y LEDS INDICADORES DE ASOCIACIÓN Y RSSI

Los diodos empleados son los mismos que el caso anterior y la tensión de polarización, proveniente del transceptor XBee es igualmente de 3.3 voltios, por lo que las condiciones son las mismas que las anteriormente calculadas y se emplean resistencias de 330 Ohmios.

2.2. MOTE BASE

2.2.1. ALIMENTACIÓN

El mote base debe proporcionar un sistema de alimentación para el microcontrolador, el transceptor XBee y las PCB funcionales que se le acoplan, por lo que podemos estimar el consumo en cada caso para determinar la corriente mínima que debe proporcionar.

De la misma forma que el coordinador, los rangos de tensión están limitados por los dispositivos a los que alimenta. La siguiente tabla muestra los dispositivos que contiene el diseño y sus posibles valores de tensión de entrada.

Dispositivo	Rango de tensiones de alimentación
Microcontrolador Atmel AVR Atmega 168	Min = 1.8 v, Max = 5v
Transceptor XBee	Min = 2.8v, Max = 3.4v

Por lo tanto, la intersección de ambos conjuntos determina que el rango de valores posibles de alimentación viene determinado por los que marca el transceptor XBee.

Un valor standard comercial que se encuentra dentro del rango de posibles valores es 3.3 voltios y es el que se ha seleccionado.

El consumo en cada mote es la suma del consumo en el mote base y en cada caso, el de la PCB funcional que tenga acoplada. Teniendo en cuenta que la principal fuente de alimentación se encuentra físicamente en el mote base y es única para todo el mote, hay que dimensionarla para el peor de los casos.

Los siguientes cálculos estiman el consumo del mote base, teniendo en cuenta una tensión de alimentación de 3.3 voltios.

MICROCOTROLADOR

Dispone de un cristal a 8 MHz como primera condición para estimar el consumo.

Así mismo se considera un modo de funcionamiento normal, sin la activación de modos de bajo consumo.

De acuerdo con el funcionamiento normal, a priori son susceptibles de estar activos los siguientes bloques periféricos internos, que contribuyen al consumo total:

Mote	Periféricos activos
Ambiental	• USART • ADC • TIMER 0, TIMER 1
Detección presencia	• USART • TIMER 0
Dimmer	• Timer 0 • Timer 2

La siguiente tabla resume el consumo de los bloques periféricos integrados, extraída a partir de la tabla que aparece en el apartado "29.3.3" del Datasheet del Atmega 168:

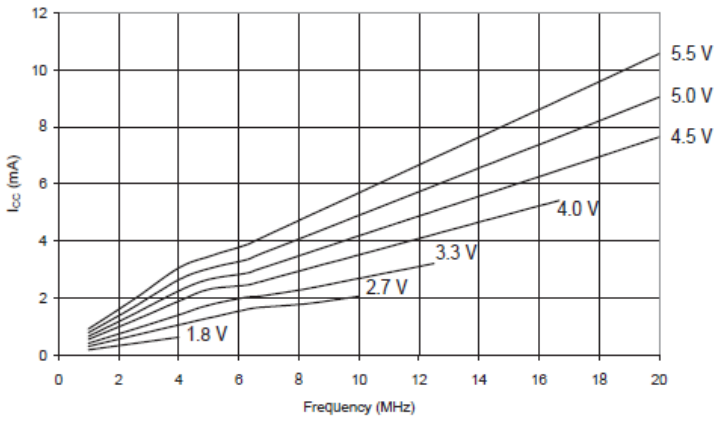


FIGURA 7: CONSUMO DEL ATMEGA168 DEPENDIENTE DE LA FRECUENCIA DEL CRISTAL

PRR bit	Additional Current consumption compared to Active with external clock (see Figure 29-93 on page 375 and Figure 29-94 on page 375)	Additional Current consumption compared to Idle with external clock (see Figure 29-98 on page 377 and Figure 29-99 on page 378)
PRUSART0	1.5%	8.9%
PRTWI	3.2%	19.5%
PRTIM2	2.4%	14.8%
PRTIM1	1.7%	10.3%
PRTIM0	0.7%	4.1%
PRSPI	2.9%	17.1%
PRADC	3.4%	20.3%

FIGURA 8: CONSUMO DE LOS PERIFÉRICOS COMO PORCENTAJE DEL AISLADO DEL ATMEGA168

A partir de la tabla 7, se determina que para una tensión de alimentación de 3.3 voltios y una frecuencia de reloj de 8 MHz, el consumo medio del microcontrolador es de 2.2 mA.

La siguiente tabla indica los consumos adicionales que se superponen al consumo del microcontrolador en modo activo.

Periférico	Consumo
USART	1.5%*2.2mA = 0.033 mA
ADC[1]	3.4%*2.2mA = 0.0748 mA
TIMER 0	0.7%*2.2mA = 0.0154 mA
TIMER 1	1.7%*2.2mA = 0.037 mA
TIMER 2	2.4%*2.2mA = 0.0528 mA

[1] SE TIENE EN CUENTA QUE LA REFERENCIA INTERNA DEL ADC ESTÁ CONECTADA A LA TENSIÓN DE ALIMENTACIÓN DEL ATMEGA.

Con la información anterior podemos estimar los siguientes consumos por mote.

Mote	Periféricos activos	Consumo estimado
Ambiental	• USART • ADC • TIMER 0, TIMER 1	2.2 mA + 0.033 mA + 0.0748 mA + 0.0154 mA + 0.037 mA = 2.36 mA
Detección presencia	• USART • TIMER 0	2.2 mA + 0.033 mA + 0.0154 mA = 2.25 mA
Dimmer	• USART • Timer 0 • Timer 2	2.2 mA + 0.033 mA + 0.0154 mA + 0.0528 mA = 2.3 mA

El consumo máximo es de 2.36 mA.

XBEE

Los consumos del transceptor XBee ya se han mencionado en el apartado "2.1.1" y como resultado se indica que el consumo máximo se especifica durante la recepción de datos, tomando un valor en torno a 50 mA.

La siguiente tabla extraída del Datasheet, referencia lo anterior.

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
I_{IN}	Input Leakage Current	V_{IN} = VCC or GND, all inputs, per pin	-	0.025	1	µA
I_{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, 140 (PRO, int)	-	mA
RX	Receive Current	VCC = 3.3 V	-	50 (XBee) / 55 (PRO)	-	mA
PWR-DWN	Power-down Current	SM parameter = 1	-	< 10	-	µA

FIGURA 9: ESPECIFICACIONES ELÉCTRICAS DEL TRANSCEPTOR XBEE

SWITCH ADG719

El consumo del switch digital es despreciable por dos motivos:

- El fabricante indica un consumo para polarización interna de máximo 1 uA.
- El consumo asociado al proceso de reset es transitorio, pues únicamente se emplea para el sistema de reseteo automático.

La siguiente figura muestra el consumo del Atmega 168 en el pin de reset.

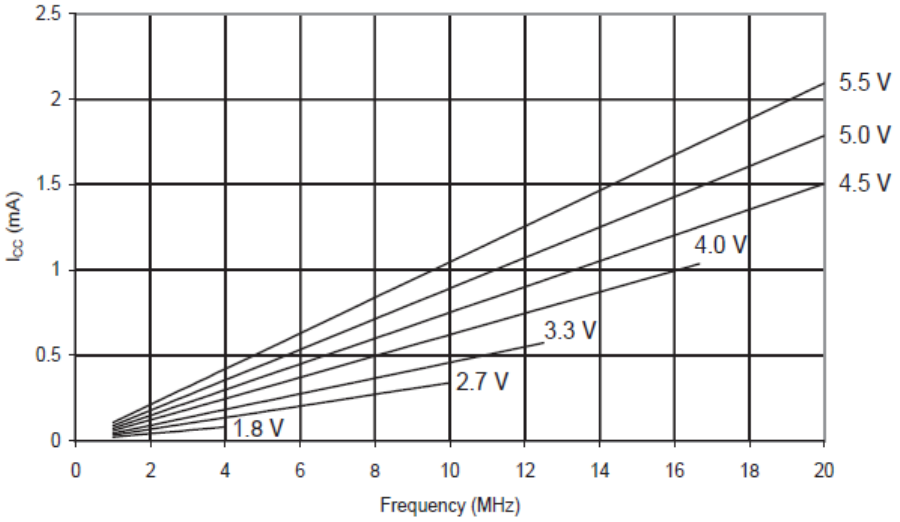


FIGURA 10: CONSUMO DEL ATMEGA 168 AL RESETEAR

Para la tensión de alimentación de 3.3 voltios y la frecuencia de 8 MHz, aparece un consumo en reset de 0.35 mA durante los 500nS que dura el reset, como se puede ver en las figuras 10 y 11.

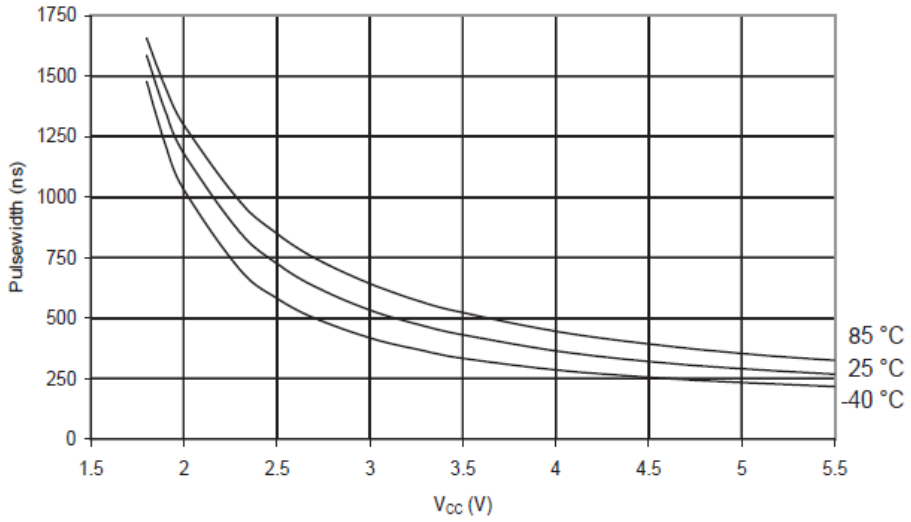


FIGURA 11: TIEMPO MÍNIMO DE RESET EN FUNCIÓN DE LA TENSIÓN DE ALIMENTACIÓN

INDICADORES

Existen varios diodos led polarizados de forma permanente o alternativa, que indican el estado del mote. Algunos de ellos se activan cuando está en modo carga de batería, por lo que no los vamos a tener en cuenta para el cálculo del consumo, pues únicamente nos referimos al consumo en funcionamiento con baterías.

El transceptor XBee dispone de un indicador de asociación, que parpadea 2 veces por segundo, según indica el Datasheet del componente en el apartado "End-device start up". Así pues suponiendo que el mote está correctamente asociado, permanecerá de forma permanente parpadeando.

Según las características de los diodos led empleados, se requiere una corriente de unos 10 mA para conseguir una luminosidad aceptable, según la siguiente tabla extraída del Datasheet.

Conventional Part No.	Lighting Color	Lens Color	I_O		I_F	V_F		λ_P	$\Delta\lambda$	I_F	I_R	
			Typ	Min		Typ	Max	Typ	Typ		Max	V_R
LNJ206R5RRX	Red	Red Diffused	0.4	0.15	10	2.03	2.6	700	100	10	5	4
LNJ206R5ARA	Ultra Bright Red	Red Diffused	5.0	1.7	10	1.72	2.5	660	20	10	10	3
LNJ306G5URA	Green	Green Diffused	2.6	0.9	10	2.03	2.6	565	30	10	10	4
LNJ806K5SRX	Soft Orange	Yellow Diffused	0.8	0.3	10	1.93	2.6	610	40	10	10	3
Unit	—	—	mcd	mcd	mA	V	V	nm	nm	mA	µA	V

FIGURA 12: ESPECIFICACIONES ELÉCTRICAS DE LOS DIODOS LED SMD

Así, teniendo en cuenta que la salida de los terminales I/O del transceptor XBee no pueden superar la tensión de alimentación, fijada a 3.3 voltios, la resistencia adecuada para el diodo es la siguiente:

$$R = \frac{V}{I} = \frac{3.3\,V}{10\,mA} = 330\,Ohm$$

Con los cálculos anteriores se puede estimar el consumo en general del mote base, sin tener en cuenta las PCB funcionales que se van a acoplar.

$$I_{mote\ base} = 2.36 + 50 + 0.35 + 10 = 62\,mA$$

Este cálculo supone un caso desfavorable, ya que supone la activación de varios periféricos integrados simultáneamente, así como el transceptor XBee recibiendo, lo que es poco probable al mismo tiempo.

A esto hay que sumarle la contribución de las PCB funcionales en cada uno de los casos, lo que incrementa el consumo final.

Mote ambiental

El mote ambiental dispone de los siguientes sensores:

- Sensor de temperatura TMP36

  El sensor está alimentado a la tensión establecida inicialmente, 3.3 voltios y el consumo en modo activo viene dado por la siguiente figura extraída del Datasheet:

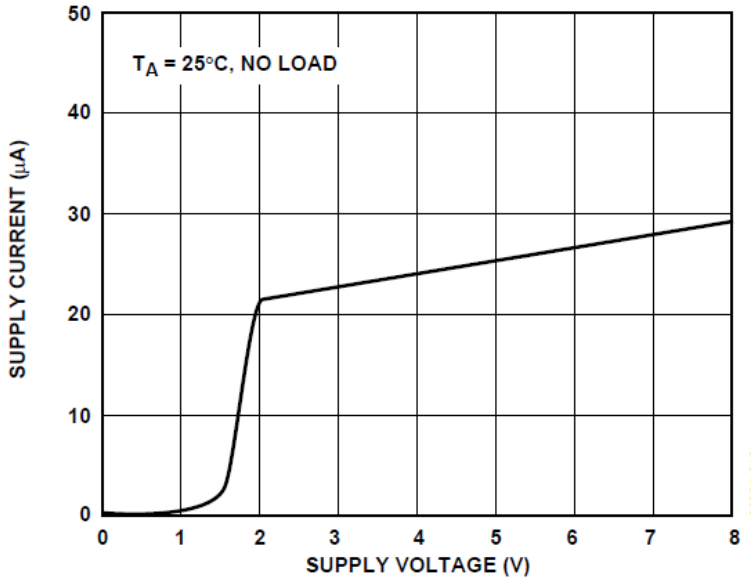


FIGURA 13: CONSUMO DEL SENSOR DE TEMPERATURA TMP36 REFERENTE A LA TENSIÓN DE ALIMENTACIÓN

  Por lo tanto, para la tensión de 3.3 voltios, el consumo está en torno a 23 uA.

- Sensor de humedad HIH 5030

  De la misma forma, el Datasheet del sensor de humedad indica que bajo las condiciones de 3.3 voltios y alimentación y 25ºC, el consumo típico es de 200 uA.

Table 1. Performance Specifications (At 3.3 Vdc supply and 25 °C [77 °F] unless otherwise noted.)

Parameter	Minimum	Typical	Maximum	Unit	Specific Note
Interchangeability (first order curve)					
0% RH to 10% RH, 90% RH to 100% RH	-7	–	7	% RH	–
11% RH to 89% RH	-3	–	3	% RH	
Accuracy (best fit straight line) 11% RH to 89% RH	-3	–	+3	% RH	4
Hysteresis	–	2	–	% RH	–
Repeatability	–	±0.5	–	% RH	–
Settling time	–	–	70	ms	–
Response time (1/e in slow moving air)	–	5	–	s	–
Stability (at 50% RH in 5 years)	–	±1.2	–	% RH	1
Voltage supply	2.7	–	5.5	Vdc	2
Current supply	–	200	500	µA	–
Voltage output (1st order curve fit)	$V_{OUT}=(V_{SUPPLY})(0.00636(\text{sensor RH}) + 0.1515)$, typical at 25 °C				
Temperature compensation	True RH = (Sensor RH)/(1.0546 – 0.00216T), T in °C				
Output voltage temp. coefficient at 50% RH, 3.3 V	–	-2	–	mV/°C	–
Operating temperature	-40[-40]	See Figure 2.	85[185]	°C[°F]	–
Operating humidity (HIH-5030)	0	See Figure 2.	100	% RH	3
Operating humidity (HIH-5031)	0	See Figure 2.	100	% RH	–
Storage temperature	-50[-58]	–	125[257]	°C [°F]	–
Storage humidity	See Figure 3.			% RH	3

FIGURA 14: ESPECIFICACIONES ELÉCTRICAS DEL SENSOR DE HUMEDAD HIH5030

- LDR

Dada la característica de la LDR y el circuito resistivo de polarización, el consumo máximo viene determinado por la menor resistencia de la LDR, esto es con la mayor iluminación a la que el dispositivo tiene el límite de sensibilidad. En esa situación el valor de la LDR es de aproximadamente 9 KOhm, según la siguiente gráfica:

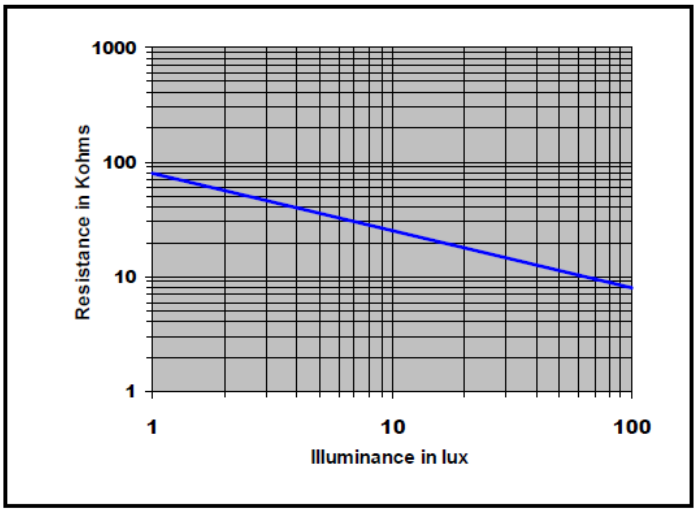


FIGURA 15: CARACTERÍSTICA DE LA LDR

Teniendo en cuenta que la resistencia de entrada del ADC es del orden de 100 MOhm, la corriente que consume es despreciable, del orden de nanoamperios, por lo que la contribución más importante al consumo es la que deriva del divisor resistivo:

$$I_{LDR} = \frac{3.3\ v}{9k + 10\ k} \approx 180\ uA$$

El conjunto de los tres sensores activos da un consumo adicional:

$$I_{add} = 23\ uA + 200\ uA + 180\ uA = 403\ uA$$

Mote detector de presencia

De acuerdo con la especificación del Datasheet, el consumo del sensor PIR, el propio fabricante lo estima en una cifra no superior a 100 uA.

Mote dimmer

En la PCB funcional hay dos elementos que suponen un consumo sobre el sistema de alimentación del mote base.

En primer lugar, el disparo del triac optoacoplado MOC3021, se efectúa desde uno de los terminales del microcontrolador Atmega 168, por lo que la corriente suministrada, proviene del sistema de alimentación.

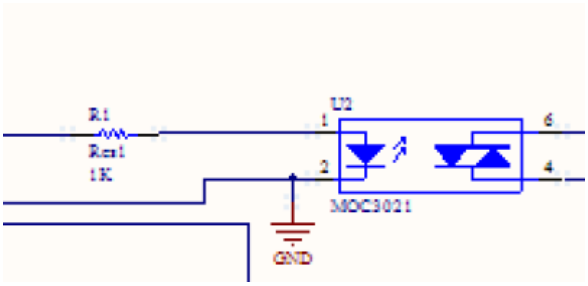


FIGURA 16: TRIAC OPTOACOPLADO MOC3021

La tensión directa del diodo IR del MOC3021 típica, está en torno a 1.15 voltios y teniendo en cuenta que la salida del terminal del microcontrolador estará entorno a 3.3 voltios, la corriente que extrae del terminal:

$$I_{trigger} = \frac{3.3 - 1.15\ v}{1\ k} = 2.15\ mA$$

Por otro lado, el optoacoplador 4N25, empleado para detectar los pasos por cero de la red supone un consumo en los momentos en los que la red monofásica se encuentra en el semiciclo positivo, pues el optoacoplador está en situación de disparo y el fototransistor de salida saturado.

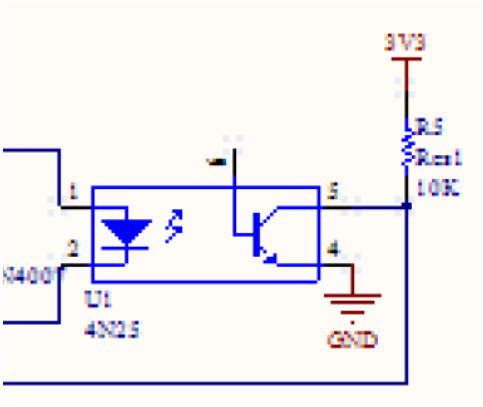


FIGURA 17: DETECTOR DE PASO POR CERO CON OPTOACOPLADOR 4N25

En esta condición, la tensión colector emisor del fototransistor es inferior a 0.5 voltios por lo tanto la corriente máxima:

$$I_{foto} = \frac{3.3 - 0.5}{10\ k} = 280\ uA$$

En conjunto, con los cálculos anteriores, el consumo máximo está en torno a 70 mA, pero no se puede descartar que el consumo de otras posibles PCB funcionales, incremente esta cifra.

Adicionalmente se ha tenido en cuenta el caso en que la tensión de la batería decaiga por debajo del límite impuesto como tensión de referencia, 3.3 voltios. Si decayera, en ese instante el sistema dejaría de funcionar sin aprovechar el límite de almacenamiento de la batería, por lo que se ha empleado un conversor buck/boost, en concreto el TPS61131 de Texas Instruments.

Este convertidor proporciona una salida principal de un buck/boost con un límite de suministro de corriente de hasta 450 mA y una salida secundaria de un regulador lineal LDO que proporciona hasta 320 mA.

Como suministro principal de energía, se emplea una batería de ión litio (Li+), que proporciona una tensión de 3.8 voltios, ajustado tanto para la etapa SEPIC como para el LDO. Por el lado del SEPIC la eficiencia se mantiene por encima del 85 %, como se puede apreciar en la figura 18, extraída del datasheet apartado “Typical characteristics” .

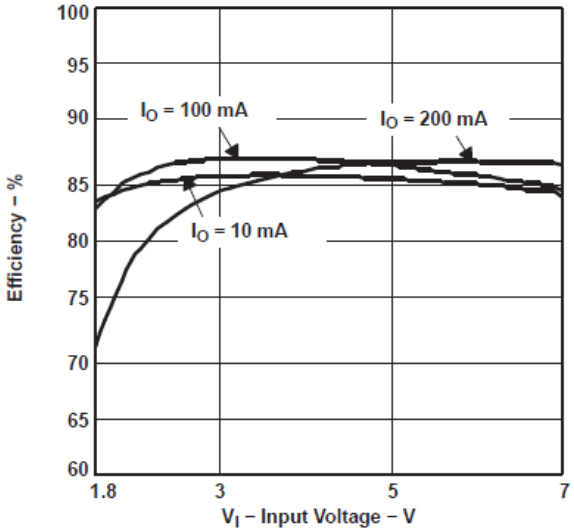


FIGURA 18: EFICIENCIA DEPENDIENTE DE TENSIÓN DE ENTRADA Y CONSUMO

Así mismo en la etapa del LDO, las pérdidas se minimizan con una caída de tensión de 0.5 voltios en el regulador a costa de limitar la corriente que puede proporcionar debido al aumento de la tensión de dropout con la corriente de salida en torno a 300 mA.

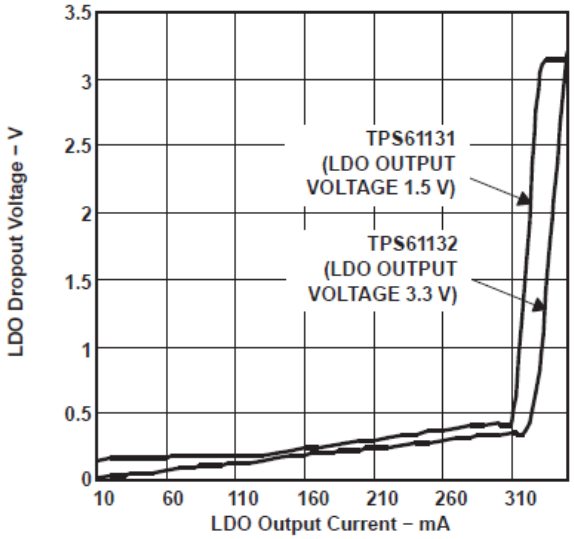


FIGURA 19: VARIACIÓN DE LA TENSIÓN DE DROPOUT EN FUNCIÓN DE LA CORRIENTE PROPORCIONADA

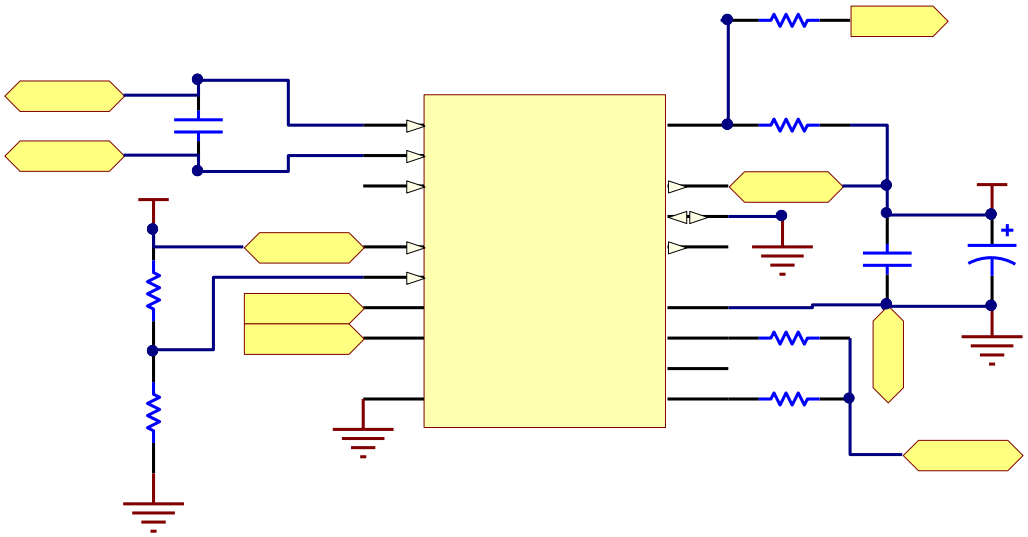


FIGURA 20: CONFIGURACIÓN DEL SEPIC TPS61131

El valor de la tensión de salida, se ajusta mediante un divisor resistivo, en concreto R8 y R9 en la figura 20, que de acuerdo con las ecuaciones proporcionadas por el fabricante en el apartado "Application information" del Datasheet:

$$R_8 = 180\ k * (\frac{V_0}{500\ mV} - 1)$$

fijando la resistencia **R9 a 180 KOhm**.

La salida necesaria es de 3.3 voltios, por lo que queda un valor para R9 de **1 MOhm.**

De acuerdo con la siguiente figura, el rendimiento del boost, cae notablemente para tensiones de entrada próximas a 1.8 voltios:

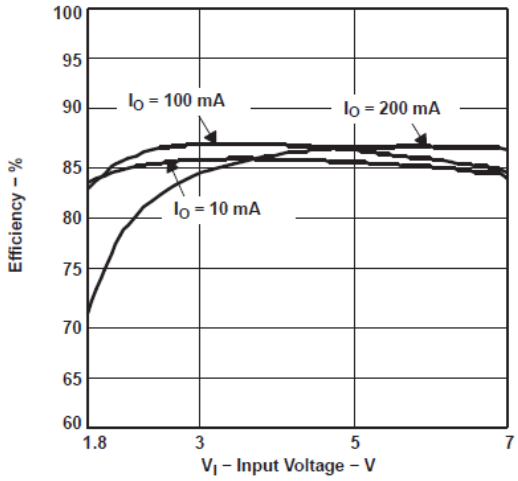


FIGURA 21: EFICIENCIA DEL TPS61131 EN FUNCIÓN DE LA TENSIÓN DE ENTRADA Y CONSUMO

Por lo tanto, para programar la entrada LBI, se configura un divisor resistivo para que active la salida LBO cuando la tensión de la batería alcance el valor de 1.8 voltios.

Según la recomendación del fabricante en el apartado "Application information", para el cálculo del divisor:

$$R_6 = 390\ k * (\frac{V_0}{500\ mV} - 1)$$

Teniendo en cuenta, Vo = 1.8 voltios, es necesaria una resistencia aproximada de **1 MOhm para R6.**

La bobina a emplear recomendada por el fabricante se encuentra en el rango 10 a 47 uH y se ha elegido una bobina de 22 uH de Cooper Electronics, DRQ74220R, de acuerdo con los modelos que también recomienda.

El condensador volante no debe tener un valor inferior a:

$$C_{min} = \frac{100}{4 * \pi^2 * f^2 * L}$$

donde f es la frecuencia de conmutación del interruptor, que en este caso el valor típico es de 500 Khz y L el valor de la inductancia, habiendo elegido 22 uH. Con esto, el valor mínimo para el condensador C11:

$$C_{min} = \frac{100}{4 * \pi^2 * (500 * 10^3)^2 * 22 * 10^{-6}} = 460\ nF$$

Se ha empleado un condensador de 10 uF.

Para controlar el rizado de la salida se dimensiona el condensador C14 para ajustarlo al valor del rizado necesario, según la ecuación:

$$C_{min} = \frac{I_{out} * V_{out}}{f * \Delta V * (V_{out} + V_{bat})}$$

donde:

$$V_{out} = 3.3\ v$$

$$f = 500\ Khz$$

$$V_{bat} = 3.8\ v$$

y suponemos una corriente de salida máxima de 450 mA y un rizado no superior a 10 mV.

$$C_{min} = \frac{450\ mA * 3.3\ v}{500 * 10^{3} Hz * 10\ mV * (3.3\ v + \ 3.7\ v)} = 42.4\ uF$$

Estableciendo un factor de seguridad de aproximadamente 2 por las pérdidas transitorias y la resistencia equivalente serie del condensador, se elige un condensador de 100 uF.

Por otro lado, se ha dispuesto un condensador de 2.2 uF tal y como indica el Datasheet en la figura 24, sin cálculo específico.

Las resistencias de pull-up R10 y R11, tienen un valor de 1 MOhm tal y como recomienda el fabricante.

2.2.2. CARGA DE BATERÍA

El mote base incluye un circuito de carga para la batería de Li+ basado en el integrado MAX1555 de Maxim.

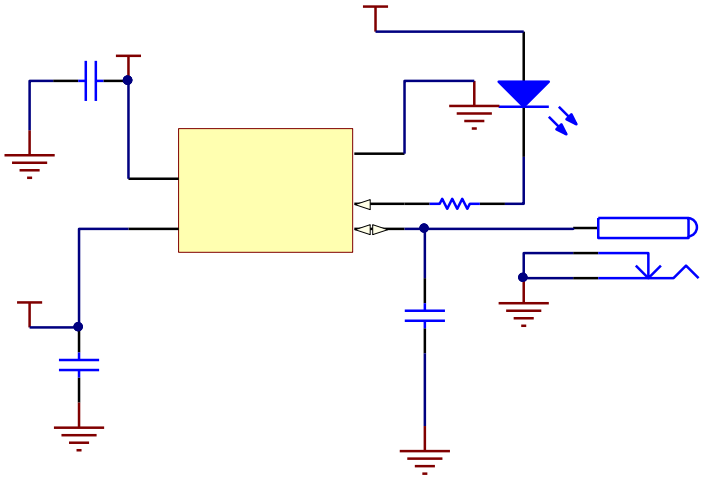


FIGURA 22: CONFIGURACIÓN DEL CARGADOR DE BATERÍAS DE LI+, MAX1555

Este dispositivo permite cargar baterías de una celda a partir de dos fuentes distintas, un puerto USB o una fuente de alimentación externa y se han implementado ambas, dado que apenas se requieren componentes externos, como una forma de ofrecer flexibilidad en este aspecto.

El conector específico para la fuente de alimentación es un terminal de barril con un diámetro del pincho de 2mm estándar.

El diseño, incluye un indicador de carga de batería, de tal forma que está activo mientras dura el proceso de carga y se desactiva una vez la batería alcanza el valor nominal.

Todos los condensadores externos se han elegido de acuerdo con la recomendación del fabricante en el Datasheet:

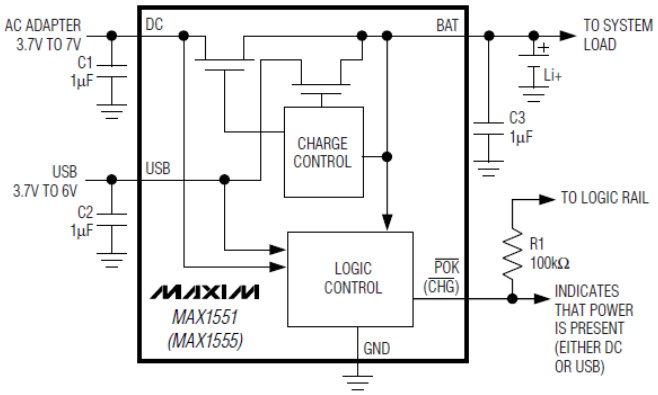


FIGURA 23: APLICACIÓN TÍPICA DEL MAX1555

2.2.3. PROCESAMIENTO

El microcontrolador elegido ha sido el Atmel Atmega 168, funcionando a una frecuencia de reloj de 8Mhz y a 3.3 voltios.

Los aspectos relacionados con el consumo se han detallado en el apartado 2.2.1.

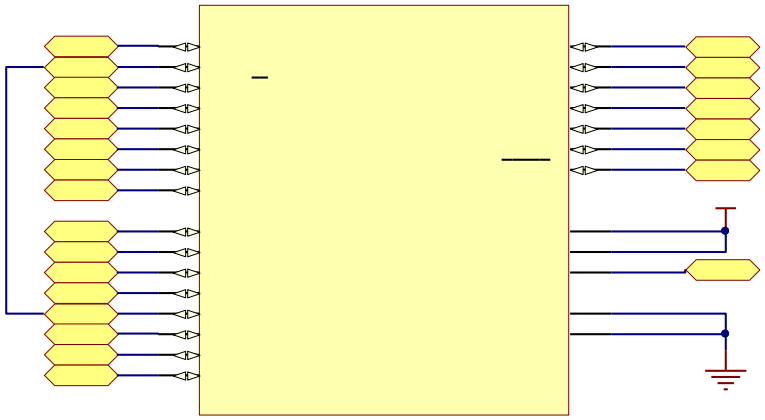


FIGURA 24: MICROCONTROLADOR ATMEGA 168

La mayoría de los terminales de I/O están mapeados a terminales externos conectables, que sirven de base a las PCB funcionales, de tal forma que establecen una conexión robusta entre los módulos.

Para establecer el clock del sistema se ha elegido un cristal de cuarzo de 8Mhz, lo que supone un bajo consumo y susceptibilidad a ruido. Dado que el ruido en principio no es un elemento cuya incidencia sea relevante, pues se trata de un diseño de baja frecuencia, prevalece el factor del bajo consumo.

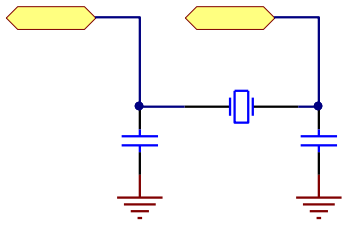


FIGURA 25: CRISTAL DE CUARZO Y CONDENSADORES PARA EL ATMEGA 168

No obstante es necesaria la colocación de dos condensadores en el rango 12 – 22 pF tal y como indica la figura 24, extraída del apartado "8.3 Low power crystal oscillator", tomando el valor de 20 pF.

Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 (pF)
0.4 - 0.9	–
0.9 - 3.0	12 - 22
3.0 - 8.0	12 - 22
8.0 - 16.0	12 - 22

FIGURA 26: RANGOS PARA LOS CONDENSADORES DEL CRISTAL

Dado que se trata de un dispositivo digital de tecnología CMOS, el consumo de corriente se produce en las transiciones entre estados lógicos, y aunque se puede establecer un consumo promedio, lo picos de corriente pueden provocar efectos adversos importantes si no se controlan. Para ello no basta con insertar un elemento capacitivo a la salida del sistema de alimentación, pues la distancia puede ser lejana y crear un bucle de corriente largo que incremente los efectos negativos.

Para gestionar los picos de corriente transitoria que suceden durante el normal funcionamiento, cuando se activa un periférico o un puerto I/O, se ha incorporado un condensador de 100 nF, para proporcionar esa corriente instantánea que la línea no es capaz de asumir.

También en la entrada de referencia del conversor A/D se ha incluido un condensador de 20pF para evitarla introducción de ruido al conversor. No obstante este terminal también está mapeado en un terminal externo por lo que no queda inhabilitado.

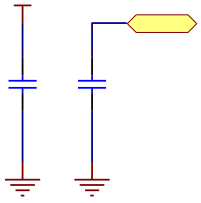


FIGURA 27: CONDENSADOR DE DESACOPLO DEL ATMEGA 168 Y CONDENSADOR DE FILTRADO PARA LA REFERENCIA ANALÓGICA

En ambos casos, para minimizar el bucle de corriente, el placement de ambos condensadores se debe realizar lo más cercano posible a los terminales que afecta, como se puede apreciar en la figura 28.

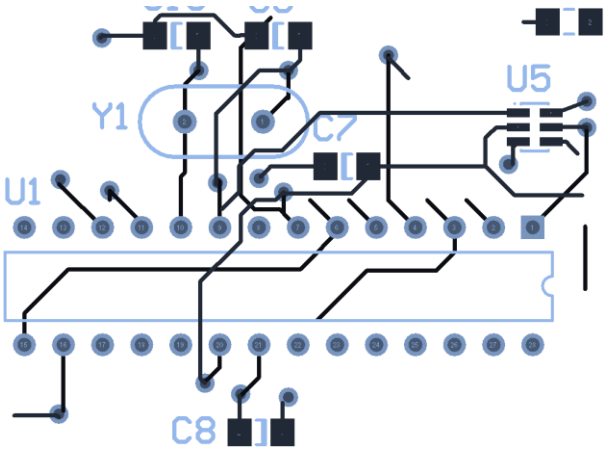


FIGURA 28: DETALLE DE RUTEO DEL CRISTAL PRÓXIMO AL MICROCONTROLADOR

El circuito de reset está formado por dos partes, un reset manual y un reset automático, para permitir algunas funciones de reprogramación inalámbrica.

El circuito de reset manual es el formado por el típico pulsador y el circuito RC que muestra la figura 29, en la que el puerto DTR está conectado a masa:

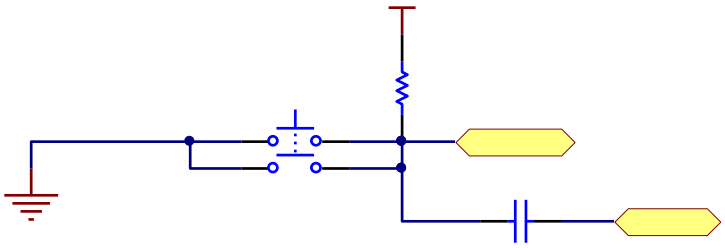


FIGURA 29: PULSADOR DE RESET MANUAL

Por otro lado, para el reset automático se ha empleado un switch controlado digitalmente por uno de los terminales del Atmega 168, el dispositivo ADG719 de Analog Devices.

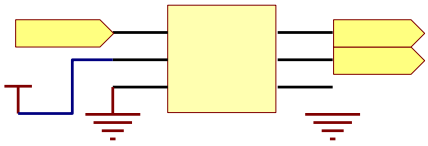


FIGURA 30: SWITCH CONTROLADO POR MICROCONTROLADOR PARA RESET AUTOMÁTICO

El terminal 1 es el de control, que está controlado por el terminal ICP del microcontrolador.

En estado de reposo, es decir, con ICP en nivel bajo, la salida RST del ADG719 está conectada al terminal 4, que a su vez corresponde a la salida del reset manual. Por defecto, el reset manual está preparado en el switch. Ahora, cuando el terminal ICP pasa a estado alto, el ADG719, cablea a la salida RST directamente a GND, lo que provoca el reset. Como el terminal ICP es controlable por programa, se trata de un reset por software.

2.2.4. COMUNICACIÓN

El bloque de comunicación es idéntico al explicado en el apartado 2.1.2, por lo que se remite al lector a dicho apartado.

El mote base incluye un bloque de 4 interruptores, empleados para permitir la configuración de modos de bajo consumo del transceptor XBee. De los cuatro interruptores, los conectados a RX y TX, siempre van a estar activos, pues son fundamentales para la comunicación.

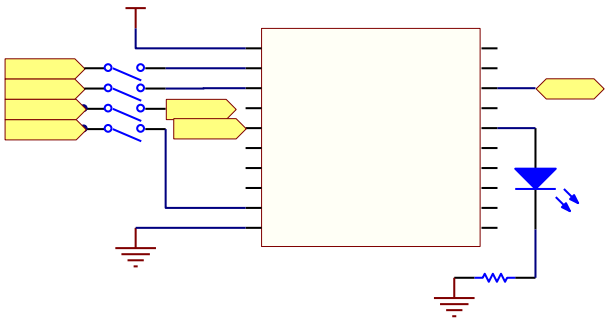


FIGURA 31: TRANSCEPTOR XBEE Y CONMUTADORES DE CONFIGURACIÓN

Para permitir al transceptor XBee los modos Hibernate y Pin Doze, de muy bajo consumo se habilita el terminal Sleep_RQ a través del bloque de interruptores, para que sea controlado por el terminal SCK del Atmega 168. De esta forma, en las aplicaciones en las que se pueda hacer uso de estos modos, el terminal está habilitado. Por otra parte, el microcontrolador, también podría emplear el bloque I2C, por lo que sería necesario el uso del terminal SCK, y deshabilitando este en el bloque de interruptores quedaría disponible.

3. FLUJOGRAMAS DE PROGRAMACIÓN

3.1. SISTEMA LOCAL

3.1.1. COMUNICACIÓN SERIE CON XBEE

Como ya se ha mencionado, una de las principales funciones del sistema local es la coordinación de los motes de la red o PAN. Esto se consigue, desde el punto de vista físico accediendo a la red mediante un transceptor inalámbrico.

Para ello se ha construido una PCB que incorpora un transceptor XBee y un adaptador serie-usb FT232RL para poder conectar el XBee con el PC a través de un puerto USB.

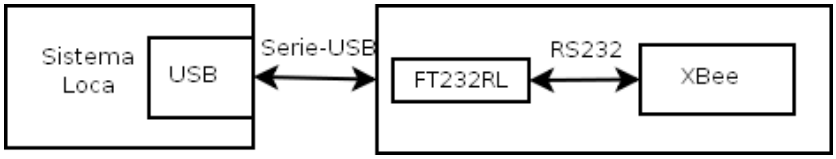


FIGURA 32: ESQUEMA DE COMUNICACIÓN PC CON XBEE

La alimentación del sistema la realiza el puerto USB del PC, por lo que se establece un límite máximo de corriente por la especificación USB de unos 500 mA. Dado el consumo límite del XBee entorno a 50 mA en el peor de los casos y el del FTDI que está en los 15 mA en operación normal, queda un margen muy amplio hasta el límite proporcionado por el terminal USB.

El XBee actúa como coordinador de la red de motes usando una topología en estrella centralizada, por lo que todos los paquetes pasan por él.

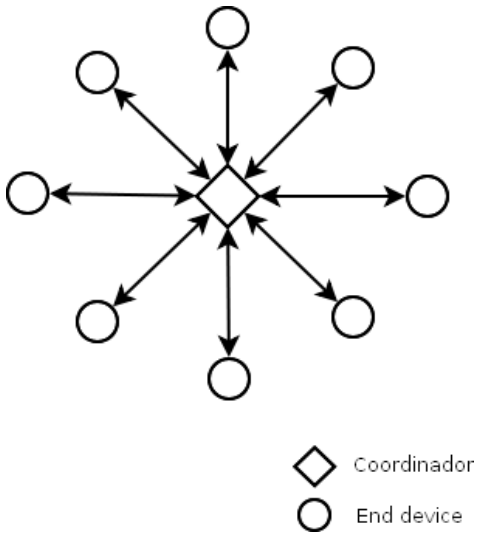


FIGURA 33: TOPOLOGÍA DE LA RED DE MOTES

De esta forma, toda la información que el XBee envíe a través de su bloque de comunicaciones serie, será recibida en el PC y estará accesible para la aplicación.

La aplicación del sistema local, actualmente está implementada en un PC, con un sistema operativo Linux, usando el lenguaje de programación Java. Se hace imprescindible el uso de una librería específica que realice dos funciones esenciales: hacer de interfaz entre el XBee y nuestra aplicación y que permita manejar las tramas estructuradas de datos que maneja el XBee, a nivel de orientación a objetos.

Para lograr todo esto se ha utilizado la única librería libre y open source que hasta este momento existe, XBee-API (http://code.google.com/p/XBee-api/), que implementa la mayoría de la funcionalidad del XBee, en lenguaje Java. Internamente se apoya en la librería también libre y open source, RXTX que da soporte de bajo nivel para el acceso a los puertos USB en lenguaje Java, usando código nativo (JNI).

Una vez conectado el XBee a través del FTDI al puerto USB, dinámicamente el sistema operativo reconoce el conjunto como un dispositivo con comunicaciones serie y lo monta, asignándole recursos en el área de dispositivos (devices).

3.1.2. GESTIÓN DE TRAMAS DE RED

Todas las tramas que llegan al coordinador, son enviadas a través del puerto USB, de tal forma que generan una interrupción en el sistema operativo. Esta interrupción es propagada hasta la JVM, de tal forma que el núcleo de la librería RXTX es notificado del hecho de la interrupción y la atiende, recogiendo los datos sin un formato específico, como una ristra de bytes. Estos bytes son nuevamente recogidos por la librería XBee-api, que los encapsula en las estructuras de tramas que marca la especificación del XBee. Además de encapsular la información de forma estructurada, provee de métodos y funciones para manejar con cierta facilidad a alto nivel dichas tramas.

En principio con todo lo anterior ya estaríamos en disposición de manejar desde la aplicación las tramas que circulan por la red. Ahora bien, hay que tener en cuenta los posibles cuellos de botella que existen desde que el XBee recibe la información hasta que esta es procesada y almacenada en su caso.

Cada vez que se recibe una trama, se genera una interrupción en el sistema y esta es finalmente atendida por la aplicación. La "atención" a las tramas que llegan implica que la aplicación está dedicada durante cierto tiempo a recibir la trama, a encapsularla y a realizar el procesamiento adecuado y todo esto lleva un tiempo, que no es siempre el mismo para todas las tramas. Aunque en principio el tráfico de datos que circula por la red de motes va a ser bajo, por la propia concepción de la red, puede darse el caso de que se iniciaran múltiples activaciones consecutivas y por lo tanto interrupciones en el sistema, mínimamente espaciadas en el tiempo, lo que puede suponer un problema, pues podrían llegar a perderse cuando el procesamiento individual es largo.

A nivel de aplicación este cuello de botella se soluciona o al menos se minimiza en buen grado, usando múltiples hilos de ejecución (threads), incrementando con ello el grado de paralelismo con el que las tareas se ejecutan. Esto se consigue haciendo uso de las utilidades para concurrencia que Java ofrece y en este sentido cabe destacar la magnífica implementación desarrollada por Doug Lea, incorporada a partir de la versión 1.5 de la JDK.

En nuestro caso, se hace uso de lo que se conoce como una "pool" de threads (ThreadPoolExecutor), un lugar donde existen instancias ya creadas de threads, listas para ser usadas. Cuando la aplicación recibe una trama, antes de iniciar el procesamiento, recupera un thread de la pool, le asigna una tarea y lanza la ejecución. Cuando la ejecución termina, el thread es liberado y devuelto a la pool. Con esto conseguimos minimizar la latencia o el tiempo de respuesta ante los eventos, por dos motivos, uno, la pool de threads no crea nuevos

threads a menos que se vacíe, con lo que se elimina el coste de creación de objetos y dos, podemos tener múltiples threads corriendo en pseudo paralelismo.

XBEEPACKETLISTENER.JAVA

```
/**
 * Interceptor de trama recibida.
 * Identifica la trama y delega el procesamiento en el parser adecuado.
 * En toda la aplicación solo vamos a tener una instancia de esta clase
 * y por ello usamos un Singleton
 * Para mejorar el rendimiento y la respuesta ante los eventos,
 * usa una Pool de Threads, que lanza un thread por cada petición.
 */
public class XBeePacketListener implements PacketListener {

	private static final Logger log = Logger.getLogger(XBeePacketListener.class);
	private static final XBeePacketListener listener = new XBeePacketListener();
	private static ThreadPoolExecutor pool = (ThreadPoolExecutor)Executors.newCachedThreadPool();

	private XBeePacketListener() {
	}

	public static XBeePacketListener getInstance(){
		return listener;
	}

	public void processResponse(XBeeResponse response) {
		log.debug("Paquete recibido...inicia worker thread");
		pool.execute(new ResponseReaderFactory().getResponseReader(response));
	}
}
```

Como se puede apreciar en el fragmento anterior, la llamada a pool.execute(..) tiene como argumento una tarea, es decir algo ejecutable. Esto en Java recibe el nombre de Runnable y es una interfaz con un único método, run(), el cual es llamado por el thread que contiene la tarea.

El siguiente esquema de clases, refleja las relaciones y los contratos entre el listener proporcionado por la librería XBee-api y la implementación propia.

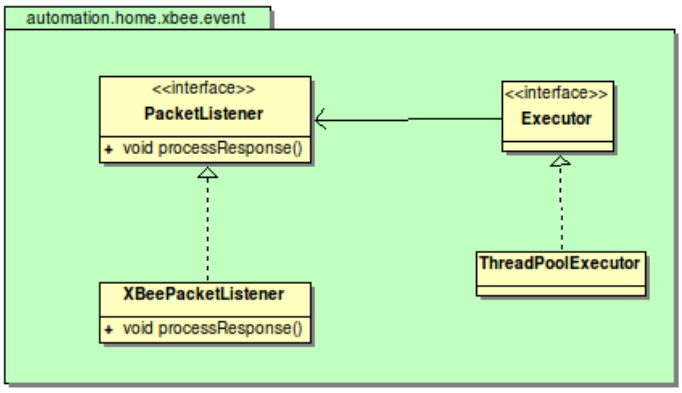


FIGURA 34: RELACIONES DE HERENCIA PARA EL LISTENER DE PAQUETES

Para poder atender a los eventos generados por la recepción de tramas en el dispositivo XBee, en la aplicación se ha de crear una instancia de una clase que sea capaz de gestionarlo. Como ya se ha mencionado, haciendo uso de la librería XBee-API, se emplea la clase XBee:

```
...
XBee xbee = new XBee();
...
```

Esta clase es una capa de abstracción sobre la proporcionada por la librería RXTX, para el manejo de datos por el puerto serie. Ofrece por lo tanto un mecanismo para manejar las interrupciones generadas, que en lenguajes de algo nivel se conoce como Listener.

El uso de este método va ligado con todo lo anterior y su uso es como sigue:

```
...
XBee.addPacketListener(XBeePacketListener.getInstance());
...
```

La secuencia de ejecución desde el punto de vista de la aplicación queda de la siguiente forma:

- Creación de una instancia para la clase XBee.
- Creación de una instancia para el listener de paquetes.
- Añadir el listener a la instancia de la clase XBee.

Con esto ya tenemos el mecanismo base para poder trabajar a alto nivel con el transceptor XBee.

El siguiente paso consiste en leer y trabajar con las tramas recibidas.

Nuevamente se ha pensado en un sistema genérico y compatible con los dispositivos XBee disponibles en el mercado. DIGI fabrica dispositivos XBee para 802.15.4, ZigBee y la variante Znet y aunque no son compatibles, nuestra aplicación ofrece la posibilidad de usar la tecnología que más convenga en cada caso. De esta forma si un problema concreto requiere del uso de ZigBee cuando 802.15.4 no cubre las necesidades, la aplicación está preparada para ello.

La problemática de una implementación genérica está en crear un sistema mínimo para procesar las tramas que sea común a todos los casos. Esto en programación orientada a objetos se puede concretar en interfaces. Una interfaz es un contrato para todos aquellos que se adhieran a ella, es decir, deben implementar de forma concreta el comportamiento que la interfaz propone.

Se ha creado una clase que proporciona la funcionalidad requerida por ThreadPoolExecutor y que además establece un criterio común para el procesamiento de tramas, en forma de método abstracto, que implementarán los lectores concretos para las tramas que se reciben.

RESPONSEREADER.JAVA

```
/**
 * Clase base para los lectores especializados en tramas concretas.
 *
 * Hace de plantilla para los que extiendan esta clase, que deben
 * implementar el método read(), para leer la trama recibida.
 * Implementa la interfaz Runnable, necesaria para el ThreadPoolExecutor.
 */

public abstract class ResponseReader implements Runnable {

	private final XBeeResponse response;

	public ResponseReader(XBeeResponse response) {
		this.response = response;
	}

	public XBeeResponse getResponse() {
		return response;
	}

	protected abstract void read();

	public void run() {
		this.read();
	}
}
```

Esta hereda la propiedad de ser ejecutable dentro de un thread al implementar Runnable y además proporciona una plantilla en forma de método read(), para las clases que la extienden, es decir, read() es un método que implementará una clase que esté preparada para trabajar con 802.15.4 pero que también implementará una clase que trabaje con Znet, eso sí, cada implementación será particular.

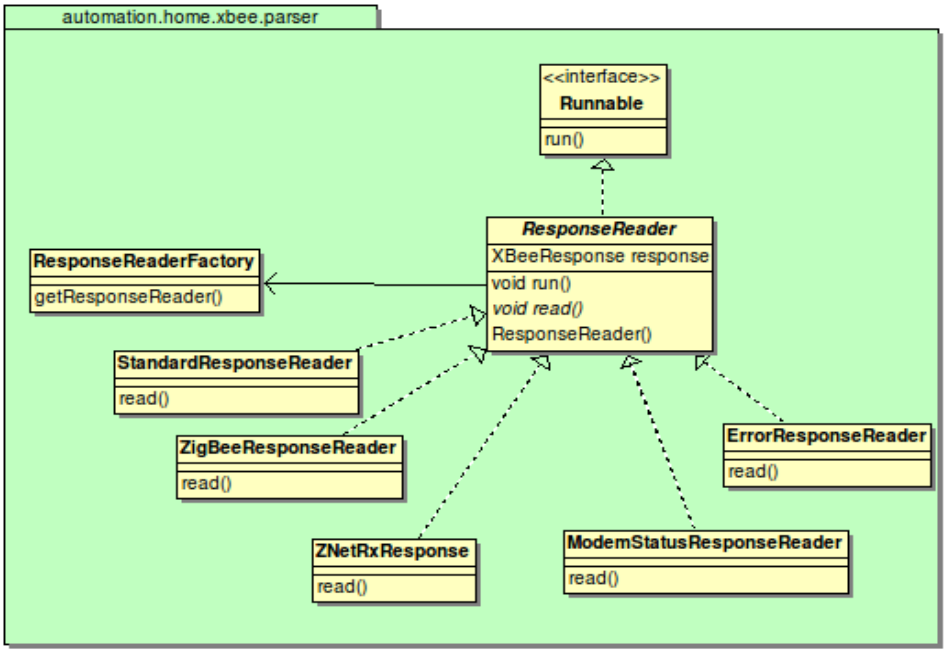


FIGURA 35: IMPLEMENTACIÓN BASE DEL LECTOR DE PAQUETES Y LAS POSIBLES IMPLEMENTACIONES

En el diagrama presentado se reflejan las relaciones de herencia e implementación entre las clases generadas, donde se pueden apreciar un lector de tramas especializado en 802.15.4 (StandardResponseReader) u otro para ZigBee (ZigBeeResponseReader).

Hay que proporcionar una forma elegante para crear instancias de estas clases, lo que generalmente se suele realizar con una "Factoría" y se ha creado la clase ResponseReaderFactory con ese propósito. Para la implementación tenemos la ayuda que la librería XBee-api nos proporciona en forma de tipos de trama, es decir, en el proceso de estructurar la información, crea un tipo concreto de trama, como puede ser RxResponse para 802.15.4 o ZnetRxResponse para XBee y Znet. Con esto es bastante sencillo generar el código de la factoría.

RESPONSEREADERFACTORY.JAVA

```
public class ResponseReaderFactory {

	private static final Logger log = Logger.getLogger(ResponseReaderFactory.class);

	public ResponseReader getResponseReader(XBeeResponse response) {
		log.debug("Recibida una trama " + response.getApiId());
		if (response instanceof RxResponse || response instanceof ZNetRxResponse) {
			log.debug("Iniciando el lector StandardResponseReader");
			return new StandardResponseReader(response);
		} else if (response instanceof ModemStatusResponse) {
			log.debug("Iniciando el lector ModemStatusResponseReader");
			return new ModemStatusResponseReader((ModemStatusResponse) response);
		} else if (response instanceof ErrorResponse) {
			log.debug("Iniciando el lector ErrorResponseReader");
			return new ErrorResponseReader((ErrorResponse) response);
		} else if (response instanceof ZNetNodeIdentificationResponse) {
			log.debug("Iniciando el lector ZNetNodeIdentificacionResponseReader");
			return new ZNetNodeIdentificacionResponseReader((ZNetNodeIdentificationResponse)
response);
		} else if (response instanceof XBeeFrameIdResponse) {
			log.debug("Iniciando el lector XBeeFrameIdResponseReader");
			return new XBeeFrameIdResponseReader((XBeeFrameIdResponse) response);
		} else {
			throw new RuntimeException("Respuesta desconocida");
		}
	}
}
```

Ya tenemos una forma de crear lectores especializados para las distintas especificaciones que DIGI ofrece en sus XBee y ahora se va a mostrar la implementación que se ha realizado sobre el caso concreto que nos atañe, que como se ha indicado anteriormente y como enuncia el título del presente proyecto, es 802.15.4.

STANDARDRESPONSEREADER.JAVA

```
public class StandardResponseReader extends ResponseReader {

	private static final Logger log = Logger.getLogger(StandardResponseReader.class);

	public StandardResponseReader(XBeeResponse response) {
		super(response);
	}

	public void read() {
		XmlDataFormat xmlDataFormat =
					new XmlDataFormatParser(super.getResponse()).parse();
		XmlResponseParser xmlResponseParser =
		new XmlResponseParserResolver().resolve(xmlDataFormat, super.getResponse());
		IResponse response = xmlResponseParser.parse();
		if (response instanceof AssociationResponse) {
			log.debug("Recibida una trama de asociación");
			AssociationResponse associationResponse =
					(AssociationResponse) xmlResponseParser.parse();
			log.debug("Información de la asociación: " +
					associationResponse.toString());
			//Persiste la información en db
			log.debug("Persiste el mote asociado en la DB");
			new DbMoteDAO().save(associationResponse.getMote());
		} else if (response instanceof SensorMeasureResponse) {
			log.debug("Recibida trama de medidas sensoriales");
			SensorMeasureResponse sensorMeasureResponse =
					(SensorMeasureResponse) xmlResponseParser.parse();
			log.debug("Información de las medidas: " +
					sensorMeasureResponse.toString());
			//Persiste la información en db
			log.debug("Persiste la información en la DB");
			new DbMeasureDAO().save(sensorMeasureResponse.getMote(),
						sensorMeasureResponse.getMeasures());
			log.debug("Comprueba las alertas programadas");
			new AlarmCheck(sensorMeasureResponse).checkAlarms();
		}
	}
}
```

Como se puede observar, la clase hereda efectivamente de ResponseReader, e implementa el método read() que lo que hace es evaluar el contenido de la trama para determinar si se trata de una trama de asociación o de una trama de datos, para en cada caso realizar las acciones necesarias. Para el caso de una trama de asociación (AssociationResponse), espera un contenido específico que identifique al nodo con sus características, esto es, sensores y actuadores instalados. Para el caso de una trama de medición (SensorMeasureResponse), espera un contenido con las mediciones asociadas a los sensores instalados en un mote. A continuación se indica la especificación que han de cumplir estas tramas en cuanto a formato, ya que son generadas en los motes y leídas en el sistema local, por lo que deben seguir unas reglas concretas.

El contenido o payload de las tramas enviadas tiene formato XML (Extensible Markup Language) que nos permite definir elementos y atributos en forma de árbol, mediante el uso de etiquetas. Actualmente es un mecanismo estándar para el intercambio de información entre sistemas heterogéneos.

Se ha generado la siguiente especificación en XML para el intercambio de información entre los motes y el coordinador, de tal forma que, además de permitir usar las implementaciones de DIGI, por separado claro, se crea un sistema de contenido de tramas formal para poder distinguir qué información es la que se envía por la red. Por ahora se han generado dos especificaciones, una para las tramas de asociación y otra para las tramas de sensado.

- Asociación

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:attribute name="fmt" minOccurs="0" maxOccurs="1">
		<xs:restriction base="xs:string">
			<xs:pattern value="1"/>
		</xs:restriction>
	</xs:attribute>
	<xs:sequence>
		<xs:element name="sensor" type="sensor" minOccurs="0"/>
		<xs:element name="actuator" type="actuator" minOccurs="0"/>
	</xs:sequence>
</xs:schema>
<xs:complexType name="sensor">
	<xs:attribute name="id">
		<xs:restriction base="xs:string">
			<xs:pattern value="[0-9A-F]{4}"/>
		</xs:restriction>
	</xs:attribute>
</xs:complexType>
<xs:complexType name="actuator">
	<xs:attribute name="id">
		<xs:restriction base="xs:string">
			<xs:pattern value="[0-9A-F]{4}"/>
		</xs:restriction>
	</xs:attribute>
</xs:complexType>
```

Con este esquema se puede constuir una trama como la siguiente:

```
<mote fmt=”1”>
	<sensor id=”12”/>
	<sensor id=”25”/>
</mote>
```

Indica que el mote tiene dos sensores instaladados, cuyos identificadores únicos en el sistema son 12 y 25. Por supuesto no se envía toda la información relacionada con los sensores pues esta ya estará en la base de datos. De esta forma, cuando se recibe esta información en el coordinador, podemos determinar que se trata de una trama de asociación y por lo tanto configurar automáticamente el mote con los sensores y actuadores que tiene instalados.

- Sensado

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:attribute name="fmt" minOccurs="0" maxOccurs="1">
		<xs:restriction base="xs:string">
			<xs:pattern value="2"/>
		</xs:restriction>
	</xs:attribute>
	<xs:sequence>
		<xs:element name="sensor" type="sensor" minOccurs="0"/>
		<xs:element name="actuator" type="actuator" minOccurs="0"/>
	</xs:sequence>
</xs:schema>
<xs:complexType name="sen" minOccurs="0">
	<xs:attribute name="id">
		<xs:restriction base="xs:string">
			<xs:pattern value="[0-9A-F]{4}"/>
		</xs:restriction>
	</xs:attribute>
	<xs:attribute name="c">
		<xs:restriction base="xs:string">
			<xs:pattern value="+-[0-9]{2}[.][0-9]{2}"/>
		</xs:restriction>
	</xs:attribute>
	<xs:attribute name="b">
		<xs:restriction base="xs:string">
			<xs:pattern value="[0-1]"/>
		</xs:restriction>
	</xs:attribute>
</xs:complexType>
```

Con este esquema se pueden construir una trama como la siguiente:

```
<mote fmt="2">
	<sensor id="12" c_meas="-12.234"/> //Sensor analógico
	<sensor id="25" b_meas="1"/>		//Sensor binario
</mote>
```

Indica las medidas que darían los sensores del mote del ejemplo anterior, siendo el sensor con id 12, un sensor que da una medida analógica y el sensor con id 25 es un sensor todo/nada.

Los dos esquemas anteriores se emplearían para validar el XML que llega, de tal forma que debe cumplir las reglas que marca. Si no las cumple, entonces no es capaz de validar y no se da por bueno el contenido de la trama. Esto realmente no se ha implementado para no añadir más capas a la aplicación, aunque en condiciones normales podría ser una medida que aportara robustez, siempre y cuando los esquemas permanezcan sencillos.

En base al análisis anterior, se ha creído conveniente el crear una herencia sencilla, de forma que haya una única interfaz para los métodos que manejan estas tramas y el siguiente diagrama lo refleja.

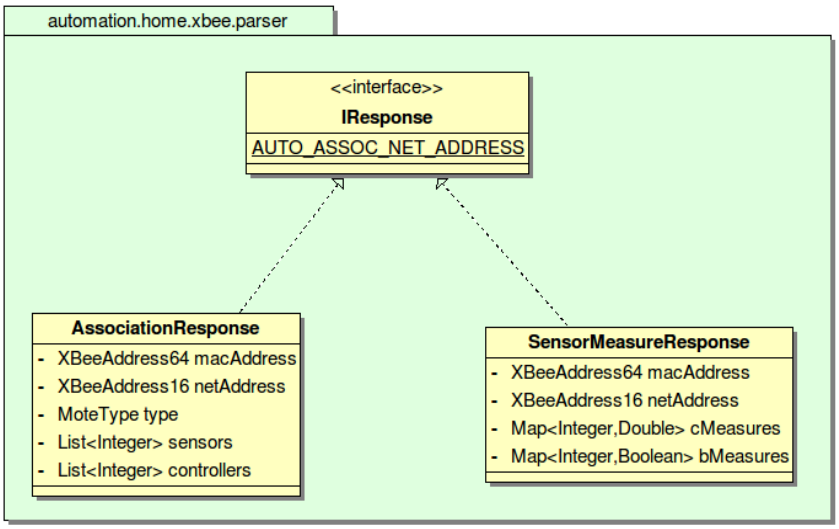


FIGURA 36: HERENCIA PARA LOS PAYLOAD

La clase que encapsula la información de asociación, contiene los siguientes atributos, mostrados en el diagrama anterior:

- macAddress: cuyo tipo es una dirección de 64 bits encapsulada a su vez en la clase XbeeAddress64.
- netAddress: cuyo tipo es una dirección de 16 bits encapsulada a su vez en la clase XbeeAddress16.
- type: tipo del mote que puede ser coordinador, sensor o controlador.
- sensors: lista con los identificadores de los sensores instalados en el mote.
- controllers: lista con los controladores instalados en el mote

ASSOCIATIONRESPONSE.JAVA

```
/**
 * Clase que encapsula la información de la trama de asociación.
 *
 * El parseador de la trama de asociación {@link XmlStandardAssociationResponseParser}
 * y el manejador del parser {@link XmlAssociationResponseHandler}, usan esta clase
 * para almacenar la información.
 */
public class AssociationResponse implements IResponse {

	private XBeeAddress64 macAddress;
	//Se la asignamos por defecto
	private XBeeAddress16 netAddress = AUTO_ASSOC_NET_ADDRESS;
	private MoteType type;				//Tipo de mote según el parseo del xml
	private List<Integer> sensors;
	private List<Integer> controllers;

	//Métodos set y get
}
```

La clase que encapsula la información de asociación, contiene los siguientes atributos, mostrados en el diagrama anterior:

- macAddress: cuyo tipo es una dirección de 64 bits encapsulada a su vez en la clase XbeeAddress64.
- netAddress: cuyo tipo es una dirección de 16 bits encapsulada a su vez en la clase XbeeAddress16.
- cMeasures: Mapa con las medidas contínuas de cada sensor.
- bMeasures: Mapa con las medidas binarias de cada sensor.
-

SENSORMEASURERESPONSE.JAVA

```
/**
 * Datos enviados por un mote sensor.
 */
public class SensorMeasureResponse implements IResponse {

	private XBeeAddress64 macAddress;
	//Se la asignamos por defecto
	private XBeeAddress16 netAddress = AUTO_ASSOC_NET_ADDRESS;
	private Map<Integer, Double> c_measures;
	private Map<Integer, Boolean> b_measures;

	//Métodos set y get
}
```

3.1.3. ARQUITECTURA SERVIDOR REMOTO

Cada uno de los sistemas locales es al mismo tiempo un servidor remoto, de tal forma que con una conexión de red, se permite el acceso a ciertas funciones restringidas. El caso más significativo es que se permite acceder a los dispositivos de la red que realizan tareas de control, como pueden ser motes que gestionan la iluminación.

Se ha empleado la tecnología cliente-servidor que ofrece Java denominada RMI o Remote Method Invocation. RMI está basada en el paso de objetos entre distintas máquinas, así como la invocación de métodos y la recepción de resultados en el cliente.

Para lograr el paso de objetos entre máquinas virtuales diferentes, evidentemente es necesario un servidor, el cual ofrece una interfaz que define un conjunto de métodos con parámetros y resultados, que está accesible para los clientes que conecten con ese servidor.

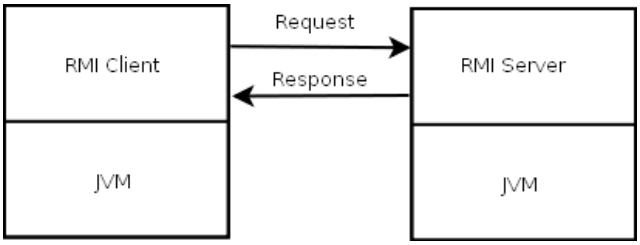


FIGURA 37: ARQUITECTURA RMI SOBRE DOS MÁQUINAS VIRTUALES DIFERENTES

Por otro lado, los clientes tienen que ser capaces de contactar con el servidor y cumplir los requisitos de acceso y seguridad, si es que los hay.

Hay que hacer la matización de que la comunicación o paso de objetos entre cliente y servidor no es directa, sino que se usan unas entidades intermedias denominadas Proxy, que se ocupan de la problemática de la comunicación a bajo nivel. Estas entidades Proxy, se ubican tanto en el cliente como en el servidor y delegan las peticiones y los resultados a servidor y cliente respectivamente.

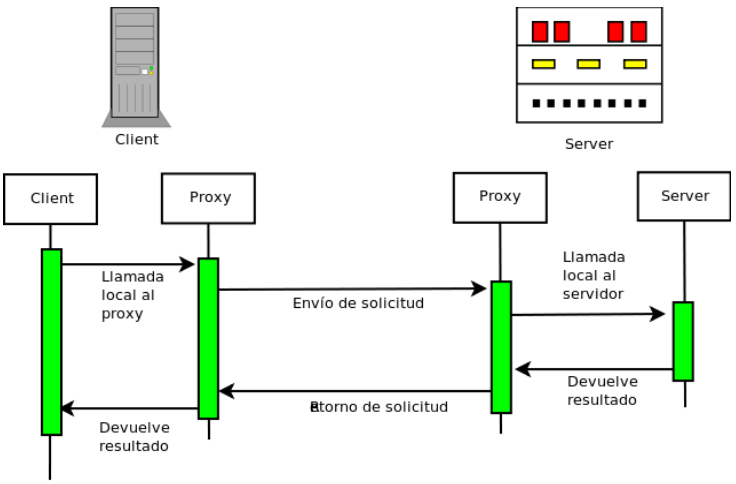


FIGURA 38: DIAGRAMA DE SECUENCIA DE UNA SOLICITUD REMOTA

La especificación RMI obliga a que se extienda la interfaz Remote para poder codificar el servidor, ya que esta permite la creación de métodos que se pueden invocar remotamente.

Así por ejemplo, para aquellos sistemas que empleen 802.15.4 en su red de motes, podrían tener una especificación del servidor remoto como esta:

XBEESTANDARDSERVICE.JAVA

```
/**
 * Interfaz de servicio remoto pasa sistemas que operan con el estándar
 * 802.15.4.
 *
 * El servicio RMI estará corriendo en la instalación inalámbrica y los
 * clientes remotos interactuarán con la instalación con esta interfaz.
 */
public interface XBeeStandardService extends Remote {

	public static final int networkID = 1; //Id de la red

	/**
	 * Envía una trama con el formato TxRequest16
	 *
	 * @param addressLow byte de menor peso de la dirección destino
	 * @param addressHigh byte de mayor pese de la dirección destino
	 * @param payload datos efectivos que contiene la trama
	 * @param options opciones de envío(ver:{@link Option})
	 * @param frameId identificador para la trama
	 * @throws RemoteException
	 */
	void sendTxRequest16(int addressLow, int addressHigh, int[] payload, int options,int frameId)
throws RemoteException;

	/**
	 * Envía una trama con el formato TxRequest64
	 *
	 * @param macAddress dirección mac del destinatario
	 * @param payload datos efectivos de la trama
	 * @param options opciones de envío(ver:{@link Option})
	 * @param frameId identificador para la trama
	 * @throws RemoteException
	 */
	void sendTxRequest64(String macAddress, int[] payload, int options, int frameId) throws
                                                                                RemoteException;
}
```

Esta es una interfaz sencilla para un servidor remoto que opera con 802.15.4 y que ofrece dos métodos para enviar remotamente tramas, empleando bien direcciones de 16 bits o direcciones de 64 bits. La implementación de los métodos remotos es bastante sencilla ya que se hace uso de la librería XBee-api y en concreto de la clase XBee de la que el servidor contiene una instancia.

XBEESTANDARDSERVICEIMPL.JAVA

```
public class XBeeStandardServiceImpl extends UnicastRemoteObject
                                                    implements XBeeStandardService {

        private static final long serialVersionUID = 1523090807995267859L;
        private static final Logger log = Logger.getLogger(XBeeStandardServiceImpl.class);
        private static final XBee xbee = new XBee();

        public void sendTxRequest16(int addressLow, int addressHigh, int[] payload,
                                    int options, int frameId) throws RemoteException {
                XBeeAddress16 xBeeAddress16 = new XBeeAddress16(addressHigh, addressLow);
                TxRequest16 txRequest16 = new TxRequest16(xBeeAddress16, frameId,
                                                    Option.get(options), payload);
                try {
                        XBee.sendAsynchronous(txRequest16);
                } catch (XBeeException ex) {
                        log.error(ex);
                }
        }

        public void sendTxRequest64(String macAddress, int[] payload, int options,
                                                    int frameId) throws RemoteException {
                XBeeAddress64 xBeeAddress64 = new XBeeAddress64(macAddress);
                TxRequest64 txRequest64 = new TxRequest64(xBeeAddress64, frameId,
                                                    Option.get(options), payload);
                try {
                        XBee.sendAsynchronous(txRequest64);
                } catch (XBeeException ex) {
                        log.error(ex);
                }
        }
}
```

Ya tenemos la funcionalidad externa mediante la cual, un cliente remoto puede enviar tramas con formatos de 16 o 64 bits a la red de motes que gestiona el sistema local.

Aprovechando la implementación de un servidor remoto, ya que es un servicio que está permanentemente activo, podemos añadir el resto de la lógica para la autoconfiguración de los motes.

3.1.4. AUTOCONFIGURACIÓN A ALTO NIVEL

Se entenderá el término autoconfiguración a alto nivel como el proceso por el cual el software de aplicación al disponer de la información necesaria, es capaz de identificar cuando un mote se añade a la red y cuál es su configuración de sensores y actuadores. Además se han de tener en cuenta otros casos como el posible reinicio de la red o la sustitución del coordinador por avería.

El proceso de autoconfiguración comienza cuando arranca el servidor local, detectando el dispositivo XBee coordinador que está conectado al puerto USB. Básicamente se puede esquematizar en el siguiente diagrama que a posteriormente se explica con fragmentos de código:

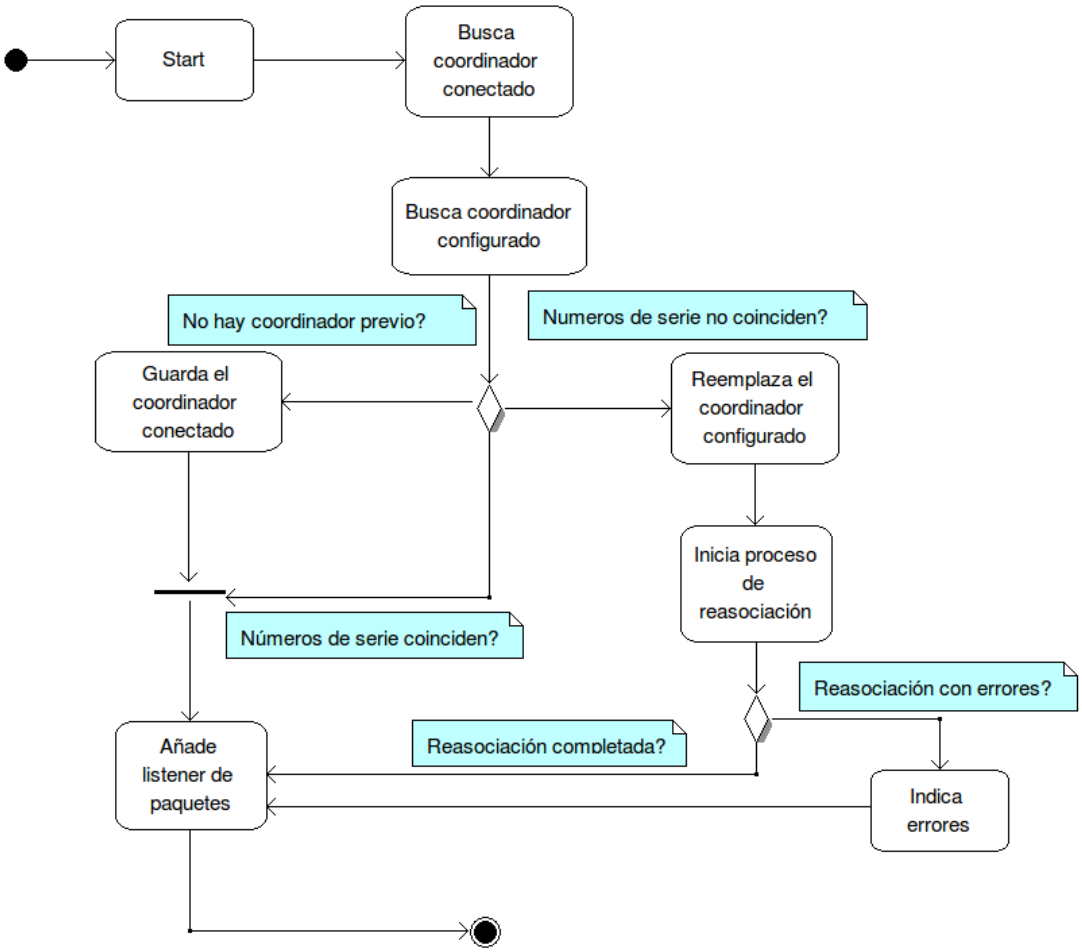


FIGURA 39: FLUJOGRAMA DEL ARRANQUE DEL SERVICIO REMOTO

1. Al arrancar el sistema, se detecta el coordinador que está conectado al puerto USB y mediante comandos AT, se obtiene el número de serie. Los comandos AT están especificados en la documentación que aporta DIGI y en concreto son SH, para obtener la parte alta del número de serie y SL para la parte baja. Este número de serie tiene 64 bits y cada parte está formada por 32 bits.

```
log.info("Iniciando el servicio remoto del coordinador XBee");
//Obtiene la información de la red a partir de la existente en la base de datos
//de la aplicación global, que está en nuestro servidor
Network network = new DbNetworkDAO().load(networkID, null);
log.info("XBee inicializado");
//Inicia la red con los datos obtenidos configurados anteriormente
XBee.open(network.getPort(), network.getBaudrate().getValue());
log.info("Conexión con el coordinador establecida");
log.info("Comprobando número de serie del coordinador");
AtCommand command = new AtCommand("SH");
AtCommandResponse response = (AtCommandResponse) XBee.sendAtCommand(command);
int[] serialHigh = response.getValue();
command = new AtCommand("SL");
response = (AtCommandResponse) XBee.sendAtCommand(command);
int[] serialLow = response.getValue();
int[] serial = new int[serialHigh.length + serialLow.length];
for (int i = 0; i < serialLow.length; i++) {
        serial[i] = serialLow[i];
        serial[i + serialLow.length] = serialHigh[i];
}
XBeeAddress64 connectedSerial = new XBeeAddress64(serial);
log.info("Dirección del coordinador conectado : " + ByteUtils.toBase16(serial));
```

2. Una vez se obtiene el número de serie, lo busca en la base de datos.

```
Mote coordinator = new DbMoteDAO().load(connectedSerial);
XBeeAddress64 currentSerial = coordinator != null ? coordinator.getSerialNumber() : null;
```

3. Si no hay coordinador previo, entonces obtiene la dirección de red del mote y guarda el número de serie obtenido y la dirección de red. Finalmente añade el listener de paquetes.

```
if (currentSerial == null) {
        //Si no hay coordinador previo lo guardaría en la base de datos
        log.debug("El sistema no tiene coordinador");
        log.debug("Obteniendo información del coordinador conectado...");
        command = new AtCommand("MY");
        response = (AtCommandResponse) XBee.sendAtCommand(command);

        coordinator = new Coordinator();
        coordinator.setSerialNumber(connectedSerial);
        coordinator.setNetworkAddress(new XBeeAddress16(response.getValue()));
        coordinator.setNetwork(network);
new DbMoteDAO().save(coordinator);
        log.debug("Nuevo coordinador guardado!");
        //y añade el listener porque van a empezar a llegar paquetes de asociación
        XBee.addPacketListener(XBeePacketListener.getInstance());
        log.info("Manejador de paquetes añadido");
}
```

4. Si no ha cambiado, simplemente añade el listener de paquetes.

```
//Si no hemos cambiado de coordinador no hace nada, pues los nodos
//ya conocen la dirección del coordinador de alguna asociación anterior
//o si es la primera vez, se asociarán ahora.
if (currentSerial.equals(connectedSerial)) {
        log.debug("No ha cambiado el coordinador y no hace falta iniciar el proceso de
reasociación.");
        //y añade el listener porque van a empezar a llegar paquetes de sensado
        XBee.addPacketListener(XBeePacketListener.getInstance());
        log.info("Manejador de paquetes añadido");
}
```

5. Si ha cambiado el coordinador, entonces reconfigura en la base de datos el nuevo coordinador y reinicia el proceso de asociación.

```
if (!currentSerial.equals(connectedSerial)) {
        //Si ha cambiado el coordinador, actualiza en DB el serial del coordinador
        //y arranca un worker thread para reasociar los nodos al nuevo coordinador.
        //Para ello tenemos que recuperar los nodos y en concreto las direcciones
        //de 64 bits y enviarles uno a uno el comando AT remoto ATDA, que fuerza
        //la desasociación para que reinicien el proceso de asociación al nuevo
        //coordinador.
        log.debug("Se ha cambiado el coordinador!");
        log.debug("Obteniendo información del coordinador conectado...");
        command = new AtCommand("MY");
        response = (AtCommandResponse) XBee.sendAtCommand(command);

        coordinator = new Coordinator();
        coordinator.setSerialNumber(connectedSerial);
        coordinator.setNetworkAddress(new XBeeAddress16(response.getValue()));
        coordinator.setNetwork(network);

        new DbMoteDAO().update(coordinator);

        log.debug("Actualizado el nuevo coordinador");
        log.debug("Iniciando el proceso de reasociacion de nodos");
        List<Mote> motes = new DbMoteDAO().loadByNetworkId(networkID);
        List<XBeeAddress64> endDevicesMacList = new ArrayList<XBeeAddress64>();
        for (Mote mote : motes) {
                log.debug(mote.getSerialNumber());
                endDevicesMacList.add(mote.getSerialNumber());
        }
        AssociationWorker associationWorker = new AssociationWorker(endDevicesMacList, XBee);
        //Usa un singleThread, pero igual se puede mejorar usando una Pool
        ExecutorService associationService = Executors.newSingleThreadExecutor();
        associationService.execute(associationWorker);
}
```

Hasta ahora se ha presentado como es el flujo una vez que el sistema arranca pero la parte importante es la de la reasociación que se da en el punto 5. En el fragmento de código presentado aparece una clase, **AssociationWorker,** que se ha creado concretamente para gestionar este proceso.

3.1.5. REASOCIACIÓN

El proceso de reasociación consiste en desasociar a cada uno de los motes que pertenecen a la red individualmente, y dejar que automáticamente se asocien.

Quien ordena la disociación de los motes es el coordinador, que envía el comando remoto DA (Dissasociate) a cada uno de los motes. Cuando un XBee se desasocia, automáticamente, por su configuración, vuelve a asociarse en el nivel de red, se une a una PAN y tiene un periodo de handshaking para obtener una dirección. Una vez que se ha asociado, el XBee envía un comando de respuesta al coordinador indicando si la asociación ha sido satisfactoria.

Principalmente esta tarea es la que realiza **AssociationWorker,** que toma todos los motes que conocemos que pertenecen a la red y va uno a uno desasociando y esperando una respuesta. Se trata de una tarea esporádica y por lo tanto se puede encapsular en un hilo de ejecución o thread, que muere cuando finaliza la tarea.

Una forma adecuada de crear esta tarea es usar el modelo de Executor de la librería estándar Java para concurrencia, de la siguiente forma:

```
AssociationWorker associationWorker = new AssociationWorker(endDevicesMacList, XBee);
ExecutorService associationService = Executors.newSingleThreadExecutor();
associationService.execute(associationWorker);
```

La utilidad AssociationWorker toma un mapa de direcciones estados, associationMap y cada segundo envía el comando AT remoto “DA” a cada una de las direcciones. Si de forma síncrona, recibe una respuesta de asociación “Ok”, entonces lo indica en el mapa de estados como un booleano cierto. Este proceso se realiza hasta que todos los motes están asociados de nuevo, momento en el que inicia el listener de paquetes por defecto XbeePacketListener.

ASSOCIATIONWORKER.JAVA

```
/**
 * Acción del worker de asociación.
 * Se ejecuta mientras no están asociados todos los nodos conocidosen el momento de iniciarse.
 * Una vez que se han asociado todos, añade el manejador de paquetes adecuado al XBee.
 */
public class AssociationWorker implements Runnable {

	private final Logger log = Logger.getLogger(this.getClass());
	private Map<XBeeAddress64, Boolean> associationMap;
	private XBee xbee;
	private static final String DISSASOCIATE_COMMAND = "DA";
	private static final int TIMEOUT = 1000;

	public AssociationWorker(List<XBeeAddress64> endDevicesMac, XBee xbee) {
		this.xbee = XBee;
		this.associationMap = new
			HashMap<XBeeAddress64,Boolean>(endDevicesMac.size());
		for (XBeeAddress64 address : endDevicesMac) {
			this.associationMap.put(address, false);
		}
	}

	private boolean associationCompleted() {
		for (XBeeAddress64 endDevice : associationMap.keySet()) {
			if (!associationMap.get(endDevice)) {
				return false;
			}
		}
		return true;
	}

	public void run() {
		while (!associationCompleted()) {
			try {
			for (XBeeAddress64 endDevice : associationMap.keySet()) {
				if (!associationMap.get(endDevice)) {
					log.debug("Reiniciando asociación con:"+ endDevice.toString());
					ZNetRemoteAtRequest dissasociateCommand =
						new ZNetRemoteAtRequest(0x01, endDevice,
						XBeeAddress16.BROADCAST, true, DISSASOCIATE_COMMAND);
					ZNetRemoteAtResponse response = (ZNetRemoteAtResponse)
					this.XBee.sendSynchronous(dissasociateCommand, TIMEOUT);
					log.debug(response);
					if (response.isOk()) {
						associationMap.put(endDevice, true);
						log.debug("Asociado el nodo con dirección:"+endDevice);
					}
				}
			}
			} catch (XBeeException ex) {
				log.error(ex);
			}

		}
		log.debug("Finalizado el proceso de reasociación de nodos");
		//Una vez realizado el proceso de asociación, iniciamos el manejador de paquetes
```

```
                XBee.addPacketListener(XBeePacketListener.getInstance());//Es un Singleton
                log.info("Manejador de paquetes añadido");
        }
}
```

3.1.6. CONEXIÓN A LA BASE DE DATOS

Para lograr la gestión de la red a alto nivel, como se ha explicado en los apartados anteriores, es necesario un mecanismo que permita guardar cierta información, como puede ser datos sobre el coordinador, las direcciones de red de los motes y los sensores que tienen instalados, información de los sensores...etc.

El mecanismo de persistencia se explica en más detalle en la sección 6 pero diremos que se trata de una base de datos relacional, donde la información se guarda en forma tabular en la que cada fila de una tabla corresponde con un registro de información.

Existe una conexión directa entre los sistemas locales y la base de datos, es decir, la base de datos es realmente un servidor que está disponible en internet y es accesible a través de una dirección.

Java ofrece un mecanismo estándar para manejo de base de datos denominado JDBC que gestiona la conexión y la ejecución de sentencias de base de datos o SQL.

En el prototipo de aplicación, se ha desarrollado una clase que permite gestionar la conexión remota con un servidor de base de datos, que físicamente estaría ubicado en el servidor global de aplicaciones.

DBLOCAL.JAVA

```
public class DBLocal {

        private final Logger log = Logger.getLogger(this.getClass());

        /**Proporciona una conexión a la base de datos.*/
        public Connection getConnection() {
                try {
                        String userName = "root";
                        String password = "admin";
                        String url = "jdbc:mysql://localhost:3306/homeAutomation";
                        Class.forName("com.mysql.jdbc.Driver").newInstance();
                        Connection conn = DriverManager.getConnection(url, userName, password);
                        return conn;
                } catch (SQLException ex) {
                        log.fatal("Problema con fuente de datos " + ex.toString());
                        throw new RuntimeException("Error de conexión a la base de datos.", ex);
                } catch (InstantiationException ex) {
                        log.fatal("Problema con fuente de datos " + ex.toString());
                        throw new RuntimeException("Error de conexión a la base de datos.", ex);
                } catch (IllegalAccessException ex) {
                        log.fatal("Problema con fuente de datos " + ex.toString());
                        throw new RuntimeException("Error de conexión a la base de datos.", ex);
                } catch (ClassNotFoundException ex) {
                        log.fatal("Problema con fuente de datos " + ex.toString());
                        throw new RuntimeException("Error de conexión a la base de datos.", ex);
                }
        }
        /** Devuelve una conexión a la base de datos. */
```

```
	public void putConnection(Connection conn) {
		try {
			if (conn != null) {
				conn.close();
			}
		} catch (SQLException ex) {
			log.error("Error cerrando conexión " + ex.toString());
			throw new RuntimeException(ex);
		}
	}
}
```

Se pueden apreciar varios detalles:

- Dispone de métodos para obtener y devolver una conexión, en el mismo sentido que una pool de conexiones gestionada por la base de datos.
- Se ha empleado MySQL como motor de base de datos y la API que ofrece:

  Class.forName("com.mysql.jdbc.Driver").newInstance();
- Para el prototipo, la base de datos es como si estuviera en el propio sistema local, pues la URL de conexión indica que la base de datos se encuentra en localhost.

  String url = "jdbc:mysql://**localhost**:3306/homeAutomation";

3.2. SISTEMA EMBEBIDO

3.2.1. LIBRERÍA XBEE

Esta librería encapsula las siguientes funciones del transceptor XBee:

- Creación de tramas con direccionamiento de 16 bits.
- Creación de tramas con direccionamiento de 64 bits.
- Creación de comandos AT
- Lectura de tramas con direccionamiento de 16 bits.
- Lectura de tramas con direccionamiento de 64 bits
- Lectura de tramas de status
- Enviar y recibir paquetes

Está estructurada de tal forma que las cabeceras de función y estructuras de datos necesarias están contenidas en un archivo con extensión .h, denominado XBee.h y las implementaciones en otro archivo con extensión .c, denominado XBee.c.

Las cabeceras de las funciones son las siguientes:

XBEE.H

```
BOOL XBee_CreateTX16Packet(XBeePacket* xbp, uint8_t frameID,
                    uint16_t destination, uint8_t options, uint8_t* data,
                    uint8_t datalength);
BOOL XBee_CreateTX64Packet(XBeePacket* xbp, uint8_t frameID,
                    uint64_t destination, uint8_t options, uint8_t* data,
                    uint8_t datalength);
void XBee_CreateATCommandPacket(XBeePacket* packet, uint8_t frameID, char* cmd,
                    uint8_t* params, uint8_t datalength);

BOOL XBee_ReadRX16Packet(XBeePacket* xbp, uint16_t* srcAddress,
                    uint8_t* sigstrength, uint8_t* options, uint8_t** data,
                    uint8_t* datalength);
BOOL XBee_ReadRX64Packet(XBeePacket* xbp, uint64_t* srcAddress,
                    uint8_t* sigstrength, uint8_t* options, uint8_t** data,
                    uint8_t* datalength);

BOOL XBee_ReadTXStatusPacket(XBeePacket* xbp, uint8_t* frameID,
                                    uint8_t* status);

BOOL XBee_ReadModemStatusPacket(XBeePacket* xbp, uint8_t* status);
BOOL XBee_ReadAtResponsePacket(XBeePacket* xbp, uint8_t* frameID,
                    char** command, uint8_t* status, int* datavalue);

uint8_t XBee_GetPacket(XBeePacket* packet/*, uint8_t byte*/);
uint8_t XBee_SendPacket(XBeePacket* packet, uint8_t datalength);
void XBee_ResetPacket(XBeePacket* packet);
```

Como se puede apreciar, todas las funciones usan un tipo de dato, XbeePacket, que es una estructura que contiene la información del paquete, según la especificación de paquetes de DIGI.

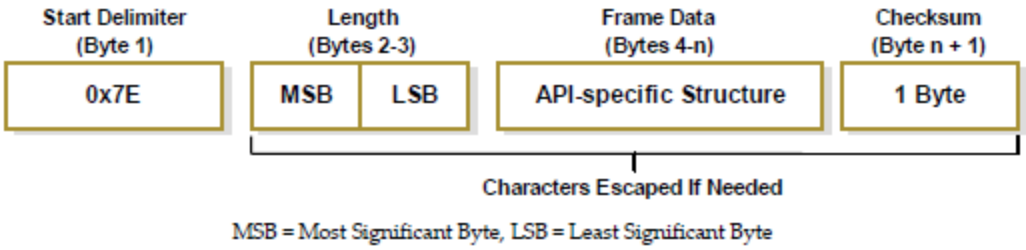


FIGURA 40: ESPECIFICACIÓN DEL FORMATO API DEL XBEE

Así, se ha diseñado una forma flexible de crear paquetes que se adapten a los distintos formatos, basándolo en el concepto de struct y union. Struct, en lenguaje C define un conjunto de datos agrupado bajo un identificador. Estos datos pueden ser heterogéneos por lo que se pueden mezclar enteros con caracteres o punteros, lo que da gran flexibilidad a la hora de crear nuevos tipos de datos compuestos.

Otro de los elementos que se ha empleado son las Union. Una Union especifica que una variable puede tomar el tipo de un conjunto de posibles tipos de datos, es decir, que el tipo concreto se establece en tiempo de ejecución. Usando esta capacidad se pueden crear las tramas de forma muy dinámica.

Veamos cómo queda la estructura XbeePacket haciendo uso del concepto de Union:

STRUCT XBEEPACKET

```
typedef struct {
        uint8_t apiId;
        union {
                uint8_t payload[100];
                XBee_TX16 tx16; /**< TX 16 packet. */
                XBee_RX16 rx16; /**< RX 16 packet. */
                XBee_TX64 tx64; /**< TX 64 packet. */
                XBee_RX64 rx64; /**< RX 64 packet. */
                XBee_ATCommand atCommand; /**< AT Command packet. */
                XBee_ATCommandResponse atResponse; /**< AT Command Response packet. */
                XBee_Remote_ATCommandRequest atRemoteRequest;
                XBee_Remote_ATCommandResponse atRemoteResponse;
                XBee_TXStatus txStatus; /**< TX status packet. */
                XBee_ModemStatusResponse modemStatusResponse;
        };
        uint8_t crc;/**< Checksum */
        uint8_t *dataPtr; /**< Puntero al paquete.  */
        int rxState; /**< Estado del parseo. */
        int length; /**< Longitud del paquete */
        int index; /**< Posición actual en la lectura.  */
}__attribute__((packed)) XBeePacket;
```

Se pueden ver elementos comunes a los paquetes como el **apiId** y el **crc**, así como variables de ayuda que indican el estado a la hora de recibir el paquete **rxState,** la longitud del paquete **length** y la posición actual durante el procesamiento **index**.

La Union, permite que la Struct contenga alguno de los tipos que indica, así por ejemplo, si se genera un paquete con direccionamiento de 16 bits, se hará uso de la variable **tx16**.

Veamos cómo son cada uno de los tipos de tramas soportados:

- Respuesta de estado del modem

```
typedef struct {
        uint8_t status; /**< respuesta - 0 (OK) or 1 (ERROR). */
} XBee_ModemStatusResponse;
```

- Comando AT

```
typedef struct {
        uint8_t frameID; /**< Id para sucesivas respuestas. */
        uint8_t command[2]; /**< Comando AT. */
        uint8_t parameters[97]; /**< Parámetros del comando. */
} XBee_ATCommand;
```

- Respuesta a comando AT

```
typedef struct {
        uint8_t frameID; /**< Id para sucesivas respuestas. */
        uint8_t command[2]; /**< Comando AT. */
        uint8_t status; /**< respuesta - 0 (OK) or 1 (ERROR). */
        uint8_t value[96]; /**< Contenido de la respuesta. */
} XBee_ATCommandResponse;
```

- Respuesta a comando AT remoto

```
typedef struct {
        uint8_t frameID; /**< Id para sucesivas respuestas. */
        uint8_t respAddr64[8];/**Dirección de 64 bit*/
        uint8_t respAddr16[2]; /**Dirección de 16 bit */
        uint8_t command[2]; /**< Comando AT. */
        uint8_t status; /**< Respuesta - 0 (OK) or 1 (ERROR). */
        uint8_t commandData[86];/**Valor de la respuesta.*/
} XBee_Remote_ATCommandResponse;
```

- Envío de comando AT remoto

```
typedef struct {
        uint8_t frameID; /**< Id para sucesivas respuestas */
        uint8_t destAddr64[8];/** Dirección de 64 bit */
        uint8_t destAddr16[2]; /** Dirección de 16 bit */
        uint8_t commandOpts; /**Opciones */
        uint8_t command[2]; /** Comando AT */
        uint8_t commandData[86]; /**Valor a establecer.*/
} XBee_Remote_ATCommandRequest;
```

- Envío de trama con direccionamiento de 64 bits

```
typedef struct {
        uint8_t frameID; /**< Id para sucesivas respuestas. */
        uint8_t destination[8]; /**< Dirección destino de 64 bits. */
        uint8_t options; /**< 0x01 (disable ACK) or 0x04 (Broadcast). */
        uint8_t data[90]; /**< Payload. */
} XBee_TX64;
```

- Recepción de trama con direccionamiento de 64 bits

```
typedef struct {
        uint8_t source[8]; /**< Dirección de 64 bits del origen. */
        uint8_t rssi; /**< Fuerza de la señal. */
        uint8_t options; /**< Opciones */
        uint8_t data[89]; /**< Payload. */
} XBee_RX64;
```

- Envío de trama con direccionamiento de 16 bits

```
typedef struct {
        uint8_t frameID; /**< Id para sucesivas respuestas. */
        uint8_t destination[2]; /**< Dirección destino de 16 bits.. */
        uint8_t options; /**< 0x01 (disable ACK) or 0x04 (Broadcast). */
        uint8_t data[96]; /**< Payload. */
} XBee_TX16;
```

- Recepción de trama con direccionamiento de 16 bits

```
typedef struct {
        uint8_t source[2]; /**< Dirección de 16 bits del origen. */
        uint8_t rssi; /**< Fuerza de la señal. */
        uint8_t options; /**< Opciones. */
        uint8_t data[95]; /**< Payload. */
} XBee_RX16;
```

- Estado de la transmisión

```
typedef struct {
        uint8_t frameID; /**< Id para sucesivas respuestas */
        uint8_t status; /**< Valores:
                                - 0 - success
                                - 1 - No ACK received
                                - 2 - CCA failure
                                - 3 - Purged. */
} XBee_TXStatus;
```

3.2.2. LIBRERÍA PARA GESTIÓN DE INFORMACIÓN

En el apartado Gestión de tramas de red, se explicó cómo se desarrolla la asociación de los motes y cómo se procesan las tramas que llegan al sistema local, desde una perspectiva de alto nivel. Ahora bien, estas tramas las han de generar los motes y por lo tanto se han de proveer de mecanismos para esta tarea. En este sentido y al igual que para el manejo del XBee, se ha creado una utilidad para crear tramas de asociación y de medida, que serán enviadas por lo motes durante el funcionamiento.

Estas utilidades como es lógico, han de seguir las convenciones o especificaciones marcadas en el punto Gestión de tramas de red en cuanto al formato de la información, que recordemos es un formato XML, con unas propiedades y atributos bien definidos. Así, se ha implementado es una utilidad a bajo nivel que genera el XML necesario tanto para las tramas de asociación como las de sensado.

El archivo de cabeceras association.h de función para la generación de tramas de asociación, define dos funciones, una para crear el payload o información de los sensores instalados y otra para los actuadores.

ASSOCIATION.H

```
/*
 * association.h
 *
 *      Define los prototipos para las funciones de
 *      creación de tramas de asociación.
 *      El resultado es un formato XML tal y como se
 *      detalla en la documentación.
 */

#ifndef ASSOCIATION_H_
#define ASSOCIATION_H_

#include <stdint.h>
#include "element.h"

#define ASSOC_MOTE_START_TAG "<mote fmt=\"1\""
#define MOTE_END_TAG "</mote>"
#define SHORT_END_TAG "\"/>"
#define SENSOR_START_TAG "<sen id=\""
#define ACTUATOR_START_TAG "<act id=\""

/**
 * Crea el XML adecuado para formar el payload de una trama de asociación
 * para un end device que únicamente contiene sensores.
 *
 * El formato es el siguiente:
 *
 *              <mote fmt="1">
 *                      <sensor id="0000"/>
 *                      <sensor id="0001"/>
 *              </mote>
 *
 * @param sensorList Puntero a la lista de sensores
 **/
char* createSensorAssociationPayload(struct Element sensorList[],uint8_t length);
```

```
/**
 * Crea el XML adecuado para formar el payload de una trama de asociación
 * para un end device que únicamente contiene actuadores.
 *
 * El formato es el siguiente:
 *
 *                  <mote fmt="1">
 *                          <actuator id="0000"/>
 *                          <actuator id="0001"/>
 *                  </mote>
 *
 * @param sensorList Puntero a la lista de actuadores
 * */
char* createActuatorAssociationPayload(struct Element actuatorList[],uint8_t length);


#endif /* ASSOCIATION_H_ */
```

Y la implementación que se ha realizado de las funciones anteriores se encuentra en el archivo association.c:

ASSOCIATION.C

```
/*
 * association.c
 *
 */
#include "association.h"
#include "element.h"
#include <string.h>
#include <stdlib.h>


char* createSensorAssociationPayload(struct Element sensorList[],uint8_t length) {
        char* xml = calloc(100,sizeof(xml));
        strncat(xml, ASSOC_MOTE_START_TAG, strlen(ASSOC_MOTE_START_TAG));
        strncat(xml, SENSOR_TYPE_TAG, strlen(SENSOR_TYPE_TAG));
        int i;
        for (i = 0; i < length; i++) {
                strncat(xml, SENSOR_START_TAG, strlen(SENSOR_START_TAG));
                strncat(xml, sensorList[i].id, strlen(sensorList[i].id));
                strncat(xml, SHORT_END_TAG, strlen(SHORT_END_TAG));
        }
        strncat(xml, MOTE_END_TAG, strlen(MOTE_END_TAG));
        return xml;
}


char* createActuatorAssociationPayload(struct Element actuatorList[],uint8_t length) {
        char* xml = calloc(100,sizeof(xml));
        strncat(xml, ASSOC_MOTE_START_TAG, strlen(ASSOC_MOTE_START_TAG));
        int i;
        for (i = 0; i < length; i++) {
                strncat(xml, ACTUATOR_START_TAG, strlen(ACTUATOR_START_TAG));
                strncat(xml, actuatorList[i].id, strlen(actuatorList[i].id));
```

```
                strncat(xml, SHORT_END_TAG, strlen(SHORT_END_TAG));
        }
        strncat(xml, MOTE_END_TAG, strlen(MOTE_END_TAG));
        return xml;
}
```

De la misma forma se han creado los archivos sensing.h y sensing.c para generar el XML de las tramas de sensado, que envían información sobre las medidas.

SENSING.H

```
/* sensing.h */

#ifndef SENSING_H_
#define SENSING_H_

#include "association.h"

#define SENSING_MOTE_START_TAG "<mote fmt=\"2\">"
#define SENSOR_C_MEAS_TAG " c=\""
#define SENSOR_B_MEAS_TAG " b=\""
#define QUOTE "\""

/**
 * Crea el XML adecuado para formar el payload de una trama de sensado
 * para un end device que únicamente contiene sensores.
 *
 * El formato es el siguiente:
 *
 *                  <mote fmt="2">
 *                          <sensor id="0000" c_meas="-12.234"/> //Sensor analógico
 *                          <sensor id="0001" b_meas="1"/>              //Sensor binario
 *                  </mote>
 *
 * @param sensorList Puntero a la lista de sensores con las medidas
 * */
char* createSensorMeasuresPayload(const struct Element sensorList[],uint8_t length);


#endif /* SENSING_H_ */
```

SENSING.C

```
/* sensing.c */

#include "sensing.h"
#include "stdlib.h"
#include "string.h"

char* createSensorMeasuresPayload(const struct Element sensorList[],uint8_t length){
		char* xml = (char*)calloc(95,sizeof(char*));
		strncat(xml, SENSING_MOTE_START_TAG, strlen(SENSING_MOTE_START_TAG));
		int i;
		//int length = sizeof(actuatorList) / sizeof(struct Element);
		for (i = 0; i < length; i++) {
			strncat(xml, SENSOR_START_TAG, strlen(SENSOR_START_TAG));
			strncat(xml, sensorList[i].id, strlen(sensorList[i].id));
			strncat(xml, QUOTE, strlen(QUOTE));
			if(sensorList[i].type == CONTINUOUS){
				strncat(xml, SENSOR_C_MEAS_TAG, strlen(SENSOR_C_MEAS_TAG));
				strncat(xml, sensorList[i].c_meas, strlen(sensorList[i].c_meas));
			}else if(sensorList[i].type == BINARY){
				strncat(xml, SENSOR_B_MEAS_TAG, strlen(SENSOR_B_MEAS_TAG));
				strncat(xml, sensorList[i].b_meas, strlen(sensorList[i].b_meas));
			}
			strncat(xml, SHORT_END_TAG, strlen(SHORT_END_TAG));
		}
		strncat(xml, MOTE_END_TAG, strlen(MOTE_END_TAG));
		return xml;
}
```

Como se puede observar, en ambos casos se está usando una tipo de dato denominado Element. Element es una struct que se ha creado para agrupar ciertos datos comunes a los sensores instalados en un mote y que sirven tanto para identificarlos como para asociar medidas concretas.

```
/** Tipo de medidas que proporciona el sensor */
enum elementType{
	CONTINUOUS = 1,
	BINARY = 2
};

/** Estructura para relacionar sensor, medidas y canales del a/d usados */
struct Element{
	const char* id; /** Id del sensor en la base de datos */
	const enum elementType type;/** Tipo de medida que proporciona */
	const char a2d_channel;/** Canal a/d que usa */
	char* c_meas;/** Vector con las medidas contínuas */
	char* b_meas;/** Vector con las medidas binarias */
};
```

3.2.3. AUTOGESTIÓN

En el lado del coordinador se deben configurar los siguientes atributos:

- **Coordinator enable(CE)**: Habilita un transceptor como coordinador.

  0 - End device

  1 - Coordinador

- **Coordinator association(A2)**: Establece las opciones de asociación en el lado del coordinador. Las posibles opciones son:

  bit 0 – Reasignación de identificador PAN

  0 – El coordinador no llevará a cabo un escaneo activo para localizar un identificador de PAN disponible, simplemente operará en la pan que tiene configurada.

  1 – El coordinador realizará en escaneo activo para determinar un identificador PAN disponible. Si el identificador PAN que tiene configurado ya está siendo usado, lo cambiará por uno que esté libre.

  bit 1 – Reasignación de canal

  0 – El coordinador realizará un escaneo de energía para determinar un canal libre. Operará en el canal que tiene configurado en el registro CH.

  1 – El coordinador realizará un escaneo de energía para encontrar un canal libre.

  bit 2 – Opciones de asociación

  0 – El coordinador no permite a ningún dispositivo que se una a la PAN.

  1 – El coordinador permite asociación.

Por lo tanto habrá que configurar el parámetro CE con valor 0x01 en cualquiera de los casos, para disponer de un coordinador que forme la PAN. Buscando la máxima flexibilidad, el parámetro A2 debe tener un valor 0x07, de tal forma que buscará un identificador PAN libre y un canal libre también, permitiendo asociación de los end device.

En el lado de los end device se debe configurar el parámetro A1 de la misma forma que se ha hecho con el A2 del coordinador, al valor 0x07, para que pueda obtener un identificador de PAN y un canal. Para seleccionar un identificador de PAN, realiza un barrido (active scan) de las PAN que alcanzan su ámbito y se queda con la de mejor señal (link quality). Entonces intenta unirse a ella enviando una solicitud de asociación (association request) y si el coordinador acepta esa solicitud, el end device establece su dirección de red de 16 bits al valor 0xFFFE y envía por la UART una trama de estado del modem(Modem status response) que indica que está asociado.

Cabe destacar que este valor lo establecen todos los end device que se asocian mediante el proceso anteriormente descrito, de tal forma que son indistinguibles a través de la dirección de red. Esta es una connotación realmente importante, pues deshabilita el uso de tramas con direccionamiento de 16 bits desde el coordinador hacia los end devices, limitando a un único direccionamiento, el de 64 bits o números de serie.

Ahora bien, los XBee disponen de dos registros de 32 bits donde pueden almacenar hasta una dirección completa de 64 bits, DH (Destination high) y DL (Destination low) y en nuestro caso, empleando el proceso de asociación descrito en el párrafo anterior, en DL queda registrada la dirección de 16 bits del coordinador, de tal forma que los end devices emplean tramas con direccionamiento de 16 bits hacia el coordinador.

Esta falta de uniformidad entre los direccionamientos derivada del empleo de la autoasociación en los dispositivos XBee es cuanto menos curiosa y requiere de un proceso previo de experimentación para llegar a estas conclusiones, pues la información que proporciona DIGI es realmente escasa.

A efectos prácticos lo que interesa para determinar si un end device está asociado es, si se recibe una trama Modem status response indicando una correcta asociación y es esto lo que se emplea en el programa de los motes.

Una vez que recibimos dicha trama, obtenemos la dirección de red del coordinador para poder direccionarle una trama con los datos de la asociación que contiene los sensores o actuadores instalados.

Al ser el proceso de asociación algo puntual, no tiene sentido el que se trate de una tarea periódica ni de algo que se ejecute en un instante fijo sino que puede tener la consideración de evento asíncrono y modelarse como un manejador de interrupción.

Se ha generado por lo tanto el código que completa la interrupción de la UART del microcontrolador de la siguiente manera:

```
/**
 * Rutina de interrupción para la recepción de tramas del XBee en formato API 2
 *
 * Tipos de trama que puede recibir:
 *              - MODEM_STATUS_RESPONSE
 *              - RX_64_RESPONSE
 *
 * La RX_16_RESPONSE no se puede dar por el uso de autoasociación, ya que
 * todos los end devices tienen como dirección de red 0xFFFE y es imposible
 * direccionar.
 *
 * Se recibirá una trama MODEM_STATUS_RESPONSE en los siguientes casos:
 *              - HARDWARE_RESET >> 0x7E|0x00|0x02|0x00|CHK
 *              - ASSOCIATED >> 0x7E|0x00|0x02|0x02|CHK
 *              - DISASSOCIATED  >> 0x7E|0x00|0x03|0x00|CHK
 *                              (solo en el caso de que se fuerce la desasociación usando ATDA)
 *
 * Si el XBee logra asociarse al coordinador, este mandará por la UART una trama
 * MODEM_STATUS con la información ASSOCIATED y procederá entonces a enviar al
 * coordinador la trama de asociación con la información necesaria para configurarse
 * en el sistema.
 * Puede ocurrir que el proceso de asociación sea masivo y que todos los nodos intenten
 * asociarse al sistema de forma simultánea. Para minimizar el impacto de este probable caso,
 * haremos que espere un breve tiempo aleatorio antes de enviar la trama.
 * Solo en el momento en que recibe una trama ASSOCIATED, inicia el funcionamiento normal
 * del mote, lo que implica que arranca los timer
 */

ISR(USART_RX_vect)
{
        cli();
        XBee_ResetPacket(&packet);
        uint8_t result = XBee_GetPacket(&packet);
```

```
	if (result == 1) {
		if (packet.apiId == XBEE_MODEM_STATUS_RESPONSE) {/
			uint8_t status;
			if (XBee_ReadModemStatusPacket(&packet, &status)) {
				if (status == ASSOCIATED) {/** Si indica asociación correcta*/
					XBee_ResetPacket(&packet);
					XBee_CreateATCommandPacket(&packet, 0x52, "DL", NULL,
                                                                        0);
					_delay_ms(5000);
					XBee_SendPacket(&packet, 0);
				}
			}
		} else if (packet.apiId == XBEE_ATCOMMANDRESPONSE) {
			uint8_t frameID;
			char* command;
			uint8_t status;
			int data;
			if (XBee_ReadAtResponsePacket(&packet, &frameID, &command, &status,
					&data)) {
				uartSendByte('C');
				if (strcmp(command, "DL") == 0) {
					mac = data;
				}
				XBee_ResetPacket(&packet);
				char* payload = createSensorAssociationPayload(sensors, 3);
				XBee_CreateTX16Packet(&packet, 0x52, mac, 0x00, payload,
                                                        strlen(payload));
				XBee_SendPacket(&packet, strlen(payload));
				free(payload);
				free(command);
				init_normal_mode();
			}
		}
		sei();
	}
}
```

El algoritmo funciona básicamente de la siguiente forma: cuando el microcontrolador detecta entrada de datos en el buffer de la USART, lanza la rutina de interrupción asociada que en este caso está identificada por USART_RX_vect. Recordemos que la familia AVR dispone de interrupciones vectorizadas con prioridades estáticas, y por ello han usado el sufijo _vect para identificarlas.

Una vez entra en la rutina de interrupción, resetea el paquete XbeePacket, es decir, limpia los valores que pueda tener y libera los punteros. La implementación de esta función es la siguiente:

```
void XBee_ResetPacket(XBeePacket* packet) {
	packet->dataPtr = (uint8_t*) packet;
	packet->crc = 0; /** Limpia el Checksum*/
	packet->rxState = XBEE_PACKET_RX_START;/** Estado inicial de recepción*/
	packet->length = 0;/** Limpia la longitud del paquete */
	packet->index = 0;/** Comienzo del paquete */
	packet->apiId = 0;
	memset(packet->payload, 0, 100);/** Reserva 100 bytes con valor 0*/
}
```

Una vez limpio el paquete se puede proceder a la lectura de la información que llega a través de la USART. Esto se realiza mediante la llamada a la función XBee_GetPacket que tiene la siguiente implementación:

```
uint8_t XBee_GetPacket(XBeePacket* packet) {
	uint8_t ret = 0;
	uint8_t byte;
	while (serialAvailable()) {
		byte = uartReadByte();
		switch (packet->rxState) {
		case XBEE_PACKET_RX_START:
			if (byte == XBEE_PACKET_STARTBYTE)
				packet->rxState = XBEE_PACKET_RX_LENGTH_1;
			break;
		case XBEE_PACKET_RX_LENGTH_1:
			packet->length = byte;
			packet->length <<= 8;
			packet->rxState = XBEE_PACKET_RX_LENGTH_2;
			break;
		case XBEE_PACKET_RX_LENGTH_2:
			packet->length += byte;
			if (packet->length > XBEE_MAX_PACKET_SIZE) {
				packet->rxState = XBEE_PACKET_RX_START;
				ret = -2;//LENGTH Error
			} else {
				packet->rxState = XBEE_PACKET_RX_PAYLOAD;
			}
			packet->crc = 0;
			break;
		case XBEE_PACKET_RX_PAYLOAD:
			*packet->dataPtr++ = byte;
			if (++packet->index >= packet->length)
				packet->rxState = XBEE_PACKET_RX_CRC;
			packet->crc += byte;
			break;
		case XBEE_PACKET_RX_CRC:
			packet->crc += byte;
			packet->rxState = XBEE_PACKET_RX_START;
			if (packet->crc == 0xFF)
				return 1;//Everything OK!
			else {
				XBee_ResetPacket(packet);
				return -1;//CRC Error
			}
		}
		_delay_ms(20);//Necesita un delay
	}
	return ret;
}
```

Esta rutina se ejecuta en un bucle mientras haya datos en el buffer de entra da de la USART. La estructura es la de una máquina de estados que es una solución elegante para detectar formatos de datos estructurados, como son las tramas API del transceptor XBee. Por lo tanto, se han definido un conjunto de estados de recepción de tramas y son los siguientes:

```
// estados de recepción de paquete
#define XBEE_PACKET_RX_START 0     /** Inicio de recepción */
#define XBEE_PACKET_RX_LENGTH_1 1 /**Recepción de la parte alta de la longitud del paquete*/
#define XBEE_PACKET_RX_LENGTH_2 2 /**Recepción de la parte baja de la longitud del paquete*/
#define XBEE_PACKET_RX_PAYLOAD 3 /** Recepción de la información o payload*/
#define XBEE_PACKET_RX_CRC 4              /** Recepción del checksum*/
```

y se ha implementado el siguiente grafo de estados:

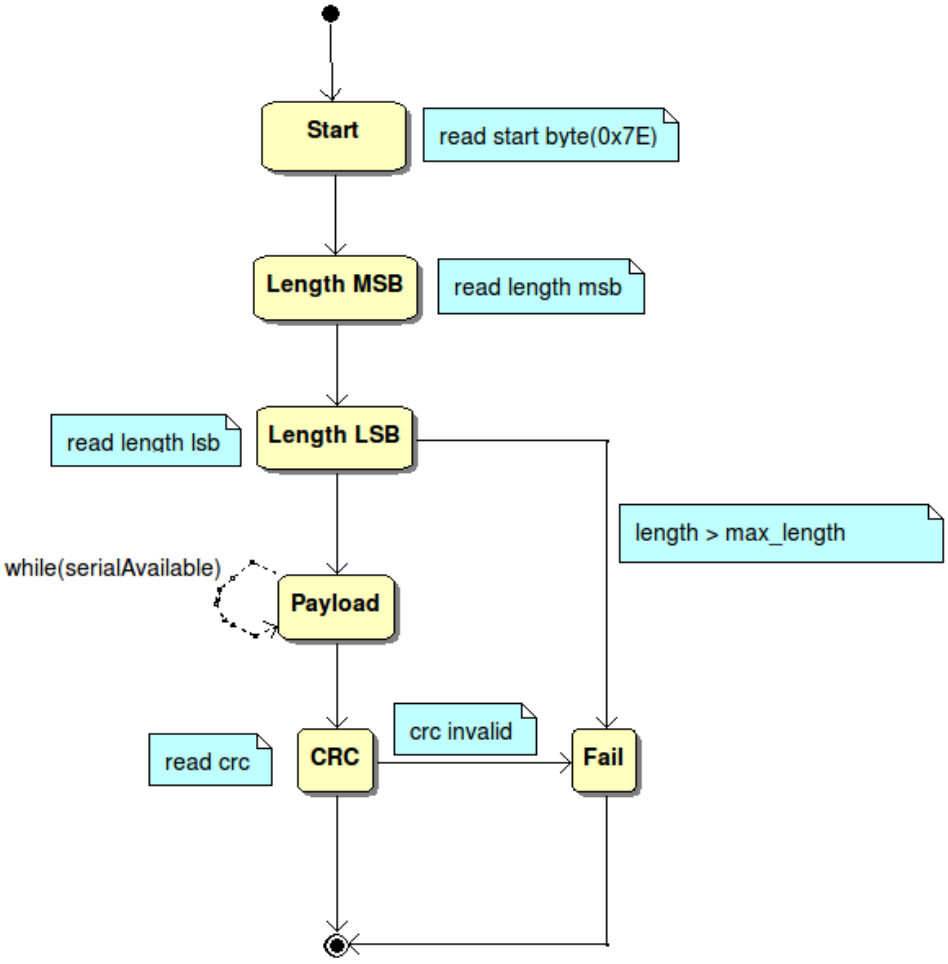


FIGURA 41: GRAFO DE ESTADOS DE RECEPCIÓN DE TRAMAS EN EL MICROCONTROLADOR

Para determinar si quedan datos en el buffer de entrada de la USART, se ha implementado una pequeña utilidad específica e inspirada en sistemas de mayor tamaño y complejidad, en lo que se conoce como BSP(Board support package), que define una interfaz de funciones de bajo nivel que son necesarias para implementar la aplicación pero cuya implementación cambia, como es normal, dependiendo del microcontrolador elegido. Así pues, hay una interfaz común con funciones útiles y su implementación se adecúa a cada caso. Esto es lo que se ha pretendido, poniendo las miras en que sería posible que cambiara el microcontrolador y por lo tanto las implementaciones de bajo nivel. Se ha creado una cabecera de funciones de utilidad muy sencillas pero que ejemplifican el uso de un BSP.

AVR_PORT.H

```
#ifndef AVR_PORT_H_
#define AVR_PORT_H_

#include <stdint.h>
#include "avrlibtypes.h"

#define ADC_V_REF 3.3                                   //Tensión de referencia del ADC
#define ADC_RESOLUTION ADC_V_REF/1024 //Resolución del ADC

double getA2dValue(char channel);

void uartSendByte(uint8_t byte);

uint8_t uartReadByte();

uint8_t serialAvailable();

uint8_t isOverrun();

void toggle();

void on();

void off();

#endif /* AVR_PORT_H_ */
```

AVR_PORT.C

```
#include "avr_port.h"
#include <stdint.h>
#include <avr/io.h>
#include <util/delay.h>
#include "a2d.h"

void uartSendByte(uint8_t byte) {
        UDR0 = byte;
        while (!(UCSR0A & (1 << UDRE0))) {
        }
}

uint8_t uartReadByte() {
        return UDR0;
}

uint8_t serialAvailable() {
        return (UCSR0A & (1 << RXC0));
}

double getA2dValue(char channel){
        unsigned short raw = a2dConvert10bit(channel);
```

```
	return raw * ADC_RESOLUTION;
}

uint8_t isOverrun() {
	return (UCSR0A & (1 << DOR0));
}

void toggle() {
	PORTB ^= _BV(PINB5);
}

void on() {
	PORTB |= _BV(PINB5);
}

void off() {
	PORTB &= ~_BV(PINB5);
}
```

3.2.4. PROGRAMA PRINCIPAL

Básicamente inicializa el sistema de I/O mediante una llamada a la función setup_io, que cada mote implementa según su funcionalidad. Seguidamente configura un modo de bajo consumo en los registros del microcontrolador e inicializa la USART para permitir autogestión y reseteo del mote. Automáticamente entra en un bucle infinito de ejecución en el que duerme y ejecuta código de interrupción.

```
/**
 * Rutina principal para los motes.
 * Inicia el I/O y establece un modo de bajo consumo
 */
int main() {
	cli();
	setup_io();
	setup_sleep_mode();
	uart_init();
	XBee_ResetPacket(&packet);
	wdt_disable();
	for (;;) {
		sei();
		sleep_enable();
		sleep_mode();
		sleep_disable();
		cli();
	}
	return 0;
}
```

3.2.5. MOTE AMBIENTAL

La configuración de sensores del mote, se establece en el código, empleando un vector de tipos Element, explicado en el punto 7.2.2.1.2:

```
#define SAMPLES 16
static XBeePacket packet;
struct Element sensors[3] = {
                { "1", CONTINUOUS, 0, "", '0' },//Temperatura
                { "2", CONTINUOUS, 1, "", '0' },//Humedad
                { "3", CONTINUOUS, 2, "", '0' }//Iluminación
};
static uint32_t mac;
```

El microcontrolador dispone de tres timer independientes integrados, dos de 8 bits y uno de 16 bits. Todos ellos son generadores de interrupciones en varios modos y disponen de escaladores previos, de tal forma que dividen la frecuencia del reloj. Los factores de escalado son: 1, 8, 64, 256 y 1024.
Teniendo en cuenta que el cristal externo es de 8 MHz, podemos hacer los siguientes cálculos:

- Sin escalado
  - Timer de 8 bits

$$f' = \frac{1}{8 * 10^6[Hz]} = 125\ ns$$
$$T_{max} = 125\ ns * 256 = 32\ us$$

  - Timer de 16 bits

$$f' = \frac{1}{8 * 10^6[Hz]} = 125\ ns$$
$$T_{max} = 125\ ns * 65535 = 8.2\ ms$$

- Con escalado máximo
  - Timer de 8 bits

$$T' = \frac{1024}{8 * 10^6[Hz]} = 128\ us$$
$$T_{max} = 128\ us * 256 = 32\ ms$$

  - Timer de 16 bits

$$T' = \frac{1024}{8 * 10^6[Hz]} = 128\ us$$
$$T_{max} = 128\ us * 65535 = 8.4\ s$$

En el mejor de los casos obtenemos un delay máximo de menos de 9 segundos, lo cual aunque suficiente para algunos casos, es poco flexible.

Una solución elegante para conseguir un delay más largo consiste en conectar en cascada dos timer, de tal forma que el primero hace de clock del segundo, con lo que se consigue un escalado mucho mayor.

Dada la arquitectura del microcontrolador, ofrece la posibilidad de usar como clock de los timer 0 y timer 1, una entrada externa, habilitando el uso de un disparo externo. Al mismo tiempo, dispone de una funcionalidad de comparación que en tiempo real, compara el valor actuar de los contadores con los almacenados en unos registros individuales, generando una interrupción, activando un pin externo o ambas a la vez.

Con estas características se puede diseñar el siguiente diagrama de bloques, que representa una conexión física entre timers, usando la salida OC1A y la entrada T0, y una conexión lógica entre el tiempo transcurrido y las interrupciones generadas.

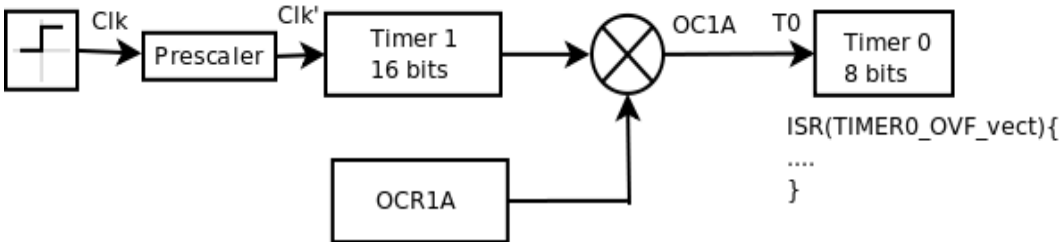


FIGURA 42: DIAGRAMA DE BLOQUES PARA LOS TIMER EN CASCADA

Además, configurando el comportamiento de OC1A como cambio (toggle), se consigue dividir a la mitad la frecuencia final.

Una vez planteado este esquema veamos cual es el rango de ciclos de trabajo que podemos conseguir.

1. Prescaler mínimo y registro de comparación mínimo
   Prescaler Timer 1 = 1
   OCR1A = 0x01

$$f' = \frac{1}{8 * 10^6[Hz]} = 125\ ns$$
$$T_{max} = 125\ ns * 256 = 32\ us$$

2. Prescaler máximo y registro de comparación máximo

   Prescaler Timer 1 = 1024
   OCR1A = 0xFFFF

$$clk' = \frac{8 * 10^6[Hz]}{1024} = 7.8\ Khz$$
$$T' = \frac{1}{clk'} = 128\ us$$
$$T_{max} = 4300\ s\ \cong 71\ min$$

Como reflejan los resultados, el abanico es bastante amplio y por lo tanto se han de ajustar los valores del prescaler y del registro OCR1A a los casos particulares.

Una vez el mote se ha asociado y configurado, se inicia el modo normal de funcionamiento, donde los timers son los que marcan el ritmo de activación.

```
/** Tras una asociación satisfactoria, inicia el modo normal*/
void init_normal_mode() {
        timer_init(); //Inicia los timers
}
```

Un ejemplo de la configuración empleada en el mote ambiental es la siguiente rutina:

```
/**
 * Configuración de los timer 0 y 1 en cascada
 * para conseguir un ciclo de trabajo suficientemente bajo de unos 4.2 segundos.
 * Prescaler por 1 y OCR1A = FFFF
 */
void timer_init() {
	//Configure Timer 1 : prescale 1
	TCCR1B &= ~_BV(CS10);
	TCCR1B |= _BV(CS11);
	TCCR1B &= ~_BV(CS12);
	//Reset TCNT
	TCNT1 = 0;
	//Configure Timer 0 : clk T0
	TCCR0B |= _BV(CS00);
	TCCR0B |= _BV(CS01);
	TCCR0B |= _BV(CS02);
	//Toggle OC1A on compare match
	TCCR1A |= _BV(COM1A0);
	//Configure OC1A toggle on compare OCR1A
	OCR1AL = 0xFF;//Set OCR1AL to max value
	OCR1AH = 0xFF;//Set OCR1AH to max value
	//Timer 0 overflow interrupt enable
	TIMSK0 |= _BV(TOIE0);
	TCNT0 = 0;
}
```

Una vez configurados los timer como se ha indicado, hay que implementar la rutina de interrupción que genera el timer 0 cuando la cuenta termina. En este caso, se aprovecha al máximo el timer 0, dejando que se desborde y ejecutando la interrupción asociada.

```
/**
 * Rutina de interrupción para el timer 0.
 * Se ejecuta periódicamente cuando la temporización expira.
 * Realiza las conversiones para los sensores, genera la trama y la envía por la USART
 */
ISR(TIMER0_OVF_vect)
{
	cli();

	a2dInit();//Inicia el a2d

	//Stop timer 1
	TCCR1B &= ~_BV(CS10);
	TCCR1B &= ~_BV(CS11);
	TCCR1B &= ~_BV(CS12);
	TCNT1 = 0;
	//Stop timer 0
	TCCR0B &= ~_BV(CS00);
	TCCR0B &= ~_BV(CS01);
	TCCR0B &= ~_BV(CS02);

	_delay_ms(10);//Espera al arranque completo del adc
	double value = 0;//Variable local para el resultado de las conversiones
	//Muestras
	sensors[0].c_meas = calloc(6, sizeof(char*));
```

```
        value = getA2dValue(sensors[0].a2d_channel,SAMPLES);
        dtostrf(100 * ((value) - 0.5), 6, 2, sensors[0].c_meas);

        sensors[1].c_meas = calloc(6, sizeof(char*));
        value = getA2dValue(sensors[1].a2d_channel,SAMPLES);
        dtostrf((((value) / 3.3) - 0.1515) / 0.00636, 6, 2, sensors[1].c_meas);

        sensors[2].c_meas = calloc(6, sizeof(char*));
        value = getA2dValue(sensors[2].a2d_channel,SAMPLES);
        dtostrf(value, 6, 2, sensors[2].c_meas);

        char* data = createSensorMeasuresPayload(sensors, 3);
        //Crea la trama con direccionamiento de 16 bits
        XBee_CreateTX16Packet(&packet, 0x52, mac, 0x00, data, strlen(data));
        //Envía la trama
        XBee_SendPacket(&packet, strlen(data));
        XBee_ResetPacket(&packet);

        //Libera la memoria
        free(sensors[0].c_meas);
        free(sensors[1].c_meas);
        free(sensors[2].c_meas);
        free(data);

        _delay_ms(10);

        //Configure Timer 1 : prescale 8
        TCCR1B &= ~_BV(CS10);
        TCCR1B |= _BV(CS11);
        TCCR1B &= ~_BV(CS12);
        //Configure Timer 0 : clk T0
        TCCR0B |= _BV(CS00);
        TCCR0B |= _BV(CS01);
        TCCR0B |= _BV(CS02);

        //power_adc_disable();
        a2dOff();

        sei();
}
```

3.2.6. MOTE DE DETECCIÓN

La configuración de sensores del mote, se establece en el código, empleando un vector de tipos Element, explicado en el punto 7.2.2.1.2:

```
static XBeePacket packet;
struct Element sensors[1] = {{ "4", BINARY, 0, "", '0' }};//PIR
static uint32_t mac;
```

Una vez que el mote se ha asociado y configurado, se inicia el modo normal de funcionamiento y para ello, se activa la interrupción que usará el PIR.

```
/** Tras una asociación satisfactoria, inicia el modo normal*/
void init_normal_mode() {
	//Port PD3 as input : INT1
	DDRD &= ~_BV(PIND3); //Realmente no es necesario según el data
	//PCINT mask
	PCMSK2 |= _BV(PCINT19);//Haciendo esto puedo usar power down si es necesario	//Enable	PCINT19 interrupt
	PCICR |= _BV(PCIE2);
}
```

La rutina de interrupción asociada simplemente genera una trama para el XBee y la envía, indicando con un valor binario que ha detectado presencia.

```
/*PIR interrupt*/
ISR(PCINT2_vect)
{
	cli();
	//Pon a true la medida
	sensors[0].b_meas = '1';
	//Crea la trama de datos
	char* data = createSensorMeasuresPayload(sensors, 1);
	XBee_CreateTX16Packet(&packet, 0x52, mac, 0x00, data, strlen(data));
	//Envía la trama
	XBee_SendPacket(&packet, strlen(data));
	//Libera la memoria
	XBee_ResetPacket(&packet);
	free(data);
	//Evita interrupciones repetitivas
	_delay_ms(5000);
	sei();
}
```

3.2.7. MOTE DIMMER

Se ha elegido el terminal INT1 y su rutina de interrupción asociada.

```
/**
 * Configura el pin INT1 como entrada de interrupción
 * para el detector de paso por cero.
 * Dada la señal que nos llega(cuadrada de la misma freq que la red),
 * tenemos que configurarlo al cambio(toggle).
 */
void zero_cross_config() {
	EICRA |= _BV(ISC10); //Configurar los registros de interrupcion para INT1 : toggle
	EICRA &= ~_BV(ISC11);
	EIMSK |= _BV(INT1); //Habilitar la máscara de interrupciones para INT1
}




/*
 * Rutina interruption de INT1
 *
 * El detector de paso por cero(Zero-cross detector), genera un pulso, que aplicado al
 * pin INT1,genera una interrupción. Ésta, arranca un timer que lo emplearemos para
 * contar el tiempo equivalente al ángulo de disparo.
 * Configuramos el timer en modo Fast PWM con prescaler a 1024 que cuenta hasta MAX(FF).
 * De esta forma, el TCNT cuenta de 0 a 255 en unos 32 ms y vuelve a cero,
 * pero como esta interrupción se ejecuta cada paso por cero, el contador se reinicia,
 * no perdiendo el sincronismo con la señal de entrada.
 * Haciendo esta interrupción NOBLOCK, en caso de que en el mismo instante nos venga
 * una interrupción del XBee(INT0),
 * esta segunda no es ignorada, sino retrasada(delay) para cuando salgamos de la ISR
 * de INT1, por lo que no perdemos el nuevo valor que hemos establecido.
 * */

ISR(INT1_vect)
{
	cli();
	//Timer 2 stopped
	TCCR2B &= ~_BV(CS20);
	TCCR2B &= ~_BV(CS21);
	TCCR2B &= ~_BV(CS22);
	TCNT2 = 0;
	OCR2A = controllers[0].current_Action;
	//Enable OC2A interrupt
	TIMSK2 |= _BV(OCIE2A);
	//Prescaler 1024 -> Cuenta de 0 a FF en unos 32ms con un clock de 8Mhz
	TCCR2B |= _BV(CS20);
	TCCR2B |= _BV(CS21);
	TCCR2B |= _BV(CS22);
	sei();
}
```

Cada vez que se ejecuta esta interrupción, es decir cada paso por cero, se para el timer, se reinicia el contador y se vuelve a iniciar.

Dada la frecuencia de la red eléctrica de 50 Hz, cada semiciclo tiene una duración de 10 ms y hay que contar tiempo durante el semiciclo. Por lo tanto el contador que elegido ha de ser capaz de contar durante ese tiempo sin desbordarse.

Empleando un contador de 8 bits y prescalando al máximo (1024) se consigue un contador de unos 31 ms, la menor cifra posible y mayor que el tiempo de semiciclo.

$$f' = \frac{8 * 10^6[Hz]}{1024} = 7.8125\ Khz$$

$$T' = \frac{1}{f'} = 30{,}51\ ms$$

Se ha elegido el Timer 2 y se ha configurado para que genere una interrupción cuando alcanza el valor contenido en el registro OCR2A, que corresponde al ángulo de disparo.

```
/**
 * Configura el timer 2(8 bits), para generar una rampa
 * usando el modo CTC.
 * Habilitaremos la interrupción de comparación con OCR2A
 * e implementaremos su rutina adecuadamente.
 * Debemos generar un pulso de disparo de duración determinada y para ello
 * nos interesa un comportamiento toggle
 */
void timer_ramp_config() {
	//Timer 2 stopped
	TCCR2B &= ~_BV(CS20);
	TCCR2B &= ~_BV(CS21);
	TCCR2B &= ~_BV(CS22);
	TCNT2 = 0;
	OCR2A = controllers[0].current_Action;
	//CTC
	TCCR2A &= ~_BV(WGM20);
	TCCR2A |= _BV(WGM21);
	TCCR2B &= ~_BV(WGM22);
	//Enable OC2A interrupt
	TIMSK2 |= _BV(OCIE2A);
}
```

Hasta ahora tenemos la detección del paso por cero, como una interrupción en el pin INT1 que arranca el Timer 2, que cuenta el tiempo necesario hasta el ángulo de disparo deseado.

Queda por resolver la generación del pulso que va hacia el triac optoacoplado y que físicamente activa el tiristor. El pulso, tiene una duración determinada cuya especificación viene dada por la figura:

Teniendo en cuenta que el pulso hacia el triac optoacoplado lo suministra el microcontrolador, la corriente debe ser suficientemente baja para no saturar la salida del pin.

Para la generación del pulso se ha empleado el mismo Timer 2 que para la rampa principal, de tal forma que lo que cambia es el tiempo que cuenta, que en el caso de la rampa principal se ha establecido en 32 ms y en el caso del pulso de disparo son 5 ms, de esta forma no es necesario emplear un segundo contador.

Se ha implementado una solución con estados, de tal forma que cuando el pin conectado a la entrada del triac está en un nivel bajo, al entrar en la interrupción lo activa y reconfigura el tiempo de rampa con el valor asociado al pulso. En el caso que se haya completado el tiempo de pulso, se desactiva el pin y se reconfigura el registro de comparación para generar la rampa.

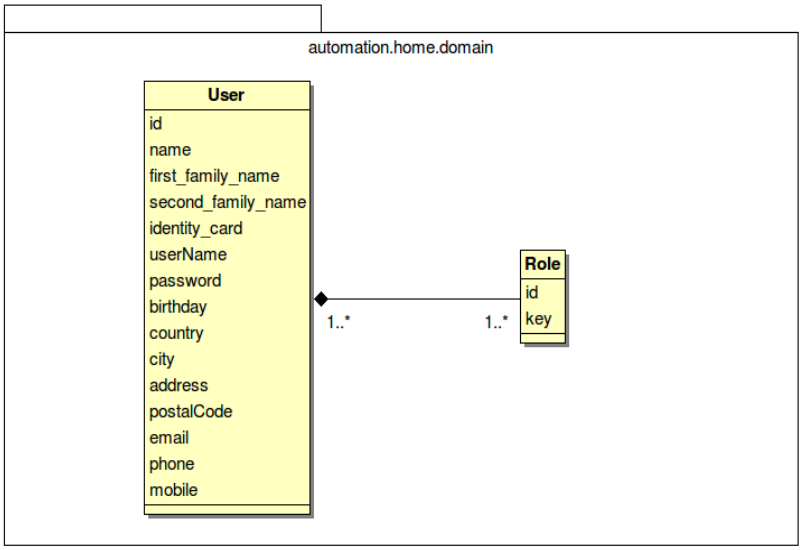
```
/**
 * Solución con estados
 */
ISR(TIMER2_COMPA_vect)
{
	cli();
	switch (pulse_state) {
	case OFF:
		PORTD |= _BV(_OUT);
		OCR2A = PULSE_WIDTH;
		pulse_state = ON;
		break;
	case ON:
		PORTD &= ~_BV(_OUT);
		OCR2A = 0xFF;
		//Disable OCIE2A interrupt
		TIMSK2 &= ~_BV(OCIE2A);
		pulse_state = OFF;
		break;
	}
	sei();
}
```

3.3. SERVIDOR DE APLICACIONES

3.3.1. MODELO ORIENTADO A OBJETOS

Los siguientes diagramas UML presentan las entidades y sus relaciones, así como algunos aspectos de la implementación orientada a objetos:

1. Usuarios y roles: Un usuario puede tener varios roles, por ejemplo, un usuario podría ser cliente de un sistema domótico y a la vez trabajador de la empresa. Así mismo, un rol puede ser usado por varios usuarios.



El modelo de usuario según el diagrama mostrado es el siguiente:

USER.JAVA

```
public class User {

        private int id;
        private IdentityCard identityCard;
        private String identityCardValue;
        private String userName;
        private String name;
        private String firstFamilyName;
        private String secondFamilyName;
        private String password;
        private Date birthday;
        private String country;
        private String city;
        private String address;
        private int postalCode;
        private int longitude;
        private int latitude;
        private byte[] photo;
        private Sex sex;
        private String email;
```

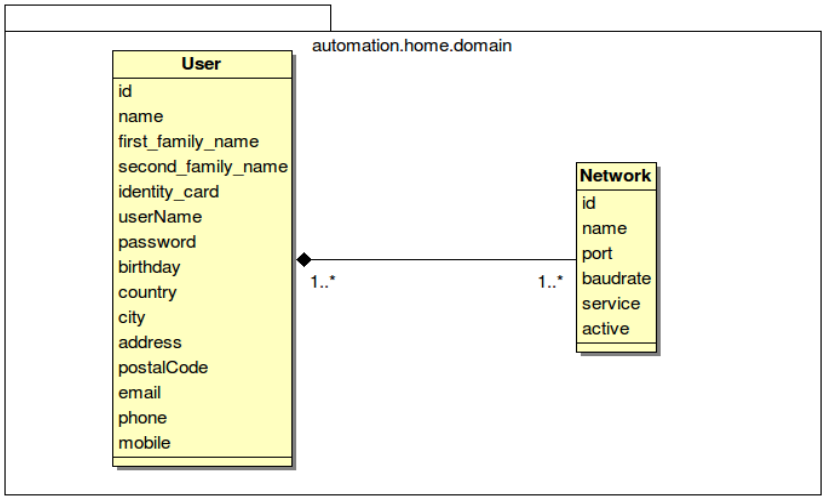
```
        private String phone;
        private String mobile;
        private List<Role> roles;
}
```

Los roles los usamos para controlar el acceso a determinadas zonas de la aplicación, por lo tanto es algo estático, un rol no cambia, lo que cambia es a quién se le asigna. Una forma elegante de programar propiedades que no cambian es mediante el uso de clases Enumerados, que listan una serie de propiedades estáticas, que las usan el resto de clases.

ROLE.JAVA

```
public enum Role {

    ADMINISTRATOR(1,"role.administrator"),
    USER(2, "role.user"),
    NETMANAGER(3, "role.netmanager");
        /** Identificador*/
        private final int id;
        /** Clave*/
        private final String key;

        Role(int id, String key) {
                this.id = id;
                this.key = key;
        }
```

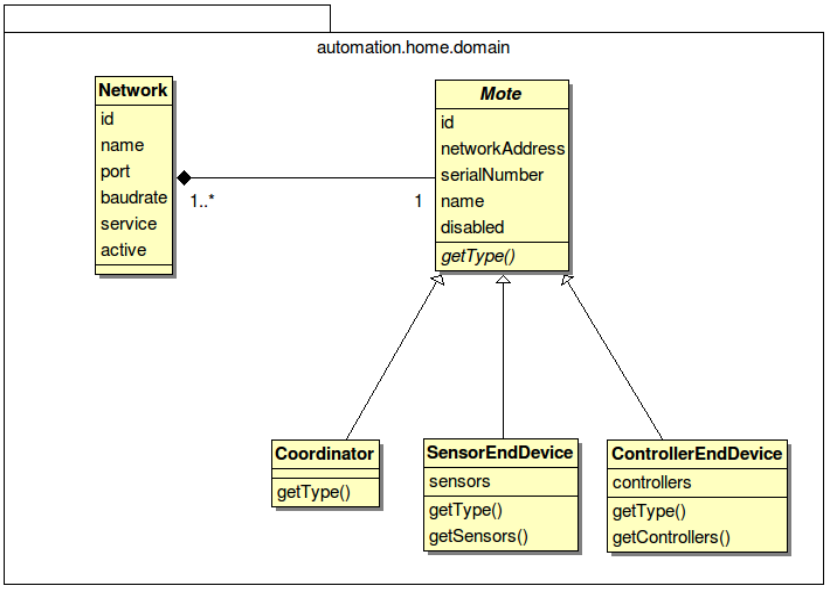
2. Usuarios y redes: Un usuario puede ser el titular de una instalación domótica y puede ser al mismo tiempo un mero usuario con privilegios restringidos de otra instalación. Y el modelo de red equivalente:

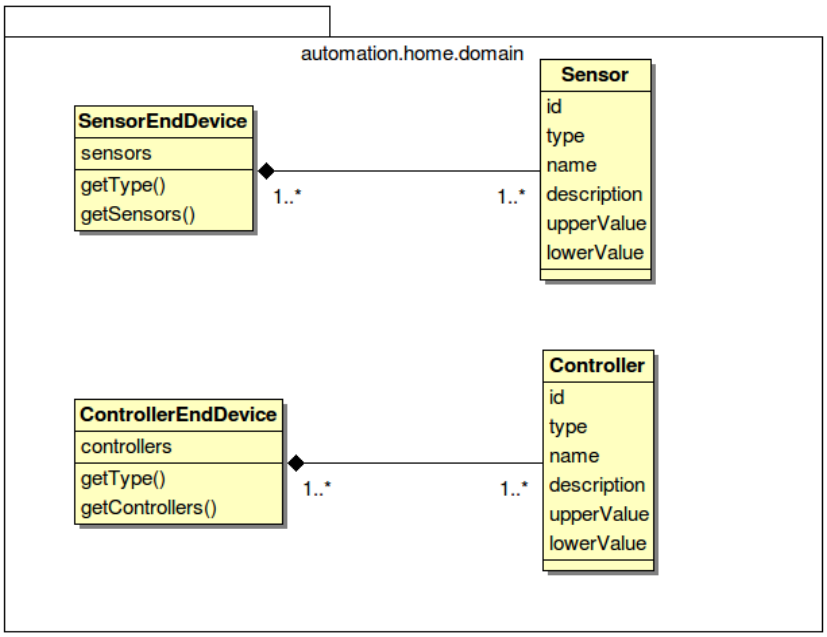


NETWORK.JAVA

```
public class Network {

        private int id;
        private String name;
        private String port;
        private String service;
        private Baudrate baudrate;
        private boolean active;
        private Mote coordinator;
}
```

3. Redes y motes: Evidentemente una red estará al menos formada un por un mote, el coordinador PAN y muy posiblemente por varios end devices.





Un mote puede tener instalados bien sensores si se trata de un mote de sensado o actuadores si es un mote de control.

El modelo de mote emplea una característica fundamental de los lenguajes orientados a objetos, la herencia, a través de la cual se tiene un modelo base que es común para todos los casos y modelos concretos que derivan del modelo base, añadiendo la funcionalidad que necesitan.

MOTE.JAVA

```
/**
 * Modela un mote genérico
 *
 * <p>Sirve como base para los tipos de motes. Cada tipo añade alguna
 * capacidad al mote base.</p>
 */
public abstract class Mote {

	private int id;			//Identificador en el sistema
	private Network network;		//Red a la que pertenece
	private XBeeAddress16 networkAddress;//Dirección de red
	private XBeeAddress64 serialNumber;//Número de serie
	private String name;			//Nombre para el usuario
	private boolean disabled;		//¿Inhabilitado?

	public abstract MoteType getType();
}
```

Define los atributos generales y un método abstracto que deberán definir las clases concretas. Se puede observar cómo se ha llevado la relación conceptual entre redes y motes al modelo de clases, usando un atributo network, que identifica la red a la que pertenece el mote.

Un coordinador es un mote básico, que no dispone de sensores ni actuadores, por lo tanto su implementación es sencilla.

COORDINATOR.JAVA

```
/**
 * Modela un coordinador
 * <p>Un coordinador es un mote básico</p>
 */
public class Coordinator extends Mote {

	public Coordinator() {
	}

	public Coordinator(int id) {
		super(id);
	}

	@Override
	public MoteType getType() {
		return MoteType.COORDINATOR;
	}
}
```

En el caso de un mote sensor, además de disponer de las características generales, tendrá un conjunto de sensores instalados, que se refleja como una lista de sensores.

SENSORENDDEVICE.JAVA

```
/**
 * Modela un mote sensor
 * <p>Dispone de un conjunto de sensores</p>
 */
public class SensorEndDevice extends Mote {

	private List<Sensor> sensors;

	public SensorEndDevice() {
		sensors = new ArrayList<Sensor>();
	}

	public SensorEndDevice(int id) {
		super(id);
	}

	public SensorEndDevice(XBeeAddress64 serialNumber) {
		super(serialNumber);
	}

	public List<Sensor> getSensors() {
		return sensors;
	}
```

```
	public void setSensors(List<Sensor> sensors) {
		this.sensors = sensors;
	}

	@Override
	public MoteType getType() {
		return MoteType.SENSOR_END_DEVICE;
	}
}
```

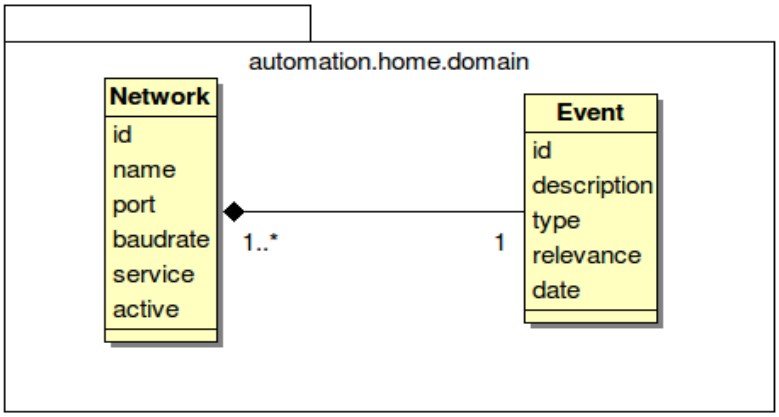
El caso de un mote de control es idéntico al de un mote sensor con la diferencia de que en este caso, dispone de actuadores instalados.

CONTROLLERENDDEVICE.JAVA

```
/**
 * Modela un mote controlador.
 *
 * <p>Dispone de un conjunto de controladores instalados</p>
 */
public class ControllerEndDevice extends Mote {

	private List<Controller> controllers;

	public ControllerEndDevice() {
	}

	public ControllerEndDevice(int id) {
		super(id);
	}

	public List<Controller> getControllers() {
		return controllers;
	}

	public void setControllers(List<Controller> controllers) {
		this.controllers = controllers;
	}

	@Override
	public MoteType getType() {
		return MoteType.CONTROLLER_END_DEVICE;
	}
}
```

Como se puede observar, cada una de las clases concretas, implementa el método abstracto getType() como más le conviene.

4. Eventos de una red: En una red suceden un conjunto de eventos asociados.



Como se puede ver, un evento se emplea únicamente para describir un hecho o una situación dentro de una red y por lo tanto pueden ser de varios tipos. Además no todos los eventos tienen la misma importancia y eso se refleja en la existencia de la propiedad relevance.

EVENT.JAVA

```
/**
 * Evento del sistema
 * Se usa para tener un log de lo que ocurre durante el funcionamiento
 * del sistema.
 * Estos eventos se podrán consultar y se muestran los últimos n eventos
 * en la ventana principal al acceder a la aplicación.
 */
public class Event {
	private int id;
	private int networkId;
	private String description;
	private EventType type;
	private EventRelevance relevance;
	private Date date;
}
```

Tipos de eventos de sistema se pueden plantear decenas, pero a modo de demostración se han implementado los siguientes:

EVENTTYPE.JAVA

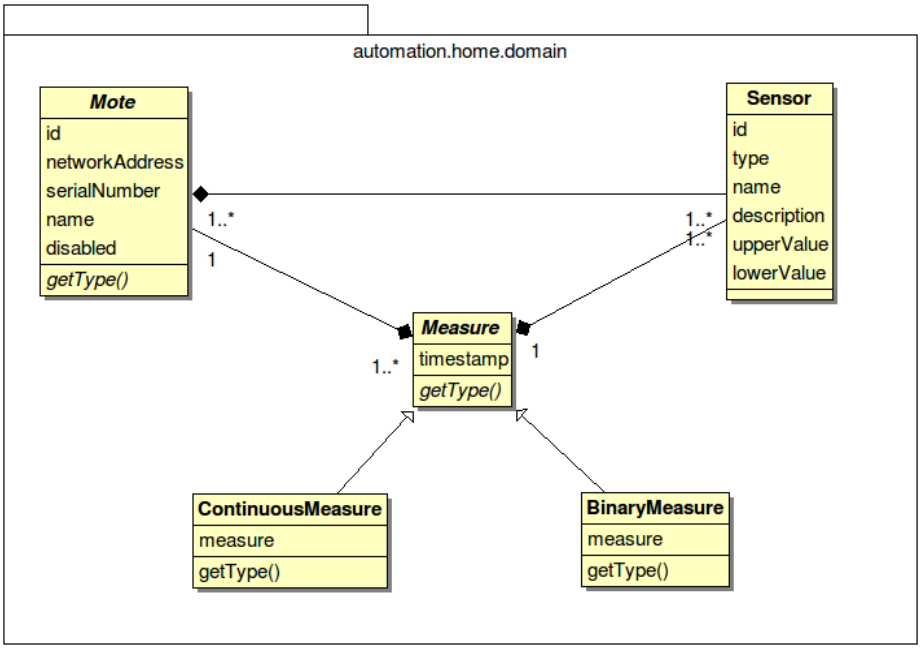
```
/** Tipos de eventos de sistema*/
public enum EventType {

        USER_ACCESS(1, "eventType.userAccess"),
        USER_EXIT(2, "eventType.userExit"),
        START_NETWORK(3, "eventType.startNetwork"),
        COORDINATOR_CHANGED(4, "eventType.coordinatorChanged"),
        MOTE_ADDED(5, "eventType.moteAdded"),
        OVER_TEMPERATURE(6,"eventType.overTemperature"),
        UNDER_TEMPERATURE(7,"eventType.underTemperature"),
        EQUAL_TEMPERATURE(8,"eventType.equalTemperature"),
        ...

        private final int id;
        private final String key;

}
```

La relevancia de los eventos está más acotada en cuato al número de casos posibles y una granularidad de cinco niveles como la que se presenta parece bastante razonable para acomodar las distintas situaciones:

EVENTRELEVANCE.JAVA

```
/** Relevancia o severidad de los eventos de sistema*/
public enum EventRelevance {

        NONE(1, "eventRelevance.none"),
        MINOR(2, "eventRelevance.minor"),
        MAJOR(3, "eventRelevance.major"),
        CRITICAL(4, "eventRelevance.critical"),
        BLOCKER(5, "eventRelevance.blocker");

        private final int id;
        private final String key;
}
```

5. Medidas: Un sensor principalmente va a proporcionar medidas que de forma general se pueden agrupar en dos grandes bloques, continuas y binarias. Las continuas van a ser proporcionadas por sensores analógicos o digitales que tienen muchos estados intermedios entre los límites de medición y las medidas binarias las proporcionan sensores que disponen de dos estados de sensado, activo e inactivo. Las medidas las vamos a considerar como entidades no aisladas de tal forma que siempre las proporciona un sensor que está instalado en un mote, de tal forma que las relacionamos en una estructura jerárquica.



Al mismo tiempo se genera otra estructura jerárquica de herencia entre los tipos de medidas, de tal forma que a partir de la clase base Measure, derivan las clases concretas para los dos tipos de medidas.

MEASURE.JAVA

```
/**
 * Modela una medida de un sensor instalado en un mote
 */
public abstract class Measure {

	private Mote mote;
	private Sensor sensor;
	private Timestamp timestamp;

}
```

La clase empleada para almacenar una medida continua, extiende la capacidad de la clase Measure, añadiendo un atributo de alta precisión de tipo Double que sigue la norma IEEE 754.

CONTINUOUSMEASURE.JAVA

```
/**
 * Modela una medida de naturaleza continua.
 *
 * <p>Una medida continua tiene infinitos estados siempre dentro del rango
 * de funcionamiento del sensor, por lo que el valor de este tipo de medida
 * se ha de considerar de la mayor precisión posible.<br />
 * Extiende la capacidad de {@link Measure} para señales analógicas.</p>
 */
public class ContinuousMeasure extends Measure {
```

```
	private double measure;

	public double getMeasure() {
		return measure;
	}

	public void setMeasure(double measure) {
		this.measure = measure;
	}

	@Override
	public MeasureType getMeasureType() {
		return MeasureType.CONTINUOUS;
	}

	@Override
	public String toString() {
		return ToStringBuilder.reflectionToString(this);
	}
}
```

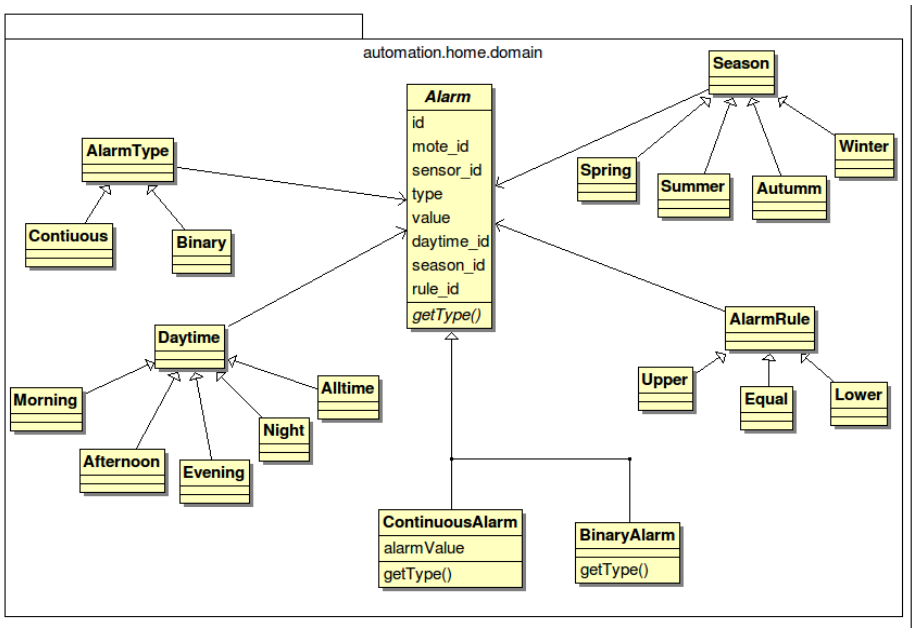
Igualmente, para las medidas de dos estados, se ha generado una clase similar a la anterior pero el atributo que almacena la medida, tiene únicamente dos estados, lo que hace evidente el uso de un tipo Boolean.

BINARYMEASURE.JAVA

```
/**
 * Modela una medida de naturaleza digital o binaria.
 *
 * <p>Una medida binaria tiene únicamente dos estado, activa e inactiva,
 * y sirve para modelar sensores que son del tipo todo-nada.
 * Dada esta naturaleza, la forma de modelarla es con un tipo
 * {@link Boolean} que tiene dos estados.
 * <br />Extiende la capacidad de {@link Measure} para señales binarias</p>
 */
public class BinaryMeasure extends Measure {

	private boolean measure;

	public boolean isMeasure() {
		return measure;
	}

	public void setMeasure(boolean measure) {
		this.measure = measure;
	}

	@Override
	public MeasureType getMeasureType() {
		return MeasureType.BINARY;
	}

	@Override
	public String toString() {
		return ToStringBuilder.reflectionToString(this);
	}
```

}

6. Alarmas: Una alarma es disparada por una medición anómala de un sensor que está instalado en un mote. Una alarma es configurable en estacionalidad y temporalidad.



La clase base de alarma presenta otros atributos adicionales que únicamente son de utilidad para el procesamiento de las alarmas y no tienen persistencia.

ALARM.JAVA

```
/**
 * Clase base para las alarmas
 * Contempla condiciones de alarma como, la estación de año o el
 * intervalo del día en que se puede producir
 */
public abstract class Alarm {

	private int id;
	private String name;
	private Mote mote;
	private Sensor sensor;
	private AlarmRule rule;
	private DayTimes dayTime;
	private Seasons season;
	private Measure measure;
	private boolean fired;
	private Event event;
}
```

Dados los dos tipos de medidas vamos a tener en consecuencia dos tipos de alarmas, que se han implementado usando herencia.

CONTINUOUSALARM.JAVA

```
/**
 * Alarma para sensores analógicos, que dan medidas contínuas.
 * Incorpora el valor a partir del cual se debe disparar la alarma y la regla
 * que lo dispara, si es mayor, menor o igual
 */
public class ContinuousAlarm extends Alarm {

	private int alarmValue;

	public void setAlarmValue(int alarmValue) {
		this.alarmValue = alarmValue;
	}

	public int getAlarmValue() {
		return alarmValue;
	}

	@Override
	public AlarmType getAlarmType() {
		return AlarmType.CONTINUOUS;
	}
}


/** Alarma para sensores binarios */
public class BinaryAlarm extends Alarm {

	@Override
	public AlarmType getAlarmType() {
		return AlarmType.BINARY;
	}
}
```

El tipo de la alarma sigue el mismo patrón que el de la medida y la implementación es similar:

ALARMTYPE.JAVA

```
/**
 * Tipos de alarmas
 * Contínua, para los sensores analógicos >> Servirá para comparar con
 * un valor
 * concreto.<br />
 * Binaria, para los sensores todo-nada >> Compara con un estado
 */
public enum AlarmType {

	CONTINUOUS(1, "alarmType.continuous"),
	BINARY(2, "alarmType.binary");
	private final int id;
	private final String key;
}
```

La regla que dispara la alarma va en función de las medidas proporcionadas por el sensor y así para un sensor que proporciona medidas continuas, una alarma se puede producir por sobre pasar un valor límite, tanto superior como inferior y el caso particular de alcanzar un valor concreto. Para un sensor binario, la alarma se puede producir por activación o desactivación, dependiendo del sensor empleado o de la lógica implementada.

ALARMRULE.JAVA

```
public enum AlarmRule {

	UPPER(1, "alarmRule.upper"),
	LOWER(2, "alarmRule.lower"),
	EQUAL(3, "alarmRule.equal"),
	ON(4, "alarmRule.on"),
	OFF(5, "alarmRule.off");

	private final int id;
	private final String key;

}
```

Como elementos de configuración de la alarma planteamos la estacionalidad, es decir, en qué estación del año estará activa una alarma concreta y la temporalidad o en qué franjas horarias estará activa. Para implementar esto de forma sencilla se han planteado dos enumerados estáticos, que pueden ser suficientes para casos sencillos.

SEASONS.JAVA

```
/** Estaciones del año */
public enum Seasons {

	ALLTIME(0, "season.allTime"),
	SPRING(1, "season.spring"),
	SUMMER(2, "season.summer"),
	AUTUMM(3, "season.autumm"),
	WINTER(4, "season.winter");

	private final int id;
	private final String key;
}
```

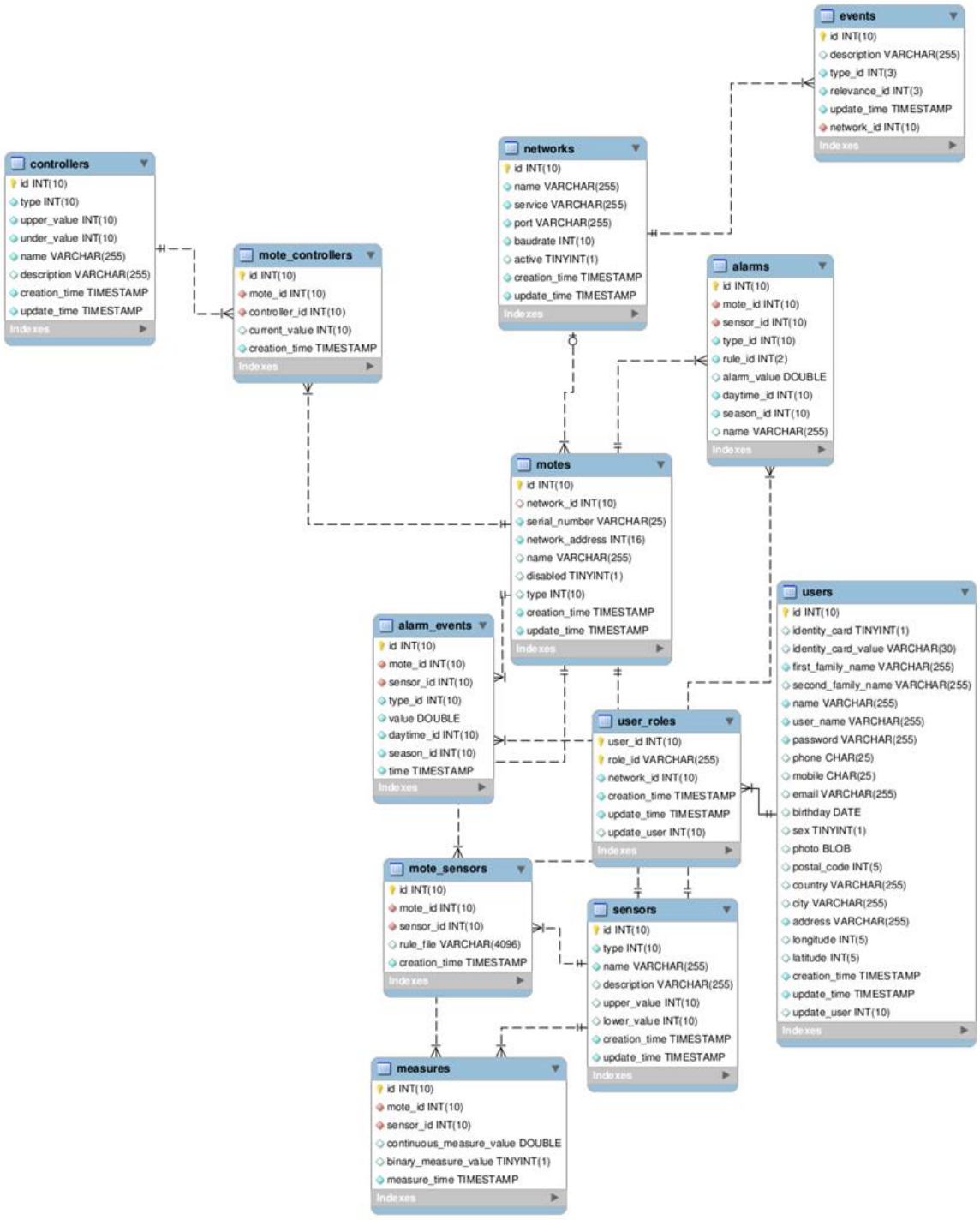
DAYTIMES.JAVA

```
/** Franjas horarias */
public enum DayTimes {

	MORNING(1, "daytime.morning"),
	AFTERNOON(2, "daytime.afternoon"),
	EVENING(3, "daytime.evening"),
	NIGHT(4, "daytime.night"),
	ALLTIME(5, "daytime.alltime");
	private int id;
	private String key;
}
```

3.3.2. MODELO RELACIONAL

El modelo relacional pretende llevar al nivel de base de datos el modelo orientado a objetos que se ha desarrollado en el anterior punto. Por lo tanto, los atributos de las clases se mapearán como columnas de las tablas y las relaciones se identificarán como claves foráneas.

El siguiente esquema presenta el resultado del mapeo de objetos sobre tablas de la base de datos y ha sido generado con la herramienta MySQLWorkbench.



Este modelo no se crea de forma automática sino que es generado a partir de scripts en lenguaje SQL que se han escrito. Básicamente son sentencias de creación de tablas en las que se describe la estructura de columnas, las claves primarias y foráneas. A continuación presentamos algunos ejemplos, y se remite al lector a que inspeccione el soporte digital si se siente interesado en el conjunto completo.

USERS.SQL

```
-- ----------------------------------------------------------------------
-- ESTRUCTURA DE LOS USUARIOS
-- ----------------------------------------------------------------------
DROP TRIGGER IF EXISTS BU_users_update_time;
DROP TRIGGER IF EXISTS BU_user_roles_update_time;
DROP TABLE IF EXISTS user_roles;
DROP TABLE IF EXISTS users;

CREATE TABLE IF NOT EXISTS users (
        id INT(10) NOT NULL auto_increment,
        identity_card TINYINT(1) DEFAULT NULL,
        identity_card_value VARCHAR(30) DEFAULT NULL,
        first_family_name VARCHAR(255) NOT NULL,
        second_family_name VARCHAR(255) DEFAULT NULL,
        name VARCHAR(255) NOT NULL,
        user_name VARCHAR(255) NOT NULL,
        password VARCHAR(255) NOT NULL,
        phone CHAR(25) DEFAULT NULL,
        mobile CHAR(25) DEFAULT NULL,
        email VARCHAR(255) DEFAULT NULL,
        birthday DATE DEFAULT NULL,
        sex TINYINT(1) DEFAULT NULL,
        photo BLOB DEFAULT NULL,
        -- Para el posicionamiento
        postal_code INT(5) DEFAULT NULL,
        country VARCHAR(255) DEFAULT NULL,
        city VARCHAR(255) DEFAULT NULL,
        address VARCHAR(255) DEFAULT NULL,

        creation_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        update_time TIMESTAMP,
        update_user INT(10),

        CONSTRAINT users_PK PRIMARY KEY (id),
        INDEX users_I_name(name, first_family_name, second_family_name),
        INDEX users_I_first_family_name(first_family_name, second_family_name, name)
) ENGINE=InnoDB;
```

MOTES.SQL

```
-- ---------------------- --
-- ESTRUCTURA DE LA RED
-- ---------------------- --
CREATE TABLE IF NOT EXISTS networks (
        id INT(10) NOT NULL auto_increment,
        name VARCHAR(255) NOT NULL,
        service VARCHAR(255) NOT NULL,
        port VARCHAR(255) NOT NULL,
        baudrate INT(10) NOT NULL,
        active BOOLEAN,
        creation_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        update_time TIMESTAMP,
        PRIMARY KEY networks_PK (id),
        INDEX networks_I_name(name)
) ENGINE=InnoDB;


-- ------------------------ --
-- ESTRUCTURA DE LOS MOTES
-- ------------------------ --
CREATE TABLE IF NOT EXISTS motes (
        id INT(10) NOT NULL auto_increment,
        network_id INT(10),
        serial_number VARCHAR(25) NOT NULL,
        network_address INT(16) NOT NULL,
        name VARCHAR(255) DEFAULT NULL,
        disabled TINYINT(1) DEFAULT 0,
        type INT(10) DEFAULT NULL,
        creation_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        update_time TIMESTAMP,
        PRIMARY KEY motes_PK (id),
        UNIQUE KEY motes_U_network_address (network_id,network_address),
        FOREIGN KEY (network_id) REFERENCES networks(id),
        INDEX motes_I_address(network_address)
) ENGINE=InnoDB;


-- --------------------------- --
-- ESTRUCTURA DE LOS SENSORES
-- --------------------------- --
CREATE TABLE IF NOT EXISTS sensors (
        id INT(10) NOT NULL auto_increment,
        type INT(10) NOT NULL,                  -- Tipo de sensor
        name VARCHAR(255) NOT NULL,
        description VARCHAR(255) DEFAULT NULL,
        upper_value INT(10) DEFAULT NULL,       -- Límite superior de las medidas
        lower_value INT(10) DEFAULT NULL,       -- Límite inferior de las medidas
        creation_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        update_time TIMESTAMP,
        PRIMARY KEY sensors_PK (id)
) ENGINE=InnoDB;
```

```
-- ---------------------------------------- --
-- ESTRUCTURA DE LOS SENSORES DE LOS MOTES
-- ---------------------------------------- --
CREATE TABLE IF NOT EXISTS mote_sensors (
        id INT(10) NOT NULL auto_increment,
        mote_id INT(10) NOT NULL,
        sensor_id INT(10) NOT NULL,
        creation_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        PRIMARY KEY mote_sensors_PK (id),
        FOREIGN KEY (mote_id) REFERENCES motes(id),
        FOREIGN KEY (sensor_id) REFERENCES sensors(id),
        INDEX mote_sensors_I_mote(mote_id),
        INDEX mote_sensors_I_sensor(sensor_id)
) ENGINE=InnoDB;

-- --------------------------
-- ESTRUCTURA DE LAS MEDIDAS
-- --------------------------
CREATE TABLE IF NOT EXISTS measures (
        id INT(10) NOT NULL auto_increment,
        mote_id INT(10) NOT NULL,
        sensor_id INT(10) NOT NULL,
        continuous_measure_value DOUBLE DEFAULT NULL,
        binary_measure_value TINYINT(1) DEFAULT NULL,
        measure_time TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
        PRIMARY KEY measures_PK (id),
        FOREIGN KEY (mote_id) REFERENCES motes(id),
        FOREIGN KEY (sensor_id) REFERENCES sensors(id),
        INDEX measures_I_mote_id(mote_id),
        INDEX measures_I_sensor_id(sensor_id)
) ENGINE=InnoDB;
```

ALARMS.SQL

```
                -- ------------------------------------
-- ESTRUCTURA DE LAS ALARMAS
-- ------------------------------------
CREATE TABLE IF NOT EXISTS alarms (
        id INT(10) NOT NULL auto_increment,
        mote_id INT(10) NOT NULL,
        sensor_id INT(10) NOT NULL,
        name VARCHAR(255) DEFAULT NULL,
        type_id INT(10) NOT NULL,
        rule_id INT(2) NOT NULL,
        alarm_value DOUBLE DEFAULT NULL,
        daytime_id INT(10) NOT NULL,
        season_id INT(10) NOT NULL,
        PRIMARY KEY alarms_PK (id),
        FOREIGN KEY (mote_id) REFERENCES motes(id),
        FOREIGN KEY (sensor_id) REFERENCES sensors(id)
) ENGINE=InnoDB;
```

EVENTS.SQL

```
                -- --------------------------------
-- EVENTOS DEL SISTEMA
-- --------------------------------
CREATE TABLE IF NOT EXISTS events(
        id INT(10) NOT NULL auto_increment,
        network_id INT(10) NOT NULL,
        description VARCHAR(255) DEFAULT NULL,
        type_id INT(3) NOT NULL,
        relevance_id INT(3) NOT NULL,
        update_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        CONSTRAINT events_PK PRIMARY KEY (id),
        CONSTRAINT FK_event_network FOREIGN KEY (network_id) REFERENCES networks(id)
)Engine=InnoDB;
```

3.3.3. VISTA

3.3.3.1. MONITORIZACIÓN DE SENSORES

El script, está programado en JavaScript, un lenguaje muy usado en aplicaciones web dinámicas para crear animaciones y mejorar la interacción del usuario.

Para mostrar unas gráficas vistosas, se ha empleado la librería que ofrece Yahoo, YUI (Yahoo User Interface) y en concreto el componente Chart. Dicho componente se configura dinámicamente para mostrar los datos adaptados a cada sensor, ya que cambian las unidades de medida o los límites de sensado en cada caso.

Para obtener la información se ha configurado una fuente de datos, lo que se conoce como DataSource, para que acceda a una dirección(Servlet) dentro de nuestro servidor y le proporcione la información relativa a las medidas que solicita. Para ello se ha empleado el componente DataSource de la librería de Yahoo, que permite realizar peticiones a distintas fuentes de datos y con distintos formatos.

En este caso hemos decidido que la información devuelta por el servlet, tenga el formato JSON (*JavaScript Object Notation*) que es un formato de intercambio de información ligero que no emplea XML. Los atributos de JSON que recibimos son una fecha (Date) y el valor(Value) de cada medida que se va a pintar.

```
<script type="text/javascript">
  //--- data
  //FIXME Para que funcione la demo, habrá que actualizar la dirección del servidor una vez tenga una IP
  var jsonData = new YAHOO.util.DataSource("http://localhost:8080/homeAutomation/open/realtimeChart.do
                        ?moteId=${mote.id}&sensorId=${sensor.id}");
  //use POST so that IE doesn't cache the data
  jsonData.connMethodPost = true;
  jsonData.responseType = YAHOO.util.DataSource.TYPE_JSON;
  jsonData.responseSchema =
        {
        resultsList: "Results",
        fields: ["Date","Value"]
  };

  formatAxisLabel = function( value )
  {
        return YAHOO.util.Number.format( value,
        {
          suffix: "${sensor.type.units}",
          thousandsSeparator: ",",
          decimalPlaces: 2

        });
  }

  formatDataTipText = function(item, index, series)
  {
        var str = series.displayName + " for " + item.date;
        str += "\n" + YAHOO.example.formatAxisLabel(item[(series.displayName).toLowerCase()]);     return str;
  }

  var yAxis = new YAHOO.widget.NumericAxis();
  yAxis.labelFunction = formatAxisLabel;
  yAxis.title = "${sensor.type.units}";
  var styleDef = {yAxis:{titleRotation:-90}}
```

```
//line chart
 <c:if test="${sensor.type.measureType == measureType.continuous}">
        yAxis.minimum = ${sensor.lowerValue};
        yAxis.maximum = ${sensor.upperValue};
        var mychart = new YAHOO.widget.LineChart("chart_${mote.id}_${sensor.id}", jsonData,
        {
          xField: "Date",
          yField: "Value",
          yAxis: yAxis,
          style: styleDef,
          polling: 2000,
          dataTipFunction:formatDataTipText,
          //only needed for flash player express install
          expressInstall: "assets/expressinstall.swf"
        });
 </c:if>
        //bar chart
 <c:if test="${sensor.type.measureType == measureType.binary}">
          yAxis.minimum = 0;
          yAxis.maximum = 1;
          var mychart = new YAHOO.widget.ColumnChart("chart_${mote.id}_${sensor.id}", jsonData,
          {
                xField: "Date",
                yField: "Value",
                yAxis: yAxis,
                style: styleDef,
                polling: 2000,
                //only needed for flash player express install
                expressInstall: "assets/expressinstall.swf"
          });
        </c:if>

</script>
```

La fuente de datos se ha configurado para que automáticamente realice una petición al servidor cada dos segundos, empleando el atributo *polling* del componente Chart. De esta forma se consigue una gráfica dinámica sin solicitar al usuario que intervenga.

En el lado del servidor hay un servlet dedicado a gestionar las peticiones del script anterior, de tal forma que recibe los parámetros de mote y sensor a través de la URL *http://localhost:8080/homeAutomation/common/jsonChart.do?moteId=${mote.id}&sensorId=${sensor.id}*, obtiene los últimos datos de la base de datos, los procesa y los devuelve en formato JSON para que el componente Chart los pinte.

REALTIMECHARTSERVLET.JAVA

```
/** Genera los datos para el monitor de tiempo real. */
public class RealtimeChartServlet extends HttpServlet {

	@Override
	protected void doPost(HttpServletRequest request,
                                                HttpServletResponse response)
			throws ServletException, IOException {
		int moteId = NumberUtil.getInt(request, "moteId");
		int sensorId = NumberUtil.getInt(request, "sensorId");
		StringBuilder json = new JsonUtil().
                                        getRealtimeMeasures(moteId, sensorId);

		response.setContentType("application/json");
		PrintWriter out = response.getWriter();
		out.write(json.toString());
	}
}
```

Se ha creado la clase de utilidad *JsonUtil*, que encapsula la funcionalidad requerida para crear el JSON.

```
	/**
	 * Genera los datos a enviar usando el formato json para el monitor de tiempo real.
	 *
	 * @param moteId mote
	 * @param sensorId sensor
	 * @return
	 */
	public StringBuilder getRealtimeMeasures(int moteId, int sensorId) {
		StringBuilder stringBuilder = new StringBuilder();
		log.info("Realtime stats...");
		//Obtiene las medidas
		List<Measure> measures = new DbMeasureDAO().load(moteId, sensorId);
		Sensor sensor = new DbSensorDAO().load(sensorId, null);
		SimpleDateFormat sdf = new SimpleDateFormat("hh:mm:ss");
		stringBuilder.append("{\"Results\":[");
		if (!measures.isEmpty()) {
			for (Measure measure : measures) {
				stringBuilder.append("{\"Date\":\"");
				stringBuilder.append(sdf.format(measure.getTimestamp()));
				stringBuilder.append("\",");
				stringBuilder.append("\"Value\":");
				stringBuilder.append("\"");
				if (sensor.getType().getMeasureType().equals(MeasureType.CONTINUOUS)) {
					double measureValue = ((ContinuousMeasure) measure).getMeasure();
					stringBuilder.append(measureValue);
				} else if (sensor.getType().getMeasureType().equals(MeasureType.BINARY)) {
					if (((BinaryMeasure) measure).isMeasure()) {
						stringBuilder.append(1);
					} else {
						stringBuilder.append(0);
					}
				}
				stringBuilder.append("\"");
				stringBuilder.append("}");
				stringBuilder.append(",");
```

```
                }
                stringBuilder.deleteCharAt(stringBuilder.lastIndexOf(","));
            }
            stringBuilder.append("]}");
            log.debug(stringBuilder.toString());
            return stringBuilder;
        }
```

3.3.3.2. HISTÓRICO DE DATOS

Desde el punto de vista de implementación, inicialmente el usuario realiza una selección del mote sensor del que quiere visitar el histórico y un intervalo temporal definido por el enumerado HistoricRange:

HISTORICRANGE.JAVA

```
public enum HistoricRange {

   ALL_TIME(1, "historicRanges.allTime", System.currentTimeMillis()),
   ONE_MONTH(2, "historicRanges.oneMonth", 30 * 24 * 60 * 1000),
   THREE_MONTHS(3, "historicRanges.threeMonths", 90 * 24 * 60 * 1000),
   SIX_MONTHS(4, "historicRanges.sixMonths", 180 * 24 * 60 * 1000),
   NINE_MONTHS(5, "historicRanges.nineMonths", 270 * 24 * 60 * 1000),
   LAST_YEAR(6, "historicRanges.lastYear", 365 * 24 * 60 * 1000),
   LAST_TWO_YEARS(7, "historicRanges.lastTwoYears", 730 * 24 * 60 * 1000),
   LAST_FIVE_YEARS(8, "historicRanges.lastFiveYears", 365 * 5 * 24 * 60 * 1000);

   private final int id;
   private final String key;
   private final long interval;//Duración del rango en milisegundos >> para el chart

   private HistoricRange(int id, String key, long interval) {
         this.id = id;
         this.key = key;
         this.interval = interval;
   }
}
```

El servlet que recibe la petición, obtiene los datos que identifican al mote, al sensor y al intervalo temporal, y genera los datos nuevamente usando el formato JSON.

STATCHARTSERVLET.JAVA

```
/**
 * Genera los datos para la estadística
 */
public class StatChartServlet extends HttpServlet {

   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
         int moteId = NumberUtil.getInt(request, "moteId");
         int sensorId = NumberUtil.getInt(request, "sensorId");
```

```
        HistoricRange historicRange =
                        HistoricRange.parse(NumberUtil.getInt(request, "historicRange"));

        StringBuilder json = new JsonUtil().getHistoricalMeasures(moteId, sensorId, historicRange);

        response.setContentType("application/json");
        PrintWriter out = response.getWriter();
        out.write(json.toString());
    }
}
```

Para generar la información JSON, se ha implementado el código necesario, dentro de la utilidad JsonUtil, que recupera las medidas del sensor instalado en el mote, y las agrupa por el intervalo temporal seleccionado.

A continuación se muestra el fragmento de la implementación que corresponde a un intervalo temporal de más de un mes y menos de un año.

```
/**
 * Genera los datos a enviar usando el formato json para los históricos
 *
 * @param moteId mote
 * @param sensorId sensor
 * @param historicRange rango
 * @return
 */
public StringBuilder getHistoricalMeasures(int moteId, int sensorId,HistoricRange historicRange) {
        List<Measure> measures = new DbMeasureDAO().loadAll(moteId, sensorId);
        Sensor sensor = new DbSensorDAO().load(sensorId, null);
        StringBuilder stringBuilder = new StringBuilder();
        Map<Integer, List<Measure>> group = new HashMap();
        if (historicRange.equals(HistoricRange.THREE_MONTHS)||
historicRange.equals(HistoricRange.SIX_MONTHS)
        || historicRange.equals(HistoricRange.NINE_MONTHS)|| historicRange.equals(HistoricRange.LAST_YEAR))
{
                for (Measure measure : measures) {//Agrupa por meses desde que tenga datos y haz medias
                        Calendar calendar = Calendar.getInstance();
                        calendar.setTimeInMillis(measure.getTimestamp().getTime());
                        if (group.containsKey(calendar.get(Calendar.MONTH))) {
                                ((List<Measure>) group.get(calendar.get(Calendar.MONTH))).add(measure);
                        } else {
                                List<Measure> m = new ArrayList<Measure>();
                                m.add(measure);
                                group.put(calendar.get(Calendar.MONTH), m);
                        }
                }
                Map<String, Object> averaged = new HashMap<String, Object>();
                Set<Integer> keys = group.keySet();
                for (Integer key : keys) {
                        List<Measure> monthlyMeasures = group.get(key);
                        if (sensor.getType().getMeasureType() == MeasureType.BINARY) {
                                averaged.put(Misc.getMonthForInt(key), monthlyMeasures.size());
                        } else {
                                double average = 0;
                                for (Measure measure : monthlyMeasures) {
                                        average += ((ContinuousMeasure) measure).getMeasure();
                                }
                                averaged.put(Misc.getMonthForInt(key), average / monthlyMeasures.size());
```

```
                }
            }
            stringBuilder.append("{\"Results\":[");
            if (!measures.isEmpty()) {
                Set<String> avKeys = averaged.keySet();
                for (String key : avKeys) {
                    stringBuilder.append("{\"Month\":\"");
                    stringBuilder.append(key);
                    stringBuilder.append("\",");
                    stringBuilder.append("\"Value\":");
                    stringBuilder.append("\"");
                    stringBuilder.append(averaged.get(key));
                    stringBuilder.append("\"");
                    stringBuilder.append("}");
                    stringBuilder.append(",");
                }
                stringBuilder.deleteCharAt(stringBuilder.lastIndexOf(","));
            }
            stringBuilder.append("]}");
        } else{. . .}
```

Básicamente, obtiene las medidas del sensor, y en una primera pasada las agrupa por meses para, en una segunda pasada, calcular la media mensual.

Finalmente, se genera el formato JSON para enviarlo a las gráficas de histograma.

En la parte que se muestra al usuario, se ha empleado la misma técnica que en el caso de la monitorización en directo, pero en este caso usando gráficos de barras, más adecuados para mostrar históricos. La forma de construir estos gráficos es la misma que en el caso anterior, con la salvedad de que ahora únicamente se realiza una única petición al servidor para obtener los datos, no de forma periódica.

A continuación se muestra el fragmento que corresponde a un intervalo temporal de más de un mes y menos de un año.

```
<c:if test="${selectedHistoricRange eq historicRangesMap['THREE_MONTHS']
                || selectedHistoricRange eq historicRangesMap['SIX_MONTHS']
                || selectedHistoricRange eq historicRangesMap['NINE_MONTHS']
                || selectedHistoricRange eq historicRangesMap['LAST_YEAR']}">
    <!--
      Los datos vienen agrupados por meses con medias mensuales
      Gráficos de columnas
       -->
    <script type="text/javascript">
        //--- data
        var jsonData = new YAHOO.util.DataSource("http://localhost:8080/homeAutomation/open/statChart.do?moteId=${mote.id}&sensorId=${sensor.id}&historicRange=${selectedHistoricRange.id}"); <%-- ?moteId=${mote.id}&sensorId=${sensor.id} --%>
        //use POST so that IE doesn't cache the data
        jsonData.connMethodPost = true;
        jsonData.responseType = YAHOO.util.DataSource.TYPE_JSON;
        jsonData.responseSchema =
            {
            resultsList: "Results",
            fields: ["Month","Value"]
```

```
		};

		formatAxisLabel = function( value )
		{
			return YAHOO.util.Number.format( value,
			{
				suffix: "${sensor.type.units}",
				thousandsSeparator: ",",
				decimalPlaces: 2

			});
		}
		//--- chart
		var yAxis = new YAHOO.widget.NumericAxis();
		// serán sensor.upperValue y sensor.lowerValue
		yAxis.minimum = ${sensor.lowerValue};
		yAxis.maximum = ${sensor.upperValue};
		yAxis.labelFunction = formatAxisLabel;
		yAxis.title = "${yAxisTitle}";

		//Style object for chart
		var styleDef =
			{
			yAxis:
				{
				titleRotation:-90
			}
		}

		var mychart = new YAHOO.widget.ColumnChart("chart_${mote.id}_${sensor.id}", jsonData,
		{
			xField: "Month",
			yField: "Value",
			yAxis: yAxis,
			style: styleDef,
			//only needed for flash player express install
			expressInstall: "assets/expressinstall.swf"
		});
	</script>

</c:if>
```

3.3.3.3. CONTROL DE ACTUADORES

Para la interfaz de usuario se ha usado el componente Slider de la librería de Yahoo, que representa un deslizador o potenciómetro lineal. Las propiedades de este componente se establecen cuando se selecciona el controlador y son las relacionadas con los valores máximo y mínimo de la acción, las unidades de control...etc.

El siguiente fragmento de código JSP crea un componente slider y le establece las propiedades del controlador.

```
<div id="slider-bg-${mote.id}-${controller.id}" class="yui-v-slider" tabindex="-1" title="Slider">
  <div id="slider-thumb-${mote.id}-${controller.id}" class="yui-slider-thumb">
        <img src="${pageContext.request.contextPath}/yui/slider/assets/thumb-bar.gif">
  </div>
</div>
```

```
<p>
  <label><fmt:message key="controller.currentValue"/></label>
        <input autocomplete="off" id="slider-converted-value-${mote.id}-${controller.id}" type="text"
        value="0" size="4" maxlength="4" readonly="true" />${controller.type.units}
</p>

<script type="text/javascript">

   (function() {
                var Event = YAHOO.util.Event,
                Dom = YAHOO.util.Dom,
                lang = YAHOO.lang,
                slider,
                bg="slider-bg-${mote.id}-${controller.id}",
                        thumb="slider-thumb-${mote.id}-${controller.id}",
                textfield="slider-converted-value-${mote.id}-${controller.id}"

                // Límite de movimiento hacia arriba
                var topConstraint = 0;

                // Límite de movimiento hacia abajo(200px)
                var bottomConstraint = 200;

                // Factor de escala para convertir a píxeles el rango de valores
                var scaleFactor = -(${(controller.upperValue - controller.lowerValue)/200});

                //Factor de incremento en cada variación(10px)
                var keyIncrement = 10;

                Event.onDOMReady(function() {

                        slider = YAHOO.widget.Slider.getVertSlider(bg,
                                                thumb, topConstraint, bottomConstraint);

                        //Valor real
                        slider.getRealValue = function() {
                                return Math.round((this.getValue()
                                                        * scaleFactor) + ${controller.upperValue});
                        }
                        slider.subscribe("change", function(offsetFromStart) {
                                .........
                                .........
                                //Servlet al que llama cada vez que se mueve el slider
                                var  sUrl =
http://localhost:8080/homeAutomation/network/mote/realTimeControlUpdate.do";
                                //Parámetros que se envían
                                var postData = "value=" + actualValue + "&moteId=" + ${mote.id} +
"&controllerId=" + ${controller.id};

                                //Conexión asíncrona con el servidor usando un POST
                                YAHOO.util.Connect.asyncRequest('POST', sUrl, callback, postData);

                        });
                        .........
                        .........
                });
   })();
```

</script>

Como se puede apreciar, cada vez que se desplaza el slider, envía una solicitud HTTP al servlet RealTimeControlUpdate, al que le pasa los parámetros de mote, controlador y el nuevo valor.

REALTIMECONTROLUPDATESERVLET.JAVA

```
public class RealTimeControlUpdateServlet extends HttpServlet {

	private final Logger log = Logger.getLogger(this.getClass());

	@Override
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		int moteId = NumberUtil.getInt(request, "moteId");
		Mote mote = new DbMoteDAO().load(moteId);
		Controller controller = new DbControllerDAO().load(NumberUtil.getInt(request,
									"controllerId"));
		//Comprueba que el controlador está en el mote
		if (!((ControllerEndDevice) mote).getControllers().contains(controller)) {
			log.error("El mote no contiene al controlador que se pasa por parámetro");
			throw new HomeFailure("El mote no contiene al controlador que se pasa por
									parámetro");
		}
		int value = NumberUtil.getInt(request, "value");
		if (controller.getLowerValue() < value && value < controller.getUpperValue()) {
			Network network = NetworkUtil.getCurrentNetwork(request);
			try {
				RmiUtil.sendTxRequest64(network, mote, new int[]{value});
			} catch (RemoteException ex) {
				log.error(ex);
			} catch (NamingException ex) {
				log.error(ex);
			}
		} else {
			log.error("El valor que se intenta establecer, está fuera del rango
									permitido");
			throw new HomeFailure("El valor que se intenta establecer, está fuera del
									rango permitido");
		}
	}
}
```

El servlet, recoge los parámetros de la petición HTTP y realiza comprobaciones de seguridad como si el mote contiene al controlador y si el valor que se pasa está dentro del rango permitido. Si pasa las comprobaciones, se envía una trama de control remotamente, haciendo uso de la utilida creada para gestionar la conexión RMI con los sistemas locales.

```
public static void sendTxRequest64(Network network,Mote mote,int[] payload)
			throws NamingException, RemoteException{
		log.debug("Solicitud de envío de trama al mote: " + mote.getId());
		log.debug("Servicio asociado: " + network.getService());
		String url = BASE_URL + network.getService();
		log.debug("URL: " + url);
		Context context = new InitialContext();
		log.debug("Iniciando el envío de la trama...");
		XBeeStandardService remoteService = (XbeeStandardService)context.lookup(url);
```

```
            remoteService.
             sendTxRequest64(XbeeUtil.getXBeeAddress64Formatted(mote.getSerialNumber()),
            payload, Option.DEFAULT_OPTION.getValue(), 0xE0);
            log.debug("Trama enviada");
    }
```

3.4. REGLAS

Desde el punto de vista de la aplicación se ha creado la clase que a continuación se presenta, la cual encapsula la comprobación de las medidas recibidas.

Inicialmente recibe como parámetro un SensorMeasureResponse (ver 7.3.2.2) que contiene las medidas que se han recibido y recupera las alarmas que se hayan configurado.

Posteriormente crea la base de conocimiento, KnowledgeBuilder, a partir de un archivo de reglas cuya estructura se analiza posteriormente.

Finalmente, se crea una instancia del motor de reglas, StatelessKnowledgeSession a la que se le pasan las medidas con sus correspondientes alarmas, así como otros parámetros de interés para el procesado y postprocesado.

ALARMCHECK.JAVA

```
public class AlarmCheck {

    private final Logger log = Logger.getLogger(this.getClass());
    private final SensorMeasureResponse sensorMeasureResponse;
    private List<Alarm> alarmEvents;
    private KnowledgeBase kbase;

    public AlarmCheck(SensorMeasureResponse sensorMeasureResponse) {
            this.sensorMeasureResponse = sensorMeasureResponse;
            this.alarmEvents = this.prepareMeasureEvents();
            KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
            kbuilder.add(ResourceFactory.newFileResource("/home/francisco/myrules.drl"),
                            ResourceType.DRL);
            if (kbuilder.hasErrors()) {
                    log.error(kbuilder.getErrors().toString());
            } else {
                    kbase = KnowledgeBaseFactory.newKnowledgeBase();
                    kbase.addKnowledgePackages(kbuilder.getKnowledgePackages());
            }
    }

    public void checkAlarms() {
            if (!alarmEvents.isEmpty()) {
                    StatelessKnowledgeSession ksession = kbase.newStatelessKnowledgeSession();
                    //Añade elementos globales al motor de reglas
                    ksession.setGlobal("log", log);

    ksession.setGlobal("currentSeason",Seasons.parse(newDate(System.currentTimeMillis())));

    ksession.setGlobal("currentDaytime",DayTimes.parse(newDate(System.currentTimeMillis())));
                    //Ejecuta
```

```
                ksession.execute(this.alarmEvents);
                //Persiste las alarmas que se han disparado >> Más eficiente hacerlo aquí que en las reglas
                new DbAlarmEventDAO().save(alarmEvents);
                //Persiste los eventos para mostrarlos en el welcome
                new DbEventDAO().save(alarmEvents);
            }else{
                log.debug("No hay alarmas configuradas");
            }
        }

        private List<Alarm> prepareMeasureEvents() {
            List<Sensor>  sensors = new
        DbSensorDAO().loadByMoteAddress(sensorMeasureResponse.getMote().getSerialNumber());
            Map<Integer, Sensor> sensorsMap = new HashMap<Integer, Sensor>(sensors.size());
            for (Sensor sensor : sensors) {
                sensorsMap.put(sensor.getId(), sensor);
            }
            List<Alarm> alarmEvents = new ArrayList<Alarm>();
            DbAlarmDAO alarmDAO = new DbAlarmDAO();
            for (Measure measure : sensorMeasureResponse.getMeasures()) {
                List<Alarm> measureAlarms = alarmDAO.load(measure.getMote(), measure.getSensor());
                for (Alarm alarm : measureAlarms) {
                    alarm.setMeasure(measure);
                    alarm.setSensor(sensorsMap.get(alarm.getSensor().getId()));
                }
                alarmEvents.addAll(measureAlarms);
            }
            log.debug(alarmEvents);
            return alarmEvents;
        }
}
```

La creación y mantenimiento del archivo de reglas que emplea el motor de Drools, no se permite a los usuarios de la aplicación, pues el algo complejo que requiere depuración antes de su puesta en producción.

Los archivos de reglas se generan usando la nomenglatura MVEL, que es un código de scripting muy similar a Java.

La estructura general de una regla en MVEL es la siguiente:

```
rule "nombre"
  atributos
  when
    Parte izquierda o condición
  then
    Parte derecha o conclusión
end
```

Dentro del archivo se pueden declarar variables globales, que están disponibles para todas las reglas contenidas así como realizar importaciones de paquetes de clases que usarán las reglas.

Por lo general, las variables globales se usan para contener las instancias de los parámetros que se pasan al motor de inferencia.

Dentro de una regla, para hacer referencia a un hecho, se denota usando el símbolo $ y las condiciones se establecen en forma de plantilla, sobre la que se aplican los valores adecuados.

En el siguiente ejemplo, se muestran las reglas para un sensor de temperatura, donde tenemos la posibilidad de crear tres formatos de reglas:

- La temperatura es mayor que una dada
- La temperatura es igual a una dada
- La temperatura es menor que una dada

A modo de ejemplo, cada vez que una regla se dispara, se crea un evento de sistema con la información del hecho que posteriormente se persistirá en la base de datos, pero podría por ejemplo, enviar un mail o un sms si la infraestructura lo permitiera.

RULES.DRL

```
package es.automation.home.rule

import es.automation.home.domain.Alarm
import es.automation.home.domain.ContinuousAlarm
import es.automation.home.domain.MeasureType
import es.automation.home.domain.SensorType
import es.automation.home.domain.DayTimes
import es.automation.home.domain.Seasons
import es.automation.home.domain.EventType
import es.automation.home.domain.EventRelevance
import es.automation.home.domain.AlarmRule
import es.automation.home.domain.Event
import java.util.ArrayList

global org.apache.log4j.Logger log
global es.automation.home.domain.Seasons currentSeason
global es.automation.home.domain.DayTimes currentDaytime


rule "check upper temperature"
 dialect "mvel"
 when
        $alarmEvent : ContinuousAlarm(measure.measureType == MeasureType.CONTINUOUS,
                                      sensor.type == SensorType.TEMPERATURE,
                                      (dayTime == currentDaytime || dayTime == DayTimes.ALLTIME),
                                      (season == currentSeason || season == Seasons.ALLTIME),
                                      (rule == AlarmRule.UPPER && measure.measure > alarmValue),
                                                                  fired == false)
 then
        $alarmEvent.setFired(true);
        Event event = new Event();
```

```
        event.setDescription("Alarma de temperatura: Límite superior de " +
                        $alarmEvent.alarmValue/100 + $alarmEvent.sensor.type.units + " superado");
        event.setNetworkId($alarmEvent.mote.network.id);
        event.setType(EventType.OVER_TEMPERATURE);
        event.setRelevance(EventRelevance.CRITICAL);
        log.debug("Añadido nuevo evento");
        $alarmEvent.setEvent(event);
        update($alarmEvent);
        log.info("Alarma de temperatura!! Límite superior excedido");
end

 rule "check lower temperature"
 dialect "mvel"
 when
        $alarmEvent : ContinuousAlarm(measure.measureType == MeasureType.CONTINUOUS,
                                sensor.type == SensorType.TEMPERATURE,
                                dayTime == currentDaytime || dayTime == DayTimes.ALLTIME,
                                season == currentSeason || season == Seasons.ALLTIME,
                                rule == AlarmRule.LOWER && measure < alarmValue,
                                                fired == false)
 then
        $alarmEvent.setFired(true);
        Event event = new Event();
        event.setDescription("Alarma de temperatura: Límite inferior de " +
                        $alarmEvent.alarmValue/100 + $alarmEvent.sensor.type.units + " superado");
        event.setNetworkId($alarmEvent.mote.network.id);
        event.setType(EventType.UNDER_TEMPERATURE);
        event.setRelevance(EventRelevance.CRITICAL);
        log.debug("Añadido nuevo evento");
        $alarmEvent.setEvent(event);
        $alarmEvent.setEvent(event);
        update($alarmEvent);
        log.info("Alarma de temperatura!! Límite inferior excedido");
end


rule "check equal temperature"
 dialect "mvel"
 when
        $alarmEvent : ContinuousAlarm(measure.measureType == MeasureType.CONTINUOUS,
                                sensor.type == SensorType.TEMPERATURE,
                                dayTime == currentDaytime || dayTime == DayTimes.ALLTIME,
                                season == currentSeason || season == Seasons.ALLTIME,
                                rule == AlarmRule.EQUAL && measure == alarmValue,
                                                fired == false)
 then
        $alarmEvent.setFired(true);
        Event event = new Event();
        event.setDescription("Alarma de temperatura: Temperatura " +
        $alarmEvent.alarmValue/100 + $alarmEvent.sensor.type.units + " alcanzada");
        event.setNetworkId($alarmEvent.mote.network.id);
        event.setType(EventType.EQUAL_TEMPERATURE);
        event.setRelevance(EventRelevance.CRITICAL);
        $alarmEvent.setEvent(event);
        log.debug("Añadido nuevo evento");
        $alarmEvent.setEvent(event);
        update($alarmEvent);
```

```
        log.info("Alarma de temperatura!! Temperatura alcanzada");
end



rule "alarm event sensed"
        dialect "mvel"
when
        $alertEvent : Alarm(fired == true)
then
        log.info("Evento de alarma sensibilizado");
end
```

4. DATOS DE COMPONENTES

4.1. ATMEL ATMEGA 168

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1K Bytes EEPROM
  - 512/1K/1K/2K Bytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 4 MHz@1.8 - 5.5V, 0 - 10 MHz@2.7 - 5.5.V, 0 - 20 MHz @ 4.5 - 5.5V
- Power Consumption at 1 MHz, 1.8V, 25°C
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

ATMEL

8-bit AVR® Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash

ATmega48A
ATmega48PA
ATmega88A
ATmega88PA
ATmega168A
ATmega168PA
ATmega328
ATmega328P

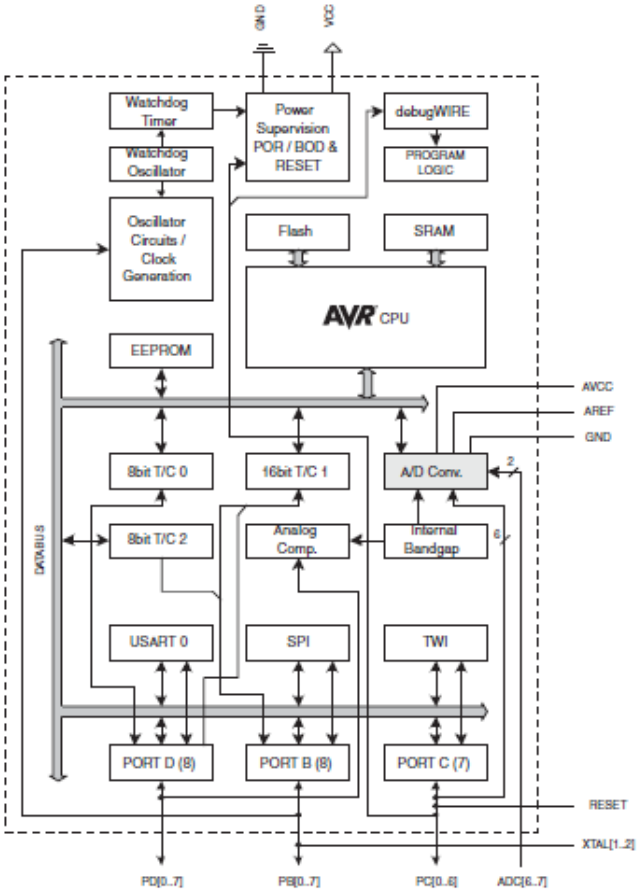
Rev. 8271C–AVR–08/10

2. Overview

The ATmega48A/48PA/88A/88PA/168A/168PA/328/328P is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega48A/48PA/88A/88PA/168A/168PA/328/328P achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

2.1 Block Diagram

Figure 2-1. Block Diagram



29.3 ATmega168PA Typical Characteristics

29.3.1 Active Supply Current

Figure 29-93. ATmega168PA: Active Supply Current vs. Low Frequency (0.1-1.0 MHz)

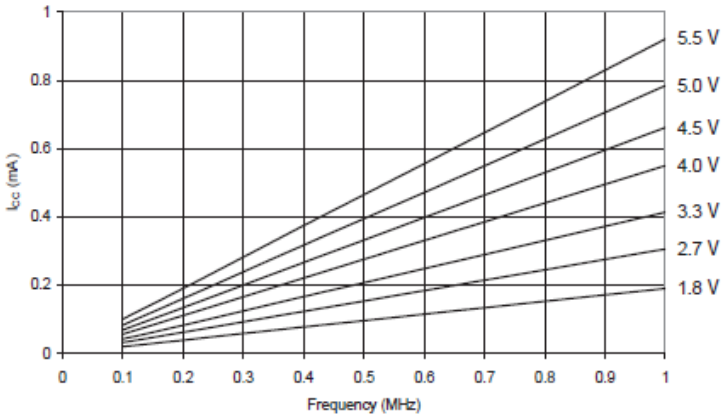
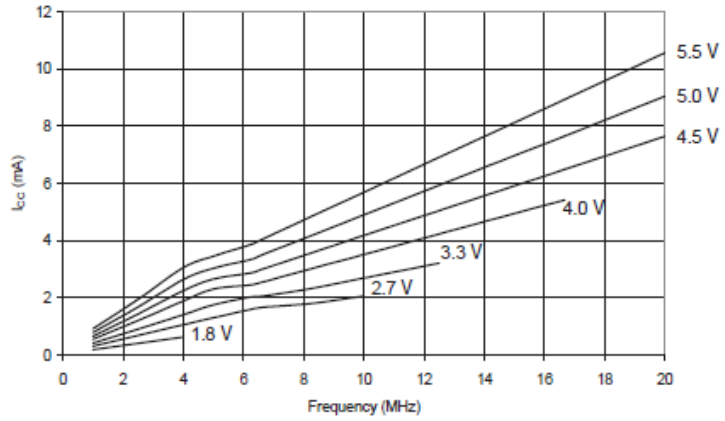


Figure 29-94. ATmega168PA: Active Supply Current vs. Frequency (1-20 MHz)



29.3.3 ATmega168PA Supply Current of IO Modules

The tables and formulas below can be used to calculate the additional current consumption for the different I/O modules in Active and Idle mode. The enabling or disabling of the I/O modules are controlled by the Power Reduction Register. See "Power Reduction Register" on page 42 for details.

Table 29-5. ATmega168PA: Additional Current Consumption for the different I/O modules (absolute values)

PRR bit	Typical numbers		
	V_{cc} = 2V, F = 1 MHz	V_{cc} = 3V, F = 4 MHz	V_{cc} = 5V, F = 8 MHz
PRUSART0	2.86 uA	20.3 uA	52.2 uA
PRTWI	6.00 uA	44.1uA	122.0 uA
PRTIM2	4.97 uA	33.2 uA	79.8 uA
PRTIM1	3.50 uA	23.0 uA	55.3 uA
PRTIM0	1.43 uA	9.2 uA	21.4 uA
PRSPI	5.01 uA	38.6 uA	111.4 uA
PRADC	6.34 uA	45.7 uA	123.6 uA

Table 29-6. ATmega168PA: Additional Current Consumption (percentage) in Active and Idle mode

PRR bit	Additional Current consumption compared to Active with external clock (see Figure 29-93 on page 375 and Figure 29-94 on page 375)	Additional Current consumption compared to Idle with external clock (see Figure 29-98 on page 377 and Figure 29-99 on page 378)
PRUSART0	1.5%	8.9%
PRTWI	3.2%	19.5%
PRTIM2	2.4%	14.8%
PRTIM1	1.7%	10.3%
PRTIM0	0.7%	4.1%
PRSPI	2.9%	17.1%
PRADC	3.4%	20.3%

23. Analog-to-Digital Converter

23.1 Features

- 10-bit Resolution
- 0.5 LSB Integral Non-linearity
- ± 2 LSB Absolute Accuracy
- 13 - 260 µs Conversion Time
- Up to 76.9 kSPS (Up to 15 kSPS at Maximum Resolution)
- 6 Multiplexed Single Ended Input Channels
- 2 Additional Multiplexed Single Ended Input Channels (TQFP and QFN/MLF Package only)
- Temperature Sensor Input Channel
- Optional Left Adjustment for ADC Result Readout
- 0 - V_{CC} ADC Input Voltage Range
- Selectable 1.1V ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler

23.2 Overview

The ATmega48PA/88PA/168PA/328P features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows eight single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND).

The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion. A block diagram of the ADC is shown in Figure 23-1 on page 251.

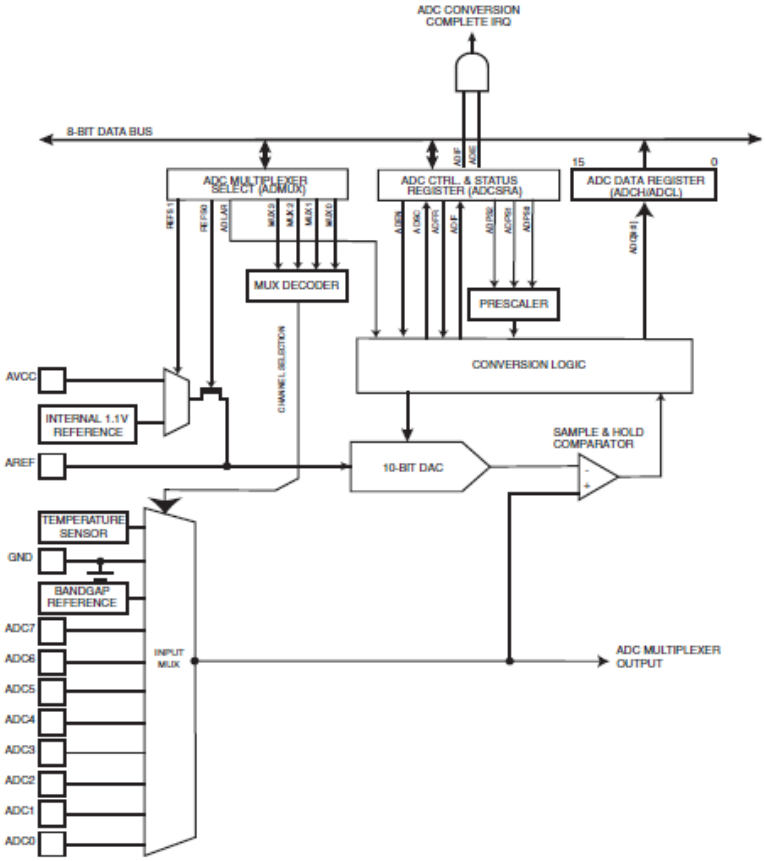
The ADC has a separate analog supply voltage pin, AV_{CC}. AV_{CC} must not differ more than ±0.3V from V_{CC}. See the paragraph "ADC Noise Canceler" on page 256 on how to connect this pin.

Internal reference voltages of nominally 1.1V or AV_{CC} are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.

The Power Reduction ADC bit, PRADC, in "Minimizing Power Consumption" on page 42 must be disabled by writing a logical zero to enable the ADC.

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AV_{CC} or an internal 1.1V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

Figure 23-1. Analog to Digital Converter Block Schematic Operation,



19. USART0

19.1 Features

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Master or Slave Clocked Synchronous Operation
- High Resolution Baud Rate Generator
- Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data OverRun Detection
- Framing Error Detection
- Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- Three Separate Interrupts on TX Complete, TX Data Register Empty and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode

19.2 Overview

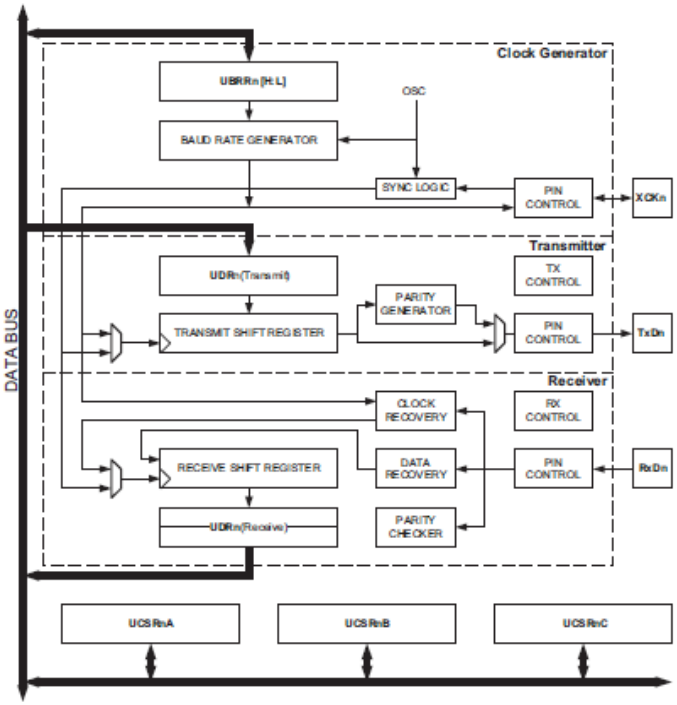
The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device.

The USART0 can also be used in Master SPI mode, see “USART in SPI Mode” on page 204. The Power Reduction USART bit, PRUSART0, in "Minimizing Power Consumption" on page 42 must be disabled by writing a logical zero to it.

A simplified block diagram of the USART Transmitter is shown in Figure 19-1 on page 177. CPU accessible I/O Registers and I/O pins are shown in bold.

The dashed boxes in the block diagram separate the three main parts of the USART (listed from the top): Clock Generator, Transmitter and Receiver. Control Registers are shared by all units. The Clock Generation logic consists of synchronization logic for external clock input used by synchronous slave operation, and the baud rate generator. The XCKn (Transfer Clock) pin is only used by synchronous transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, Parity Generator and Control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The Receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recovery units, the Receiver includes a Parity Checker, Control logic, a Shift Register and a two level receive buffer (UDRn). The Receiver supports the same frame formats as the Transmitter, and can detect Frame Error, Data OverRun and Parity Errors.

Figure 19-1. USART Block Diagram[1]



ATmega48PA/88PA/168PA/328P

14. 8-bit Timer/Counter0 with PWM

14.1 Features

- Two Independent Output Compare Units
- Double Buffered Output Compare Registers
- Clear Timer on Compare Match (Auto Reload)
- Glitch Free, Phase Correct Pulse Width Modulator (PWM)
- Variable PWM Period
- Frequency Generator
- Three Independent Interrupt Sources (TOV0, OCF0A, and OCF0B)

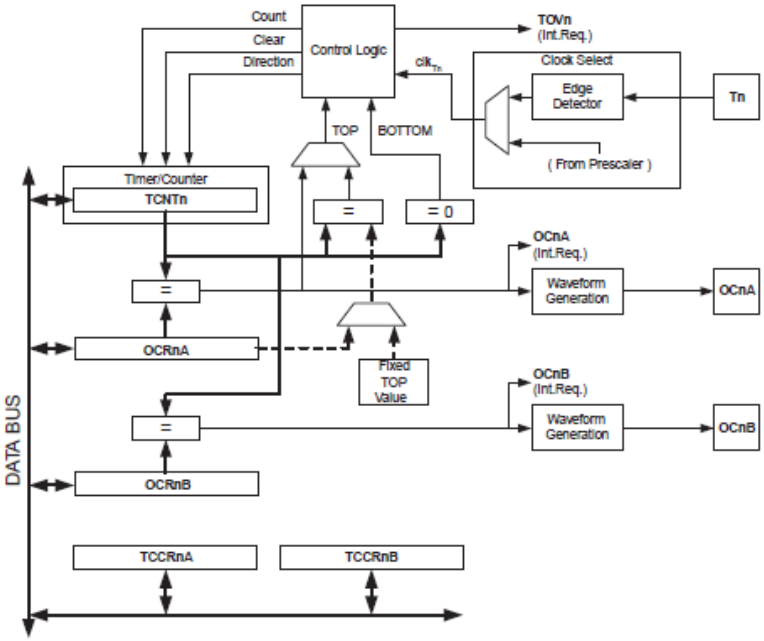
14.2 Overview

Timer/Counter0 is a general purpose 8-bit Timer/Counter module, with two independent Output Compare Units, and with PWM support. It allows accurate program execution timing (event management) and wave generation.

A simplified block diagram of the 8-bit Timer/Counter is shown in Figure 14-1. For the actual placement of I/O pins, refer to "Pinout ATmega48PA/88PA/168PA/328P" on page 2. CPU accessible I/O Registers, including I/O bits and I/O pins, are shown in bold. The device-specific I/O Register and bit locations are listed in the "Register Description" on page 106.

The PRTIM0 bit in "Minimizing Power Consumption" on page 42 must be written to zero to enable Timer/Counter0 module.

Figure 14-1. 8-bit Timer/Counter Block Diagram



9. Power Management and Sleep Modes

Sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application's requirements.

When enabled, the Brown-out Detector (BOD) actively monitors the power supply voltage during the sleep periods. To further save power, it is possible to disable the BOD in some sleep modes. See "BOD Disable" on page 40 for more details.

9.1 Sleep Modes

Figure 8-1 on page 26 presents the different clock systems in the ATmega48PA/88PA/168PA/328P, and their distribution. The figure is helpful in selecting an appropriate sleep mode. Table 9-1 shows the different sleep modes, their wake up sources BOD disable ability.

Table 9-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

	Active Clock Domains					Oscillators		Wake-up Sources							
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

4.2.XBEE

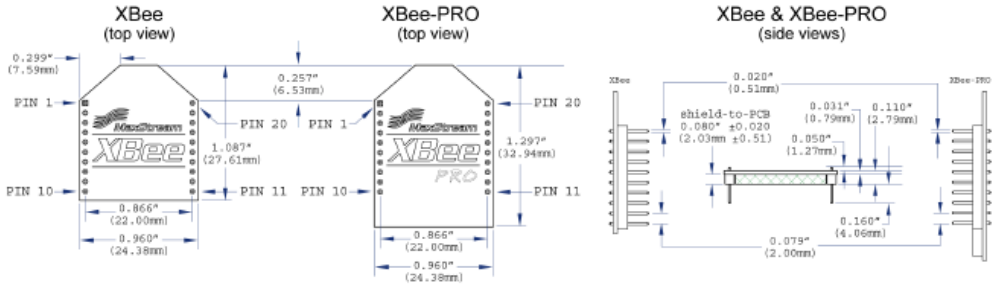
Specifications

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

Specification	XBee	XBee-PRO
Performance		
Indoor/Urban Range	Up to 100 ft (30 m)	Up to 300 ft. (90 m), up to 200 ft (60 m) International variant
Outdoor RF line-of-sight Range	Up to 300 ft (90 m)	Up to 1 mile (1600 m), up to 2500 ft (750 m) international variant
Transmit Power Output (software selectable)	1mW (0 dBm)	63mW (18dBm)* 10mW (10 dBm) for International variant
RF Data Rate	250,000 bps	250,000 bps
Serial Interface Data Rate (software selectable)	1200 bps - 250 kbps (non-standard baud rates also supported)	1200 bps - 250 kbps (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)	250mA (@3.3 V) (150mA for international variant) RPSMA module only: 340mA (@3.3 V) (180mA for international variant)
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 85° C (industrial)	-40 to 85° C (industrial)
Antenna Options	Integrated Whip, Chip or U.FL Connector, RPSMA Connector	Integrated Whip, Chip or U.FL Connector, RPSMA 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	R201WW07215214	R201WW08215111 (Max. 10 dBm transmit power output)*
Austraila	C-Tick	C-Tick

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / CONFIG	Input	UART Data In
4	DIO8*	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

The XBee and XBee-PRO RF Modules are pin-for-pin compatible.



API Frame Specifications

Two API modes are supported and both can be enabled using the AP (API Enable) command. Use the following AP parameter values to configure the module to operate in a particular mode:

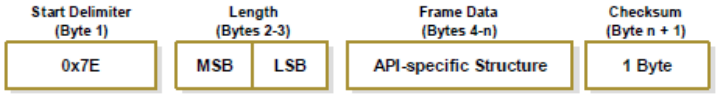
- AP = 0 (default): Transparent Operation (UART Serial line replacement) API modes are disabled.
- AP = 1: API Operation
- AP = 2: API Operation (with escaped characters)

Any data received prior to the start delimiter is silently discarded. If the frame is not received correctly or if the checksum fails, the data is silently discarded.

API Operation (AP parameter = 1)

When this API mode is enabled (AP = 1), the UART data frame structure is defined as follows:

Figure 3-01. UART Data Frame Structure:

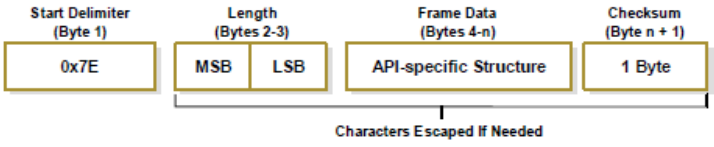


MSB = Most Significant Byte, LSB = Least Significant Byte

API Operation - with Escape Characters (AP parameter = 2)

When this API mode is enabled (AP = 2), the UART data frame structure is defined as follows:

Figure 3-02. UART Data Frame Structure - with escape control characters:



MSB = Most Significant Byte, LSB = Least Significant Byte

Escape characters. When sending or receiving a UART data frame, specific data values must be escaped (flagged) so they do not interfere with the UART or UART data frame operation. To escape an interfering data byte, insert 0x7D and follow it with the byte to be escaped XOR'd with 0x20.

4.3.FTDI FT232RL

The FT232R is a USB to serial UART interface with the following advanced features:

- Single chip USB to asynchronous serial data transfer interface.
- Entire USB protocol handled on the chip. No USB specific firmware programming required.
- Fully integrated 1024 bit EEPROM storing device descriptors and CBUS I/O configuration.
- Fully integrated USB termination resistors.
- Fully integrated clock generation with no external crystal required plus optional clock output selection enabling a glue-less interface to external MCU or FPGA.
- Data transfer rates from 300 baud to 3 Mbaud (RS422, RS485, RS232) at TTL levels.
- 256 byte receive buffer and 128 byte transmit buffer utilising buffer smoothing technology to allow for high data throughput.
- FTDI's royalty-free Virtual Com Port (VCP) and Direct (D2XX) drivers eliminate the requirement for USB driver development in most cases.
- Unique USB FTDIChip-ID™ feature.
- Configurable CBUS I/O pins.
- Transmit and receive LED drive signals.
- FIFO receive and transmit buffers for high data throughput.
- Synchronous and asynchronous bit bang interface options with RD# and WR# strobes.
- Device supplied pre-programmed with unique USB serial number.
- Supports bus powered, self powered and high-power bus powered USB configurations.
- Integrated +3.3V level converter for USB I/O.
- Integrated level converter on UART and CBUS for interfacing to between +1.8V and +5V logic.
- True 5V/3.3V/2.8V/1.8V CMOS drive output and TTL input.
- Configurable I/O pin output drive strength.
- Integrated power-on-reset circuit.
- Fully integrated AVCC supply filtering - no external filtering required.
- UART signal inversion option.
- +3.3V to +5.25V Single Supply Operation.
- Low operating and USB suspend current.
- Low USB bandwidth consumption.
- UHCI/OHCI/EHCI host controller compatible.
- USB 2.0 Full Speed compatible.
- -40°C to 85°C extended operating temperature range.
- Available in compact Pb-free 28 Pin SSOP and QFN-32 packages (both RoHS compliant).

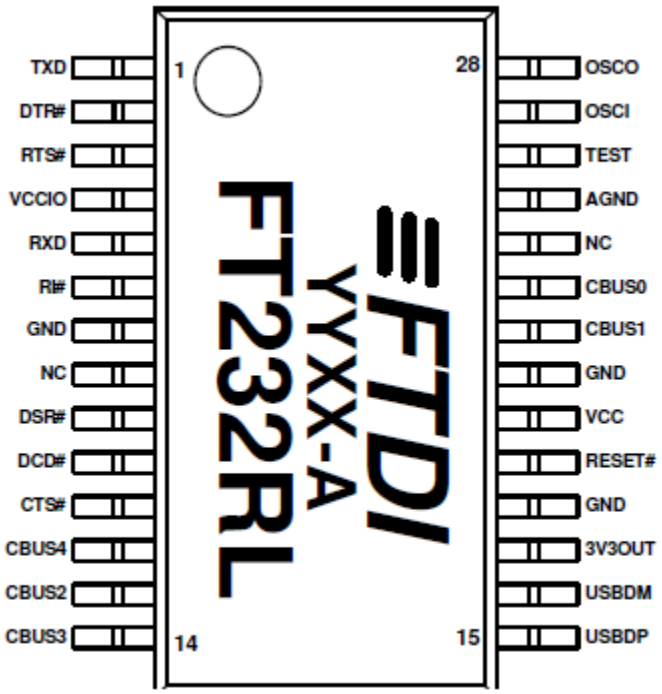
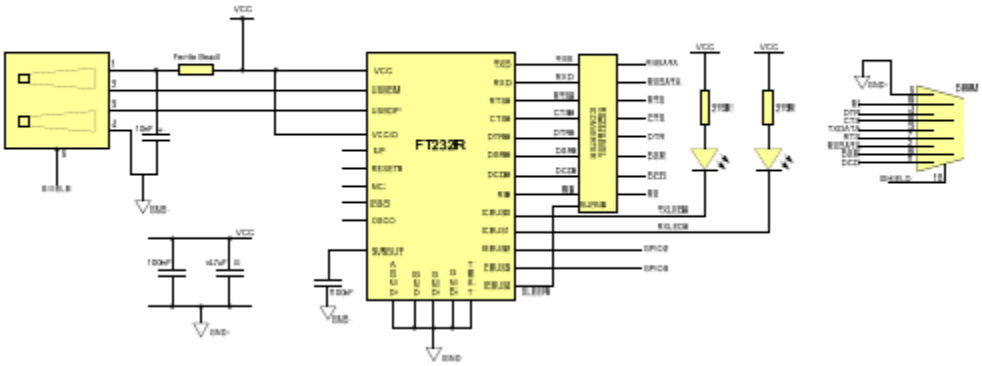


Figure 3.1 SSOP Package Pin Out and Schematic Symbol

7.1 USB to RS232 Converter



7.5 LED Interface

Any of the CBUS I/O pins can be configured to drive an LED. The FT232R has 3 configuration options for driving LEDs from the CBUS. These are TXLED#, RXLED#, and TX&RXLED#. Refer to Section 3.5 for configuration options.

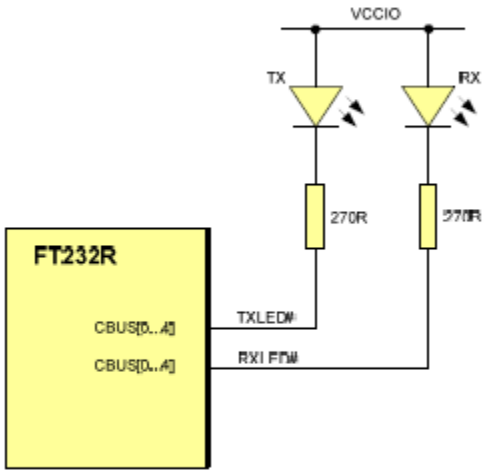


Figure 7.5 Dual LED Configuration

An example of using the FT232R to drive LEDs is shown in Figure 7.5. In this application one of the CBUS pins is used to indicate transmission of data (TXLED#) and another is used to indicate receiving data (RXLED#). When data is being transmitted or received the respective pins will drive from tri-state to low in order to provide indication on the LEDs of data transfer. A digital one-shot is used so that even a small percentage of data transfer is visible to the end user.

4.4.TPS79333

TEXAS INSTRUMENTS
www.ti.com

DBV 5 DBV 6 YEQ, YZQ

TPS793xx

SLVS348K–JULY 2001–REVISED OCTOBER 2007

ULTRALOW-NOISE, HIGH PSRR, FAST RF 200mA LOW-DROPOUT LINEAR REGULATORS IN NanoStar™ WAFER CHIP SCALE AND SOT23

FEATURES

- 200mA RF Low-Dropout Regulator With Enable
- Available in Fixed Voltage Versions from 1.8V to 4.75V and Adjustable (1.22V to 5.5V)
- High PSRR (70dB at 10kHz)
- Ultralow-Noise (32μV_{RMS}, TPS79328)
- Fast Start-Up Time (50µs)
- Stable With a 2.2µF Ceramic Capacitor
- Excellent Load/Line Transient Response
- Very Low Dropout Voltage (112mV at 200mA, TPS79330)
- 5- and 6-Pin SOT23 (DBV) and NanoStar Wafer Chip Scale (YEQ, YZQ) Packages

APPLICATIONS

- RF: VCOs, Receivers, ADCs
- Audio
- Cellular and Cordless Telephones
- Bluetooth®, Wireless LAN
- Handheld Organizers, PDAs

DESCRIPTION

The TPS793xx family of low-dropout (LDO) low-power linear voltage regulators features high power-supply rejection ratio (PSRR), ultralow-noise, fast start-up, and excellent line and load transient responses in NanoStar wafer chip scale and SOT23 packages. NanoStar packaging gives an ultrasmall footprint as well as an ultralow profile and package weight, making it ideal for portable applications such as handsets and PDAs. Each device in the family is stable, with a small 2.2µF ceramic capacitor on the output. The TPS793xx family uses an advanced, proprietary BiCMOS fabrication process to yield extremely low dropout voltages (for example, 112mV at 200mA, TPS79330). Each device achieves fast start-up times (approximately 50µs with a 0.001µF bypass capacitor) while consuming very low quiescent current (170µA typical). Moreover, when the device is placed in standby mode, the supply current is reduced to less than 1µA. The TPS79328 exhibits approximately 32μV_{RMS} of output voltage noise at 2.8V output with a 0.1µF bypass capacitor. Applications with analog components that are noise-sensitive, such as portable RF electronics, benefit from the high PSRR and low-noise features as well as the fast response time.

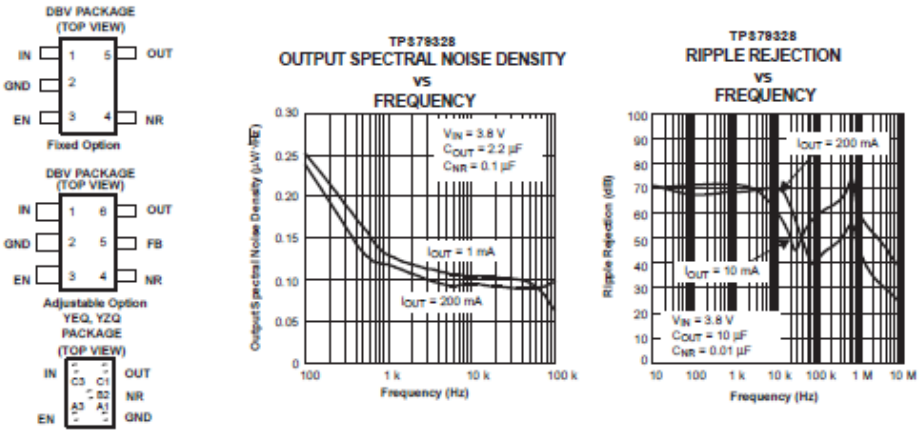


Figure 1.

ELECTRICAL CHARACTERISTICS

Over recommended operating temperature range T_J = –40°C to +125°C, $V_{EN} = V_{IN}$, $V_{IN} = V_{OUT(nom)} + 1V$[1], I_{OUT} = 1mA, C_{OUT} = 10µF, C_{NR} = 0.01µF (unless otherwise noted). Typical values are at +25°C.

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
V_{IN} Input voltage[1]				2.7		5.5	V
I_{OUT} Continuous output current				0		200	mA
V_{FB} Internal reference (TPS79301)				1.201	1.225	1.250	V
Output voltage range (TPS79301)				V_{FB}		5.5 – V_{DO}	V
Output voltage	TPS79318	0µA < I_{OUT} < 200mA,	2.8V < V_{IN} < 5.5V	1.764	1.8	1.836	V
	TPS79325	0µA < I_{OUT} < 200mA,	3.5V < V_{IN} < 5.5V	2.45	2.5	2.55	V
	TPS79328	0µA < I_{OUT} < 200mA,	3.8V < V_{IN} < 5.5V	2.744	2.8	2.856	V
	TPS793285	0µA < I_{OUT} < 200mA,	3.85V < V_{IN} < 5.5V	2.793	2.85	2.907	V
	TPS79330	0µA < I_{OUT} < 200mA,	4V < V_{IN} < 5.5V	2.94	3	3.06	V
	TPS79333	0µA ≤ I_{OUT} < 200mA,	4.3V < V_{IN} < 5.5V	3.234	3.3	3.366	V
	TPS793475	0µA < I_{OUT} < 200mA,	5.25V < V_{IN} < 5.5V	4.655	4.75	4.845	V
Line regulation (ΔV_{OUT}%/ΔV_{IN})[1]		V_{OUT} + 1V < V_{IN} ≤ 5.5V			0.05	0.12	%/V
Load regulation (ΔV_{OUT}%/ΔI_{OUT})		0µA < I_{OUT} < 200mA,	T_J = +25°C		5		mV
Dropout voltage[2] ($V_{IN} = V_{OUT(nom)} – 0.1V$)	TPS79328	I_{OUT} = 200mA			120	200	mV
	TPS793285	I_{OUT} = 200mA			120	200	
	TPS79330	I_{OUT} = 200mA			112	200	
	TPS79333	I_{OUT} = 200mA			102	180	
	TPS793475	I_{OUT} = 200mA			77	125	
Output current limit		V_{OUT} = 0V		285		600	mA
GND pin current		0µA < I_{OUT} < 200mA			170	220	µA
Shutdown current[3]		V_{EN} = 0V, 2.7V < V_{IN} < 5.5V			0.07	1	µA
FB pin current		V_{FB} = 1.8V				1	µA
Power-supply ripple rejection	TPS79328	f = 100Hz, T_J = +25°C,	I_{OUT} = 10mA		70		dB
		f = 100Hz, T_J = +25°C,	I_{OUT} = 200mA		68		
		f = 10kHz, T_J = +25°C,	I_{OUT} = 200mA		70		
		f = 100kHz, T_J = +25°C,	I_{OUT} = 200mA		43		
Output noise voltage (TPS79328)		BW = 200Hz to 100kHz, I_{OUT} = 200mA	C_{NR} = 0.001µF		55		μV_{RMS}
			C_{NR} = 0.0047µF		36		
			C_{NR} = 0.01µF		33		
			C_{NR} = 0.1µF		32		
Time, start-up (TPS79328)		R_L = 14Ω, C_{OUT} = 1µF	C_{NR} = 0.001µF		50		µs
			C_{NR} = 0.0047µF		70		
			C_{NR} = 0.01µF		100		
High level enable input voltage		2.7V < V_{IN} < 5.5V		1.7		V_{IN}	V
Low level enable input voltage		2.7V < V_{IN} < 5.5V		0		0.7	V
EN pin current		V_{EN} = 0V		–1		1	µA
UVLO threshold		V_{CC} rising		2.25		2.65	V
UVLO hysteresis					100		mV

(1) Minimum V_{IN} is 2.7V or V_{OUT} + V_{DO}, whichever is greater.
(2) Dropout is not measured for the TPS79318 and TPS79325 since minimum V_{IN} = 2.7V.
(3) For adjustable versions, this parameter applies only after V_{IN} is applied; then V_{EN} transitions high to low.

APPLICATION INFORMATION

The TPS793xx family of low-dropout (LDO) regulators has been optimized for use in noise-sensitive battery-operated equipment. The device features extremely low dropout voltages, high PSRR, ultralow output noise, low quiescent current (170µA typically), and enable-input to reduce supply currents to less than 1µA when the regulator is turned off.

A typical application circuit is shown in Figure 22.

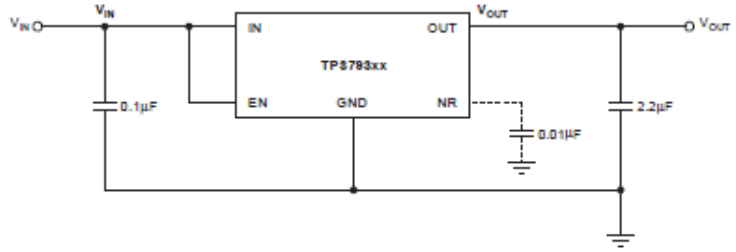


Figure 22. Typical Application Circuit

External Capacitor Requirements

A 0.1µF or larger ceramic input bypass capacitor, connected between IN and GND and located close to the TPS793xx, is required for stability and improves transient response, noise rejection, and ripple rejection. A higher-value input capacitor may be necessary if large, fast-rise-time load transients are anticipated or the device is located several inches from the power source.

Like most low-dropout regulators, the TPS793xx requires an output capacitor connected between OUT and GND to stabilize the internal control loop. The minimum recommended capacitance is 2.2µF. Any 2.2µF or larger ceramic capacitor is suitable, provided the capacitance does not vary significantly over temperature. If load current is not expected to exceed 100mA, a 1.0µF ceramic capacitor can be used.

The internal voltage reference is a key source of noise in an LDO regulator. The TPS793xx has an NR pin which is connected to the voltage reference through a 250kΩ internal resistor. The 250kΩ internal resistor, in conjunction with an external bypass capacitor connected to the NR pin, creates a low-pass filter to reduce the voltage reference noise and, therefore, the noise at the regulator output. In order for the regulator to operate properly, the current flow out of the NR pin must be at a minimum, because any leakage current creates an IR drop across the internal resistor, thus creating an output error. Therefore, the bypass capacitor must have minimal leakage current. The bypass capacitor should be no more than 0.1µF to ensure that it is fully charged during the quickstart time provided by the internal switch shown in the Functional Block Diagrams.

As an example, the TPS79328 exhibits only $32\mu V_{RMS}$ of output voltage noise using a 0.1µF ceramic bypass capacitor and a 2.2µF ceramic output capacitor. Note that the output starts up slower as the bypass capacitance increases due to the RC time constant at the NR pin that is created by the internal 250kΩ resistor and external capacitor.

Board Layout Recommendation to Improve PSRR and Noise Performance

To improve ac measurements like PSRR, output noise, and transient response, it is recommended that the board be designed with separate ground planes for V_{IN} and V_{OUT}, with each ground plane connected only at the GND pin of the device. In addition, the ground connection for the bypass capacitor should connect directly to the GND pin of the device.

Power Dissipation and Junction Temperature

Specified regulator operation is assured to a junction temperature of +125°C; the maximum junction temperature should be restricted to +125°C under normal operating conditions. This restriction limits the power dissipation the regulator can handle in any given application. To ensure the junction temperature is within acceptable limits, calculate the maximum allowable dissipation, $P_{D(max)}$, and the actual dissipation, P_D, which must be less than or equal to $P_{D(max)}$.

The maximum power dissipation limit is determined using Equation 1:

$$P_{D(max)} = \frac{T_J max - T_A}{R_{\theta JA}} \quad (1)$$

Where:

- $T_J$max is the maximum allowable junction temperature.
- $R_{\theta JA}$ is the thermal resistance junction-to-ambient for the package (see the Dissipation Ratings Table).
- T_A is the ambient temperature.

The regulator dissipation is calculated using Equation 2:

$$P_D = (V_{IN} - V_{OUT}) \times I_{OUT} \quad (2)$$

Power dissipation resulting from quiescent current is negligible. Excessive power dissipation triggers the thermal protection circuit.

Programming the TPS79301 Adjustable LDO Regulator

The output voltage of the TPS79301 adjustable regulator is programmed using an external resistor divider as shown in Figure 23. The output voltage is calculated using Equation 3:

$$V_{OUT} = V_{REF} \times \left(1 + \frac{R_1}{R_2}\right) \quad (3)$$

Where:

- V_{REF} = 1.2246V typ (the internal reference voltage)

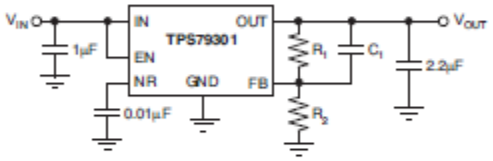
Resistors R_1 and R_2 should be chosen for approximately 50µA divider current. Lower value resistors can be used for improved noise performance, but the solution consumes more power. Higher resistor values should be avoided as leakage current into/out of FB across R_1/R_2 creates an offset voltage that artificially increases/decreases the feedback voltage and thus erroneously decreases/increases V_{OUT}. The recommended design procedure is to choose R_2 = 30.1kΩ to set the divider current at 50µA, C_1 = 15pF for stability, and then calculate R_1 using Equation 4:

$$R_1 = \left(\frac{V_{OUT}}{V_{REF}} - 1\right) \times R_2 \quad (4)$$

In order to improve the stability of the adjustable version, it is suggested that a small compensation capacitor be placed between OUT and FB. For voltages less than 1.8V, the value of this capacitor should be 100pF. For voltages greater than 1.8V, the approximate value of this capacitor can be calculated as shown in Equation 5:

$$C_1 = \frac{(3 \times 10^{-7}) \times (R_1 + R_2)}{(R_1 \times R_2)} \quad (5)$$

The suggested value of this capacitor for several resistor ratios is shown in the table below. If this capacitor is not used (such as in a unity-gain configuration) or if an output voltage less than 1.8V is chosen, then the minimum recommended output capacitor is 4.7µF instead of 2.2µF.



OUTPUT VOLTAGE PROGRAMMING GUIDE

OUTPUT VOLTAGE	R_1	R_2	C_1
1.22V	short	open	0pF
2.5V	31.6kΩ	30.1kΩ	22pF
3.3V	51kΩ	30.1kΩ	15pF
3.6V	59kΩ	30.1kΩ	15pF

Figure 23. TPS79301 Adjustable LDO Regulator Programming

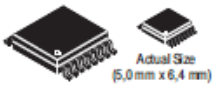
Regulator Protection

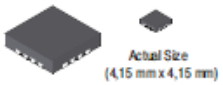
The TPS793xx PMOS-pass transistor has a built-in back diode that conducts reverse current when the input voltage drops below the output voltage (for example, during power-down). Current is conducted from the output to the input and is not internally limited. If extended reverse voltage operation is anticipated, external limiting might be appropriate.

The TPS793xx features internal current limiting and thermal protection. During normal operation, the TPS793xx limits output current to approximately 400mA. When current limiting engages, the output voltage scales back linearly until the overcurrent condition ends. While current limiting is designed to prevent gross device failure, care should be taken not to exceed the power dissipation ratings of the package or the absolute maximum voltage ratings of the device. If the temperature of the device exceeds approximately +165°C, thermal-protection circuitry shuts it down. Once the device has cooled down to below approximately +140°C, regulator operation resumes.

4.5.TPS61131

TEXAS INSTRUMENTS
www.ti.com





TPS61130
TPS61131
TPS61132

SLVS431B – JUNE 2002 – REVISED JANUARY 2008

SYNCHRONOUS SEPIC / FLYBACK CONVERTER WITH 1.1A SWITCH AND INTEGRATED LDO

FEATURES

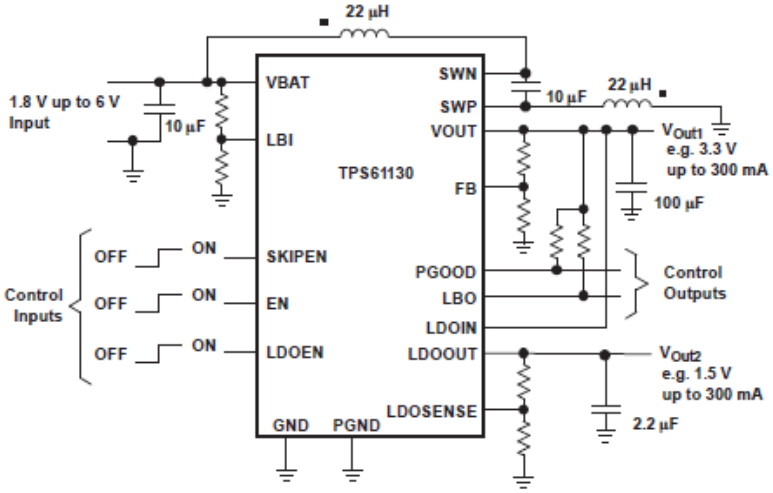
- Synchronous, Up To 90% Efficient, SEPIC Converter With 300-mA Output Current From 2.5-V Input
- Integrated 200-mA Reverse Voltage Protected LDO for DC/DC Output Voltage Post Regulation or Second Output Voltage
- 40-µA (Typical) Quiescent Current
- Input Voltage Range: 1.8-V to 5.5-V
- Fixed and Adjustable Output Voltage Options up to 5.5-V
- Power Save Mode for Improved Efficiency at Low Output Power
- Low Battery Comparator
- Power Good Output
- Low EMI-Converter (Integrated Antiringing Switch)
- Load Disconnect During Shutdown
- Overtemperature Protection
- Available in a Small 4mm x 4mm QFN-16 or in a TSSOP-16 Package

APPLICATIONS

- All Single Cell Li, Dual or Triple Cell Battery or USB Powered Products as MP-3 Player, PDAs, and Other Portable Equipment
- Dual Input or Dual Output Mode
- High Efficient Li-Ion to 3.3-V Conversion

DESCRIPTION

The TPS6113x devices provide a complete power supply solution for products powered by either a one-cell Li-Ion or Li-Polymer, or two- to four-cell Alkaline, NiCd, or NiMH batteries. The devices can generate two regulated output voltages. It provides a simple and efficient buck-boost solution for generating 3.3 V out of an input voltage that can be both higher and lower than the output voltage. The converter provides a maximum output current of at least 300 mA with supply voltages down to 1.8 V. The implemented SEPIC converter is based on a fixed frequency, pulse-width-modulation (PWM) controller using a synchronous rectifier to obtain maximum efficiency. The maximum peak current in the SEPIC switch is limited to a value of 1600 mA.



ELECTRICAL CHARACTERISTICS

over recommended free-air temperature range and over recommended input voltage range (typical at an ambient temperature range of 25°C) (unless otherwise noted)

DC/DC STAGE	PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_I	Input voltage range			1.8		6.5	V
V_O	Adjustable output voltage range (TPS61130)			2.5		5.5	V
V_{ref}	Reference voltage			485	500	515	mV
f	Oscillator frequency			400	500	600	kHz
I_{SW}	Switch current limit		VOUT= 3.3 V	1100	1300	1600	mA
	Startup current limit				0.4 x I_{SW}		mA
	SWN switch on resistance		VOUT= 3.3 V		200	350	mΩ
	SWP switch on resistance		VOUT= 3.3 V		250	500	mΩ
	Total accuracy (including line and load regulation)					±3	%
	DC/DC quiescent current	into VBAT	I_O= 0 mA, V_{EN} = VBAT = 1.8 V, VOUT = 3.3 V, ENLDO = 0 V		10	25	µA
		into VOUT	I_O = 0 mA, V_{EN} = VBAT = 1.8 V, VOUT = 3.3 V, ENLDO = 0 V		10	25	µA
	DC/DC shutdown current		V_{EN}= 0 V		0.2	1	µA
LDO STAGE	**PARAMETER**		**TEST CONDITIONS**	**MIN**	**TYP**	**MAX**	**UNIT**
$V_{I(LDO)}$	Input voltage range			1.8		7	V
$V_{O(LDO)}$	Adjustable output voltage range (TPS61130)			0.9		5.5	V
$I_{O(max)}$	Output current			200	320		mA
	LDO short circuit current limit					500	mA
	Minimum voltage drop		I_O =200 mA			300	mV
	Total accuracy (including line and load regulation)		I_O≥ 1 mA			±3%	
	Line regulation		LDOIN change from 1.8 V to 2.6 V at 100 mA, LDOOUT = 1.5 V			0.6%	
	Load regulation		Load change from 10% to 90%, LDOIN = 3.3 V			0.6%	
	LDO quiescent current		LDOIN = 7 V, VBAT = 1.8 V, EN = VBAT		20	30	µA
	LDO shutdown current		LDOEN = 0 V, LDOIN = 7 V		0.1	1	µA
CONTROL STAGE	**PARAMETER**		**TEST CONDITIONS**	**MIN**	**TYP**	**MAX**	**UNIT**
V_{IL}	LBI voltage threshold		V_{LBI} voltage decreasing	490	500	510	mV
	LBI input hysteresis				10		mV
	LBI input current		EN = VBAT or GND		0.01	0.1	µA
	LBO output low voltage		V_O = 3.3 V, I_{OI} = 100 µA		0.04	0.4	V
	LBO output low current				100		µA
	LBO output leakage current		V_{LBO} = 7 V		0.01	0.1	µA
V_{IL}	EN, SKIPEN input low voltage					0.2 × VBAT	V
V_{IH}	EN, SKIPEN input high voltage			0.8 × VBAT			V
V_{IL}	LDOEN input low voltage					0.2 × V_{LDOIN}	V
V_{IH}	LDOEN input high voltage			0.8 × V_{LDOIN}			V

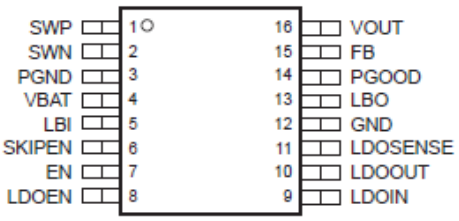
ELECTRICAL CHARACTERISTICS (continued)

over recommended free-air temperature range and over recommended input voltage range (typical at an ambient temperature range of 25°C) (unless otherwise noted)

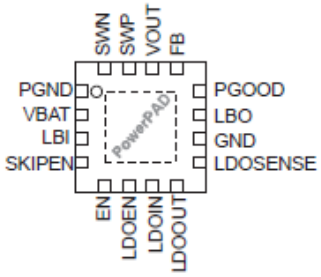
CONTROL STAGE (continued)					
PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
EN, SKIPEN input current	Clamped on GND or VBAT		0.01	0.1	µA
Power-Good threshold	V_O = 3.3 V	0.9 × V_o	0.92 × V_o	0.95 × V_o	V
Power-Good delay			30		µs
Power-Good output low voltage	V_O = 3.3 V, I_{OI} = 100µA		0.04	0.4	V
Power-Good output low current			100		µA
Power-Good output leakage current	V_{PG} = 7 V		0.01	0.1	µA
Over-Temperature protection			140		°C
Over-Temperature hysteresis			20		°C

PIN ASSIGNMENTS

PW PACKAGE
(TOP VIEW)



RSA PACKAGE
(TOP VIEW)



TPS61130
MAXIMUM SEPIC CONVERTER OUTPUT CURRENT
vs
INPUT VOLTAGE

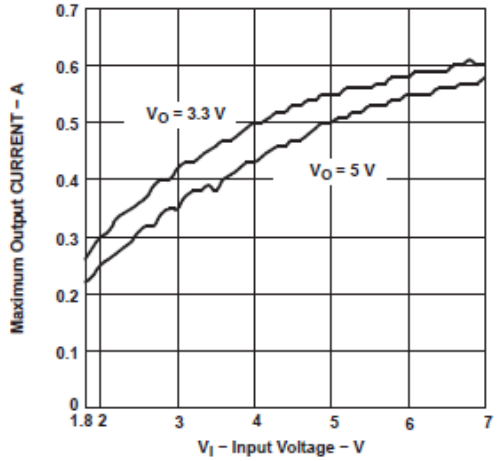


Figure 1.

TPS61130
MAXIMUM SEPIC CONVERTER OUTPUT CURRENT
vs
INPUT VOLTAGE

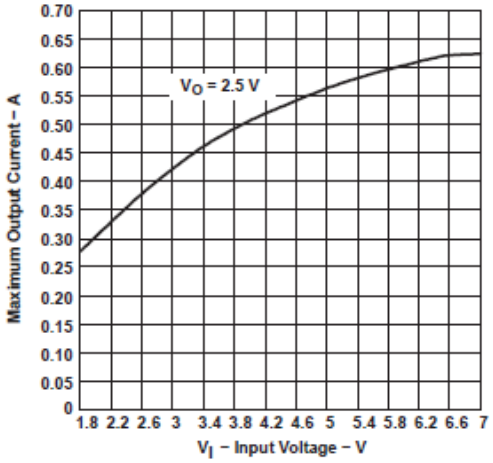


Figure 2.

TPS61130
SEPIC CONVERTER EFFICIENCY
vs
OUTPUT CURRENT

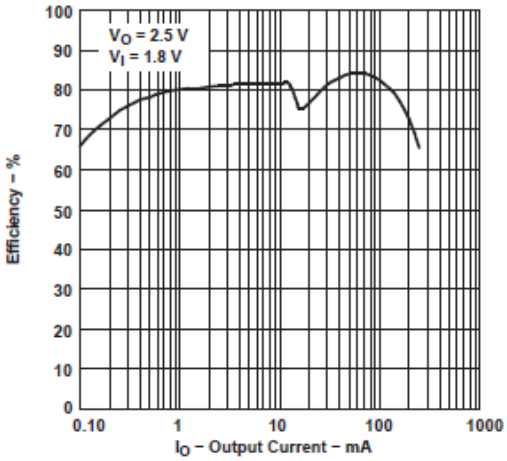


Figure 3.

TPS61132
SEPIC CONVERTER EFFICIENCY
vs
OUTPUT CURRENT

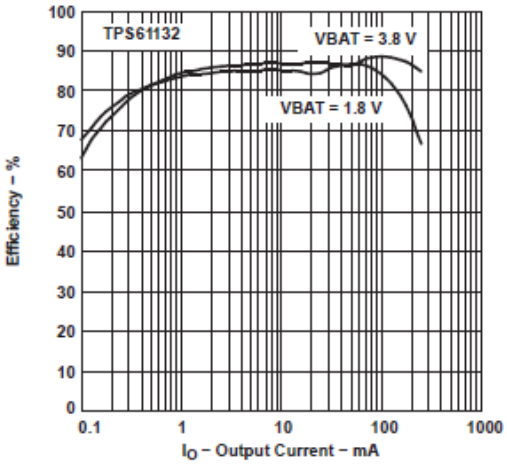


Figure 4.

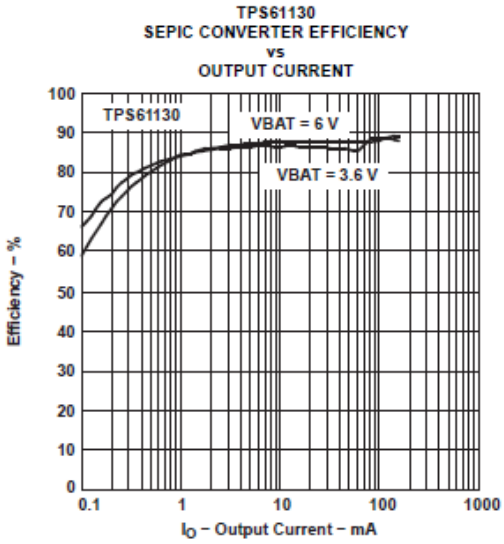


Figure 5.

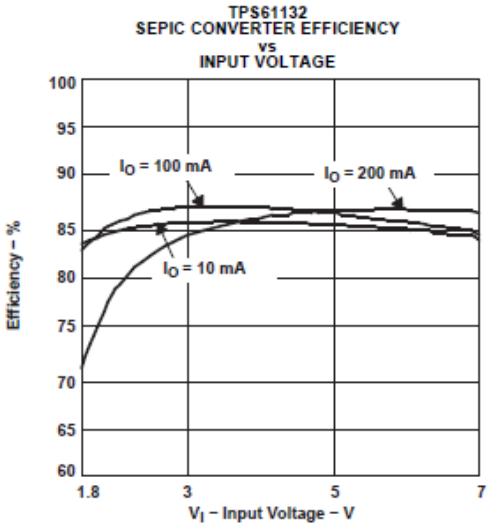


Figure 6.

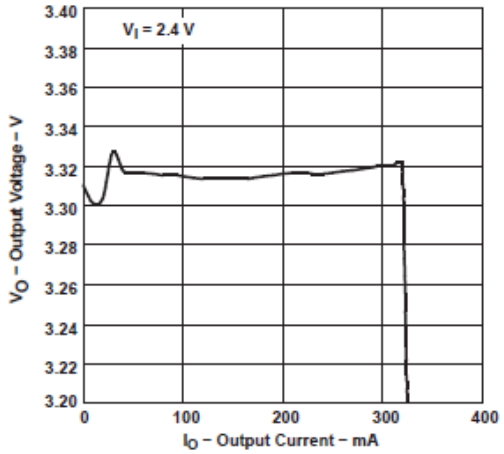


Figure 7.

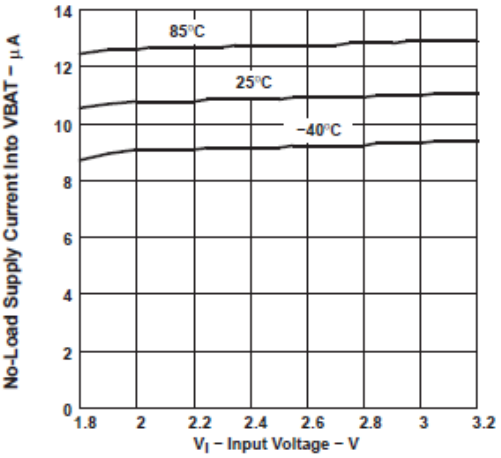


Figure 8.

APPLICATION INFORMATION

DESIGN PROCEDURE

The TPS6113x dc/dc converters are intended for systems powered by a dual up to 4 cell NiCd or NiMH battery with a typical terminal voltage between 1.8 V and 6.5 V. They can also be used in systems powered by one-cell Li-Ion with a typical stack voltage between 2.5 V and 4.2 V. Additionally, two up to four primary alkaline battery cells can be the power source in systems where the TPS6113x is used.

Programming the Output Voltage

DC/DC Converter

The output voltage of the TPS61130 dc/dc converter section can be adjusted with an external resistor divider. The typical value of the voltage on the FB pin is 500 mV. The maximum recommended value for the output voltage is 5.5 V. The current through the resistive divider should be about 100 times greater than the current into the FB pin. The typical current into the FB pin is 0.01 µA and the voltage across R6 is typically 500 mV. Based on those two values, the recommended value for R6 should be lower than 500 kΩ, in order to set the divider current at 1 µA or higher. Because of internal compensation circuitry the value for this resistor should be in the range of 200 kΩ. From that, the value of resistor R3, depending on the needed output voltage (V_O), can be calculated using Equation 1:

$$R3 = R6 \times \left(\frac{V_O}{V_{FB}} - 1\right) = 180\ k\Omega \times \left(\frac{V_O}{500\ mV} - 1\right) \quad (1)$$

If as an example, an output voltage of 3.3 V is needed, a 1-MΩ resistor should be chosen for R3. If for any reason the value for R6 is chosen significantly lower than 200 kΩ additional capacitance in parallel to R3 is recommended. The required capacitance value can be easily calculated using Equation 2.

$$C_{parR3} = 20\ pF \times \left(\frac{200\ k\Omega}{R6} - 1\right) \quad (2)$$

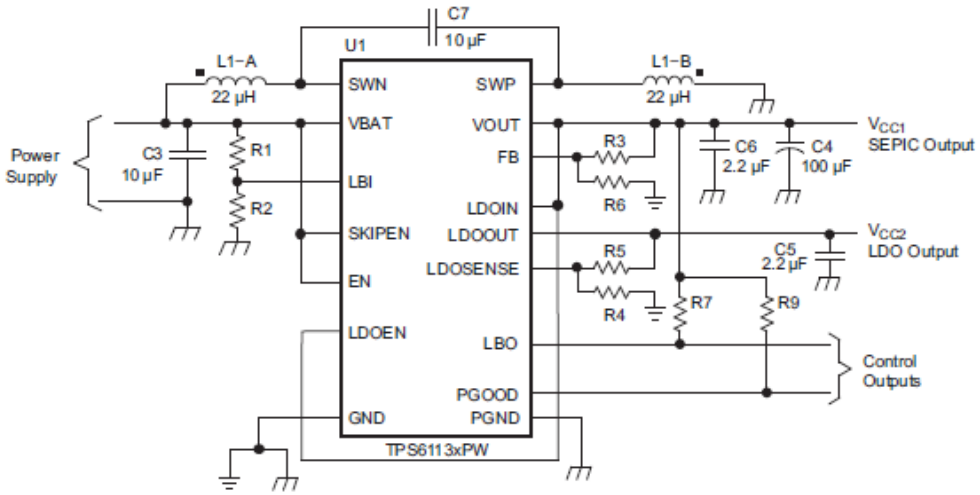


Figure 24. Typical Application Circuit for Adjustable Output Voltage Option

LDO

Programming the output voltage of the LDO follows almost the same rules as at the dc/dc converter section. The maximum recommended output voltage of the LDO is 5.5 V. Since reference and internal feedback circuitry are similar, as they are at the dc/dc converter section, R4 also should be in the 200-kΩ range. The calculation of the value of R5 can be done using the following Equation 3:

$$R5 = R4 \times \left(\frac{V_O}{V_{FB}} - 1\right) = 180\ k\Omega \times \left(\frac{V_O}{500\ mV} - 1\right) \quad (3)$$

If as an example, an output voltage of 1.5 V is needed, a 360 kΩ-resistor should be chosen for R5.

Programming the LBI/LBO Threshold Voltage

The current through the resistive divider should be about 100 times greater than the current into the LBI pin. The typical current into the LBI pin is 0.01 µA, and the voltage across R2 is equal to the LBI voltage threshold that is generated on-chip, which has a value of 500 mV. The recommended value for R2 is therefore in the range of 500 kΩ. From that, the value of resistor R1, depending on the desired minimum battery voltage V_{BAT}, can be calculated using Equation 4.

$$R1 = R2 \times \left(\frac{V_{BAT}}{V_{LBI-threshold}} - 1\right) = 390\ k\Omega \times \left(\frac{V_{BAT}}{500\ mV} - 1\right) \quad (4)$$

The output of the low battery supervisor is a simple open-drain output that goes active low if the dedicated battery voltage drops below the programmed threshold voltage on LBI. The output requires a pullup resistor with a recommended value of 1 MΩ. The maximum voltage which is used to pull up the LBO outputs should not exceed the output voltage of the dc/dc converter. If not used, the LBO pin can be left floating or tied to GND.

Inductor Selection

A SEPIC converter normally requires three main passive components for storing energy during the conversion. Two inductors, a flying capacitor, and a storage capacitor at the output are required. To select the two inductors, it is recommended to keep the possible peak inductor current below the current limit threshold of the power switch in the chosen configuration. For example, the typical current limit threshold of the TPS6113x's switch is 1300 mA at an output voltage of 3.3 V. The highest peak current through the switch is the sum of the two inductor currents and depends on the output load, the input (V_{BAT}), and the output voltage (V_{OUT}). Estimation of the maximum average inductor current of each inductor can be done using Equation 5:

$$I_{L1-A} = I_{L1-B} = I_{OUT} \times \frac{V_{OUT}}{V_{BAT} \times 0.8} \quad (5)$$

For example, for an output current of 300 mA at 3.3 V, at least 680 mA of average current flows through each of the the inductors at a minimum input voltage of 1.8 V.

The second parameter for choosing the inductor is the desired current ripple in the inductor. Normally, it is advisable to work with a ripple of around ±20% of the average inductor current. A smaller ripple reduces the magnetic hysteresis losses in the inductor, as well as output voltage ripple and EMI. But in the same way, regulation time at load changes rises. In addition, a larger inductor increases the total system costs. With those parameters, it is possible to calculate the value for the inductor by using Equation 6:

$$L1 - A = L1 - B = \frac{V_{BAT} \times V_{OUT}}{\Delta I_L \times f \times \left(V_{OUT} + V_{BAT}\right)} \quad (6)$$

Parameter *f* is the switching frequency andΔI_L is the ripple current in the inductor, i.e., 40% ΔI_L. In this example, the desired inductance is in the range of 20 µH. With this calculated value and the calculated currents, it is possible to choose a suitable inductor. In typical applications a 22 µH inductance is recommended. The device

has been optimized to operate with inductance values between 10 µH and 47 µH. Nevertheless operation with higher inductance values may be possible in some applications. Detailed stability analysis is recommended. Care has to be taken that load transients and losses in the circuit can lead to higher currents as estimated in Equation 6. Also, the losses in the inductor caused by magnetic hysteresis losses and copper losses are a major parameter for total circuit efficiency.

The following inductor series from different suppliers have been used with the TPS6113X converters:

List of Inductors

VENDOR	RECOMMENDED INDUCTOR SERIES	COUPLED INDUCTOR SERIES
Coilcraft	LPS4012	LPD4012
	LPS3015	
Cooper Electronics Technologies	DR73	DRQ73
	DR74	DRQ74
EPCOS	B82462G	
Sumida	CDRH5D18	
Wurth Electronik	7447789___	744878220
	7447779___	744877220

Capacitor Selection

Input Capacitor

At least a 10-µF input capacitor is recommended to improve transient behavior of the regulator and EMI behavior of the total power supply circuit. A ceramic capacitor or a tantalum capacitor with a 100-nF ceramic capacitor in parallel, placed close to the IC, is recommended.

Flying Capacitor DC/DC Converter

In the normal operating mode, the *flying* capacitor (C7) must be large enough so that the voltage across the capacitor is small. This means the resonance frequency formed by the *flying* capacitor and the inductors must be at least ten times lower than the switching frequency (see Equation 7).

$$C_{min} = \frac{100}{4\pi^2 f^2 L} \quad (7)$$

Where L is the inductance of L1-A or L1-B.

To optimize efficiency, capacitors with very low ESR such as ceramic capacitors are recommended. The voltage rating of the *flying* capacitor must be higher than the input voltage V_{BAT}.

Output Capacitor DC/DC Converter

The major parameter necessary to define the output capacitor is the maximum allowed output voltage ripple of the converter. This ripple is determined by two parameters of the capacitor, the capacitance and the ESR. It is possible to calculate the minimum capacitance needed for the defined ripple, supposing that the ESR is zero, by using Equation 8:

$$C_{min} = \frac{I_{OUT} \times V_{OUT}}{f \times \Delta V \times \left(V_{OUT} + V_{BAT}\right)} \quad (8)$$

Parameter *f* is the switching frequency and ΔV is the maximum allowed ripple.

With a chosen ripple voltage of 15 mV, a minimum capacitance of 26 µF is needed. The total ripple is larger due to the ESR of the output capacitor. This additional component of the ripple can be calculated using Equation 9:

$$\Delta V_{ESR} = I_{OUT} \times R_{ESR} \quad (9)$$

An additional ripple of 24 mV is the result of using a tantalum capacitor with a low ESR of 80 mΩ. The total ripple is the sum of the ripple caused by the capacitance and the ripple caused by the ESR of the capacitor. In this example, the total ripple is 39 mV. Additional ripple is caused by load transients. This means that the output

4.6. MAX1555

19-2902; Rev 0; 7/03

MAXIM

SOT23 Dual-Input USB/AC Adapter 1-Cell Li+ Battery Chargers

MAX1551/MAX1555

General Description

The MAX1551/MAX1555 charge a single-cell lithium-ion (Li+) battery from both USB* and AC adapter sources. They operate with no external FETs or diodes, and accept operating input voltages up to 7V.

On-chip thermal limiting simplifies PC board layout and allows optimum charging rate without the thermal limits imposed by worst-case battery and input voltage. When the MAX1551/MAX1555 thermal limits are reached, the chargers do not shut down, but progressively reduce charging current.

The MAX1551 includes a $\overline{POK}$ output to indicate when input power is present. If either charging source is active, $\overline{POK}$ goes low. The MAX1555 instead features a $\overline{CHG}$ output to indicate charging status.

With USB connected, but without DC power, charge current is set to 100mA (max). This allows charging from both powered and unpowered USB hubs with no port communication required. When DC power is connected, charging current is set at 280mA (typ). No input-blocking diodes are required to prevent battery drain.

The MAX1551/MAX1555 are available in 5-pin thin SOT23 packages and operate over a -40°C to +85°C range.

Applications

PDAs
Wireless Appliances
Cell Phones
Digital Cameras

Features

- Charge from USB or AC Adapter
- Automatic Switchover when AC Adapter is Plugged In
- On-Chip Thermal Limiting Simplifies Board Design
- Charge Status Indicator
- 5-Pin Thin SOT23 Package
- Protected by U.S. Patent #6,507,172

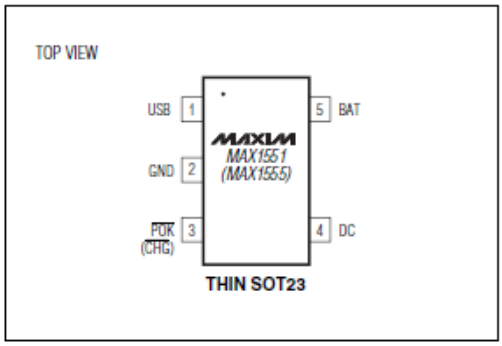
Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX1551EZK-T	-40°C to +85°C	5 Thin SOT23-5
MAX1555EZK-T	-40°C to +85°C	5 Thin SOT23-5

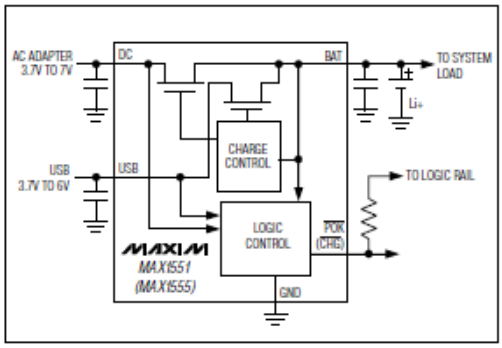
Selector Guide

PART	TOP MARK	FEATURES
MAX1551EZK	ADRT	$\overline{POK}$ Output
MAX1555EZK	ADRU	$\overline{CHG}$ Output

Pin Configuration



Typical Operating Circuit



*Protected by U.S. Patent #6,507,172.

SOT23 Dual-Input USB/AC Adapter 1-Cell Li+ Battery Chargers

ABSOLUTE MAXIMUM RATINGS

DC to GND .....0 to +8V
DC to BAT .....0 to +7V
BAT, $\overline{CHG}$, $\overline{POK}$, USB to GND .....-0.3V to +7V
Continuous Power Dissipation (T_A = +70°C)
5-Pin Thin SOT23 (derate 9.1mW/°C above +70°C) 727mW

Operating Temperature Range .....-40°C to +85°C
Junction Temperature Range .....-40°C to +150°C
Storage Temperature Range .....-65°C to +150°C
Lead Temperature (soldering, 10s) .....+300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

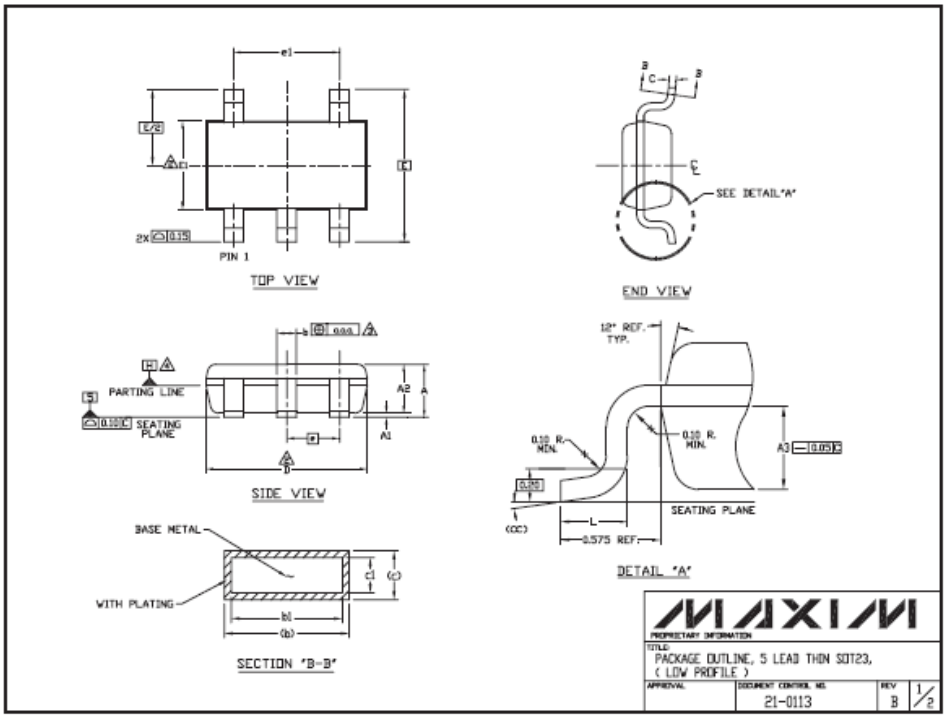
ELECTRICAL CHARACTERISTICS

(V_{DC} = 5V, V_{USB} = 0, I_{BAT} = 0, C_{BAT} = 1µF, **T_A = 0°C to +85°C**, unless otherwise noted.)

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
DC					
DC Voltage Range	(Note 1)	3.7		7.0	V
DC to BAT Voltage Range		0.1		6.0	V
DC Undervoltage Lockout Threshold	Input rising, 430mV hysteresis, V_{BAT} = 3V (Note 1)	3.75	3.95	4.15	V
DC Supply Current			1.75	3	mA
DC to BAT On-Resistance	V_{DC} = 3.7V, V_{BAT} = 3.6V		1	2	Ω
DC to BAT Dropout Voltage	When charging stops, V_{BAT} = 4V, DC falling, 200mV hysteresis	30	60	90	mV
USB					
USB Voltage Range	(Note 1)	3.7		6.0	V
USB Undervoltage Threshold	Input rising, 430mV hysteresis, V_{DC} = 0, V_{BAT} = 3V (Note 1)	3.75	3.95	4.15	V
USB Supply Current	V_{USB} = 5V, V_{DC} = 0		1.65	3	mA
USB to BAT On-Resistance	V_{USB} = 3.7V, V_{BAT} = 3.6V, V_{DC} = 0		2	4	Ω
USB to BAT Dropout Voltage	When charging stops, V_{BAT} = 4V, USB falling, 200mV hysteresis, V_{DC} = 0	30	60	90	mV
BAT					
BAT Regulation Voltage	V_{DC} or V_{USB} = 5V	4.158	4.2	4.242	V
DC Charging Current	V_{BAT} = 3.3V, V_{USB} = 0, V_{DC} = 5V	220	280	340	mA
USB Charging Current	V_{BAT} = 3.3V, V_{DC} = 0, V_{USB} = 5V	80	90	100	mA
BAT Prequal Threshold	V_{BAT} rising, 100mV hysteresis	2.9	3	3.1	V
Prequalification Charging Current	V_{BAT} = 2.8V	20	40	80	mA
BAT Leakage Current	V_{DC} = V_{USB} = 0, V_{BAT} = 4.2V			5	µA
$\overline{POK}$, $\overline{CHG}$, AND THERMAL LIMIT					
$\overline{CHG}$ Threshold	Charge current where $\overline{CHG}$ goes high, I_{BAT} falling, 50mA hysteresis	25	50	100	mA
$\overline{CHG}$, $\overline{POK}$ Logic-Low Output	$I_{\overline{CHG}}$, $I_{\overline{POK}}$ = 10mA		150	300	mV
$\overline{CHG}$, $\overline{POK}$ Leakage Current	$V_{\overline{CHG}}$, $V_{\overline{POK}}$ = 6V, T_A = +25°C		0.001	1	µA
Thermal-Limit Temperature	Charge current reduced by 17mA/°C above this temperature		+110		°C

Package Information

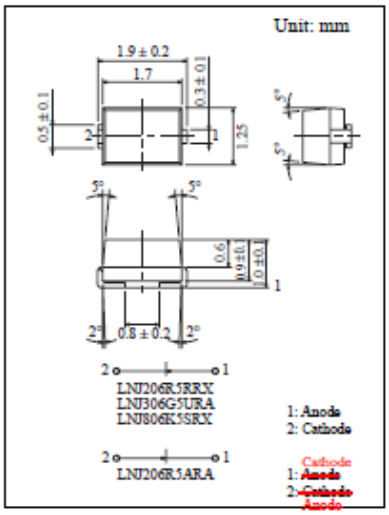
(The package drawing(s) in this data sheet may not reflect the most current specifications. For the latest package outline information, go to **www.maxim-ic.com/packages**.)



4.7. DIODOS LED SMD

Surface Mounting Chip LED

S – J Type



Conventional Part No.	Global Prat No.	Lighting Color
LNJ206R5RRX	LNJ206R5RRX	Red
LNJ206R5ARA	LNJ206R5ARA	Ultra Bright Red
LNJ306G5URA	LNJ306G5URA	Green
LNJ806K5SRX	LNJ806K5SRX	Soft Orange

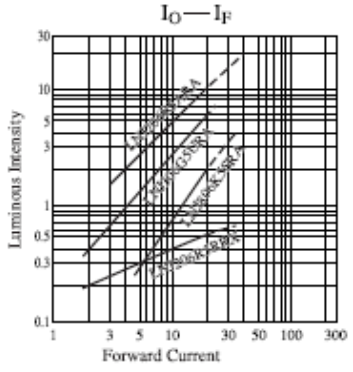
■ Absolute Maximum Ratings (T_a = 25°C)

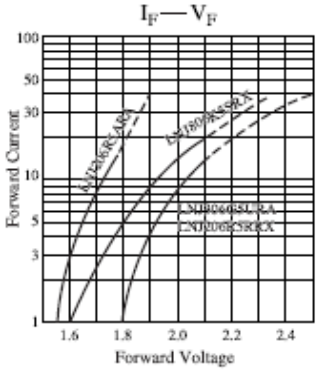
Lighting Color	P_D(mW)	I_F(mA)	I_{FP}(mA)*	V_R(V)	T_{opr}(°C)	T_{stg}(°C)
Red	60	20	60	4	−25 ~ +85	−30 ~ +100
Ultra Bright Red	60	20	60	3	−25 ~ +85	−30 ~ +100
Green	60	20	60	4	−25 ~ +85	−30 ~ +100
Soft Orange	60	20	60	3	−25 ~ +85	−30 ~ +100

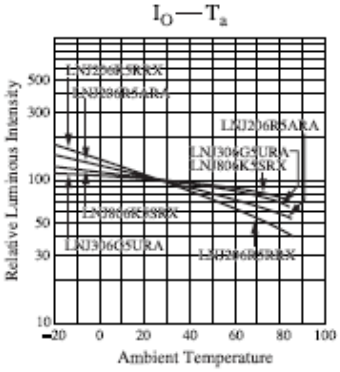
Pulse width 1 msec. The condition of I_{FP} is duty 10%, Pulse width 1 msec

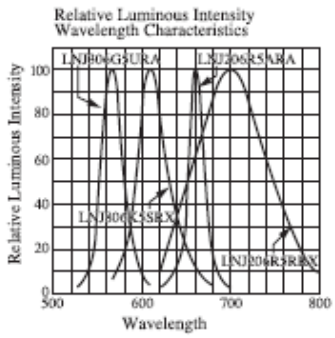
■ Electro–Optical Characteristics (T_a = 25°C)

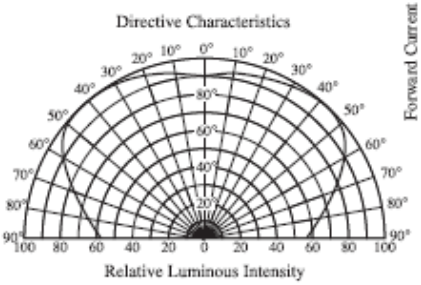
Conventional Part No.	Lighting Color	Lens Color	I_O		I_F	V_F		λ_P	$\Delta\lambda$	I_F	I_R	
			Typ	Min		Typ	Max	Typ	Typ		Max	V_R
LNJ206R5RRX	Red	Red Diffused	0.4	0.15	10	2.03	2.6	700	100	10	5	4
LNJ206R5ARA	Ultra Bright Red	Red Diffused	5.0	1.7	10	1.72	2.5	660	20	10	10	3
LNJ306G5URA	Green	Green Diffused	2.6	0.9	10	2.03	2.6	565	30	10	10	4
LNJ806K5SRX	Soft Orange	Yellow Diffused	0.8	0.3	10	1.93	2.6	610	40	10	10	3
Unit	—	—	mcd	mcd	mA	V	V	nm	nm	mA	μA	V

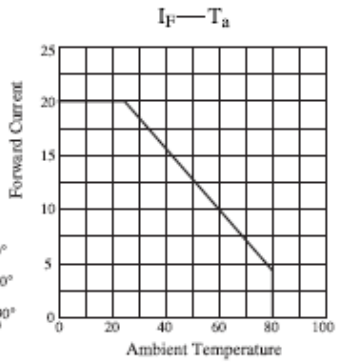












4.8.SWITCH ADG719

ANALOG DEVICES

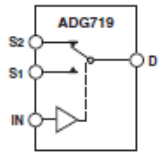
CMOS 1.8 V to 5.5 V, 2.5 Ω 2:1 Mux/SPDT Switch in SOT-23

ADG719

FEATURES
1.8 V to 5.5 V Single Supply
4 Ω (Max) On Resistance
0.75 Ω (Typ) On Resistance Flatness
Automotive Temperature Range: –40°C to +125°C
–3 dB Bandwidth > 200 MHz
Rail-to-Rail Operation
6-Lead SOT-23 Package and 8-Lead μSOIC Package
Fast Switching Times:
t_{ON} = 12 ns
t_{OFF} = 6 ns
Typical Power Consumption (< 0.01 μW)
TTL/CMOS Compatible

APPLICATIONS
Battery-Powered Systems
Communication Systems
Sample-and-Hold Systems
Audio Signal Routing
Video Switching
Mechanical Reed Relay Replacement

FUNCTIONAL BLOCK DIAGRAM



SWITCHES SHOWN FOR A LOGIC "1" INPUT

GENERAL DESCRIPTION

The ADG719 is a monolithic CMOS SPDT switch. This switch is designed on a submicron process that provides low power dissipation yet gives high switching speed, low on resistance, and low leakage currents.

The ADG719 can operate from a single-supply range of 1.8 V to 5.5 V, making it ideal for use in battery-powered instruments and with the new generation of DACs and ADCs from Analog Devices.

Each switch of the ADG719 conducts equally well in both directions when on. The ADG719 exhibits break-before-make switching action.

Because of the advanced submicron process, –3 dB bandwidths of greater than 200 MHz can be achieved.

The ADG719 is available in a 6-lead SOT-23 package and an 8-lead μSOIC package.

PRODUCT HIGHLIGHTS

1. 1.8 V to 5.5 V Single-Supply Operation. The ADG719 offers high performance, including low on resistance and fast switching times, and is fully specified and guaranteed with 3 V and 5 V supply rails.
2. Very Low R_{ON} (4 Ω Max at 5 V and 10 Ω Max at 3 V). At 1.8 V operation, R_{ON} is typically 40 Ω over the temperature range.
3. Automotive Temperature Range: –40°C to +125°C
4. On Resistance Flatness ($R_{FLAT(ON)}$) (0.75 Ω typ).
5. –3 dB Bandwidth > 200 MHz.
6. Low Power Dissipation. CMOS construction ensures low power dissipation.
7. Fast t_{ON}/t_{OFF}.
8. Tiny 6-lead SOT-23 and 8-lead μSOIC packages.

ADG719

ABSOLUTE MAXIMUM RATINGS[1]

(T_A = 25°C, unless otherwise noted.)

V_{DD} to GND ..... –0.3 V to +7 V
Analog, Digital Inputs[2] ..... –0.3 V to V_{DD} + 0.3 V or
..... 30 mA, Whichever Occurs First
Peak Current, S or D ..... 100 mA
(Pulsed at 1 ms, 10% Duty Cycle Max)
Continuous Current, S or D ..... 30 mA
Operating Temperature Range
Industrial (B Version) ..... –40°C to +125°C
Storage Temperature Range ..... –65°C to +150°C
Junction Temperature ..... 150°C
µSOIC Package, Power Dissipation ..... 315 mW
θ_{JA} Thermal Impedance ..... 206°C/W
θ_{JC} Thermal Impedance ..... 44°C/W
SOT-23 Package, Power Dissipation ..... 282 mW
θ_{JA} Thermal Impedance ..... 229.6°C/W
θ_{JC} Thermal Impedance ..... 91.99°C/W
Lead Temperature, Soldering
Vapor Phase (60 sec) ..... 215°C
Infrared (15 sec) ..... 220°C
ESD ..... 1 kV

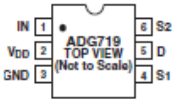
NOTES
[1] Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those listed in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device relisibility. Only one absolute maximum rating may be applied at any one time.
[2] Overvoltages at IN, S, or D will be clamped by internal diodes. Current should be limited to the maximum ratings given.

Table I. Truth Table

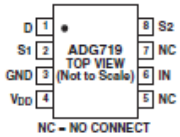
ADG719 IN	Switch S1	Switch S2
0	ON	OFF
1	OFF	ON

PIN CONFIGURATIONS

6-Lead SOT-23 (RT-6)



8-Lead µSOIC (RM-8)



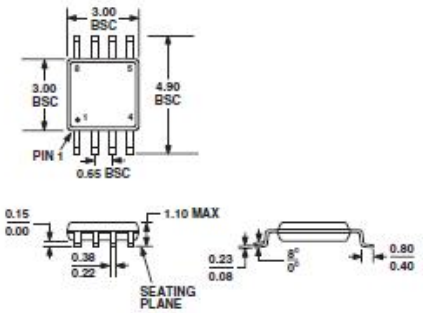
NC = NO CONNECT

TERMINOLOGY

V_{DD}	Most Positive Power Supply Potential
GND	Ground (0 V) Reference
S	Source Terminal. May be an input or output.
D	Drain Terminal. May be an input or output.
IN	Logic Control Input
R_{ON}	Ohmic Resistance between D and S
ΔR_{ON}	On Resistance Match between Any Two Channels i.e., R_{ON} max – R_{ON} min
$R_{FLAT(ON)}$	Flatness is defined as the difference between the maximum and minimum value of on resistance, as measured over the specified analog signal range.
I_S (Off)	Source Leakage Current with the Switch Off
I_D, I_S (On)	Channel Leakage Current with the Switch On
V_D (V_S)	Analog Voltage on Terminals D and S
C_S (Off)	Off Switch Source Capacitance
C_D, C_S (On)	On Switch Capacitance
t_{ON}	Delay between Applying the Digital Control Input and the Output Switching On
t_{OFF}	Delay between Applying the Digital Control Input and the Output Switching Off
t_D	Off Time or On Time Measured between the 90% Points of Both Switches, when Switching From One Address State to Another
Crosstalk	A Measure of Unwanted Signal That Is Coupled through from One Channel to Another as a Result of Parasitic Capacitance
Off Isolation	A Measure of Unwanted Signal Coupling through an Off Switch
Bandwidth	The Frequency at Which the Output is Attenuated by –3 dBs
On Response	The Frequency Response of the On Switch
Insertion Loss	Loss due to On Resistance of Switch

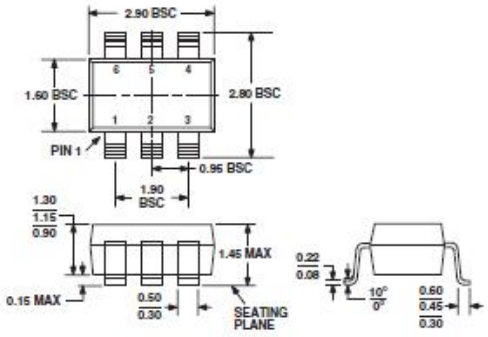
OUTLINE DIMENSIONS

8-Lead Small Outline Package [MSOP]
(RM-8)
Dimensions shown in millimeters



COMPLIANT TO JEDEC STANDARDS MO-187AA

6-Lead Plastic Surface Mount Package [SOT-23]
(RT-6)
Dimensions shown in millimeters



COMPLIANT TO JEDEC STANDARDS MO-178AB

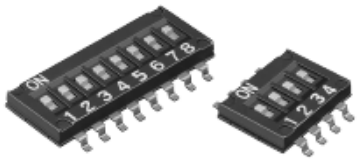
4.9.INTERRUPTORES A6H-1109

OMRON

Half-pitch DIP Switch
A6H

Ultra-low Profile, Half-pitch, Surface-mounting DIP Switch

- Very low profile of 1.55 mm
- Half-pitch (1.27-mm) design allows greater compactness and reduces mounting space by 63%. (compared with conventional models)
- Washable, seal tape models available
- Embossed taping models available
- RoHS Compliant



Ordering Information

Number of poles	Part numbers: Standard models, Tube packaging	Quantity per tube	Standard models, Embossed taping packaging	Quantity per reel	Models with seal tape, Tube packaging	Quantity per tube	Models with seal tape, Embossed taping packaging	Quantity per reel
2	A6H-2101	125	A6H-2101-P	4,000	A6H-2102	125	A6H-2102-P	4,000
4	A6H-4101	75	A6H-4101-P		A6H-4102	75	A6H-4102-P	
6	A6H-6101	54	A6H-6101-P		A6H-6102	54	A6H-6102-P	
8	A6H-8101	40	A6H-8101-P		A6H-8102	40	A6H-8102-P	
10	A6H-0101	33	A6H-0101-P		A6H-0102	33	A6H-0102-P	

Note: 1. Small reels of 500 pieces are also available. Order "-PM" version instead of "-P".
2. Switches cannot be water washed.

Specifications

■ Characteristics

Switching capacity		25 mA at 24 VDC
Minimum permissible load		10 μA at 3.5 VDC
Contact resistance		200 mΩ max.
Insulation resistance		100 MΩ min. (at 100 VDC)
Dielectric strength		300 VAC for 1 min between terminals
Operating force		30 to 500 gf (0.3 to 4.9 N)
Vibration resistance	**Malfunction durability**	10 to 55 Hz, 1.5-mm double amplitude
Shock resistance	**Malfunction durability**	300 m/s^2 min.
Service life	**Mechanical**	1,000 operations min.
	Electrical	1,000 operations min.
Ambient operating temperature		–20 to 70°C at 60% RH max. (with no icing or condensation)
Ambient operating humidity		35% to 95% (at 5 to 35°C)
Weight		0.06 g (2 poles), 0.09 g (4 poles), 0.12 g (6 poles), 0.15 g (8 poles), 0.18 g (10 poles)

Note: Data shown are of initial value.

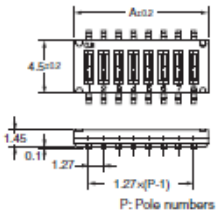
OMRON

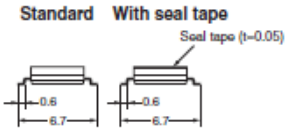
Dimensions

Note: 1. All units are in millimeters unless otherwise indicated.
2. Unless otherwise specified, a tolerance of ±0.4 mm applies to all dimensions.

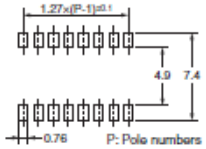
■ Standard

A6H-□101
A6H-□101-P





Dimensions of PCB Pad (Top View)

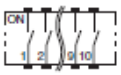


■ With Seal Tape

A6H-□102
A6H-□102-P

No. of poles	Model		Dimension A
	Standard	With seal tape	
2	A6H-2101	A6H-2102	3.77
4	A6H-4101	A6H-4102	6.31
6	A6H-6101	A6H-6102	8.85
8	A6H-8101	A6H-8102	11.39
10	A6H-0101	A6H-0102	13.93

Internal connections (top view)



4.10. TMP36

ANALOG DEVICES

Low Voltage Temperature Sensors

TMP35/TMP36/TMP37

FEATURES

Low voltage operation (2.7 V to 5.5 V)
Calibrated directly in °C
10 mV/°C scale factor (20 mV/°C on TMP37)
±2°C accuracy over temperature (typ)
±0.5°C linearity (typ)
Stable with large capacitive loads
Specified –40°C to +125°C, operation to +150°C
Less than 50 µA quiescent current
Shutdown current 0.5 µA max
Low self-heating

APPLICATIONS

Environmental control systems
Thermal protection
Industrial process control
Fire alarms
Power system monitors
CPU thermal management

GENERAL DESCRIPTION

The TMP35/TMP36/TMP37 are low voltage, precision centigrade temperature sensors. They provide a voltage output that is linearly proportional to the Celsius (centigrade) temperature. The TMP35/ TMP36/TMP37 do not require any external calibration to provide typical accuracies of ±1°C at +25°C and ±2°C over the –40°C to +125°C temperature range.

The low output impedance of the TMP35/TMP36/TMP37 and its linear output and precise calibration simplify interfacing to temperature control circuitry and ADCs. All three devices are intended for single-supply operation from 2.7 V to 5.5 V maximum. The supply current runs well below 50 µA, providing very low self-heating—less than 0.1°C in still air. In addition, a shutdown function is provided to cut the supply current to less than 0.5 µA.

The TMP35 is functionally compatible with the LM35/LM45 and provides a 250 mV output at 25°C. The TMP35 reads temperatures from 10°C to 125°C. The TMP36 is specified from –40°C to +125°C, provides a 750 mV output at 25°C, and operates to 125°C from a single 2.7 V supply. The TMP36 is functionally compatible with the LM50. Both the TMP35 and TMP36 have an output scale factor of 10 mV/°C.

FUNCTIONAL BLOCK DIAGRAM

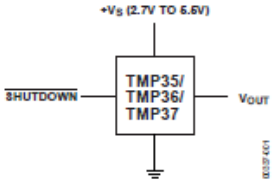


Figure 1.

PIN CONFIGURATIONS

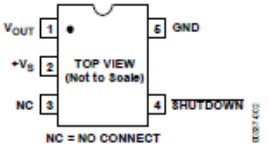


Figure 2. RJ-5 (SOT-23)

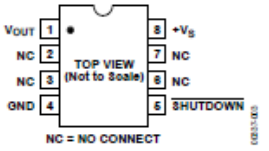


Figure 3. R-8 (SOIC_N)

Figure 4. T-3 (TO-92)

The TMP37 is intended for applications over the range of 5°C to 100°C and provides an output scale factor of 20 mV/°C. The TMP37 provides a 500 mV output at 25°C. Operation extends to 150°C with reduced accuracy for all devices when operating from a 5 V supply.

The TMP35/TMP36/TMP37 are available in low cost 3-lead TO-92, 8-lead SOIC_N, and 5-lead SOT-23 surface-mount packages.

SPECIFICATIONS

V_S = 2.7 V to 5.5 V, −40°C ≤ T_A ≤ +125°C, unless otherwise noted.

Table 1.

Parameter[1]	Symbol	Test Conditions/Comments	Min	Typ	Max	Unit
ACCURACY						
TMP35/TMP36/TMP37 (F Grade)		T_A = 25°C		±1	±2	°C
TMP35/TMP36/TMP37 (G Grade)		T_A = 25°C		±1	±3	°C
TMP35/TMP36/TMP37 (F Grade)		Over rated temperature		±2	±3	°C
TMP35/TMP36/TMP37 (G Grade)		Over rated temperature		±2	±4	°C
Scale Factor, TMP35		10°C ≤ T_A ≤ 125°C		10		mV/°C
Scale Factor, TMP36		−40°C ≤ T_A ≤ +125°C		10		mV/°C
Scale Factor, TMP37		5°C ≤ T_A ≤ 85°C		20		mV/°C
		5°C ≤ T_A ≤ 100°C		20		mV/°C
		3.0 V ≤ V_S ≤ 5.5 V				
Load Regulation		0 µA ≤ I_L ≤ 50 µA				
		−40°C ≤ T_A ≤ +105°C		6	20	m°C/µA
		−105°C ≤ T_A ≤ +125°C		25	60	m°C/µA
Power Supply Rejection Ratio	PSRR	T_A = 25°C		30	100	m°C/V
		3.0 V ≤ V_S ≤ 5.5 V		50		m°C/V
Linearity				0.5		°C
Long-Term Stability		T_A = 150°C for 1 kHz		0.4		°C
SHUTDOWN						
Logic High Input Voltage	V_{IH}	V_S = 2.7 V	1.8			V
Logic Low Input Voltage	V_{IL}	V_S = 5.5 V			400	mV
OUTPUT						
TMP35 Output Voltage		T_A = 25°C		250		mV
TMP36 Output Voltage		T_A = 25°C		750		mV
TMP37 Output Voltage		T_A = 25°C		500		mV
Output Voltage Range			100		2000	mV
Output Load Current	I_L		0		50	µA
Short-Circuit Current	I_{SC}	Note 2			250	µA
Capacitive Load Driving	C_L	No oscillations[2]	1000	10000		pF
Device Turn-On Time		Output within ±1°C, 100 kΩ‖100 pF load[2]		0.5	1	ms
POWER SUPPLY						
Supply Range	V_S		2.7		5.5	V
Supply Current	I_{SY} (ON)	Unloaded			50	µA
Supply Current (Shutdown)	I_{SY} (OFF)	Unloaded		0.01	0.5	µA

[1] Does not consider errors caused by self-heating.
[2] Guaranteed but not tested.

TYPICAL PERFORMANCE CHARACTERISTICS

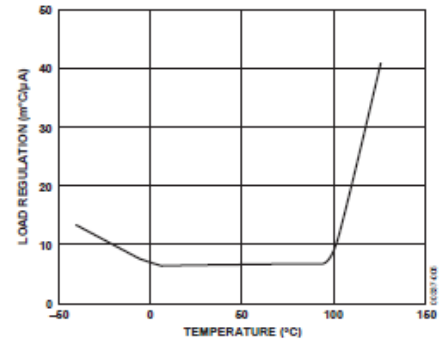


Figure 5. Load Regulation vs. Temperature (m°C/µA)

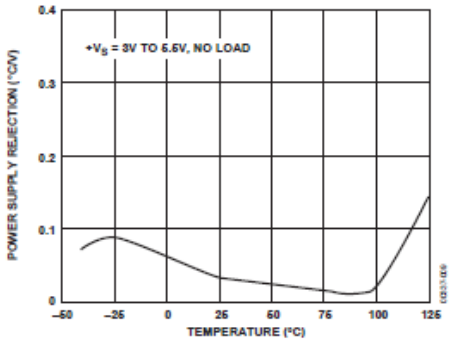


Figure 8. Power Supply Rejection vs. Temperature

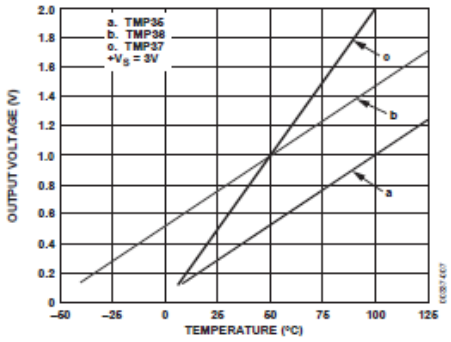


Figure 6. Output Voltage vs. Temperature

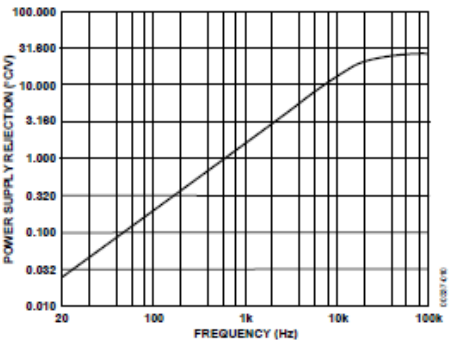


Figure 9. Power Supply Rejection vs. Frequency

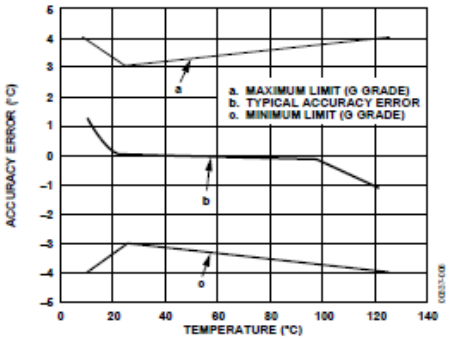


Figure 7. Accuracy Error vs. Temperature

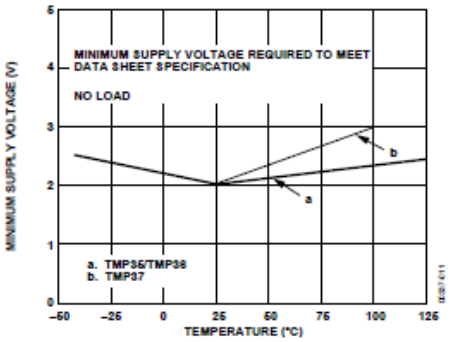


Figure 10. Minimum Supply Voltage vs. Temperature

4.11. HIH5030

Honeywell

HIH-5030/5031 Series

Low Voltage Humidity Sensors

DESCRIPTION

The HIH-5030/5031 Series Low Voltage Humidity Sensors operate down to 2.7 Vdc, often ideal in battery-powered systems where the supply is a nominal 3 Vdc.

The HIH 5030/5031 complements our existing line of 5 Vdc SMD (Surface Mount Device) humidity sensors. SMD packaging on tape and reel allows for use in high volume, automated pick and place manufacturing, eliminating lead misalignment to printed circuit board through-holes.

The HIH-5030/5031 Series Humidity Sensors are designed specifically for high volume OEM (Original Equipment Manufacturer) users.

Direct input to a controller or other device is made possible by this sensor's near linear voltage output. With a typical current draw of only 200 µA, the HIH-5030/5031 Series is ideally suited for many low drain, battery operated systems.

Tight sensor interchangeability reduces or eliminates OEM production calibration costs.

The HIH-5030/5031 Series delivers instrumentation-quality RH (Relative Humidity) sensing performance in a competitively priced, solderable SMD.

The HIH-5030 is a covered integrated circuit humidity sensor. The HIH-5031 is a covered, condensation-resistant, integrated circuit humidity sensor that is factory-fitted with a hydrophobic filter allowing it to be used in many condensing environments including industrial, medical and commercial applications.

The RH sensor uses a laser trimmed, thermoset polymer capacitive sensing element with on-chip integrated signal conditioning.

The sensing element's multilayer construction provides excellent resistance to most application hazards such as condensation, dust, dirt, oils and common environmental chemicals.

Sample packs are available. See order guide.

FEATURES

- Operates down to 2.7 Vdc, often ideal in battery-powered systems where the supply is a nominal 3 Vdc.
- Tape and reel packaging allows for use in high volume pick and place manufacturing (1,000 units per tape and reel)
- Molded thermoset plastic housing
- Near linear voltage output vs %RH
- Laser trimmed interchangeability
- Low power design
- Enhanced accuracy
- Fast response time
- Stable, low drift performance
- Chemically resistant

POTENTIAL APPLICATIONS

Industrial

- Air compressors
- Battery-powered systems
- Drying equipment
- HVAC (includes air conditioning, air movement, thermostats, humidifiers, de-humidifiers, humidistats, enthalpy sensing)
- OEM assemblies
- Office automation equipment
- Process equipment
- Refrigeration (includes bulk and transport systems)
- Telecommunications cabinets
- Weather stations and meteorology equipment

Medical

- Hospital air compressors
- Infant incubators
- Microenvironments
- Sleep apnea equipment
- Treadmill stress monitoring equipment

HIH-5030/5031 Series

Table 1. Performance Specifications (At 3.3 Vdc supply and 25 °C [77 °F] unless otherwise noted.)

Parameter	Minimum	Typical	Maximum	Unit	Specific Note
Interchangeability (first order curve)					
0% RH to 10% RH, 90% RH to 100% RH	-7	–	7	% RH	–
11% RH to 89% RH	-3	–	3	% RH	
Accuracy (best fit straight line) 11% RH to 89% RH	-3	–	+3	% RH	4
Hysteresis	–	2	–	% RH	–
Repeatability	–	±0.5	–	% RH	–
Settling time	–	–	70	ms	–
Response time (1/e in slow moving air)	–	5	–	s	–
Stability (at 50% RH in 5 years)	–	±1.2	–	% RH	1
Voltage supply	2.7	–	5.5	Vdc	2
Current supply	–	200	500	µA	–
Voltage output (1st order curve fit)	$V_{OUT}=(V_{SUPPLY})(0.00636(\text{sensor RH}) + 0.1515)$, typical at 25 °C				
Temperature compensation	True RH = (Sensor RH)/(1.0546 – 0.00216T), T in °C				
Output voltage temp. coefficient at 50% RH, 3.3 V	–	-2	–	mV/°C	–
Operating temperature	-40[-40]	See Figure 2.	85[185]	°C[°F]	–
Operating humidity (HIH-5030)	0	See Figure 2.	100	% RH	3
Operating humidity (HIH-5031)	0	See Figure 2.	100	% RH	–
Storage temperature	-50[-58]	–	125[257]	°C [°F]	–
Storage humidity	See Figure 3.			% RH	3

Specific Notes:

1. Includes stress outside of recommended operating zone.
2. Device is tested at 3.3 Vdc and 25 °C.
3. Non-condensing environment. When liquid water falls on the humidity sensor die, output goes to a low rail condition indicating no humidity.
4. Total accuracy including interchangeability is ±3 %RH.

General Notes:

- Sensor is ratiometric to supply voltage.
- Extended exposure to ≥90 % RH causes a reversible shift of 3 % RH.
- Sensor is light sensitive. For best performance, shield sensor from bright light.

Figure 1. Operating Environment (Non-condensing environment for HIH-5030 catalog listings only.)

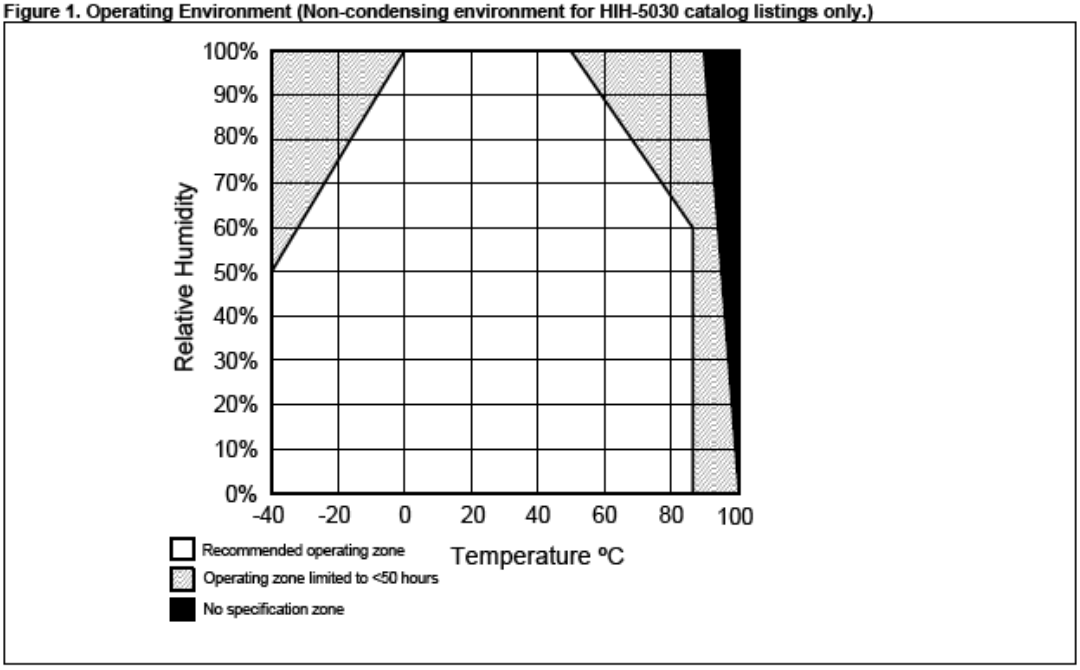


Figure 2. Storage Environment (Non-condensing environment for HIH-5030 catalog listings only.)

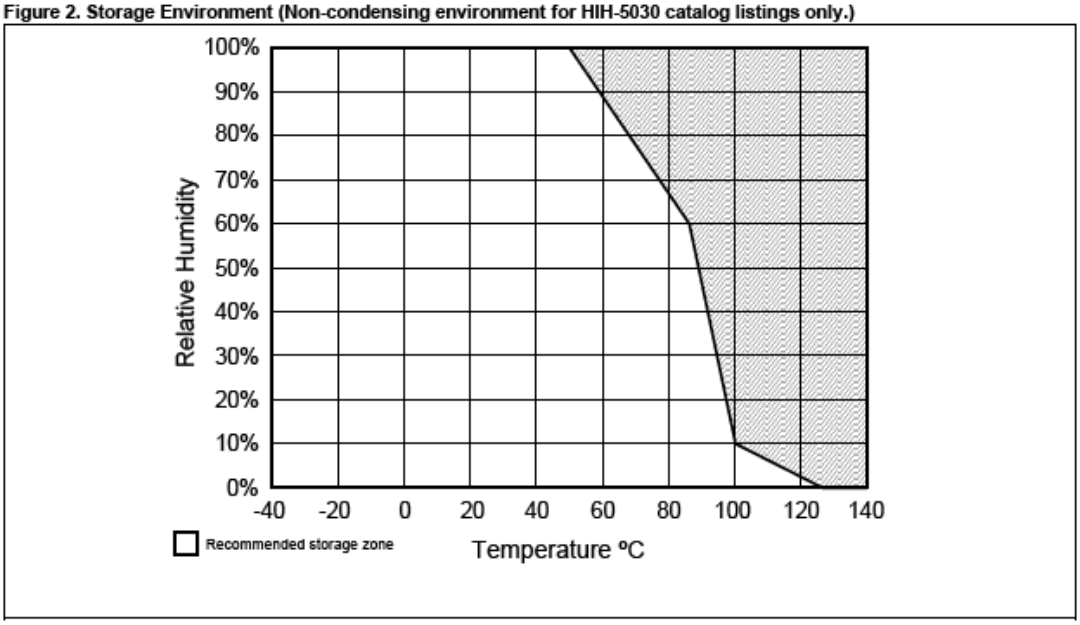


Figure 5. HIH-5030 Mounting Dimensions (For reference only. mm/[in])

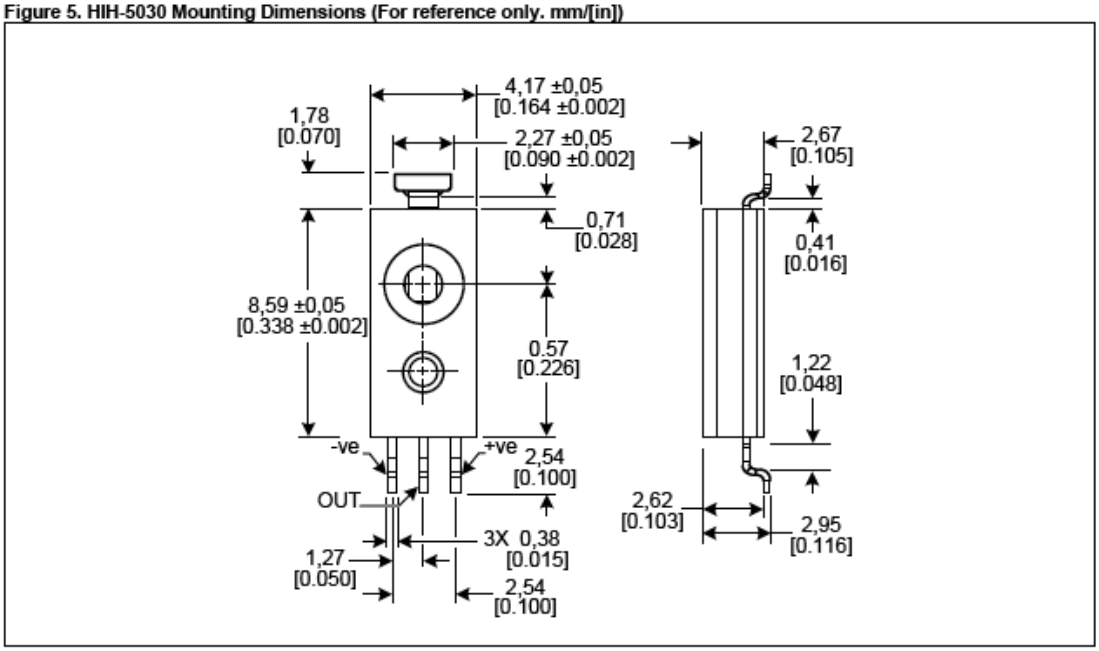
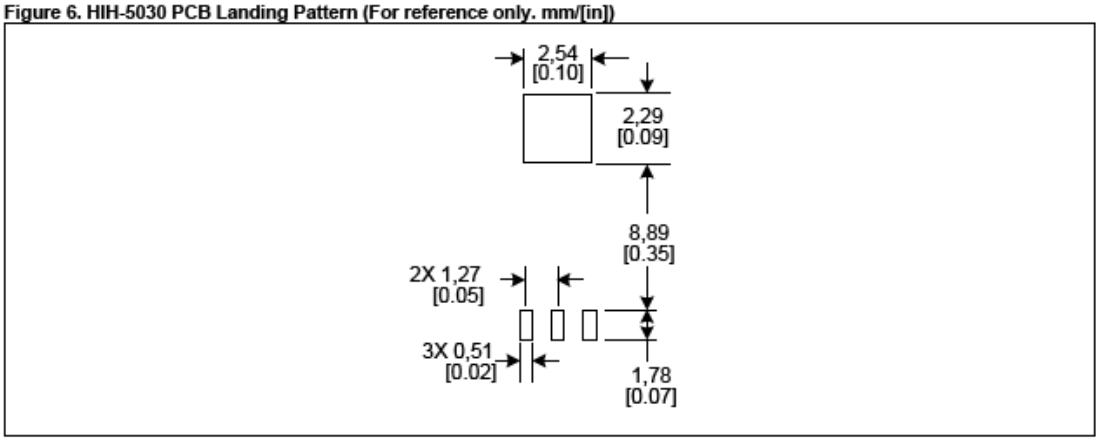


Figure 6. HIH-5030 PCB Landing Pattern (For reference only. mm/[in])



4.12. LDR

Figure 5. HIH-5030 Mounting Dimensions (For reference only. mm/[in])

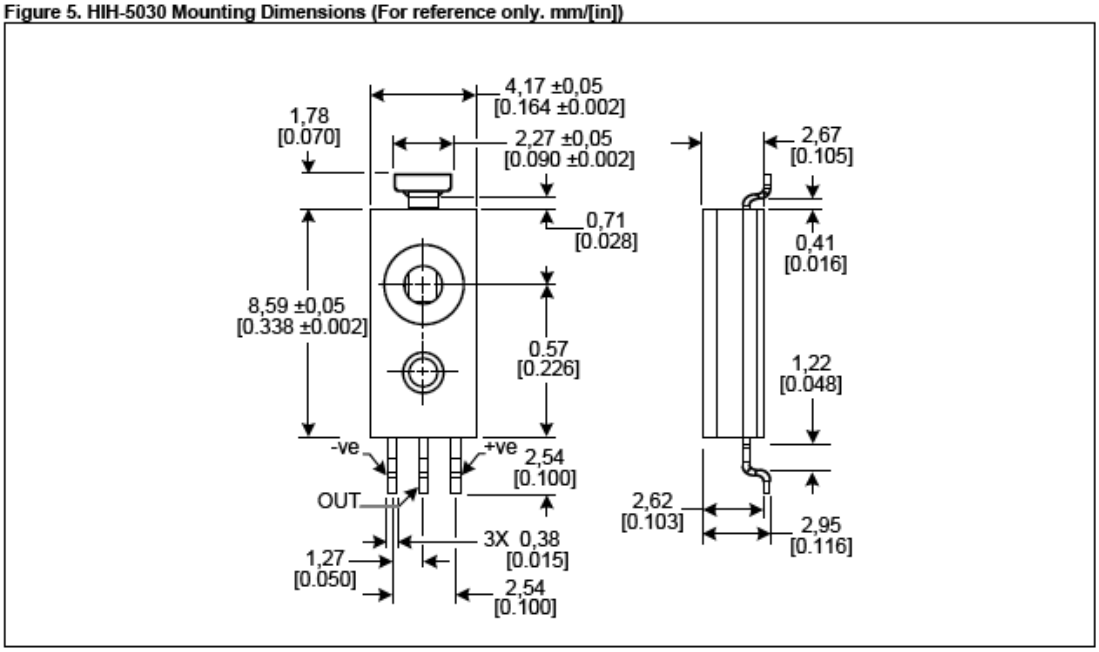
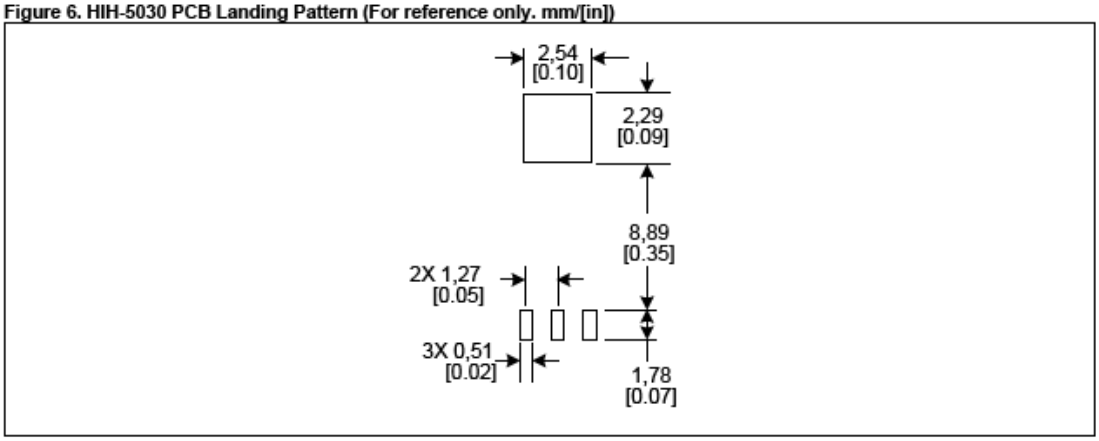


Figure 6. HIH-5030 PCB Landing Pattern (For reference only. mm/[in])



4.13. SENSOR PIR PARALLAX

PARALLAX

Web Site: www.parallax.com
Forums: forums.parallax.com
Sales: sales@parallax.com
Technical: support@parallax.com

Office: (916) 624-8333
Fax: (916) 624-8003
Sales: (888) 512-1024
Tech Support: (888) 997-8267

PIR Sensor (#555-28027)

General Description

The PIR (Passive Infra-Red) Sensor is a pyroelectric device that detects motion by measuring changes in the infrared levels emitted by surrounding objects. This motion can be detected by checking for a high signal on a single I/O pin.

Features

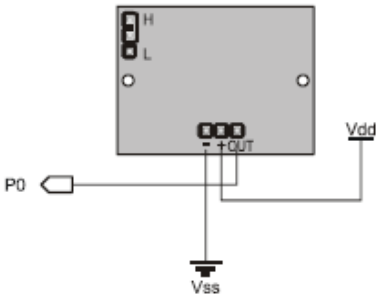
- Single bit output
- Small size makes it easy to conceal
- Compatible with all Parallax microcontrollers
- 3.3V & 5V operation with <100uA current draw

Application Ideas

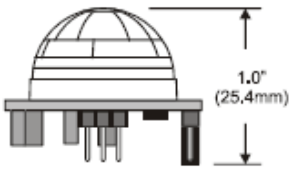
- Alarm Systems
- Halloween Props

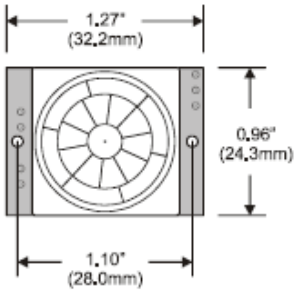
Quick Start Circuit

Note: The sensor is active high when the jumper (shown in the upper left) is in either position.



Module Dimensions





Theory of Operation

Pyroelectric devices, such as the PIR sensor, have elements made of a crystalline material that generates an electric charge when exposed to infrared radiation. The changes in the amount of infrared striking the element change the voltages generated, which are measured by an on-board amplifier. The device contains a special filter called a Fresnel lens, which focuses the infrared signals onto the element. As the ambient infrared signals change rapidly, the on-board amplifier trips the output to indicate motion.

Pin Definitions and Ratings

Pin	Name	Function
-	GND	Connects to Ground or Vss
+	V+	Connects to Vdd (3.3V to 5V) @ ~100uA
OUT	Output	Connects to an I/O pin set to INPUT mode (or transistor/MOSFET)

Jumper Setting

Position	Mode	Description
H	Retrigger	Output remains HIGH when sensor is retriggered repeatedly. Output is LOW when idle (not triggered).
L	Normal	Output goes HIGH then LOW when triggered. Continuous motion results in repeated HIGH/LOW pulses. Output is LOW when idle.

Connecting and Testing

Connect the 3-pin header to your circuit so that the minus (-) pin connects to ground or Vss, the plus (+) pin connects to Vdd and the OUT pin connects to your microcontroller's I/O pin. One easy way to do this would be to use a standard servo/LCD extension cable, available separately from Parallax (#805-00002). This cable makes it easy to plug sensor into the servo headers on our Board Of Education or Professional Development Board. If you use the Board Of Education, be sure the servo voltage jumper (located between the 2 servo header blocks) is in the Vdd position, not Vin. If you do not have this jumper on your board you should manually connect to Vdd through the breadboard. You may also plug the sensor directly into the edge of the breadboard and connect the signals from there. Remember the position of the pins when you plug the sensor into the breadboard.

Calibration

The PIR Sensor requires a 'warm-up' time in order to function properly. This is due to the settling time involved in 'learning' its environment. This could be anywhere from 10-60 seconds. During this time there should be as little motion as possible in the sensors field of view.

Sensitivity

The PIR Sensor has a range of approximately 20 feet. This can vary with environmental conditions. The sensor is designed to adjust to slowly changing conditions that would happen normally as the day progresses and the environmental conditions change, but responds by making its output high when sudden changes occur, such as when there is motion.

4.14. OPTOACOPLADOR 4N25

FAIRCHILD
SEMICONDUCTOR

GlobalOptoisolator™

VDE | UL | CSA | SETI | SEMKO | DEMKO | NEMKO | BABT

6-Pin DIP Optoisolators Transistor Output

4N25
4N26
4N27
4N28

The 4N25, 4N26, 4N27 and 4N28 devices consist of a gallium arsenide infrared emitting diode optically coupled to a monolithic silicon phototransistor detector.

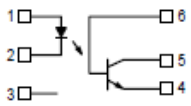
- Most Economical Optoisolator Choice for Medium Speed, Switching Applications
- Meets or Exceeds All JEDEC Registered Specifications
- *To order devices that are tested and marked per VDE 0884 requirements, the suffix "V" must be included at end of part number. VDE 0884 is a test option.*

Applications

- General Purpose Switching Circuits
- Interfacing and coupling systems of different potentials and impedances
- I/O Interfacing
- Solid State Relays

STANDARD THRU HOLE

SCHEMATIC



PIN 1. LED ANODE
2. LED CATHODE
3. N.C.
4. EMITTER
5. COLLECTOR
6. BASE

MAXIMUM RATINGS (T_A = 25°C unless otherwise noted)

Rating	Symbol	Value	Unit
INPUT LED			
Reverse Voltage	V_R	3	Volts
Forward Current — Continuous	I_F	60	mA
LED Power Dissipation @ T_A = 25°C with Negligible Power in Output Detector Derate above 25°C	P_D	120 1.41	mW mW/°C
OUTPUT TRANSISTOR			
Collector–Emitter Voltage	V_{CEO}	30	Volts
Emitter–Collector Voltage	V_{ECO}	7	Volts
Collector–Base Voltage	V_{CBO}	70	Volts
Collector Current — Continuous	I_C	150	mA
Detector Power Dissipation @ T_A = 25°C with Negligible Power in Input LED Derate above 25°C	P_D	150 1.76	mW mW/°C
TOTAL DEVICE			
Isolation Surge Voltage(1) (Peak ac Voltage, 60 Hz, 1 sec Duration)	V_{ISO}	7500	Vac(pk)
Total Device Power Dissipation @ T_A = 25°C Derate above 25°C	P_D	250 2.94	mW mW/°C
Ambient Operating Temperature Range	T_A	−55 to +100	°C
Storage Temperature Range	T_{stg}	−55 to +150	°C
Soldering Temperature (10 sec, 1/16" from case)	T_L	260	°C

1. Isolation surge voltage is an internal device dielectric breakdown rating. For this test, Pins 1 and 2 are common, and Pins 4, 5 and 6 are common.

FAIRCHILD SEMICONDUCTOR

ELECTRICAL CHARACTERISTICS (T_A = 25°C unless otherwise noted)(1)

Characteristic		Symbol	Min	Typ(1)	Max	Unit
INPUT LED						
Forward Voltage (I_F = 10 mA)	T_A = 25°C T_A = –55°C T_A = 100°C	V_F	— — —	1.15 1.3 1.05	1.5 — —	Volts
Reverse Leakage Current (V_R = 3 V)		I_R	—	—	100	µA
Capacitance (V = 0 V, f = 1 MHz)		C_J	—	18	—	pF
OUTPUT TRANSISTOR						
Collector–Emitter Dark Current (V_{CE} = 10 V, T_A = 25°C	4N25,26,27 4N28	I_{CEO}	— —	1 1	50 100	nA
(V_{CE} = 10 V, T_A = 100°C)	All Devices	I_{CEO}	—	1	—	µA
Collector–Base Dark Current (V_{CB} = 10 V)		I_{CBO}	—	0.2	—	nA
Collector–Emitter Breakdown Voltage (I_C = 1 mA)		$V_{(BR)CEO}$	30	45	—	Volts
Collector–Base Breakdown Voltage (I_C = 100 µA)		$V_{(BR)CBO}$	70	100	—	Volts
Emitter–Collector Breakdown Voltage (I_E = 100 µA)		$V_{(BR)ECO}$	7	7.8	—	Volts
DC Current Gain (I_C = 2 mA, V_{CE} = 5 V)		h_{FE}	—	500	—	—
Collector–Emitter Capacitance (f = 1 MHz, V_{CE} = 0)		C_{CE}	—	7	—	pF
Collector–Base Capacitance (f = 1 MHz, V_{CB} = 0)		C_{CB}	—	19	—	pF
Emitter–Base Capacitance (f = 1 MHz, V_{EB} = 0)		C_{EB}	—	9	—	pF
COUPLED						
Output Collector Current (I_F = 10 mA, V_{CE} = 10 V)	4N25,26 4N27,28	I_C (CTR)(2)	2 (20) 1 (10)	7 (70) 5 (50)	— —	mA (%)
Collector–Emitter Saturation Voltage (I_C = 2 mA, I_F = 50 mA)		$V_{CE(sat)}$	—	0.15	0.5	Volts
Turn–On Time (I_F = 10 mA, V_{CC} = 10 V, R_L = 100 Ω)(3)		t_{on}	—	2.8	—	µs
Turn–Off Time (I_F = 10 mA, V_{CC} = 10 V, R_L = 100 Ω)(3)		t_{off}	—	4.5	—	µs
Rise Time (I_F = 10 mA, V_{CC} = 10 V, R_L = 100 Ω)(3)		t_r	—	1.2	—	µs
Fall Time (I_F = 10 mA, V_{CC} = 10 V, R_L = 100 Ω)(3)		t_f	—	1.3	—	µs
Isolation Voltage (f = 60 Hz, t = 1 sec)(4)		V_{ISO}	7500	—	—	Vac(pk)
Isolation Resistance (V = 500 V)(4)		R_{ISO}	10^{11}	—	—	Ω
Isolation Capacitance (V = 0 V, f = 1 MHz)(4)		C_{ISO}	—	0.2	—	pF

1. Always design to the specified minimum/maximum electrical limits (where applicable).
2. Current Transfer Ratio (CTR) = I_C/I_F x 100%.
3. For test circuit setup and waveforms, refer to Figure 11.
4. For this test, Pins 1 and 2 are common, and Pins 4, 5 and 6 are common.

TYPICAL CHARACTERISTICS

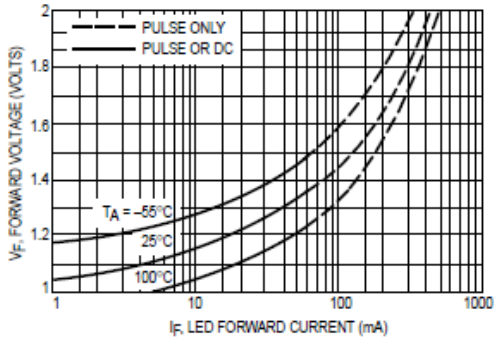


Figure 1. LED Forward Voltage versus Forward Current

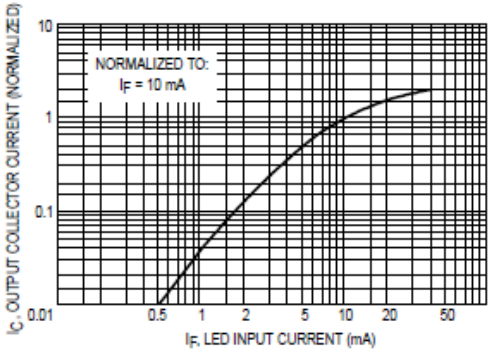


Figure 2. Output Current versus Input Current

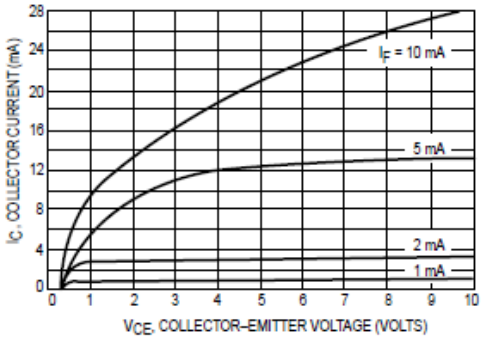


Figure 3. Collector Current versus Collector–Emitter Voltage

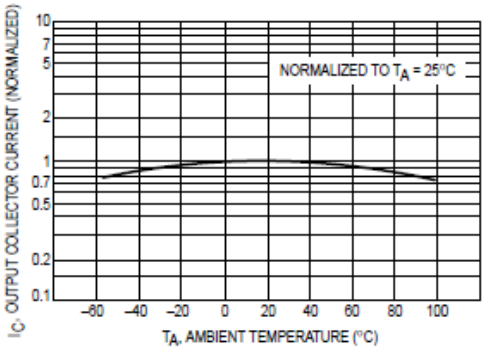


Figure 4. Output Current versus Ambient Temperature

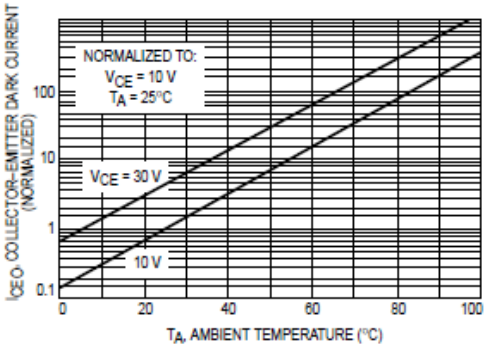


Figure 5. Dark Current versus Ambient Temperature

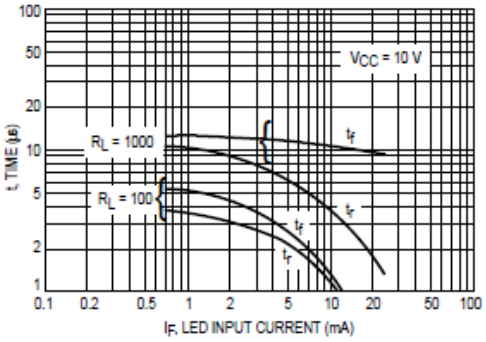


Figure 6. Rise and Fall Times (Typical Values)

4.15. TRIAC OPTOACOPLADO MOC3021

MOTOROLA
SEMICONDUCTOR TECHNICAL DATA

Order this document by MOC3020/D

GlobalOptoisolator™

VDE | UL | CSA | SETI | SEMKO | DEMKO | NEMKO | BABT

6-Pin DIP Random-Phase Optoisolators Triac Driver Output (400 Volts Peak)

MOC3021 [IFT = 15 mA Max]
MOC3022 [IFT = 10 mA Max]
MOC3023* [IFT = 5 mA Max]

*Motorola Preferred Device

The MOC3020 Series consists of gallium arsenide infrared emitting diodes, optically coupled to a silicon bilateral switch.

- ***To order devices that are tested and marked per VDE 0884 requirements, the suffix "V" must be included at end of part number. VDE 0884 is a test option.***

They are designed for applications requiring isolated triac triggering.

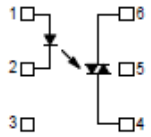
Recommended for 115/240 Vac(rms) Applications:

- Solenoid/Valve Controls
- Lamp Ballasts
- Interfacing Microprocessors to 115 Vac Peripherals
- Motor Controls
- Static ac Power Switch
- Solid State Relays
- Incandescent Lamp Dimmers

STYLE 6 PLASTIC

STANDARD THRU HOLE
CASE 730A–04

SCHEMATIC



1. ANODE
2. CATHODE
3. NC
4. MAIN TERMINAL
5. SUBSTRATE
   DO NOT CONNECT
6. MAIN TERMINAL

MAXIMUM RATINGS (T_A = 25°C unless otherwise noted)

Rating	Symbol	Value	Unit
INFRARED EMITTING DIODE			
Reverse Voltage	V_R	3	Volts
Forward Current — Continuous	I_F	60	mA
Total Power Dissipation @ T_A = 25°C Negligible Power in Triac Driver Derate above 25°C	P_D	100 1.33	mW mW/°C
OUTPUT DRIVER			
Off–State Output Terminal Voltage	V_{DRM}	400	Volts
Peak Repetitive Surge Current (PW = 1 ms, 120 pps)	I_{TSM}	1	A
Total Power Dissipation @ T_A = 25°C Derate above 25°C	P_D	300 4	mW mW/°C
TOTAL DEVICE			
Isolation Surge Voltage(1) (Peak ac Voltage, 60 Hz, 1 Second Duration)	V_{ISO}	7500	Vac(pk)
Total Power Dissipation @ T_A = 25°C Derate above 25°C	P_D	330 4.4	mW mW/°C
Junction Temperature Range	T_J	−40 to +100	°C
Ambient Operating Temperature Range(2)	T_A	−40 to +85	°C
Storage Temperature Range(2)	T_{stg}	−40 to +150	°C
Soldering Temperature (10 s)	T_L	260	°C

1. Isolation surge voltage, V_{ISO}, is an internal device dielectric breakdown rating. For this test, Pins 1 and 2 are common, and Pins 4, 5 and 6 are common.
2. Refer to Quality and Reliability Section in Opto Data Book for information on test conditions.

MOC3021 MOC3022 MOC3023

ELECTRICAL CHARACTERISTICS (T_A = 25°C unless otherwise noted)

Characteristic		Symbol	Min	Typ	Max	Unit
INPUT LED						
Reverse Leakage Current (V_R = 3 V)		I_R	—	0.05	100	μA
Forward Voltage (I_F = 10 mA)		V_F	—	1.15	1.5	Volts
OUTPUT DETECTOR (I_F = 0 unless otherwise noted)						
Peak Blocking Current, Either Direction (Rated V_{DRM}(1))		I_{DRM}	—	10	100	nA
Peak On–State Voltage, Either Direction (I_{TM} = 100 mA Peak)		V_{TM}	—	1.8	3	Volts
Critical Rate of Rise of Off–State Voltage (Figure 7, Note 2)		dv/dt	—	10	—	V/μs
COUPLED						
LED Trigger Current, Current Required to Latch Output (Main Terminal Voltage = 3 V(3))	MOC3021	I_{FT}	—	8	15	mA
	MOC3022		—	—	10	
	MOC3023		—	—	5	
Holding Current, Either Direction		I_H	—	100	—	μA

1. Test voltage must be applied within dv/dt rating.
2. This is static dv/dt. See Figure 7 for test circuit. Commutating dv/dt is a function of the load–driving thyristor(s) only.
3. All devices are guaranteed to trigger at an I_F value less than or equal to max I_{FT}. Therefore, recommended operating I_F lies between max I_{FT} (15 mA for MOC3021, 10 mA for MOC3022, 5 mA for MOC3023) and absolute max I_F (60 mA).

TYPICAL ELECTRICAL CHARACTERISTICS

T_A = 25°C

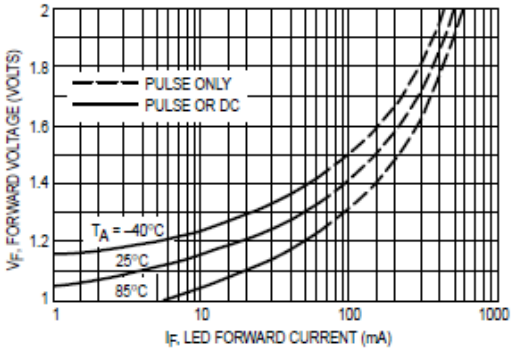


Figure 1. LED Forward Voltage versus Forward Current

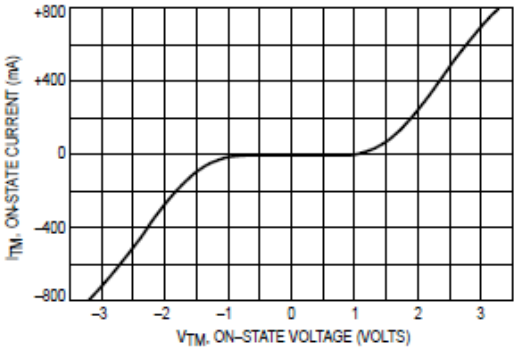


Figure 2. On–State Characteristics

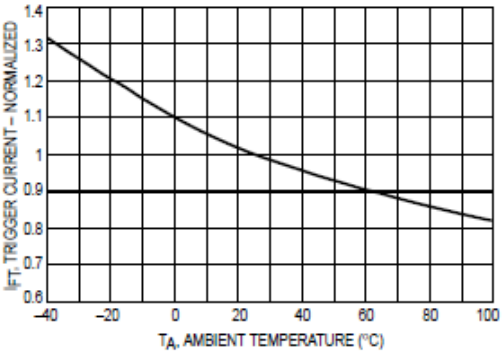


Figure 3. Trigger Current versus Temperature

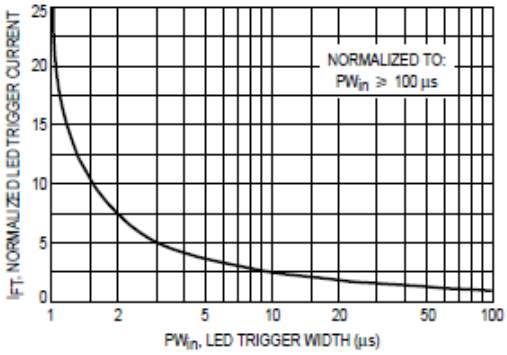


Figure 4. LED Current Required to Trigger versus LED Pulse Width

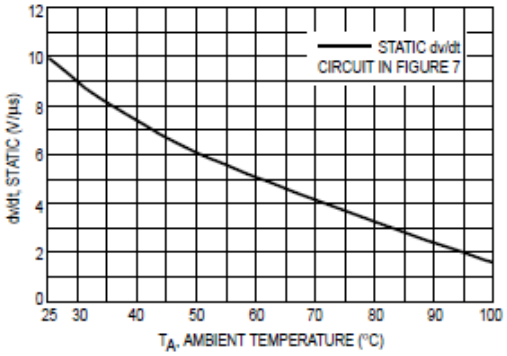


Figure 5. dv/dt versus Temperature

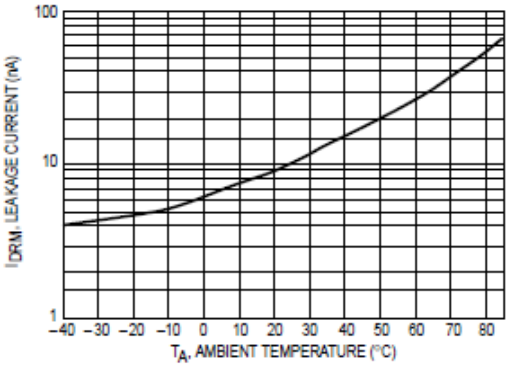
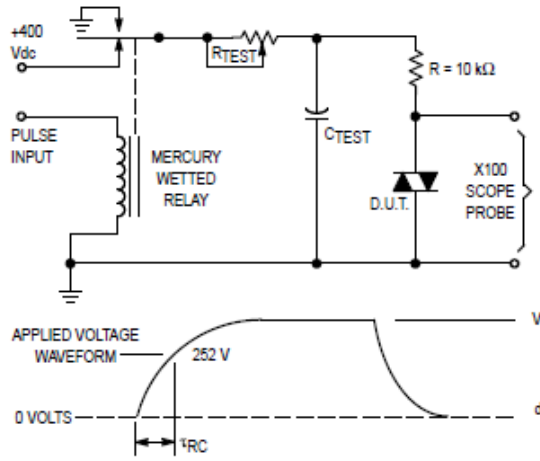


Figure 6. Leakage Current, I_{DRM} versus Temperature



1. The mercury wetted relay provides a high speed repeated pulse to the D.U.T.
2. 100x scope probes are used, to allow high speeds and voltages.
3. The worst–case condition for static dv/dt is established by triggering the D.U.T. with a normal LED input current, then removing the current. The variable R_{TEST} allows the dv/dt to be gradually increased until the D.U.T. continues to trigger in response to the applied voltage pulse, even after the LED current has been removed. The dv/dt is then decreased until the D.U.T. stops triggering. τ_{RC} is measured at this point and recorded.

Figure 7. Static dv/dt Test Circuit