Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO DE FIN DE GRADO

Título: Diseño de varios sensores y actuadores domóticos que utilicen comunicación inalámbrica ZigBee

Autor: Gustavo Alejandro Lanfranconi Peña
Tutor: Alberto Hamilton Castro

Índice de contenido

Índice de tablas

Índice de figuras

1. Introducción y objetivos

1.1. Resumen

El trabajo plantea la creación de una red domótica de bajo consumo mediante la utilización del protocolo inalámbrico ZigBee. Se han utilizado los dispositivos XBee del fabricante Digi, estudiando el máximo aprovechamiento de sus capacidades para evitar así la necesidad de añadir microcontroladores en cada nodo, reduciendo su coste, su consumo y su complejidad. Por este motivo, el control queda centralizado en el nodo coordinador, facilitando la reconfiguración del funcionamiento de los dispositivos.

La viabilidad de este esquema se ha demostrado realizando una red de pequeño tamaño donde se han puesto a prueba sensores y actuadores digitales. Se han diseñado los circuitos electrónicos y el software de control pensando en un funcionamiento genérico de esta red.

1.2. Abstract

This project proposes the creation of a home automation network with low consumption, by using the ZigBee wireless protocol. XBee S2 models from the manufacturer Digi are the devices used. A study was made to prove the maximum use of its capabilities, in order to avoid the need of adding microcontrollers in each node, thus reducing its cost, energy consumption and complexity. Because of this, the network control is centralized at the coordinator node, which facilitates the devices' behavior reconfiguration.

The viability of this scheme has been proven by the realization of a small sized network, where some digital sensors and actuators have been tested. The electronic circuits and control software have been designed thinking of a generic behavior of the network.

1.3. Introducción

El presente informe se corresponde al trabajo de fin de grado TFG020 del Grado en Ingeniería Electrónica Industrial y Automática, consistente en el diseño de varios sensores y actuadores domóticos que utilicen comunicación inalámbrica ZigBee. El objetivo del trabajo consiste en crear una red de sensores y actuadores para la posible automatización de una casa, empleando el

protocolo ZigBee de comunicación.

La domótica se define como un conjunto de sistemas que permiten automatizar las distintas instalaciones de una vivienda, aportando a sus inquilinos comodidad, seguridad y comunicación. Históricamente, producto de las más innovadoras obras de ciencia ficción, la domótica se comenzó a extender tras el avance de las tecnologías de la información. Aunque originalmente los proyectos domóticos no eran más que futuristas diseños expuestos en las Ferias Mundiales, la invención y popularización de los microcontroladores redujo significativamente el coste económico y la complejidad de los sistemas, permitiendo aplicarlos en cada vez más hogares y dispositivos.

Los sistemas domóticos se comunican y conectan entre sí, realizando tomas de medidas relevantes para alcanzar las características anteriores, y actuando en función de estas medidas. Normalmente se tiene un nodo central que coordina todas las operaciones que realiza la instalación, y que recibe datos del entorno mediante sensores. Los principales servicios que una instalación de este tipo proporciona se agrupan en cinco tipos:

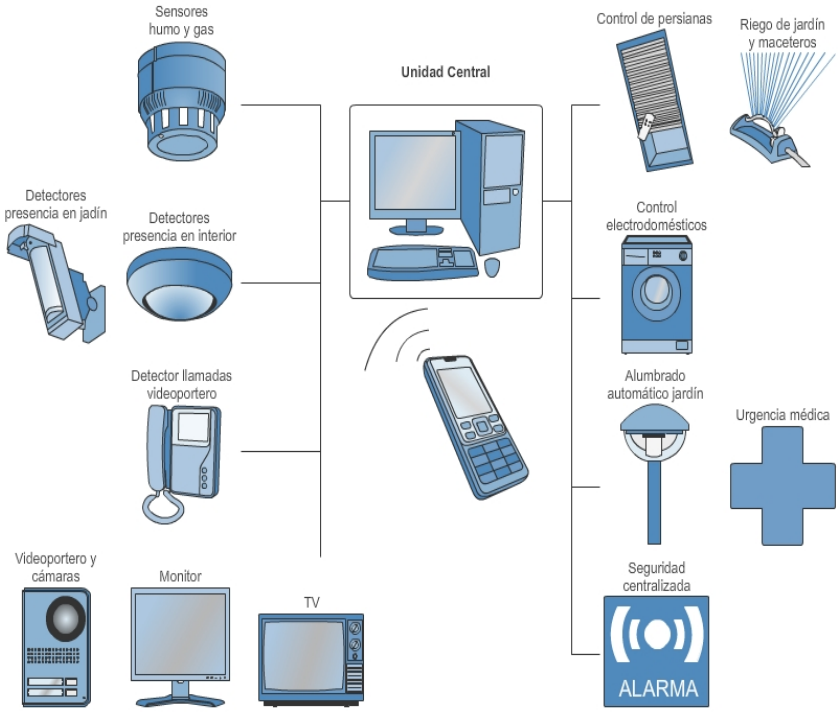

Figura 1. Esquema básico de una red domótica.

- Ahorro energético: mediante un sistema domótico, un hogar puede mantener las condiciones de luminosidad, temperatura o humedad de forma precisa y con el menor gasto de energía posible, además de controlar directamente el consumo de electricidad de la vivienda desconectando los dispositivos que no sean imprescindibles.

- Comodidad y confort: acciones como el encendido y apagado automático de las luces,

interfaces remotas de control de sistemas domésticos o el guardado de configuraciones personalizadas de cualquier parámetro modificable para cada miembro de la casa mejoran la calidad de vida de las personas y repercute positivamente en su bienestar.

- Sistemas de seguridad: alarmas, sistemas de videovigilancia o detectores de incendios o humos integrados en el sistema de la casa pueden no sólo proteger los bienes materiales, sino además a las propias personas, como es el caso de un sistema de alerta médica automatizado para personas de avanzada edad o con dolencias graves.

- Comunicaciones: dado que un sistema de este tipo debe estar continuamente conectado, es fácil integrar y controlar redes para el usuario, ya sea para acceder remotamente al sistema, recibir o proporcionar asistencia y mantenimiento a distancia, o recibir información en tiempo real sobre cualquier instalación de la vivienda.

- Accesibilidad: las personas que padecen algún tipo de incapacidad de cualquier tipo pueden ver su calidad de vida sensiblemente incrementada gracias a sistemas automatizados que les ayuden a poder valerse por sí mismos y asistirles en lo que necesiten.

Figura 2. Control automático de temperatura en una habitación.

El principal inconveniente a día de hoy de este tipo de instalaciones es que puede conllevar un gasto económico y energético considerable, que no todas las familias pueden permitirse. Sin embargo, recientemente se ha comenzado a extender el uso de sistemas y protocolos de comunicación como ZigBee, que el presente trabajo aborda, y que permiten realizar instalaciones de baja complejidad, coste y consumo. El objeto del trabajo consiste en emplear este protocolo, implementado en los

dispositivos Xbee, para simular una instalación domótica empleando el menor número posible de componentes adicionales a las propias radios, como podría ser el caso de un microcontrolador para cada nodo. Se trata en definitiva de poner a prueba las capacidades de estos módulos sin ningún elemento adicional, para comprobar su viabilidad como alternativa de bajo coste a la hora de automatizar cualquier proceso. Concretamente, se han elegido los nodos XBee S2 para formar una red con un nodo central coordinador y varios nodos sensores/actuadores.

2. Conocimientos previos

2.1. Protocolo ZigBee

2.1.1. Introducción

El protocolo de comunicación ZigBee se basa en el estándar IEEE 802.15.4 de comunicación, el cual define el nivel OSI 1 (capa física) y el control de acceso al medio (MAC) de la capa de enlace de datos para redes inalámbricas de área personal con bajas tasas de transmisión de datos. El protocolo ZigBee, partiendo de dicho estándar, desarrolla las capas superiores del modelo OSI y permite la creación de redes inalámbricas en configuración de malla con un bajo consumo de energía.

Figura 3. Utilidades del protocolo ZigBee.

Este protocolo fue concebido a finales del pasado milenio, en 1998, cuando diversos instaladores y empresas del sector concluyeron que tanto el protocolo WiFi como el Bluetooth no podrían emplearse en varias aplicaciones. A raíz de estas necesidades, se fundó la ZigBee Alliance [8], una organización abierta sin ánimo de lucro que agrupaba inicialmente a 25 empresas y compañías, llegando en la actualidad a más de 400, entre las que se incluyen empresas de renombre como Philips, Schneider Electric o Texas Instrument. Esta organización creó y desarrolló el protocolo, específicamente diseñado para las necesidades particulares de redes inalámbricas de bajo coste y baja potencia. Para usos no comerciales, la especificación ZigBee es gratuita para el público general, y cualquier empresa que entre al nivel más básico de la alianza, los llamados *Adopters,* tienen el derecho a acceder a especificaciones todavía no lanzadas al público y crear y comercializar productos usando este protocolo.

Así, en el mercado se pueden encontrar diversos productos de muy variadas marcas y finalidades [9], con distintos protocolos (propios o versiones de ZigBee) pero todos basados en el IEEE

802.15.4. Estos dispositivos pueden ir desde pequeños transceptores como el EasyBee de FlexiPanel Ltd. o el pequeño circuito integrado MRF24J40MA de Microchip Technology, hasta dispositivos mayores que incluyen un microcontrolador, como los módulos ZigBit 2.4 GHz de Atmel o el PAN4570 de Panasonic. La elección de uno u otro dependerá de las características que requiera el proyecto en el que se necesite.

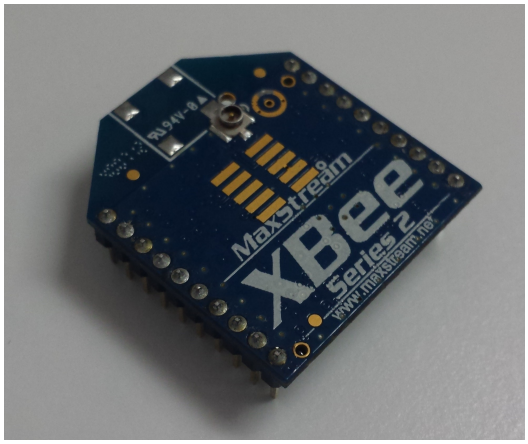

Figura 4. Módulo XBee Series 2.

Las radios que se emplearán en este trabajo, debido a que sus características y especificaciones se adecúan a los requerimientos del trabajo, y dada su popularidad y extensión, serán los módulos Xbee de Digi International. Esta empresa, una de las marcas miembro de la alianza, fabrica dispositivos para redes inalámbricas cuyas radios no sólo toleran ZigBee sino otros protocolos de comunicación tales como WiFi. La marca de los módulos empleados es MaxStream, empresa que fue comprada por Digi en 2006. Los módulos XBee de la serie 1 soportaban directamente el estándar IEEE en su forma nativa, pero a partir de la serie 2 (como las radios que se emplearán en este trabajo) se comenzó a implementar el protocolo ZigBee, lo que permite que la serie 2 tenga nuevas funcionalidades además de las básicas que otorga el uso del estándar, como las siguientes [1]:

- Tablas de ruteo para definir cómo enviar mensajes entre dos radios a través de una serie de nodos intermedios.
- Capacidad de creación de redes Ad hoc de forma automatizada y sin intervención humana
- Autorreparación, esto es, en caso de que uno o más nodos no estén en la red, ésta se reconfigura para reparar cualquier ruta rota de forma automática.

2.1.2. Características del protocolo

Las radios toleran de forma nativa redes en estrella o árbol, así como topología en malla. Esto permite al sistema la transmisión de datos entre largas distancias sin necesidad de un controlador central capaz de conectarse a todos los miembros de la red. El protocolo opera en varias bandas de

frecuencias de radio ISM (reservadas para el uso industrial, científico y médico), dependiendo del lugar geográfico y de las necesidades de velocidad de transmisión. A su vez, cada banda tiene varios canales distintos habilitados para su uso. Todos estos datos de la capa física del protocolo se definen en el estándar IEEE [2], el cual ha sido objeto de varias reformas para mejorar estas características. Así, se pueden tener transmisiones con velocidades desde 20 kb/s a 868 MHz hasta 250 kb/s en la banda de 2.4 GHz.

Una red ZigBee cuenta siempre con hasta tres tipos de dispositivos [1]:

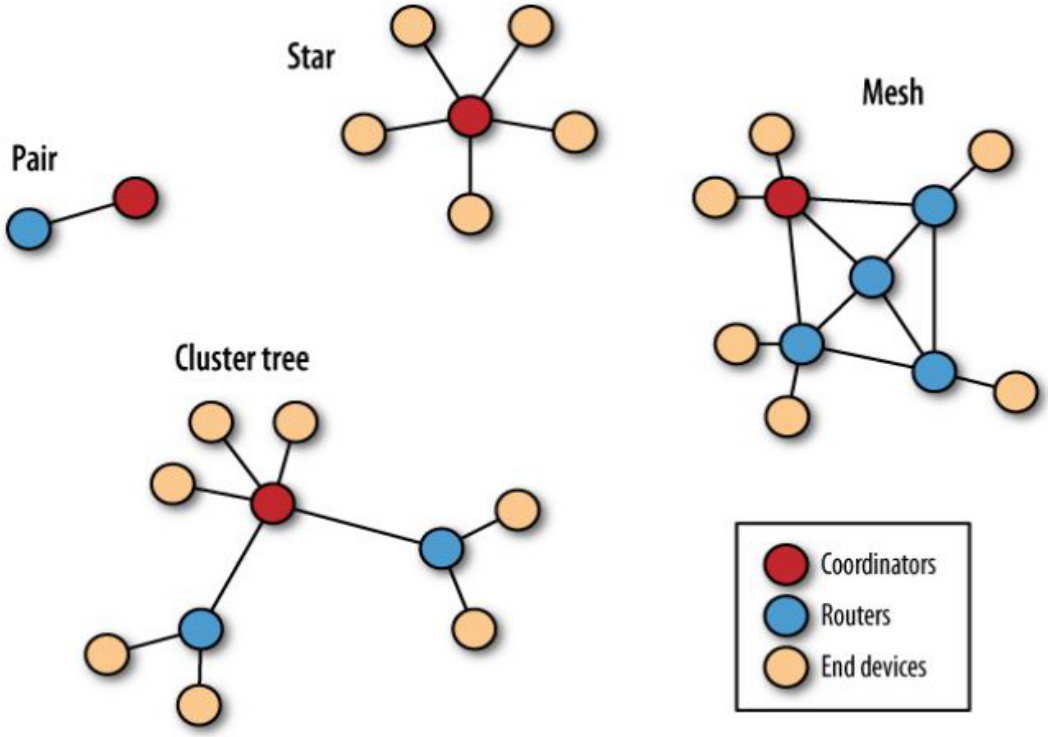

Figura 5. Esquema de los distintos miembros de la red y las topologías posibles.

- Un único coordinador que se encarga de crear la red, administrar las direcciones y llevar a cabo funciones de seguridad y mantenimiento de la red.

- Routers, nodos con plenas capacidades de funcionamiento. Pueden unirse a redes existentes, enviar y recibir información o pasarla hacia otros destinos, permitiendo que se pueda establecer una comunicación fiable entre otros dos puntos que estén a demasiada distancia como para conectarse directamente. Un mensaje se va transmitiendo de uno a otro hasta llegar al destino final. Para cumplir con estas funciones los nodos que sean routers deben

estar siempre encendidos y disponibles, por lo que suelen estar directamente alimentados mediante una toma de corriente.

- Dispositivo final, un nodo router con funciones reducidas. Estos dispositivos actúan como terminaciones de una red, y sus capacidades incluyen unirse a redes y enviar o recibir información. Como no necesitan estar continuamente en funcionamiento, requieren un hardware más barato y con menor consumo energético. Para esto, los dispositivos finales pueden entrar periódicamente en modo de ahorro de energía o modo sueño. Todo dispositivo final necesita un router o coordinador como dispositivo padre, el cual ayuda a los dispositivos a unirse a las redes y almacena los mensajes dirigidos a ellos en caso de que estén en estado de hibernación.

Este protocolo tolera direccionado de dispositivos y direccionado en la capa de aplicación [3]. El direccionado de dispositivos especifica la dirección del dispositivo de destino para el cual se destina un paquete de datos, mientras que el direccionado en la capa de aplicación sirve para indicar un receptor particular de aplicación, junto con un campo de texto llamado *Cluster ID*.

El direccionamiento de dispositivos consta de dos direcciones:

- Dirección de red de 16 bits: se asigna a un nodo cuando éste se une a una red. El protocolo requiere que los datos se envíen a esta dirección, lo que obliga a conocerla antes de poder comunicarse. Esta dirección es única para cada nodo de la red, pero es variable, puede cambiar si se produce una de estas situaciones:

    1. Si un dispositivo final no se puede comunicar con su padre, necesita abandonar la red y reconectarse para encontrar un nuevo padre.

    2. Si el tipo de dispositivo cambia de router a dispositivo final o viceversa, este debe desconectarse de la red y volverse a unir como un dispositivo del nuevo tipo.

- Dirección de 64 bits: cada nodo posee una dirección permanente de 64 bits única y fijada de fábrica, la cual identifica completamente al dispositivo.

A pesar de esto, el protocolo incluye la funcionalidad para emitir mensajes en modo *broadcast*, de forma que cualquier dispositivo que esté conectado reciba el mismo mensaje. También se puede

realizar un *multicast* o direccionado a un grupo.

Del mismo modo, cada red o PAN (*Personal Area Network*) creada tendrá su propia dirección [1] para diferenciarla de otras redes. Todas estas direcciones, junto con el canal de frecuencia correcto, permiten identificar a, y comunicarse con, cualquier elemento miembro de una PAN en un conjunto de varias redes coexistiendo en un mismo entorno, sin interferir las unas con las otras.

2.1.3. Alternativas a ZigBee

Existen otras alternativas al protocolo ZigBee en el mercado, aplicables tanto a nivel doméstico como industrial. Las alternativas para domótica se resumen en la Tabla 1 [10]:

Tabla 1. Protocolos de comunicación inalámbrica en un entorno no industrial

Característica	ZigBee	Wireless USB	Z-Wave	ISM genérico
Rango de frecuencias	2.4 – 2.483 GHz	2.4 – 2.483 GHz	868.42 Mhz (en Europa)	135 – 650, 431 – 478, 862 – 956 MHz
Máximo ratio de transferencia	250 kb/s	62.5 kb/s	9.6 kb/s	Entre 25 y 384 kb/s
Máximo rango TX/RX	Entre 10 y 50 metros	Entre 10 y 50 metros	Más de 100 metros	Más de 100 metros
Sensibilidad RX típica	-87 a -98 dBm	-97 dBm	-104 dBm	-112 a -117 dBm
Potencia TX típica	0 a +5 dBm	+4 dBm	-20 a +5 dBm	+13 dBm
Capacidad de formar mallas	Sí	No	Sí	No

Wireless USB, de Cypress Semiconductor, es un protocolo útil para aplicaciones punto a punto (P2P) y multipunto a punto (MP2P), diseñada principalmente para periféricos de ordenador (ratones y teclados inalámbricos), aunque también es útil para juegos, juguetes, mandos a distancia y otras aplicaciones potenciales como electrónica de consumo, domótica, entretenimiento doméstico o sistemas de monitorización de salud. Al actuar como un dispositivo de interfaz humana, la conectividad inalámbrica es completamente transparente para el diseñador, y no requiere de drivers.

Una alternativa reciente es Z-Wave, de Zensys. Principalmente enfocado a domótica, específicamente al control y monitorización de sistemas domésticos, es menos flexible pero más económico que ZigBee. Opera en la banda UHF de frecuencias, que le otorga un buen rango, pero su velocidad de transmisión es baja. Se apoya en otra organización grupal, la Z-Wave Alliance, que cuenta con más de 125 miembros, además de inversiones por parte de Intel y Cisco. Si bien es una alternativa bastante posible, se ha optado por el protocolo ZigBee por tener mucho más material de apoyo y estudios sobre él, al ser un protocolo más veterano.

A efectos comparativos, se añade las características genéricas básicas de un dispositivo transceptor de banda ISM (industrial, científica y médica). Adicionalmente, si bien WiFi y Bluetooth también son protocolos inalámbricos, no se enfocan en control y monitorización de sistemas, siendo excesivamente potentes y sobrecualificados para las necesidades del presente trabajo. Existen otros protocolos y sistemas para redes domóticas que no son inalámbricos pero que también se emplean con cierta frecuencia, como el estándar KNX. Aunque sus características pueden ser comparables o incluso superiores en rendimiento a estos protocolos inalámbricos, al emplear cableado son menos prácticas y no son el objeto de este trabajo.

En entornos industriales el protocolo ZigBee también es empleado, aunque en este tipo de aplicaciones puede sufrir problemas debido a la gran cantidad de fuentes de ruido e interferencias. Las alternativas a ZigBee en entornos industriales se muestran en la Tabla 2 [11]:

Tabla 2. Protocolos de comunicación inalámbrica en entornos industriales			
Protocolo	Descripción	Ventajas	Desventajas
ZigBee	Estándar IEEE 802.15.4 para redes en malla enfocado al control y monitorización, domótica y automatismo de sistemas de energía.	Muy bajo consumo energético, soporte de diversas topologías.	No puede trabajar con un elevado número de nodos en el tiempo de ciclo especificado.
Wireless HART	Extensión de protocolo HART diseñado específicamente para control y monitorización. Añadido al protocolo HART general como parte de la especificación HART 7.	Empleo de estándar IEEE 802.15.4, salto de frecuencias, caminos de datos redundantes, mecanismos de reintento, redes en malla.	Banda de frecuencias ya usada por otros protocolos, se traduce en menor fiabilidad.
Ultrawideband (UWB)	Protocolo de corto alcance basado en la transmisión de impulsos muy cortos emitidos en secuencias periódicas. Sus aplicaciones incluyen redes personales y multimedia.	Buena capacidad de localización, capacidad de compartir bandas previamente dedicadas escondiendo las señales bajo el ruido, alta transmisión de datos con bajo consumo, seguridad.	No viable para largas distancias o para medir datos de zonas no seguras debido a los altos picos de energía de los pulsos.
IETF 6LoWPAN	Su objetivo es la implementación de comunicación IP estándar a través de redes IEEE 802.15.4 de bajo consumo empleando IPv6.	Comunicación directa con otros dispositivos IP locales o por redes IP, arquitectura y seguridad ya existentes, modelo y servicios de nivel de aplicación ya establecidos, soporte IP ya extendido.	Complejidad de la trama IP.
Bluetooth Low Energy	Parte de la especificación Bluetooth como una tecnología de ultrabajo consumo para dispositivos de direccionamiento con muy poca batería.	Transmisión de datos de hasta 1 Mb/s a una distancia de entre 5 y 10 m en la banda de 2.45 Ghz.	Paquete de longitud variable distinto al Bluetooth estándar, diferente esquema de modulación.

2.1.4. Redes PAN en el protocolo ZigBee

Las redes en el protocolo ZigBee [3] se crean cuando un coordinador selecciona un canal y una ID de red, aunque también se cuenta con la funcionalidad para emplear un canal y una ID de PAN preferida, que se configurará previamente por el usuario. Cuando esto se ha completado, los routers y dispositivos finales pueden empezar a unirse a la PAN, llegando a formar parte de la misma y heredando la ID de red. Todos ellos reciben una dirección de 16 bits dentro de la PAN, que es siempre 0 para el coordinador.

El protocolo soporta ruteo en malla, que permite a su vez que, si es necesario, los paquetes de datos se transmitan o "salten" por múltiples nodos para poder alcanzar el destino. Como ya se explicó anteriormente, esto permite crear redes de gran extensión pero manteniendo la comunicación entre todos los dispositivos.

El coordinador, al ser el responsable de crear la red, debe estar presente inicialmente al crearla. Para ello, el coordinador realiza una serie de escaneos para comprobar el nivel de actividad de radiofrecuencia en distintos canales, además de descubrir cualquier red PAN cercana que esté en funcionamiento. El escaneo consiste en medir los niveles de energía en distintas frecuencias, descartando los que se detecten con excesivo nivel de energía de la lista de potenciales canales para iniciar una red.

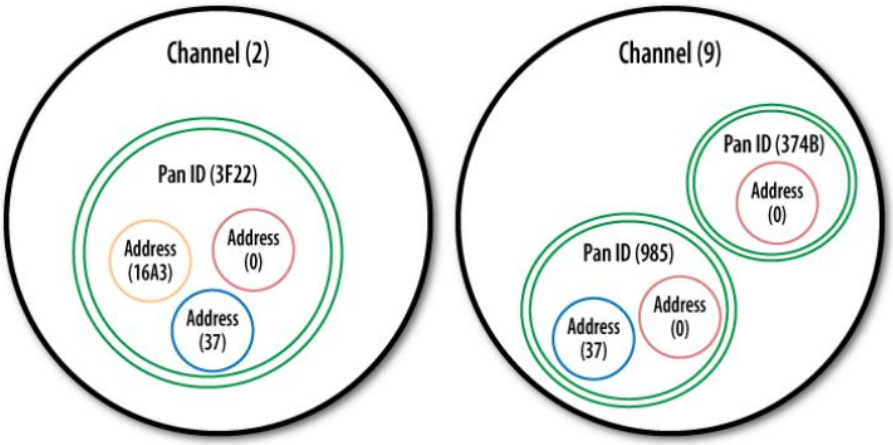

Figura 6. Diagrama de Venn que describe el direccionamiento en redes PAN [1].

Cuando este primer escaneo se completa, el coordinador revisará los canales no descartados en busca de redes PAN ya existentes. Esto se realiza enviando una solicitud de baliza de un único salto en modo *broadcast*. Cualquier coordinador o router cercano responderá enviando una trama de baliza que contiene información acerca de la red PAN en la que está el emisor, incluyendo la ID de

la red, y si el dispositivo admite o no conexiones. Cuando el coordinador ya ha completado la búsqueda, analiza todas las balizas recibidas e intenta iniciar una red en un canal y con una ID sin usar. Si la consigue crear, entonces permitirá a routers y/o dispositivos finales unirse a la red. El coordinador mantendrá el canal y la ID de la PAN incluso tras reinicios.

Para unirse a la PAN, los routers o dispositivos finales primero deben descubrirla, para lo cual realizan un escaneo exactamente igual al que realiza el coordinador al intentar crear una red. Como resultado de este escaneo, reciben una serie de balizas de otros dispositivos ZigBee cercanos, que serán analizados para poder encontrar una red válida para unirse. Si se encuentra alguna, intentarán realizar la conexión enviando una solicitud de asociación al dispositivo al que desean conectarse. Los routers y los dispositivos finales se pueden configurar para poder unirse a cualquier red PAN o sólo a una red PAN con una ID concreta, pero en cualquier caso primero necesitarán localizar a un coordinador o router de la misma que permita conexiones.

El hecho de que un coordinador o router admita o no conexiones dependerá de dos datos: de su atributo de permiso de unión y del número de dispositivos hijos que ya tiene conectados. En cuanto al primer dato, todo coordinador o router que se quiera que pueda recibir conexiones debe tener el atributo correspondiente configurado para tal fin. Este atributo puede estar de modo que:

- siempre admita uniones.
- que las admita durante un corto periodo de tiempo.
- que no admita ninguna conexión más.

En cuanto al segundo dato, como los dispositivos finales dependen de su padre para almacenar paquetes de radiofrecuencia entrantes, estos sólo pueden administrar un número finito de hijos, que si se alcanza hará que el dispositivo no permita más conexiones.

Por último, si las medidas de seguridad están activas, el coordinador empezará usando una clave de encriptación AES (*Advanced Encryption Standard*) de 128 bits. Sólo los dispositivos que cuenten con la misma clave podrán comunicarse en la PAN, por lo que los candidatos a unirse deben obtener dicha clave. Esta puede ya venir preinstalada o ser recibida durante la unión.

2.2. XBee Series 2

2.2.1. Introducción: ¿por qué XBee?

Los módulos XBee son la opción escogida para realizar la implementación de una red de sensores y actuadores domóticos. Estos módulos son perfectos para cubrir las necesidades del trabajo, ya que como se describirá con mas detalle a continuación, cuentan con la capacidad de recoger directamente información de sensores tanto digitales como analógicos, y transmitirla a otros nodos sin necesidad de emplear microcontroladores adicionales. Asimismo, también poseen salidas digitales y de PWM que les permiten realizar actuaciones básicas por sí solos. Las ventajas que aportan estas características son claras:

- Menor tamaño y complejidad, lo que les permite ser empleados en una gran variedad de situaciones y en espacios muy reducidos. El peso también se ve mermado, por lo que puede emplearse incluso para sensores que se puedan llevar en la ropa o en el cuerpo.
- Consumo de energía mucho más reducido.
- Ahorro económico al emplear menor número de componentes.

Sin embargo, no emplear microcontroladores también tiene sus inconvenientes, principalmente que no tiene acceso a ningún tipo de lógica, limitándose simplemente a enviar y recibir datos o cambiar remotamente el estado de sus pines. Es decir, la lógica debe implementarse en un dispositivo (microcontrolador o superior) conectado normalmente al nodo coordinador. Si en ese momento falla la conexión, el nodo dejará de actuar como se espera. Además de esto, no se puede incrementar el número de entradas o salidas disponibles. A pesar de ello, para las necesidades de una red doméstica sencilla, el XBee es una opción muy interesante y versátil.

2.2.2. Características principales

Algunas de las características principales de los dispositivos XBee Series 2, tanto en su versión normal como el modelo PRO, de mayor rendimiento [3], se resumen en los siguientes puntos:

- Alto rendimiento con bajo coste. El rango de alcance en entornos interiores o urbanos es de hasta 40 m, alcanzando los 120 m en exteriores y con línea de visión directa. El XBee PRO tiene un rango mayor (unos 100 metros en interior y hasta 1.6 km en exterior y con línea

directa de visión).

- Bajo consumo energético. El módulo XBee tiene un consumo máximo de corriente de 40 mA a 3.3 V. Si no está ejecutando ninguna tarea pero está encendido, su consumo es de 15 mA, pero si se configura para entrar en modo de hibernación, el consumo en dicho modo es menor de 1 µA. El módulo PRO tiene un consumo mayor de energía debido a sus mejoradas prestaciones.

Figura 7. Modelo XBee S2 PRO, con características mejoradas.

- Creación avanzada de redes y alta seguridad. El sistema de interconexión cuenta con procesos de reintento y confirmación de recepción, múltiples direcciones disponibles, admite topologías punto a punto, estrella y *peer to peer* y tiene la capacidad de autorutear, autorreparar y tolerar fallos.

- Fácil de usar. Cuenta con dos modos de funcionamiento (transparente y API) para enviar órdenes AT y configuraciones. Tiene un amplio catálogo de comandos y cuenta con un software gratuito de configuración y testeo de los dispositivos, denominado X-CTU. Además, funciona con frecuencias estandarizadas y se ha manufacturado bajo el estándar ISO 9001:2000.

Otras especificaciones destacables son las siguientes:

- Ratio de transmisión de datos seriales variable por software entre 1200 y 230400 bps.

- Temperaturas de operación comprendidas entre los -40ºC y los 85ºC.

- Cumplimiento de la normativa RoHS de tratamiento de sustancias peligrosas en dispositivos electrónicos.

La radio cuenta con un firmware que permite especificar al dispositivo qué rol va a cumplir en la red, así como una serie de parámetros de configuración que recogen todas las funciones y opciones. Estas opciones incluyen la definición de parámetros como la dirección de 16 bits, el alias del

dispositivo, el estado de los pines, muestreos de señales y muchas otras características del circuito integrado. El cambio en estos parámetros puede realizarse mediante el software X-CTU o mediante mensajes recibidos por el módulo tanto por conexión serial directa como a través de la propia red.

2.2.3. Descripción física

El dispositivo que se empleará en este trabajo será una radio XBee ZNet 2.5 OEM RF Module, anteriormente conocida como Serie 2. Se trata de un pequeño circuito integrado de 2.5 x 2.8 cm, con 20 pines de inserción y tamaño de paso de 2 mm. Implementa el protocolo ZigBee y se emplea principalmente para proyectos con la necesidad específica de una red inalámbrica sensorial de bajo consumo y coste. Opera en la banda de frecuencias ISM de 2.4 GHz y se alimenta mediante una fuente de tensión continua comprendida entre los 2.1 V y los 3.6 V. Su funcionamiento nominal es a 3.3 V.

Existe una versión superior en cuanto a capacidades de potencia entregada a la salida, alcance o sensibilidad, entre otros, de nombre XBee PRO ZNet 2.5 OEM RF Module. Esta versión por ejemplo tiene un voltaje de entrada con un menor rango de posibles valores (sólo entre 3.0 V y 3.4 V), pero cuenta con una salida de potencia de 50 mW comparados con los 2 mW que entrega la versión normal. Debido a las diferencias económicas entre ambos y a los escasos requerimientos del presente trabajo, se ha optado por trabajar con el módulo básico XBee 2.5.

El XBee cuenta con 20 pines, cuyas funciones se describen a continuación en la Tabla 3. La información se corresponde tanto al módulo XBee básico como al XBee PRO. Las señales activas a nivel bajo aparecen suprarrayadas.

Tabla 3. Asignación de pines para el XBee PRO ZNet 2.5			
Nº pin	Nombre	Tipo	Descripción
1	VCC	-	Fuente de alimentación
2	DOUT	Salida	Salida de datos UART
3	DIN / $\overline{\text{CONFIG}}$	Entrada	Entrada de datos UART
4	DIO12	Ambos	E/S digital 12
5	RESET	Entrada	Reseteo módulo (pulso mín. 200 ns)
6	PWM0 / RSSI / DIO10	Ambos	PWM salida 0 / Indicador fuerza señal RSSI / E/S digital 10
7	PWM / DIO11	Ambos	E/S digital 11
8	[reservado]	-	No conectar
9	$\overline{\text{DTR}}$ / SLEEP_RQ / DIO8	Ambos	Control de hibernación por pin/ E/S digital 8
10	GND	-	Tierra
11	DIO4	Ambos	E/S digital 4
12	$\overline{\text{CTS}}$ / DIO7	Ambos	Control flujo Clear-to-send / E/S digital 7
13	ON / $\overline{\text{SLEEP}}$ / DIO9	Salida	Indicador de estado del módulo / E/S digital 9
14	[reservado]	-	No conectar
15	Associate / DIO5	Ambos	Indicador asociado / E/S digital 5
16	$\overline{\text{RTS}}$ / DIO6	Ambos	Control flujo Request-to-send / E/S digital 6
17	AD3 / DIO3	Ambos	Entrada analógica 3 / E/S digital 3
18	AD2 / DIO2	Ambos	Entrada analógica 2 / E/S digital 2
19	AD1 / DIO1	Ambos	Entrada analógica 1 / E/S digital 1
20	AD0 / DIO0 / Commissioning Button	Ambos	Entrada analógica 0 / E/S digital 0 / Botón de comisionado

El dispositivo requiere 4 conexiones para poder funcionar: VCC, GND, DOUT y DIN, siendo estos dos últimos los pines de conexión serial a otro dispositivo. Para poder actualizar el firmware, en cambio, se necesitan conectar los siguientes pines: VCC, GND, DIN, DOUT, RTS y DTR.

2.2.4. Comunicación entre dispositivos

Como ya se introdujo anteriormente, los módulos XBee ZNet 2.5 se pueden comunicar con un dispositivo anfitrión a través de un puerto serial asíncrono. A través de este puerto, se pueden

conectar a una UART compatible en voltaje y lógica, o bien a un dispositivo serial mediante un traductor de niveles.

Los módulos cuentan con buffers para acumular datos seriales recibidos y datos de radiofrecuencia. El buffer de recepción serial acumula caracteres seriales recibidos y los mantiene hasta que sean procesados, mientras que el buffer de transmisión serial hace lo propio con los datos recibidos vía enlace de radiofrecuencia, que serán enviados por la UART. Mediante los pines activos a nivel bajo CTS y RTS se puede mantener un control del flujo de datos, ya que permiten activar o desactivar los buffers. Así, si está activo el control de flujo CTS y si el buffer de recepción serial está a 17 bytes de llenarse, se cambiará el estado del pin a nivel alto, lo cual indica al dispositivo anfitrión que debe dejar de enviar datos. Sólo si el buffer pasa a tener 34 o más bytes libres es cuando el pin CTS volverá a su estado inicial. Por otro lado, con el control de flujo RTS activo, los datos almacenados en el buffer de transmisión serial no se enviarán mientras el pin RTS se encuentre a nivel alto.

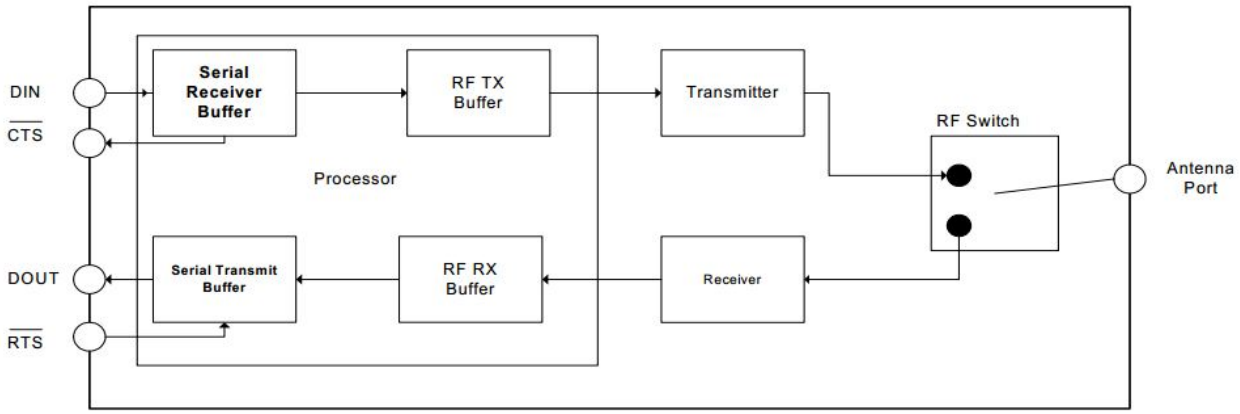

Figura 8. Diagrama de flujo de datos interno [3].

Estos módulos XBee, para comunicarse con un dispositivo anfitrión a través de sus pines seriales, soportan dos protocolos de comunicación: transparente y API (*Application Programming Interface*), que se comentarán a continuación. Es importante, no obstante, destacar que el protocolo que emplean los módulos XBee para comunicarse entre sí dentro de las redes que crean es siempre el mismo.

2.2.5. Modos de operación

Mientras esté encendido, el módulo XBee estará en uno de los cinco estados siguientes:

- Estado desocupado: cualquier momento en el que la radio no esté ni transmitiendo ni recibiendo datos. En este estado, el dispositivo está comprobando la red por si recibe datos de radiofrecuencia válidos. Cambiará a alguno de los otros cuatro estados en función de que se desencadene un suceso concreto.

- Estado de transmisión: el módulo entrará en este estado e intentará realizar una transmisión cuando los datos seriales contenidos en el buffer de recepción serial están listos para ser empaquetados. La dirección de destino indicará qué nodo o nodos recibirán dichos paquetes. Previamente, el dispositivo se asegurará de que el ruteo hacia la dirección especificada está establecido, realizando las operaciones necesarias para averiguar esta dirección o el ruteo en caso de que no se tengan. El paquete se descartará si no se consigue establecer esta conexión.

  El nodo receptor envía un mensaje de confirmación de vuelta al nodo emisor, que indicará al mismo que se ha recibido correctamente el paquete. Si dicho mensaje de confirmación no se ha recibido, el emisor volverá a transmitir el paquete. Se puede dar la circunstancia de que a pesar de que se ha recibido el paquete no se envíe un mensaje de recepción, lo que causaría que el emisor volviese a enviar los datos. Esto puede dar problemas porque el XBee no incluye una funcionalidad para filtrar paquetes de datos duplicados.

- Estado de recepción: si se reciben datos de RF válidos a través de la antena, el módulo pasará a este estado, en el cual los datos se envían al buffer de transmisión serial.

- Estado de comandos: cuando se envía la secuencia del modo comando, el módulo pasará a interpretar los caracteres seriales entrantes como comandos, lo cual permitirá modificar y/o leer parámetros del módulo. Estos comandos se pueden enviar en modo transparente o en modo API, y se detallará en el apartado correspondiente del presente informe.

- Estado de hibernación: este estado, como ya se vio anteriormente, sólo es válido para dispositivos finales. El XBee ZNet 2.5 puede hibernar durante un ciclo periódico definido en

los parámetros de configuración, o puede ser activado o desactivado mediante transiciones en uno de los pines del dispositivo.

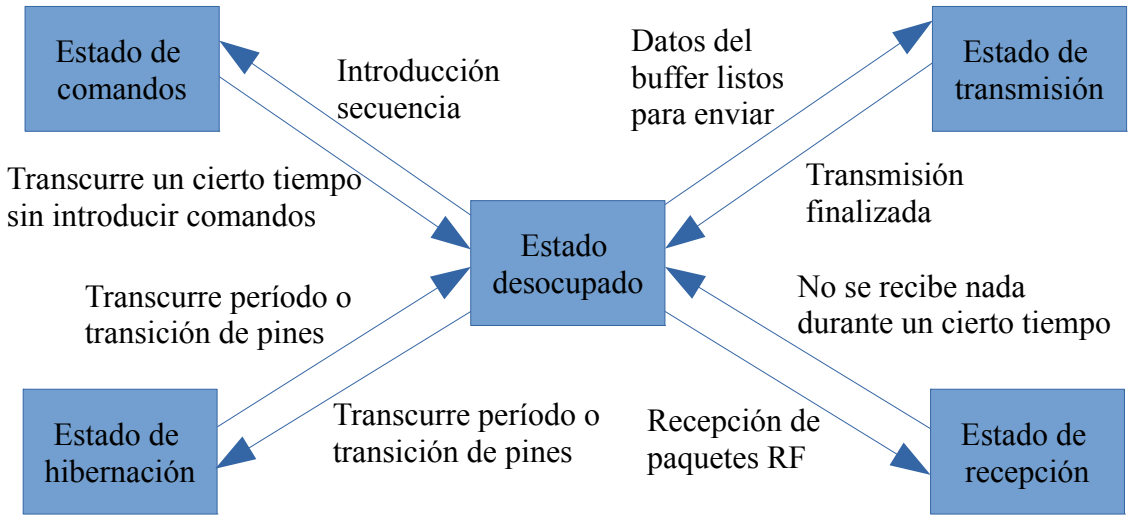

Figura 9. Diagrama de flujo de los estados del XBee.

2.2.6. Protocolo transparente

Los módulos se comportan como si se tratase de una línea serial. Cualquier dato de un UART recibido a través del pin DIN se pone en la cola para ser transmitido vía radiofrecuencia al nodo de destino configurado en los parámetros de configuración, y del mismo modo, si se recibe cualquier dato de radiofrecuencia proveniente de dicho nodo, se enviará directamente por el pin DOUT. Los datos recibidos se almacenarán en un buffer serial hasta que sean enviados, que ocurrirá en caso de que no se reciban más datos durante el tiempo especificado (timeout) en los parámetros de configuración, que se llegue al máximo de caracteres que pueden enviarse en un único paquete, o que se reciba una secuencia de comandos específica que activa la transmisión. Sólo los módulos que tengan las versiones de firmware 1.0xx (si es coordinador) o 1.2xx (si es router o dispositivo final) podrán trabajar en modo transparente.

2.2.6.1. Modo de comandos

El modo transparente también permite al usuario enviar comandos AT al dispositivo. Los comandos AT son órdenes introducidas como caracteres desde un terminal, que permiten leer y modificar los parámetros de configuración y otros datos del módulo XBee al que está conectado. Su funcionamiento es similar al del conjunto de comandos Hayes empleado para comunicarse con

módems antiguamente. Teniendo un dispositivo configurado para que emplee el protocolo transparente, la forma predeterminada de pasar a usar el modo de comandos es introducir por terminal la secuencia "+++". Existen parámetros que permiten modificar tanto el carácter que se debe introducir tres veces consecutivas, como el tiempo de espera antes, durante y después de la introducción de dicha secuencia, y son modificables mediante estos propios comandos AT. Al entrar al modo de comando, se inicia un temporizador de *timeout* que contará un tiempo especificado en dichos parámetros de opciones, saliendo de este modo en caso de que transcurra este tiempo.

En cualquier caso, si la secuencia se completa satisfactoriamente, el módulo responderá enviando el mensaje "OK" a través del pin DOUT. Puede ocurrir que dicho mensaje no llegue en el momento esperado, ya que si el módulo se encuentra todavía transmitiendo o recibiendo datos seriales, se produce un retardo al enviarlo. Además, en ocasiones se pueden producir errores que impidan entrar al modo de comandos, casi siempre producidos por una discrepancia entre las velocidades de transmisión del módulo XBee y el dispositivo. El valor por defecto de esta velocidad de transmisión es de 9600 bps.

Todos los comandos AT comienzan por las letras "AT" (Attention), seguidos de dos caracteres que indican el comando que se desea emplear y, de necesitarlos, otros parámetros del comando. La sintaxis general es la siguiente, donde el retorno de carro (*carriage return*) corresponde a presionar la tecla ENTER:

Prefijo "AT" + comando ASCII de dos caracteres + parámetro (opcional) + retorno de carro

Ejemplo: ATDT1F<CR>

Algunos de los comandos AT más empleados o más relevantes se recogen en la Tabla 4:

Tabla 4. Comandos AT del modo de comandos [3]			
Comando	Nombre y descripción	Rango de parámetros	Valor predeterminado
WR	Escribir. Escribe parámetros a la memoria no volátil, conservando los cambios a pesar de reseteos posteriores.	-	-
FR	Reseteo de software. Responde con un “OK” y tras dos segundos resetea el módulo.	-	-
DH	Dirección de destino alta. Establece (o muestra si no se pasa ningún parámetro) los 32 bits más significativos de la dirección de destino empleada para la transmisión.	0 - 0xFFFFFFFF	0
DL	Dirección de destino baja. Establece o muestra los 32 bits menos significativos de la dirección de destino empleada para la transmisión. Se suele emplear en conjunto con DH.	0 – 0xFFFFFFFF	0xFFFF ó 0
MY	Dirección de red de 16 bits. Recibe la dirección de red de 16 bits del dispositivo.	0 – 0xFFFE (sólo lectura)	0xFFFE
NI	Identificador de nodo. Almacena un identificador en forma de cadena de caracteres. El registro sólo acepta caracteres ASCII imprimibles. Un <CR> o llegar al límite de caracteres termina el comando.	Cadena de caracteres ASCII de 20 bits	Espacio de caracteres ASCII (0x20)
ID	ID de la PAN. Permite fijar o leer la ID de la PAN.	0 – 0x3FFF, 0xFFFF	0x0234
ND	Descubrir nodo. Descubre e informa de todos los módulos de RF descubiertos. Se informa de la dirección de 64 bits, la de red de 16 bits, el identificador de nodo, el tipo de dispositivo, etc.	Valor de NI o MY opcional	-
Dx	Configuración de la E/S digital x. Selecciona o lee las opciones de la E/S digital especificada. Cada una puede tener sus opciones exclusivas.	Depende de la E/S	Depende de la E/S

IC	Detección de cambios en E/S digitales. Establece o lee los pines digitales a monitorizar para detectar cambios en el estado lógico, emitiendo un mensaje cuando suceda. IC es una máscara de bits que permite especificar los pines a monitorizar.	0 – 0xFFFF	0
CT	Timeout de modo de comandos. Fija o muestra el periodo de inactividad tras el cual el módulo sale automáticamente del modo de comandos AT y regresa al estado desocupado.	2 – 0x028F (x100 ms)	0x64 (100d, 10 segundos)

2.2.7. Protocolo API

Alternativa al modo transparente que, mediante el uso de un API (interfaz de programación de aplicaciones) basada en tramas, es capaz de ampliar el nivel de interacción de una aplicación anfitriona con las capacidades de red del módulo. En el modo API, todos los datos entrantes o salientes se agrupan en tramas que definen operaciones o eventos dentro del módulo. Una aplicación externa puede enviar tramas de datos que contienen la dirección y otra información necesaria al módulo, en lugar de utilizar el modo de comandos para modificar direcciones. El módulo responderá enviando otra trama con paquetes de estado entre otros datos. Las versiones de firmware que toleran el modo API son la 1.1xx para el coordinador y la 1.3xx para los routers y dispositivos finales.

Las ventajas de este modo frente al modo transparente son muy numerosas:

- Modos adicionales a los comandos AT, como la recepción de mensajes de estado de entradas y salidas digitales de uno o más nodos remotos, permitiendo crear una red de sensores que transmitan información sobre variables.
- Configuración a distancia de nodos mediante comandos AT remotos.
- Direccionamiento de mensajes mucho más rápido y capacidad de envío de mensajes en modo *broadcast*.
- Respuestas a paquetes enviados para tener confirmación de la correcta recepción de los mismos, y posibilidad de reintentar el envío del paquete en caso de no recibir el mensaje de

respuesta.

- CRC para determinar la integridad de los datos recibidos y proteger frente a fallos y perturbaciones.

Estas y muchas otras funcionalidades son el motivo de que sea el modo API el escogido para realizar la comunicación con el coordinador en este trabajo, ya que se requiere de funciones que el modo transparente no es capaz de proporcionar.

La estructura general de una trama API es la mostrada en la Figura 10:

Figura 10. Estructura de las tramas API [3].

Delimitador de inicio + Longitud + Estructura de datos específica de cada API + CRC

El CRC o *ckecksum* es un byte añadido al final del paquete que permite mantener la integridad de los mensajes. Se calcula restando al valor 0xFF la suma de todos los bytes del paquete, sin contar la longitud y el encabezado. La verificación de integridad se realiza sumando dichos bytes y comprobando que el resultado da 0xFF. En caso contrario, el paquete no se procesa.

El campo de longitud ocupa dos bytes y especifica el número de bytes que se contienen en la estructura de datos específica, sin contar el CRC. Esta estructura específica comienza con la ID del comando a ejecutar. Los distintos tipos de paquetes que se pueden trabajar en el modo API se identifican con esa ID, y todos los posibles IDs se recogen en la Tabla 5:

Tabla 5. Nombres y valores de las tramas API	
Nombre de la trama	Valor
Estado del módem	0x8A
Comando AT	0x08
Comando AT – Poner en cola valor del parámetro	0x09
Respuesta a comando AT	0x88
Solicitud de comando remoto	0x17
Respuesta a comando remoto	0x97
Petición de transmisión ZigBee	0x10
Trama de comando de direccionamiento explícito de ZigBee	0x11
Estado de transmisión de ZigBee	0x8B
Recepción de paquete ZigBee(AO=0)	0x90
Indicador de Rx explícito ZigBee (AO=1)	0x91
Indicador de muestreo de datos Rx de E/S ZigBee	0x92
Indicador de lectura de sensor XBee (AO=0)	0x94
Indicador de identificación de nodo (AO=0)	0x95

De entre todas estas tramas, las más empleadas en este trabajo son las siguientes:

- Comando AT. Esta trama permite enviar un comando AT al dispositivo conectado mediante la conexión serial en modo API, por lo que el paquete no incluye ni necesita un campo de direccionamiento. Sí incluye la opción de recibir un mensaje de confirmación por parte del dispositivo, el cual estaría identificado con la trama 0x88 (respuesta de comando AT).

- Solicitud de comando remoto. Esta trama es una de las más importantes, pues permite enviar un comando AT a un dispositivo remoto que forme parte de la red en la que se encuentra el módulo conectado. Para esta trama sí se necesita un direccionamiento, que puede ser la dirección específica de un destinatario o bien un valor predeterminado que se interpreta

como un mensaje a todos los miembros de la red. Si se desea, el destinatario enviará un mensaje de confirmación tras la recepción del mensaje identificado con la trama 0x97 (respuesta a comando remoto). Tanto en esta trama como en la anterior, se puede enviar casi cualquier comando AT, lo que demuestra las capacidades adicionales del modo API (direccionamiento, CRC, confirmación de mensaje) frente a los comandos AT por separado.

- Muestreo de datos de E/S. Este mensaje también es de suma importancia, pues recoge el estado de las entradas o salidas digitales de un nodo XBee remoto en un momento determinado. Este mensaje, identificado con la ID de trama 0x92, puede configurarse para ser enviado cuando se produzca un cambio en el estado lógico de una entrada, de forma que desde el controlador se tenga constancia del estado del sistema y, mediante lógica, tomar decisiones al respecto. El paquete contiene la dirección del remitente del mismo, las entradas y/o salidas que está monitorizando y el estado de las mismas.

2.2.8. Software de configuración

Para poder inicializar y descargar el firmware en los dispositivos, el fabricante suministra un software que permite realizar la conexión con ellos y poder modificar sus parámetros. El programa X-CTU es la herramienta oficial de configuración, diagnóstico y testeo de los módulos XBee. Sólo está disponible de forma nativa para el sistema operativo Windows. Sus utilidades incluyen una ventana de terminal integrada, con capacidad para poder mostrar en pantalla caracteres hexadecimales y ASCII, que permite la comunicación entre el usuario y el dispositivo, así como construir y enviar paquetes de datos en cualquiera de los dos formatos; un test de rango de alcance de señales de radiofrecuencia; funcionalidad para cargar, escribir o sustituir el firmware de la radio, o

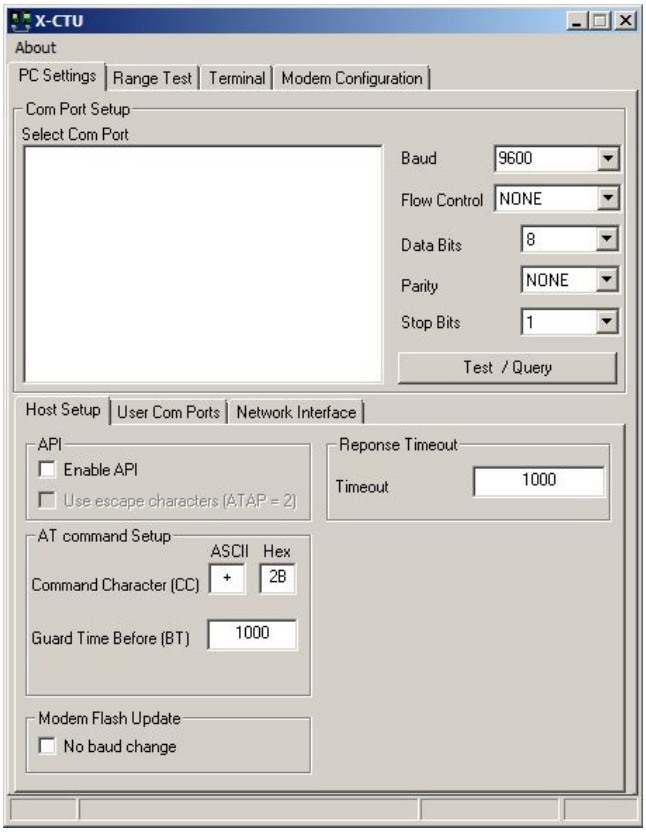

Figura 11. Interfaz del programa X-CTU de Digi.

bien revertirlo a su configuración de fábrica, mostrando en pantalla una ayuda y descripción de los comandos y parámetros de configuración; o crear, guardar y modificar perfiles de configuración con los parámetros más utilizados.

Existen otros programas alternativos al X-CTU [1], que permiten comunicarse con los dispositivos mediante una conexión serial, creando una ventana de terminal que habilita el envío y recepción de comandos y paquetes de datos. Estos programas permitirán configurar parámetros mediante el empleo de comandos AT, o bien visualizar los paquetes de datos recibidos o transmitidos. Sin embargo, ninguno permite modificar el firmware del mismo. Es por ello necesario utilizar el X-CTU para, al menos, configurar y guardar el firmware de cada uno de los módulos a utilizar, lo que obligaría a instalar programas adicionales en sistemas operativos que no sean Windows. Por ejemplo, en distribuciones Linux, se puede emplear el software Wine, que permite ejecutar programas de Windows, e instalar el X-CTU mediante el mismo. La Tabla 6 recoge algunas características de los programas de terminal alternativos al X-CTU. Es importante señalar que estos programas sólo son útiles en modo transparente, ya que para trabajar con un módulo XBee en el modo API se requiere de un programa que maneje las tramas que se envían o reciben a través de la conexión serial, como puede ser el propio X-CTU.

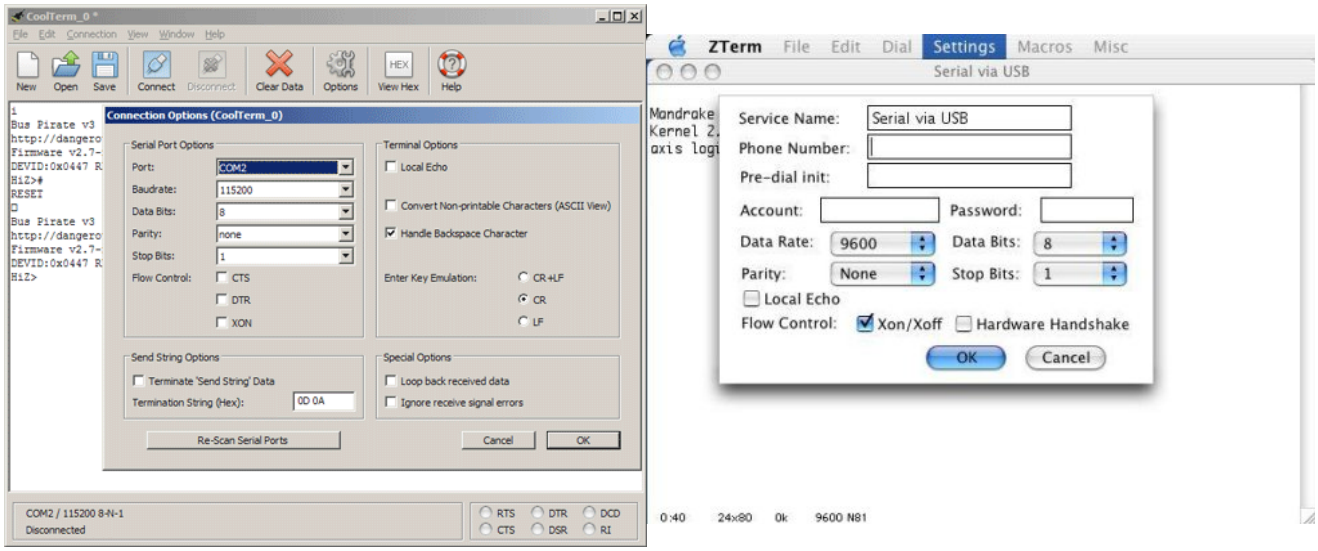

Figura 12. Interfaces de algunos programas alternativos a X-CTU (Izquierda, CoolTerm; Derecha, ZTerm)

Tabla 6. Programas de terminal para configuración de parámetros			
Nombre	Sistemas operativos	Licencia	Características
CoolTerm	Windows, Mac	Open Source, gratuito	Simple, incluye las funcionalidades básicas necesarias.
HyperTerminal	Windows	De pago	Viene de serie en SO Windows antiguos.
Tera Term	Windows	Open Source, gratuito	Amplia variedad de funciones, alternativa gratuita a HyperTerminal.
ZTerm	Mac	De pago	Actualizado por última vez en 2002 pero estable, ampliamente extendido.
screen	Linux, Mac	-	Desde línea de comandos, permite acceso directo a puertos seriales, incluyendo USB.
RealTerm	Windows	Gratuito	Diseñado para capturar, controlar y hacer debug de flujos de datos complejos.
Termite	Windows	Gratuito	Interfaz sencilla estilo chat para terminales RS232, histórico de comandos, soporte de velocidades de transmisión no estándar.
PuTTY	Windows, Linux	Gratuito	Implementación de Telnet y SSH junto con un emulador de terminal xterm.
MacWise	Mac	De pago	Emulador de terminal empleando Telnet y SSH. Incluye funcionalidades de teléfono, teclas programables y más.
Minicom	Linux	GPL	Emulador de terminal sobre consola o terminal de texto.
GtkTerm	Linux	GPL	Emulador de terminal para entornos X.

3. Trabajo desarrollado

En el anterior capítulo se presentaron las diferentes características tanto del protocolo ZigBee de comunicación inalámbrica como del dispositivo XBee que, mediante este protocolo, permite la creación de redes de sensores y actuadores a distancia, constituyendo una red domótica de bajo consumo. En este capítulo se presentará el desarrollo tanto del hardware como del software realizados para la consecución de los objetivos propuestos para este trabajo, así como los estudios necesarios para dicho desarrollo.

3.1. Alimentación

Para poder realizar las pruebas necesarias, en primer lugar es necesario construir el circuito básico que permita al XBee recibir alimentación. Según el manual del dispositivo [3], el XBee S2 estándar requiere de una alimentación de 3.3 V de tensión continua, y consume como máximo 40 mA en estado de funcionamiento. El consumo del dispositivo en modo desocupado se reduce a los 15 mA, mientras que en modo de hibernación el aparato necesita menos de 1 μA para mantener el estado.

3.1.1. Alternativas de alimentación

A continuación se recoge un estudio de las posibles formas existentes en el mercado para alimentar un módulo XBee. La elección de una u otra dependerá del uso y las funcionalidades deseadas para el trabajo. Estas alternativas presentan distintas ventajas y desventajas que serán enumeradas en el subapartado de conclusiones.

3.1.1.1. Fuente de alimentación continua

Una fuente de alimentación regulable (de laboratorio) permite entregar directamente 3.3 V mediante el ajuste del voltaje de salida. La corriente que puede entregar una fuente típica es más que suficiente para alimentar incluso al XBee PRO, de mayores necesidades energéticas.

3.1.1.2. Conexión USB

Un puerto USB, tanto normal como mini-USB o micro-USB, en cualquiera de sus versiones (1.0, 2.0, 3.0) entrega un voltaje de 5 V por sus pines de alimentación. Este voltaje siempre es continuo y fijo a 5 V, y la corriente que suministra puede variar entre 0.5 y 0.9 A en dispositivos generales,

aunque para cargadores puede alcanzar los 5 A. Si se desea emplear esta alimentación, se deberá reducir el voltaje a 3.3 V.

La conexión USB también puede provenir de adaptadores de corriente alterna, ampliamente extendidos en el mercado y cuyas características de salida se asimilan a las de un puerto USB de un dispositivo común.

3.1.1.3. Adaptador de corriente alterna

Un adaptador de corriente alterna, cargador o conversor AC/DC es un dispositivo eléctrico compuesto de un transformador que reduce el voltaje de 230 V de una red doméstica a un valor menor, de un rectificador de onda completa que permite convertir el voltaje alterno de 50 Hz de la red en voltaje continuo con un cierto rizado residual, y de un filtro que permite eliminar dicho rizado. Más recientemente, nuevos modelos de adaptadores emplean transistores como interruptores para obtener una onda cuadrada a la entrada de un esquema como el descrito anteriormente, consiguiendo de esta forma un adaptador de menor tamaño y una mayor eficiencia energética.

Figura 13. Adaptador de corriente alterna universal.

Comúnmente empleados para conectar cualquier dispositivo a un enchufe doméstico típico, ya sea para su funcionamiento o para cargar las baterías del mismo si las tuviese, pueden entregar valores de voltaje muy variados, incluso directamente 3.3 V. La corriente que entregan también es normalmente suficiente para las necesidades del dispositivo. Si el voltaje a la salida no es el adecuado para las necesidades del módulo, hay que reducirlo o ampliarlo hasta los 3.3 V mediante un circuito adicional.

3.1.1.4. Baterías

Una batería es una forma barata y rápida de lograr alimentar el dispositivo sin necesidad de emplear tomas de corriente, logrando un montaje portable. Estas permiten almacenar energía eléctrica en forma de energía química, liberándose al producirse una reacción entre los compuestos electrolíticos

que las forman. Hay dos tipos principales de baterías: primarias o desechables, que se agotan tras su uso y no pueden recuperar su carga, y secundarias o recargables, que mediante elementos químicos distintos sí permiten volver a almacenar energía en ellas (un número limitado de veces), aunque a costa de un menor voltaje a la salida.

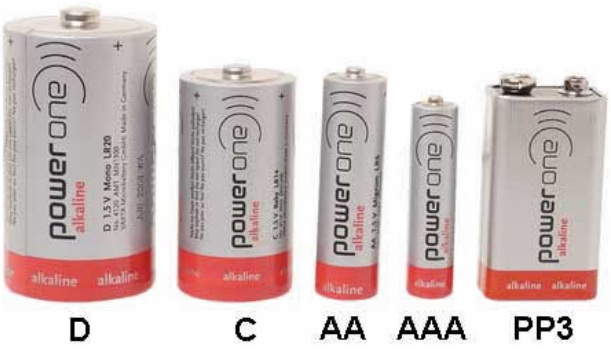

Figura 14. Algunos de los tipos de pilas más habituales.

Existe un amplio abanico de baterías de ambos tipos en el mercado, desde pilas de botón para relojes hasta baterías de litio para teléfonos móviles, pero todas ellas entregan un voltaje continuo a la salida. El principal problema de las baterías es su duración, ya que acaban agotándose tras su uso. Cada batería tiene una capacidad, medida normalmente en mAh (miliamperios-hora), que indica la cantidad de carga eléctrica que puede entregar al voltaje nominal. Esta capacidad depende del tamaño de la pila, de sus componentes, del voltaje y también de la carga a la que está sometida, y es un factor determinante en la viabilidad del uso de esta alimentación. Para determinar la duración de la batería, se aplica la siguiente expresión aproximada:

$$D = C / I * 0.7 \quad (1)$$

En la expresión anterior, D es la duración de la batería en horas, C la capacidad en amperios-hora e I la corriente que consume el dispositivo que alimenta en amperios. El factor de 0.7 da cuenta de factores externos que pueden afectar a la vida útil de la batería, como la temperatura de trabajo o el empleo de baterías de distinto tipo, carga o fabricante.

A continuación se presenta la Tabla 7, con un estudio sobre algunos de los tamaños de baterías comerciales más extendidos y su duración estimada en distintas condiciones de uso del dispositivo, calculadas a partir de la aplicación de la expresión aproximada (1) y los datos de consumo recogidos en el manual del XBee. La duración en modo hibernación se calcula suponiendo un ciclo de trabajo en el que de cada minuto permanezca 59.5 segundos en hibernación y 0.5 en transmisión, enviando en ese tiempo datos de sensado. Sabiendo que el ciclo de trabajo en porcentaje se calcula como Ton / Toff *100, el ciclo de trabajo será de 0.5 / 59.5 = 0.0084 = 0.84 %. Para hallar por tanto

la duración en modo hibernación, dividimos la duración en horas de las baterías en modo de máximo consumo (de transmisión) entre el ciclo de trabajo.

Tabla 7. Estudio sobre empleo de baterías para alimentación

Tipo de batería	Voltaje (V)	Capacidad (mAh)	Duración (h) en estado			Nº baterías
			Transmisión @ 40 mA	Desocupado @ 15 mA	Hibernación @ 1 µA	
AA alcalina	1.5	2700	67.5	180	8035.71	3
AAA alcalina	1.5	1200	30	80	3571.43	3
AA NiMH	1.2	1700 - 2900	42.5 – 72.5	113.3 – 193.3	5060 - 8631	3
AAA NiMH	1.2	800 - 1000	20 - 25	53.3 – 66.7	2381 - 2976	3
C alcalina	1.5	8000	200	533.33	23809.52	3
D alcalina	1.5	12000	300	800	35714.29	3
SR44 (De botón)	1.5	110 - 150	2.75 – 3.75	7.33 – 10	328 - 447	3
A23	12	40	1	2.667	119.05	1
9V	9	565	14.125	37.667	1681.55	1

A partir de toda esta información se puede escoger una u otra batería, teniendo en cuenta además factores como el precio, el tamaño que ocupa el conjunto o la disponibilidad en el mercado. Existen muchas otras baterías, como las de litio empleadas en electrónica de consumo, que también podrían ser interesantes debido a su tamaño y prestaciones.

3.1.1.5. Conclusiones

Cada alternativa plantea una serie de inconvenientes únicos que, junto con los beneficios o particularidades que aporta, pueden inclinar la balanza hacia uno u otro lado. La Tabla 8 plantea una serie de posibles beneficios y desventajas que cada una posee.

Tabla 8. Ventajas e inconvenientes de las alternativas de alimentación

Alimentación	Ventajas	Inconvenientes
Fuente de alimentación	Obtención directa del voltaje requerido, suministro constante y fijo, visualización en tiempo real de voltaje e intensidad.	Tamaño, necesidad de toma de corriente, poca practicidad para un uso doméstico, requiere de cables.
Conexión USB, cargador USB	Voltaje constante y fijo, tamaño reducido, puerto estándar en muchos sistemas informáticos y electrónicos, alternativa barata y muy disponible.	Necesita de un dispositivo con un puerto USB libre y que éste se encuentre encendido, requiere de un cable.
Adaptador de AC	Uso muy sencillo, abundancia en el mercado, no necesita otro dispositivo.	Requiere toma de corriente, trabaja con voltajes potencialmente peligrosos, riesgo de interferencias si es de mala calidad, usa cables.
Baterías	Sin cables, tamaño reducido, muy baratas y fáciles de usar, disponibilidad casi asegurada, no necesita dispositivos adicionales.	No es un suministro ilimitado, fácilmente afectado por factores externos, el voltaje no es constante, la eliminación de algunas baterías es difícil por sus productos químicos.

3.1.2. Regulación de tensión

Sea cual sea la alimentación escogida, el valor de tensión que entrega a la salida puede no ser el adecuado para el módulo XBee. En tal caso, es necesario convertir ese valor de voltaje a los 3.3 V requeridos. Para ello, existen dos alternativas principales.

- Convertidor DC/DC magnético: mediante un circuito conmutador y bobinas se pueden implementar circuitos que permiten reducir (circuitos *buck*) o aumentar (circuitos *boost*) el voltaje a la entrada, con una muy alta eficiencia energética. Se basan en el almacenamiento y extracción de energía de forma temporal en un campo magnético, con ciclos de trabajo de

alta frecuencia. Se emplean cuando existen muchos circuitos con necesidades de voltaje distintas, cuando se desea un voltaje a la salida superior al de entrada o cuando la eficiencia es un factor crítico de diseño. Aunque su eficiencia es muy alta, son circuitos complejos, más caros y grandes, y pueden sufrir problemas de ruidos debido a las conmutaciones. Existen variantes que permiten invertir el voltaje (*buck-boost*).

- Regulador de tensión lineal: estos circuitos permiten mantener un voltaje constante a la salida para entradas de voltaje variables. Su funcionamiento se basa en actuar como una resistencia variable, ajustando un divisor de tensión para mantener siempre la salida a un voltaje concreto. La energía se disipa en forma de calor, por lo que es un método más ineficiente que un convertidor magnético, y además sólo pueden obtener tensiones de salida menores que a la entrada, y dicho voltaje de entrada debe ser superior en una cierta cantidad para poder operar correctamente. A cambio, son muy baratos, sencillos (no requieren apenas componentes adicionales) y pequeños, y no generan ningún ruido. A pesar de todo, los reguladores lineales aumentan su eficiencia cuanto menor sea la diferencia entre el voltaje de entrada y el de salida, por lo que para ciertos casos puede ser más conveniente que los conmutados.

Para el presente trabajo se ha elegido emplear un regulador de tensión lineal, al ser más simple y barato que la alternativa conmutada. La eficiencia no es un factor significativo del diseño, y se utilizará una fuente continua que en todo momento el voltaje que suministra a la entrada es superior al de salida en una cantidad suficiente como para asegurar el correcto funcionamiento. En el mercado existe una gran variedad de reguladores de tensión como circuitos integrados en distintos encapsulados (TO-220, DIP), generalmente entregando +3, ±5, ±6, ±9, ±12 o ±15 V sin necesidad de componentes adicionales, salvo condensadores de desacoplo para proteger el circuito de posibles ruidos. Una variante muy popular es un regulador lineal variable, un CI que permite ajustar el voltaje de salida mediante el uso de resistencias externas de valores específicos. Concretamente, el regulador lineal LM317, inventado en la década de los 70, permite

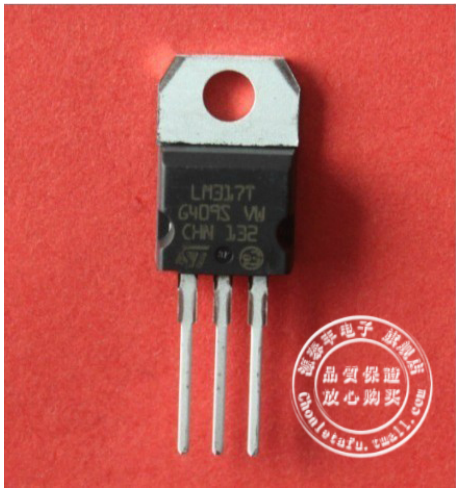

Figura 15. Regulador lineal LM317T.

obtener cualquier voltaje en el intervalo de 1.25 a 37 V. Será necesario emplear un regulador de esta familia debido a que el voltaje deseado no se puede conseguir con un regulador fijo.

El regulador LM317T es un LM317 con encapsulado TO-220. En su hoja de características [4] se recoge el circuito básico para la regulación, incluyendo condensadores de desacoplo, y la ecuación que permite averiguar los valores de las resistencias externas:

$$V_O = V_{REF} (1 + R_2 / R_1) + I_{ADJ} R_2 \quad (2)$$

El fabricante informa en la hoja de datos que el componente ha sido diseñado para que la corriente I_{ADJ} sea mínima y constante respecto a cambios de carga, por lo que habitualmente se suele descartar ese término de la ecuación. Por tanto, si se desea una tensión de salida V_O = 3.3 V, y siendo V_{REF} = 1.25 V, puede obtenerse la relación entre R_1 y R_2 despejando en la ecuación (2). Así, para una R_2 de un valor estándar de 3900 Ω, la resistencia R_1 adquiere un valor de 2400 Ω.

3.1.3. Montaje escogido

Para el presente trabajo se ha implementado un regulador de tensión LM317T a la entrada de la alimentación de ambos XBee, configurado para que su voltaje a la salida sea el que el módulo necesita. Incluye condensadores de desacoplo para filtrar posibles ruidos y perturbaciones. Como no se disponía de una resistencia de 2400 Ω, se ha optado por colocar en serie dos resistores de 2200 Ω y 220 Ω. Las fuentes de alimentación que se emplean son principalmente tres: mediante fuente de alimentación de continua, empleada para las pruebas en laboratorio (tanto en protoboard como en el prototipo), por conexión USB para poder ser alimentado de un ordenador, y por baterías para tener la opción de movilidad. Para poder conectar esta alimentación, se cuenta con un conector de tornillo de dos pines y 0.1” de paso, así como dos pines con orientación en ángulo recto. Adicionalmente, a efectos de diagnóstico e identificación, se ha añadido un diodo LED rojo en paralelo a la entrada, conectado en serie a una resistencia de 1000 Ω, de forma que se pueda comprobar visualmente si existe tensión a la entrada del circuito. El circuito implementado se muestra en la Figura 16.

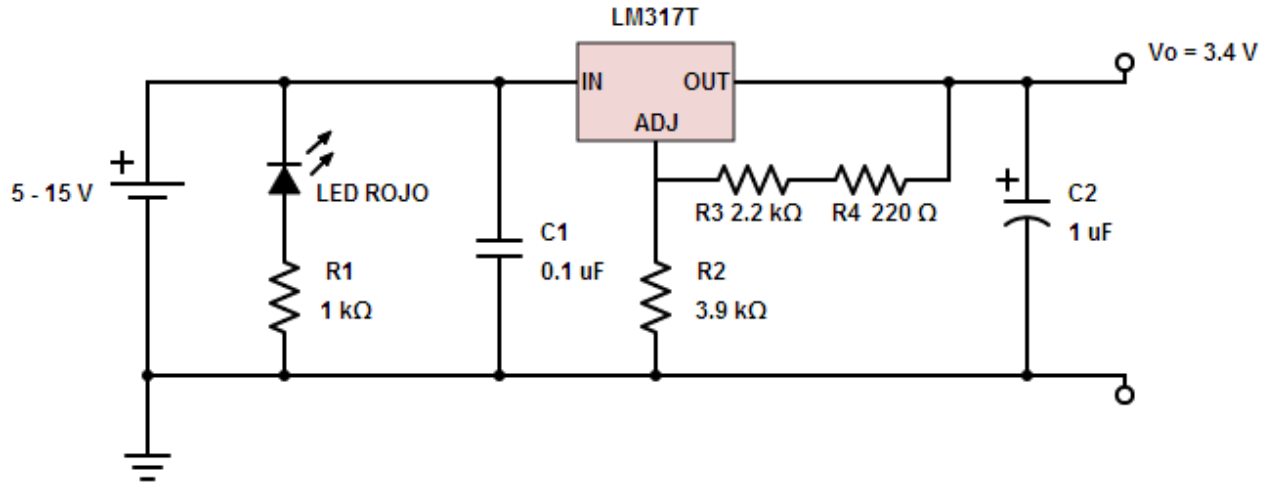

Figura 16. Diagrama esquemático del circuito de alimentación.

3.2. Comunicación

El XBee no suele emplearse por sí solo. Normalmente se suelen emplear sus cualidades como radio para sistemas que realicen procesos más complejos, que requieran de conectividad inalámbrica con otros sistemas. Para poder comunicarse y conectarse con el resto del sistema, los XBee necesitan algún tipo de interfaz de comunicación. Así, por ejemplo, aunque el módulo reciba alimentación y funcione, necesitará configurar su firmware para poder adoptar el rol que se desee en la red. Para ello, es necesario que se conecte a un PC equipado con el software X-CTU, para lo cual necesita de una interfaz adecuada.

El módulo XBee cuenta con un UART (*Universal Asynchronous Receiver/Transmitter*), punto de entrada y salida de datos del módulo desde un dispositivo anfitrión. El módulo UART también realiza tareas como la comprobación de la paridad y el sincronismo, necesarias para la comunicación.

A la hora de realizar la conexión con otro dispositivo, estas son las opciones principales:

3.2.1. Directamente mediante UART

Si el dispositivo al que se conecta el XBee también cuenta con un UART, se pueden conectar directamente los puertos de entrada y salida entre sí mediante una conexión de lógica CMOS de entre 2.8 y 3.4 V. Si los niveles de voltaje no son los adecuados, será necesario adaptarlos, ya sea

mediante un divisor de tensión, resistencias *pull-up* u otras interfaces de conversión. Un ejemplo de este método es el módulo XBee para placas Arduino, que proporciona capacidades inalámbricas a esta última. Como ambos dispositivos cuentan con UART, lo único que se debe hacer es controlar que los valores de voltaje sean los adecuados. En caso de conexiones a microcontroladores que trabajan a 3.3 V ,como BeagleBoard Black o Raspberry Pi, la conexión es directa.

3.2.2. USB

Un puerto USB es la forma más rápida de realizar la conexión entre un ordenador y un XBee. El principal problema a solventar es que el módulo XBee por sí solo únicamente tiene capacidad para enviar datos seriales, ya que opera con el UART. Para solucionarlo se requiere convertir UART a USB y viceversa, normalmente mediante un circuito integrado. Existen varios fabricantes para estos CIs, entre los que podemos destacar:

- Silicon Labs oferta circuitos integrados que soportan protocolos de conversión USB-Serial, como el CP2102, CP2103 o CP2104, cada uno con sus características concretas y aplicaciones específicas. Estos pueden adquirirse por separado o ya integrados en una placa que proporciona los puertos adecuados para la conexión.

- FTDI Chip fabrica el circuito integrado FT232R, ampliamente extendido, que implementa el protocolo USB en el propio chip, incluye generación de reloj sin cristales, conversión a niveles TTL o CMOS o una opción para invertir la señal UART, entre otras. Requiere de un driver que se instala en el dispositivo de destino, y su reducido tamaño y versatilidad permite incluso integrarlo en el cabezal de un cable USB normal.

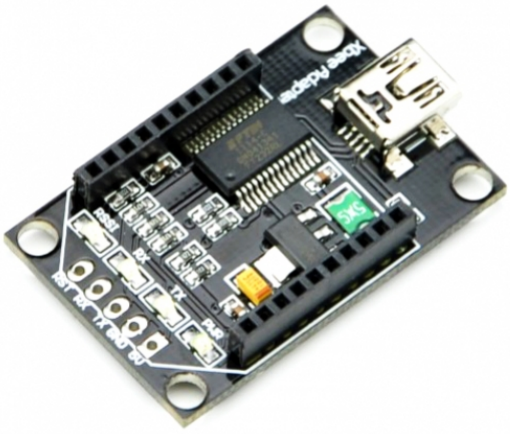

Figura 17. Adaptador USB basado en un FT232RL.

Implementando un circuito empleando este tipo de integrados, se puede conectar el módulo a un dispositivo y además alimentarlo. Existen varias opciones en el mercado de adaptadores USB que

incorporan estos circuitos y otras funcionalidades, como el USB Explorer de SparkFun, el Adapter Kit de Adafruit o el XBee to USB Adapter de Gravitech. Algunos de estos fabricantes también proporcionan el esquemático del adaptador entre la documentación existente en la web. Del mismo modo, existen adaptadores para convertir un puerto USB en serial, y conectar cualquier otro dispositivo al puerto serial hembra que éste incluye.

3.2.3. Serial

Alternativamente a la conexión USB, existe la opción de emplear directamente una conexión serial mediante una interfaz adecuada. Existen tres estándares seriales principales:

- RS-232, creado a finales de los 60 para implementar la comunicación entre terminales y módems, cuenta con dos líneas de datos y cinco de control. Se caracteriza por su baja velocidad de transmisión y conectores de gran tamaño. Los niveles de voltaje que se traducen en unos o ceros lógicos son los siguientes:
  - El 1 lógico va de -5 a -25 V en los emisores, y de -3 a -25 V en los receptores.
  - El 0 lógico, por otro lado, va de +5 a +25 V en emisores, y de +3 a +25 en receptores.
  - Los voltajes no definidos, por tanto, serán ±5 V en emisores y ±3 V en receptores.

  En la mayoría de PCs el rango de voltajes está comprendido entre los ±13 V.
- RS-422, que emplea señales diferenciales a través de 4 cables. Al no haber punto de referencia a masa, no es afectado por el ruido electrostático. El rango de voltajes también es menor, de -2 a -6 V y de +6 a -2 V. Estos cambios lo hacen más rápido y con mayor alcance que el RS-232.
- RS-485, que sólo requiere dos conductores. Permite conectar muchos más dispositivos e implementa un estado de alta impedancia cuando no se está enviando ningún dato.

De todos ellos, el más común en ordenadores y otros dispositivos es el RS-232. Sin embargo, el voltaje de este es mucho mayor que el que trabaja el módulo XBee, y además trabajan a lógicas inversas (el 1 lógico corresponde a Vcc en la lógica TTL y a -13 V en el RS-232) por lo que se requiere de una interfaz de conversión entre ambos valores. El método más empleado es usar un

MAX232, un circuito integrado creado por primera vez por la empresa Maxim, que mediante el uso de condensadores externos de 0.1 µF permite realizar el cambio de voltaje y de lógica. Otras formas alternativas, más baratas, se basan en usar transistores o inversores para invertir la señal y bombas de carga para subir los voltajes a niveles de RS-232.

Debido a que el XBee trabaja con lógica CMOS y no TTL, el MAX232 no podría emplearse para esta aplicación en concreto. Sin embargo, en el mercado también existe una versión CMOS del mismo circuito integrado, denominado MAX3232. Uno de los modelos de este CI disponibles, el MAX3232CPE, es un transceptor (transmisor y receptor que comparten circuitería) para voltajes de entrada de entre 3 y 5.5 V, que con la ayuda de 4 condensadores de 0.1 µF situados en los pines 1 a 6 permite manipular las señales entrantes y salientes y adecuar los niveles de voltaje. Con este circuito se realizan las conversiones de las señales seriales TX, RX, CTS y RTS y las señales CMOS del XBee DIN, DOUT, CTS y RTS.

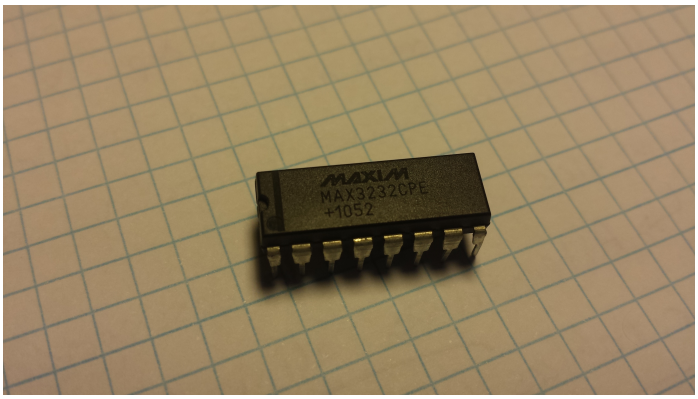

Figura 18. Circuito integrado MAX3232.

3.2.4. Montaje escogido

Para poder lograr comunicar los módulos XBee con un ordenador equipado con el X-CTU, se ha creado un módulo de comunicación que implementa el MAX3232CPE y los componentes que necesita, junto con pines para introducir las señales del XBee y del ordenador. La señal DTR no emplea este integrado para su transmisión, sino que se comunica directamente la proveniente del cable serial con la del módulo XBee, mediante un transistor NPN 2N2222A y resistencias. El esquema concreto es el que se muestra en la Figura 19.

Las conexiones de este módulo constan de alimentación (VCC + GND), 4 pines provenientes del XBee conectados al MAX3232, 6 pines salientes del MAX3232 más la señal DTR serial, a los que se conecta el cable RS-232, y un pin para la señal DTR del XBee. Para el envío de una de las señales seriales, se ha requerido el uso de una resistencia de 2 Ω a modo de cable.

El XBee puede funcionar sin necesidad de este módulo, y dado que se empleará sobre todo para

ajustar el firmware y los parámetros de configuración, sólo se ha desarrollado un único módulo de comunicación.

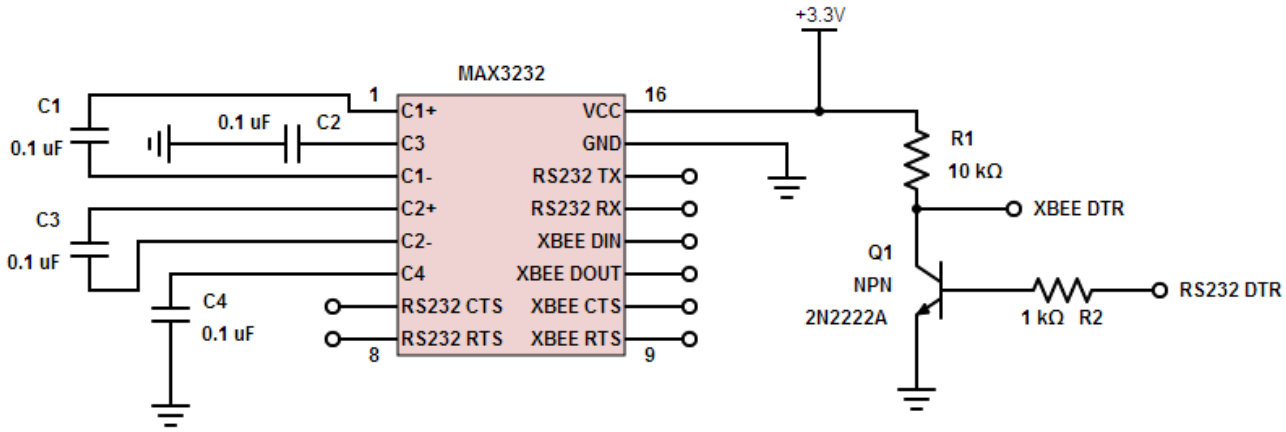

Figura 19. Diagrama esquemático del circuito de comunicación.

3.3. Sensores y actuadores

3.3.1. Sensores

El propósito de un módulo sensor es recoger información del entorno para poder tomar decisiones de actuación en base a estos datos. Un módulo XBee cuenta con un total de diez pines configurables como entradas o salidas digitales, y cuatro de ellos tienen además la capacidad de actuar como entradas analógicas. Esto le permite al XBee trabajar con sensores digitales de tipo interruptor, así como sensores analógicos de temperatura, presión, luminosidad, aceleración, humedad, etc.

En el mercado existen muchos tipos de sensores, basados en diferentes procesos y fundamentos físicos, para poder medir distintas magnitudes. La mayoría funciona en el rango 3.3 V – 5 V, por lo que o bien se pueden conectar directamente a las entradas del XBee o bien convertir a CMOS el voltaje. Las necesidades de corriente de estos sensores suelen ir desde los µA hasta mA, y operan en el mismo rango de voltaje que entregan a la salida. Una consideración importante de diseño es que los pines analógicos del XBee sólo leen entre 0 y 1.2 V, y los voltajes desde 1.2 V hasta Vcc son ignorados y considera el mismo valor máximo. Esto obliga a emplear un divisor de tensión para reducir en 2/3 el voltaje, de forma que caiga al rango 0-1.2 V.

Los más empleados desde el punto de vista domótico son los siguientes:

Figura 20. Sensor de movimiento PIR.

- De movimiento y de presencia: los sensores de movimiento más comunes son los PIR (*passive infrared sensor*), que detectan cambios en la radiación de luz infrarroja dentro de su campo de visión. En caso de que se desee detectar la presencia de una persona aunque no se esté moviendo, se necesita un tipo de sensor que sea de presencia.

- De luminosidad: también se los conoce como fotosensores o fotodiodos. Se emplean para monitorizar el nivel de iluminación, normalmente en conjunción con un sensor de presencia para poder activar la iluminación artificial de una habitación sólo si es necesaria. También se emplean para acciones que deben ocurrir en ciertos momentos, como al ocaso o al alba.

- De temperatura y de humedad: termómetros de pequeño tamaño permiten saber la temperatura de una habitación. Los sensores de humedad detectan la concentración de vapor de agua en el ambiente. Combinados, posibilitan el control de sistemas de climatización, aire acondicionado o deshumidificadores.

- Sensores de alarma de incendios: los hay de tipo óptico, que se activan si partículas de humo interrumpen el haz de luz que estos emiten; de ionización, que detectan partículas ionizadas en el aire y por tanto poseen mayor sensibilidad que los ópticos; y de calor, que detecta temperaturas anómalas. Normalmente los sensores que se instalan combinan un sensor óptico y uno de ionización para evitar en la medida de lo posible falsos positivos.

- Sensores de monóxido de carbono: durante un incendio, es casi tan peligroso el envenenamiento por inhalación de CO que el propio fuego. Esto es debido a que el CO es inodoro, incoloro y muy tóxico (inhibe la fijación de oxígeno en sangre). Los sensores de CO detectan este gas en el ambiente, y se suelen combinar con los

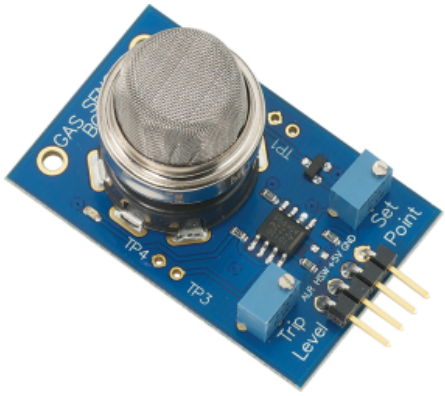

Figura 21. Sensor de gases.

sensores de alarma de incendios para ofrecer una cobertura casi total.

- De contacto: estos sensores funcionan como un interruptor, enviando una corriente entre dos superficies cuando éstas están en contacto. Permiten monitorizar, entre otras opciones, si una ventana o una puerta está cerrada o no. Las utilidades de estos sensores van desde alarmas antirrobo hasta el control de la calefacción de una habitación.

- De proximidad: una utilidad muy común de estos sensores es en lugar de interruptores. Por ejemplo, pueden sustituir a un interruptor de pared, activándose y encendiendo la iluminación simplemente con aproximar la mano al sensor.

- De uso de energía: extremadamente útiles para reducir el consumo de energía de una vivienda. Estos sensores monitorizan el consumo exacto de electricidad de cada dispositivo conectado a la red eléctrica, permitiendo de este modo comprobar qué aparatos consumen más y cuáles conviene desconectar o mantener enchufados.

- De fugas y de inundaciones: normalmente se instalan bajo los baños y los fregaderos, o en otras zonas con riesgo de fugas elevado. Combinado con un actuador en forma de válvula, pueden parar una fuga de agua y avisar de la misma, de forma que se impida que empeore.

Figura 22. Sensor de agua. Permite la detección de lluvia o de pérdidas.

Se pueden implementar muchos otros sensores, como micrófonos o sensores de identificación, pero todos actúan del mismo modo, ya sea enviando una señal digital cuando sucede el evento que los active, o enviando un valor de voltaje que depende de la variable que esté midiendo. Estos datos se introducirán en el XBee por los pines que estén configurados como entradas, y se enviarán a otros nodos que lo necesiten.

Para simular el comportamiento de un sensor digital, se ha empleado un pulsador de botón, el cual al ser presionado actuaría como un sensor activo. Para simular una entrada analógica se emplea un potenciómetro.

3.3.2. Actuadores

Los actuadores son dispositivos que generan una acción tras recibir una orden de un controlador, mediante la conversión de energía eléctrica, mecánica o hidráulica. De entre los principales tipos de actuadores existentes en el mercado y empleados en domótica se pueden destacar los siguientes.

3.3.2.1. LEDs

La forma más sencilla de actuación. Un diodo LED permite proporcionar información visual al usuario sobre el estado de un sistema, iluminar elementos en sustitución de bombillas incandescentes tradicionales, e incluso pueden actuar en bandas fuera del espectro visible, como diodos infrarrojos. Se basa en un diodo que, al recibir voltaje en sus bornes, permite a los electrones recombinarse con los huecos o cargas positivas, liberando energía en forma de fotones.

3.3.2.2. Interruptores

Un interruptor es un componente eléctrico que permite desviar o interrumpir una corriente eléctrica. Normalmente es un elemento operado manualmente, aunque los que interesan desde el punto de vista automático son controlados autónomamente mediante alguna variable o directamente por un controlador. Los principales tipos que se manejan en este trabajo son:

- Relé: interruptor accionados por un electroimán. Al aplicar una tensión en los bornes de una bobina, esta genera un campo magnético que atrae una armadura férrea hacia el núcleo. Esto hace que el terminal de entrada deje de estar conectado al terminal normalmente cerrado y pase a establecer contacto con el terminal normalmente abierto. Al cesar el voltaje, el relé vuelve al estado de reposo. El relé permite manejar cargas de potencia más elevadas que otros tipos de interruptores, por lo que se suele emplear para el control de lámparas o motores. Un contactor es un relé que funciona a muy altos valores de potencia.

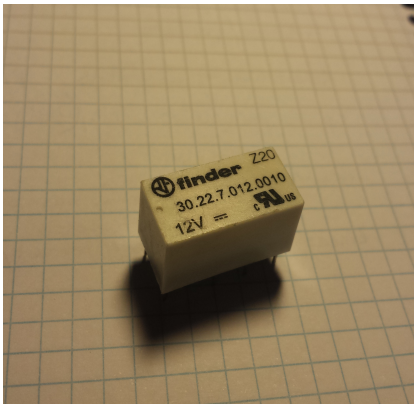

Figura 23. Relé Finder de 12 V y 2 A. Componente empleado en el trabajo.

- Interruptor MOSFET: el transistor MOSFET o de efecto de campo, basado en la

modificación del ancho de un canal conductor (de electrones en un tipo N o de cargas positivas en un tipo P), puede actuar como un interruptor controlado por tensión que se suministra a través de uno de sus terminales. Cuando la tensión suministrada a dicho terminal (puerta o *gate*) adquiere un valor suficientemente alto, el canal conductor generado provoca que queden técnicamente cortocircuitados los otros dos terminales del transistor, comportándose en ese estado como un interruptor cerrado. En este momento, el voltaje V_{GS} entre la puerta y la fuente (*source*) es mucho mayor que el voltaje umbral (voltaje a partir del cual comienza a permitir el paso de corriente a través del canal), y circula la corriente máxima posible. Del mismo modo, si el voltaje en la puerta cae a cero, el voltaje V_{GS} pasa a ser menor que el umbral y no se genera un canal conductor. La intensidad cae a cero y el transistor está actuando como un interruptor abierto. Este comportamiento se puede visualizar gráficamente en la curva característica del transistor en la que se representa I_D frente a V_{DS}.

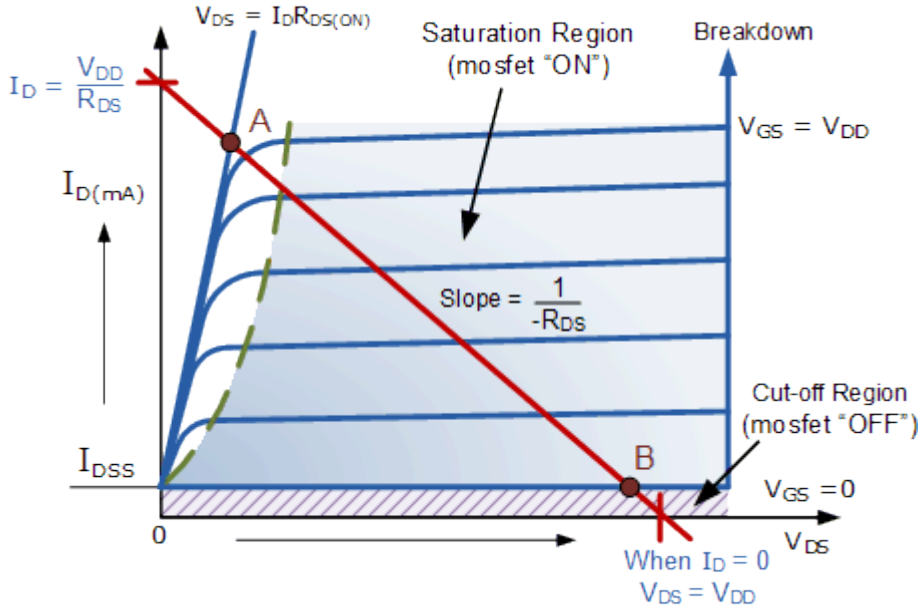

Figura 24. Curva característica de un MOSFET de enriquecimiento.

Estos interruptores pueden usarse para corriente continua y alterna, pero en este último caso conviene aislar la señal de control de la puerta, mediante un optoacoplador u otros elementos aislantes.

- Interruptor BJT: su funcionamiento es similar al MOSFET, buscando actuar únicamente en las zonas de saturación y corte del transistor. Cuando el voltaje en la base es menor de 0.7 V

y no circula corriente por la misma, las uniones base-emisor y colector-base están inversamente polarizadas, y el transistor pasa a estar operando en la región de corte, es decir, como un interruptor abierto. Si el voltaje en la base es mayor de 0.7 V y circula la corriente suficiente por la misma, ambas uniones estarán polarizadas directamente. La corriente que circula por el transistor es máxima, con una caída de tensión entre el colector y el emisor idealmente nula. En definitiva, se comporta como un interruptor cerrado. Este tipo de interruptores se emplea sólo para corriente continua.

- TRIAC: también conocido como tríodo para corriente alterna, permite conducir la corriente en cualquier sentido cuando está cerrado. Esta característica bidireccional los hace muy adecuados para el control de corriente alterna, permitiendo controlar cargas de potencia considerable con corrientes de control muy bajas. Se suelen emplear para controlar la iluminación de una lámpara, la velocidad de un motor eléctrico, y en general en aplicaciones domóticas que utilizan corriente alterna.

3.3.2.3. Motores de corriente continua

Los motores DC convierten energía eléctrica en mecánica. Se construyen con dos imanes permanentes fijados, típicamente, a la carcasa del motor o estátor, y con un bobinado de cobre en el eje del motor o rotor. La interacción magnética de ambos hace que el eje comience a moverse. Al trabajar con corriente continua es muy sencillo controlar su velocidad (reduciendo o aumentando el voltaje) y sentido de giro (invirtiendo la polaridad), responden con rapidez y son muy baratos.

Figura 25. Motor de corriente continua de 12 V.

3.3.2.4. Motores de corriente alterna

Los motores de corriente alterna pueden ser de dos tipos, síncronos (si giran a la velocidad que determina la frecuencia de la corriente) y asíncronos (si su velocidad es algo menor a la que determina la frecuencia). Unos polos fijos en el rotor siguen las variaciones de polaridad de los devanados del estátor experimentadas por la corriente alterna, causando así una rotación. Esta depende de la frecuencia de la corriente alterna y del número de polos.

3.3.2.5. Motores paso a paso

Este tipo de motores es ideal para automatismos que requieran movimientos muy precisos. Se caracterizan porque se mueven un paso cada vez que reciben un pulso de corriente. Este paso se mide en grados de giro, y puede ir desde 90º (motor de 4 pasos) hasta 1.8º (motor de 200 pasos). Se pueden fijar en una posición si hay corriente por una o más de sus bobinas, o bien estar completamente libres si ninguna bobina está alimentada. El rotor se compone de varios imanes permanentes, mientras que el estátor se compone de bobinas excitadas siguiendo un control preciso, del cual se encarga un controlador.

Existen muchos otros actuadores de diverso tipo, basados o no en los anteriores, como válvulas hidráulicas, alarmas sonoras, pistones o émbolos, resistencias calefactoras...

3.3.3. Montaje escogido

El circuito que se ha decidido montar para realizar las pruebas de funcionamiento consta de lo siguiente:

3.3.3.1. Sensores

A nivel de sensores, se ha implementado un pulsador para simular el comportamiento de un sensor digital conectado al pin 11 (entrada/salida digital 4). El botón al ser oprimido crea una conexión entre VCC y tierra a través de una resistencia de 390 Ω, lo que hace que el estado lógico del pin (que está conectado entre el pulsador y tierra) cambie de 1 a 0. Al soltar el botón, el estado lógico

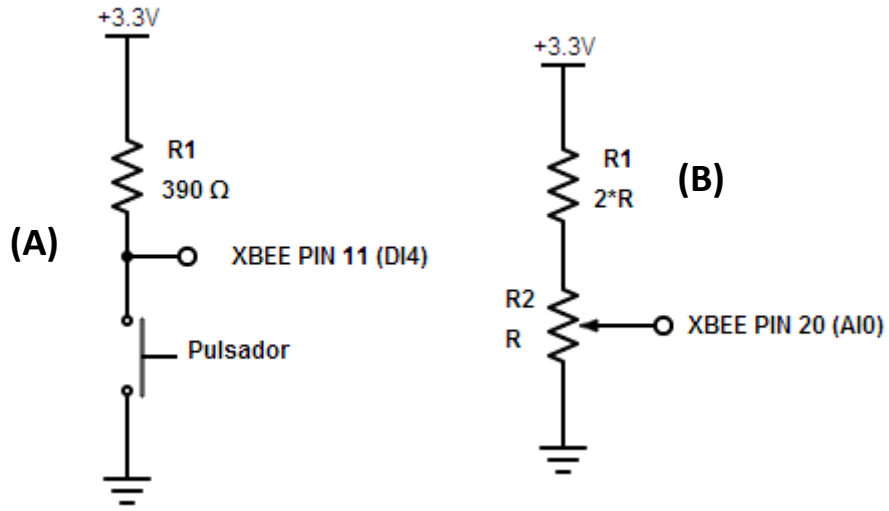

Figura 26. Diagramas esquemáticos del sensor digital simulado (A) y del sensor analógico (B).

lo que provoca que se tengan 3.3 V en extremos de la resistencia debido a que no cae tensión. El esquema se puede apreciar en la Figura 26(A).

Adicionalmente, se ha preparado un pequeño módulo con un potenciómetro que permiteº implementar un divisor de tensión regulable. La salida de este divisor se envía a un pin de entrada analógica (por ejemplo, el pin 20), y permite observar el comportamiento del XBee con entradas de tipo analógico. El valor de voltaje en esta entrada cambiará de forma continua mientras se regule el potenciómetro, y el objetivo a perseguir es muestrear esta entrada para obtener los valores analógicos que está midiendo. La resistencia superior debe ser el doble que el máximo que ofrece el potenciómetro, de forma que los valores de voltaje queden dentro del rango que acepta el XBee. El esquema se recoge en la Figura 26(B).

3.3.3.2. Actuadores

El actuador más sencillo consta de un interruptor compuesto de un BJT y un MOSFET conectados en cascada. Se emplean dos transistores porque el MOSFET empleado, un IRF9520 con empaquetamiento TO-220, requiere de un voltaje mayor que el que los niveles CMOS pueden proporcionar para poder crear el canal conductor y así cerrarse. El segundo transistor, un BF199 NPN, actuará permitiendo introducir el voltaje adecuado a la entrada del MOSFET cuando reciba un 1 lógico (3.3 V). En conjunto, actúan como un único interruptor al cual se le conectó a modo de prueba un motor de corriente continua de pequeño tamaño, que a su vez mueve las aspas de un ventilador de ordenador.

Este interruptor se instala en una pequeña placa de puntos junto con los conectores necesarios para poder operar: un pin transmite la señal de control desde el XBee, mientras que un conector de tornillo de 4 pines posibilita la conexión de la alimentación necesaria para mover el ventilador y la conexión del propio ventilador. Esta alimentación, en el caso de este actuador, será de entre 12 y 15 V.

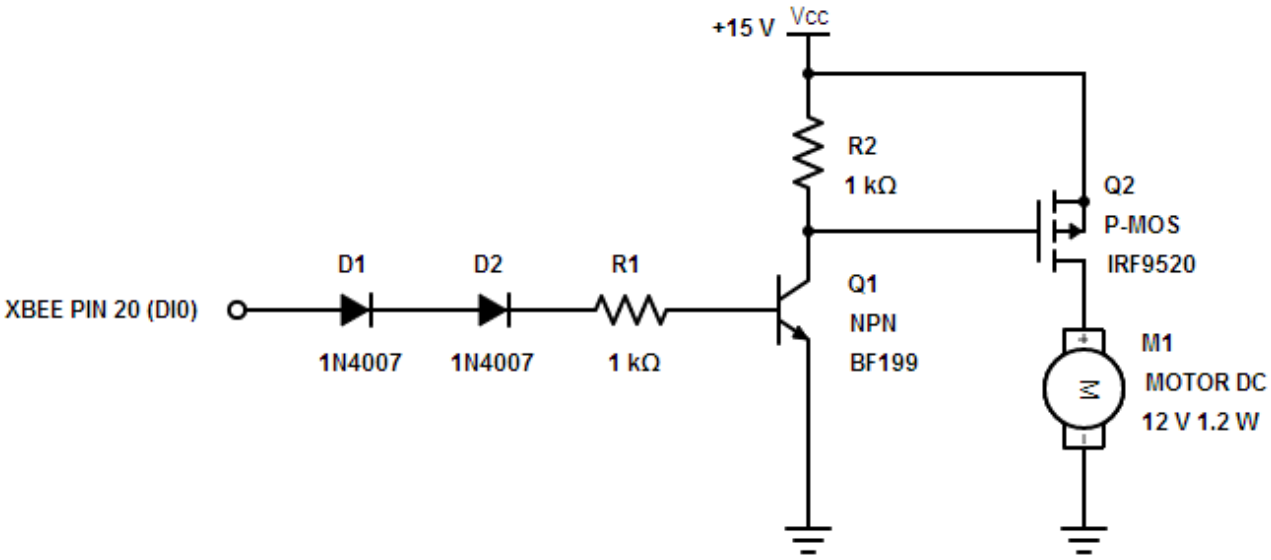

Figura 27. Diagrama esquemático del circuito interruptor.

Adicionalmente, para demostrar las aptitudes del módulo XBee para trabajar con elementos de mayor potencia, se ha creado un módulo con un relé Finder 30.22.7.012.0010, de 12 V y 2 A. Este relé tiene la misma huella que un circuito integrado de 16 pines, por lo que puede ser montado en un zócalo, aportando practicidad y facilidad de reposición en caso de que se averíe. El relé sólo tiene 8 pines, situados en las posiciones 1, 4, 6, 8, 9, 11, 13 y 16. Los pines 1 y 16 corresponden a los extremos de la bobina del relé, mientras que los pines 4, 6 y 8 por un lado y 9, 11 y 13 por el otro son las posibles conexiones, normalmente cerradas en el caso de 4-6 y 13-11 y normalmente abiertas en el caso de 4-8 y 13-9. Esto significa que, cuando la bobina recibe el voltaje suficiente (12 V, aunque empíricamente se ha comprobado que conmuta con un voltaje considerablemente menor), las conexiones pasan de estar cerradas a estar abiertas, creándose una conexión entre los pines 4-8 y 13-9.

Para proteger el circuito de los picos negativos de tensión producidos al trabajar con cargas inductivas, como por ejemplo un motor o la propia bobina del relé, se sitúa un diodo 1N4007 en paralelo a la bobina del relé, con su cátodo en el pin 16. Esto obliga a que dicho pin sea la entrada de la alimentación, siendo el pin 1 la salida a tierra, ya que en caso contrario se tendría un cortocircuito causado por el diodo. Por otro lado, para proteger el relé de tensiones de alimentación excesivamente elevadas, se ha implementado un regulador de tensión lineal LM317T que permite mantener siempre 12 V a la entrada de la bobina. Se han empleado como resistencias una R_1 de 270

Ω y una R_2 de 2440 Ω, compuesta de dos resistencias de 2200 Ω y 220 Ω en serie. Una resistencia de baja tolerancia de 1 Ω realiza la conexión entre las resistencias R_1 y R_2.

Dos filas de 5 pines hembra a cada lado del zócalo permiten poder conectar los elementos que el relé controlaría. Como sólo los pines 4/13, 6/11 y 8/9 tienen conectividad con el relé, los pines 2 y 4 de cada fila no se usan. Además, se tiene un conector de tornillo de tres pines para la alimentación, que normalmente provendrá del módulo interruptor.

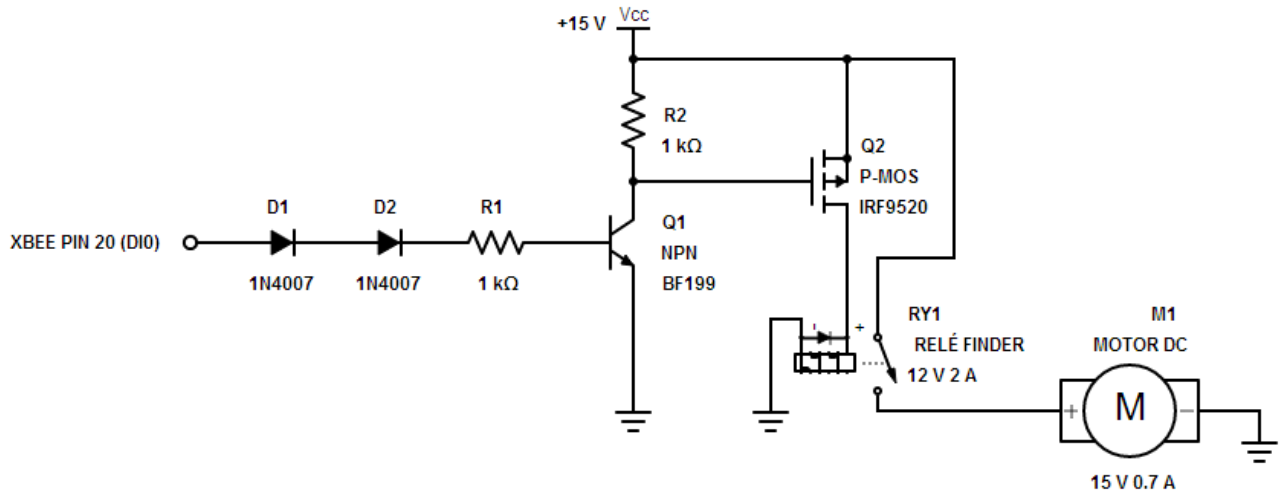

Figura 28. Diagrama esquemático del circuito de relé.

3.4. Montaje final

Una vez probados, se han realizado los montajes de los esquemas recogidos en una placa de puntos de baquelita REPRO CT-19, de 2.54 mm de paso. El módulo XBee tiene un tamaño de paso de 2 mm, por lo que se necesita una placa de conversión que permita conectarlo a una placa estándar [6].

El circuito construido consta de los siguientes elementos:

- Módulo principal (Figura 29), en el cual se monta el XBee. Este módulo se basa en uno de los adaptadores seriales comerciales disponibles en el mercado [5], e incluye el regulador de tensión para la alimentación junto con sus conectores de entrada, pines verticales para la conexión serial y la alimentación del módulo de comunicación, diodos LED de diagnóstico conectados a los pines DIN, DOUT y RSSI, y un pulsador que permite hacer un reinicio rápido del XBee. Para poder realizar las conexiones, se ha alternado entre crear pistas de

estaño en la cara inferior y emplear resistencias de valores bajos (2 Ω o menos) a modo de cables.

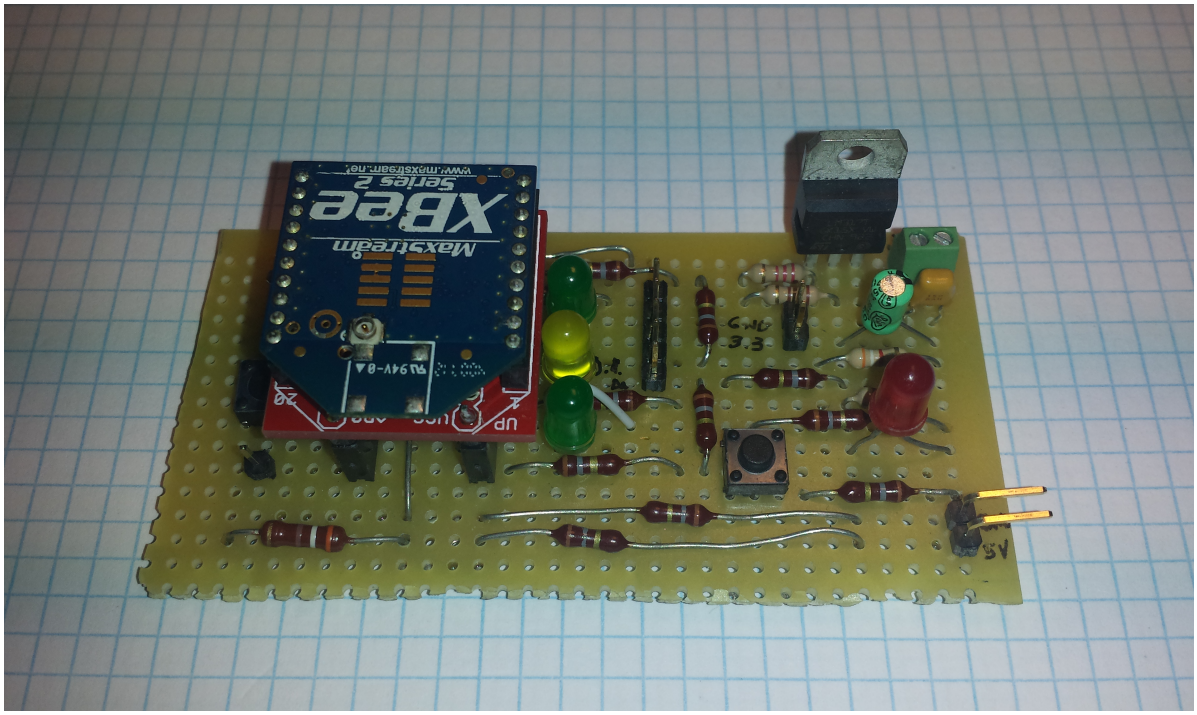

Figura 29. Módulo principal con el XBee (nodo remoto).

- Módulo de comunicaciones (Figura 30), que incluye el circuito integrado MAX3232, el cual realiza el acondicionamiento de las señales seriales del XBee y del ordenador.

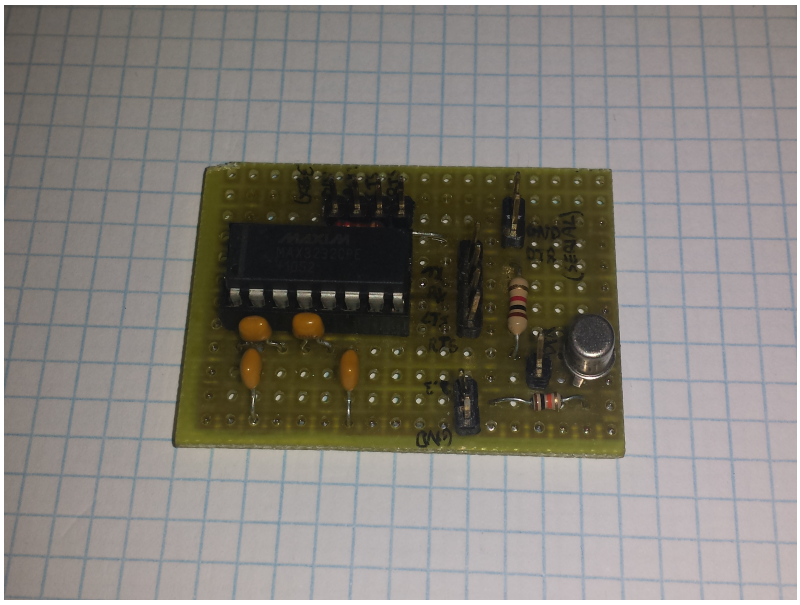

Figura 30. Módulo de comunicaciones.

- Módulo de interruptor (Figura 31), en el que se ha implementado el interruptor basado en los transistores BJT y MOSFET (en la imagen, con encapsulado TO-220).

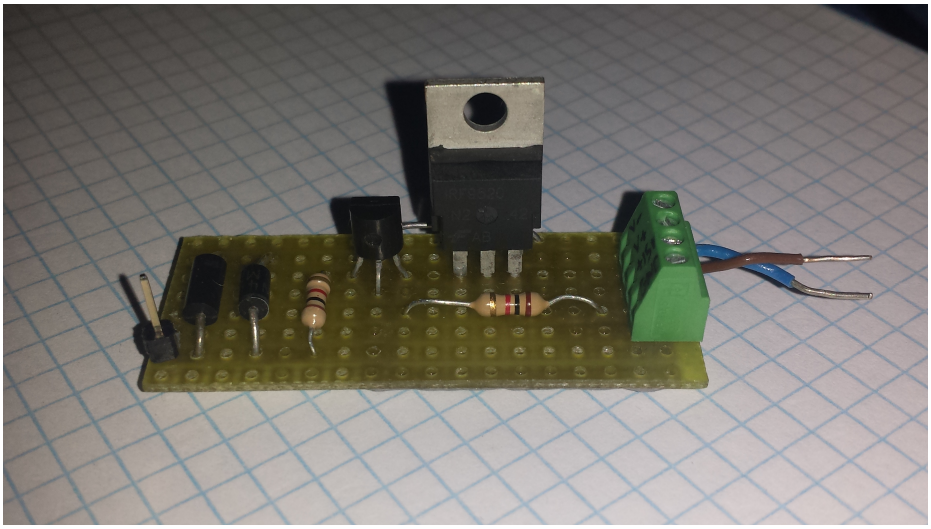

Figura 31. Módulo de interruptor.

- Módulo de relé (Figura 32), donde se ha montado el relé Finder con su regulador de tensión, el diodo de protección y los conectores de entrada.

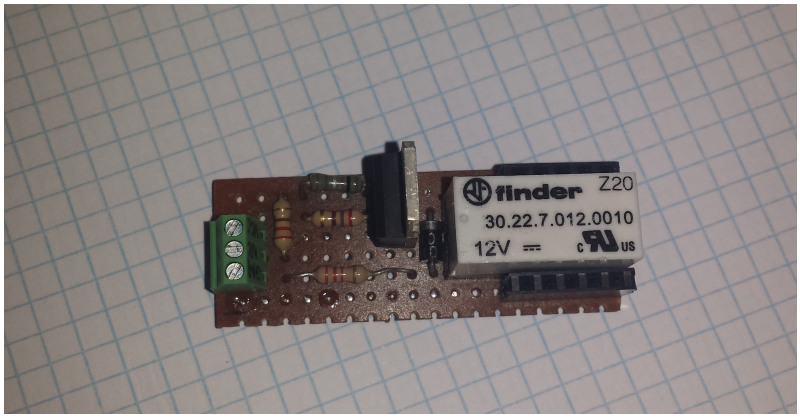

Figura 32. Módulo de relé.

El módulo principal es ligeramente diferente en cada XBee. El coordinador (ver Figura 33 izquierda) cuenta con un diodo LED conectado al pin 11 (E/S digital 4) que sirve para, mediante software, indicar al usuario si el interruptor del nodo remoto está cerrado o abierto. El nodo remoto,

por otro lado (ver Figura 33 derecha), cuenta con el pulsador que simula el sensor digital, y un pin conectado a la E/S digital 0 a través de la cual se enviará la señal de control al módulo interruptor.

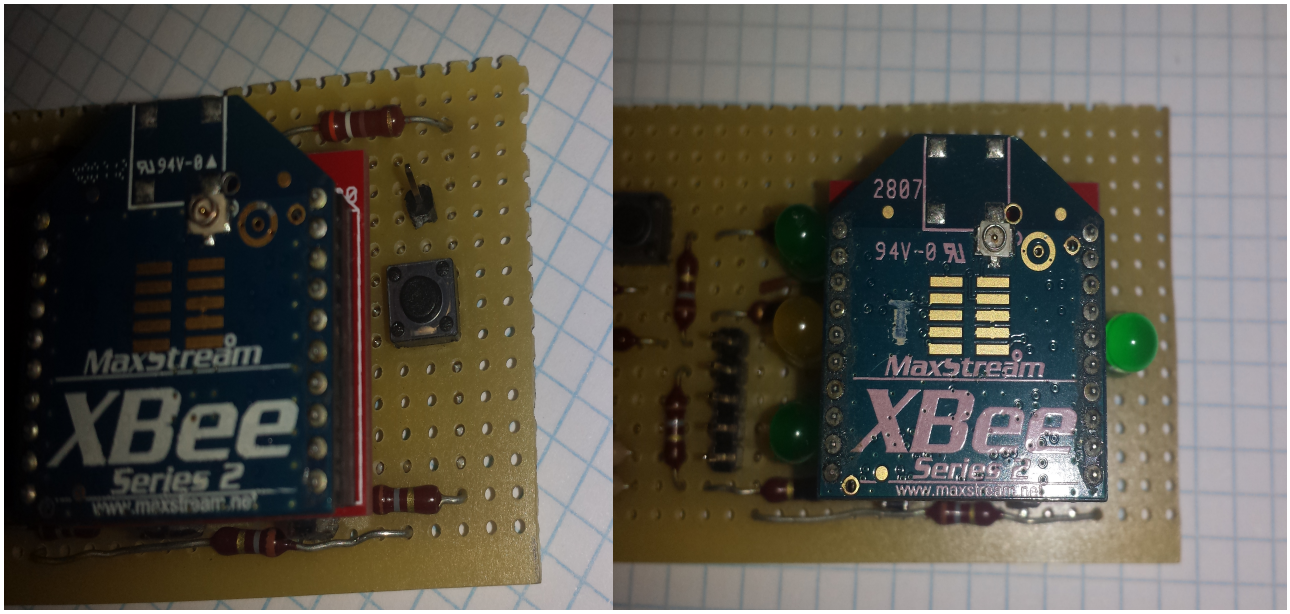

Figura 33. Detalle del pulsador del nodo remoto (izquierda) y detalle del LED de confirmación del coordinador (derecha).

Los diodos LED de diagnóstico a la izquierda de cada Xbee permiten identificar rápidamente si se está transmitiendo o recibiendo información serial o por radiofrecuencia. Los diodos de los pines DOUT y RSSI serán verdes, mientras que el diodo del pin DIN será amarillo. El diodo de DOUT estará encendido siempre que esté conectado el módulo XBee a la alimentación, mientras que el diodo de DIN sólo estará encendido si se tiene conectado al menos al módulo de comunicación con el MAX3232. Por otro lado, el diodo RSSI se encenderá durante unos segundos siempre que se reciba un mensaje por radiofrecuencia.

3.5. Configuración de los módulos XBee

3.5.1. Introducción

Los módulos XBee cuentan con un firmware que especifica el papel que cumple el dispositivo en la red ZigBee. Además, tienen una serie de parámetros que especifican diferentes opciones relacionadas con el módulo. Para poder configurar el firmware y los parámetros, se necesita el programa X-CTU anteriormente introducido.

La interfaz del X-CTU cuenta con cuatro pestañas:

- PC Settings: en esta pestaña se introducen los parámetros necesarios para la comunicación entre el XBee y el PC: velocidad de transmisión, control de flujo, paridad, etc. La opción *Test/Query* permite realizar una prueba de conexión con el XBee al otro lado del puerto serial y averiguar su firmware y el tipo de hardware que es, mostrando por pantalla un error en caso de que falle. Adicionalmente permite la creación de puertos seriales del usuario.

- Range Test: aquí se incluye un test de rango para probar la distancia efectiva de transmisión de información de un dispositivo XBee mediante una pasarela o *gateway* proporcionada por el fabricante.

- Terminal: esta es la pestaña donde se tiene el terminal de envío y recepción de paquetes de datos seriales. Se puede visualizar el estado de las líneas seriales de estado, forzar el estado digital de algunas de esas líneas, abrir o cerrar el puerto serial, construir un paquete de datos para ser enviado (tanto en formato ASCII como en hexadecimal), limpiar la pantalla de datos y mostrar u ocultar los datos en formato hexadecimal.

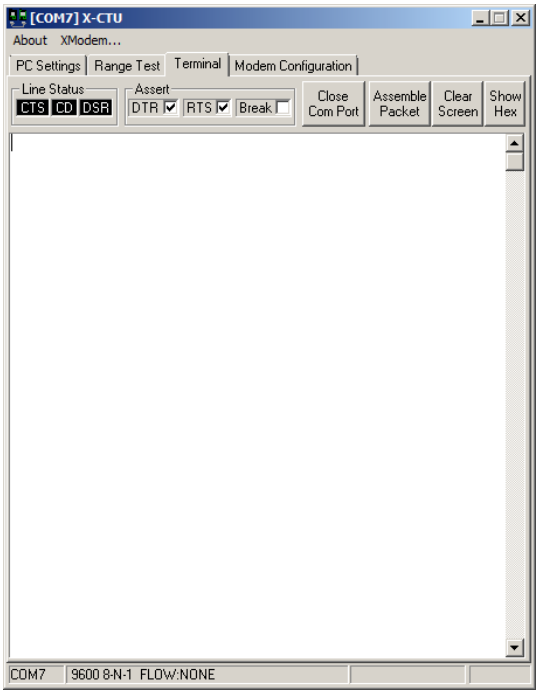

Figura 34. Pestaña de terminal del programa X-CTU.

- Modem Configuration: esta pestaña es donde se recogen los parámetros del módulo y su firmware. Mediante las opciones se pueden leer, escribir o restaurar los parámetros, actualizar el firmware del sistema, ver los valores por defecto de cada parámetro, guardar o cargar perfiles personalizados con los parámetros preferidos del usuario, modificar el firmware del dispositivo según el tipo de hardware que sea, visualizar un mensaje de ayuda para cada parámetro y cuáles han sido modificados, guardados o permanecen en sus valores predeterminados, etc.

3.5.2. Configuración del nodo remoto

El nodo remoto es un dispositivo XB24-B configurado como router o dispositivo final en modo transparente (ZNET 2.5 ROUTER/END DEVICE AT). La versión de firmware de este modo es la

1.247. Los parámetros que se han modificado respecto a los predeterminados se enumeran a continuación, agrupados según aparecen en el X-CTU.

En el grupo de parámetros de red, se ha modificado la ID de la red PAN al valor 1234 (ATID 1234), y se ha habilitado la verificación de canal (ATJV 1). Si esta verificación está activa y si el dispositivo funciona como router, éste comprobará si existe un coordinador en el mismo canal después de unirse para asegurarse de que está operando en un canal válido, abandonando el canal si no encuentra ninguno.

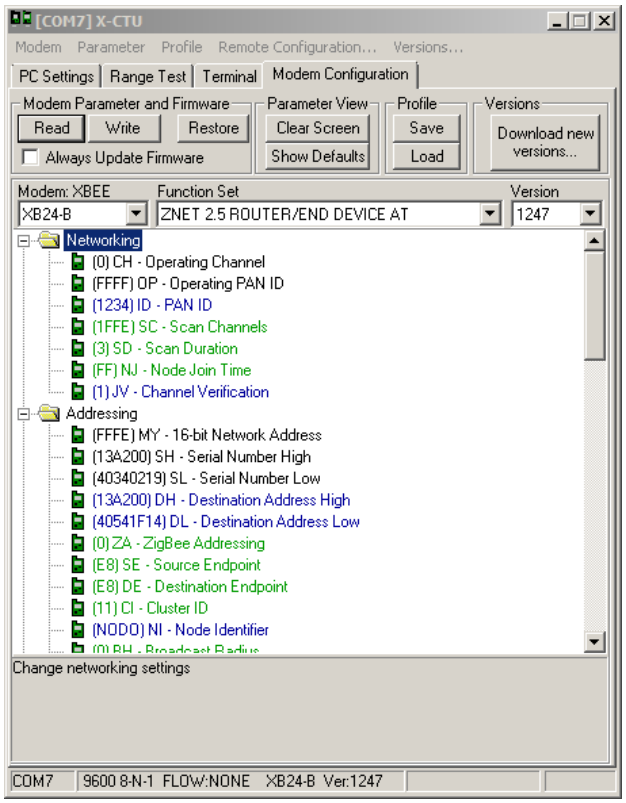

Figura 35. Configuración de algunos de los parámetros del nodo remoto.

En el apartado de direccionamiento, se ha indicado la dirección de destino como la dirección de 64 bits del coordinador (ATDH 13A200, ATDL 40541F14), y se ha identificado el nodo con la cadena de caracteres NODO (ATNI NODO). Los grupos de opciones *RF Interfacing* y *Security* se han dejado en los parámetros por defecto, mientras que en el siguiente grupo (*Serial interfacing*) se ha configurado la salida digital D7 como deshabilitada (ATD7 0). En los ajustes de entradas y salidas se ha asignado la E/S digital 4 como una entrada digital (ATD4 3).

Por último, en el grupo de parámetros de muestreo de entradas/salidas, se modificará el parámetro de detección de cambio de entradas o salidas digitales (*Digital I/O Change Detection*). El valor de este parámetro es una máscara de bits que permite indicar qué E/S digitales provocarán el envío de un mensaje de estado al coordinador en una trama API con el identificador 0x92 (*ZigBee IO Data Sample Rx Indicator*), cada vez que se produzca un cambio en su estado lógico. En dicha trama informará de quién envía el mensaje, qué entradas digitales está monitorizando y el estado en el que se encuentran. La máscara se compone asignando un 1 a las entradas o salidas a monitorizar, interpretando el resultado como un número hexadecimal [7]. La Tabla 9 recoge la forma de

construir esta máscara.

Tabla 9. Composición de la máscara de bits								
1er byte:	N/A	N/A	N/A	D12	D11	D10	N/A	N/A
2º byte:	D7	D6	D5	D4	D3	D2	D1	D0

Así, como en este caso se desea monitorizar únicamente la entrada digital 4 previamente configurada, se tendrá un 1 en el bit marcado por D4. El resultado será 00000000 00010000, que en hexadecimal corresponde a 0x10. Por tanto, el parámetro de la opción será 10 (ATIC 10). Las entradas digitales también tienen una máscara, que se construye e interpreta del mismo modo, pero al sólo disponerse de 4 entradas sólo emplea los 4 bits menos significativos de un único byte.

Con esto, una vez configurados los parámetros y guardados los cambios (opcionalmente también en un perfil para poder realizar *backups*) se completa la configuración del nodo remoto. A partir de este momento el nodo se conectará a la misma red que el otro módulo, y enviará un mensaje de estado a la dirección del coordinador cada vez que se oprima el botón conectado al pin 11, lo que será empleado por el programa informático para saber que ha ocurrido un suceso.

3.5.3. Configuración del coordinador

El coordinador, al igual que su homólogo, es un dispositivo XB24-B, aunque en esta ocasión configurado como coordinador en modo API (ZNET 2.5 COORDINATOR API), versión de firmware 1.147.

Al igual que el nodo, la ID de la red PAN ha sido asignada al valor 1234. La dirección de 64 bits de destino, en cambio, se ha configurado como la del nodo (0013A200 40340219), y se ha identificado con la cadena de caracteres COORD. El resto de parámetros se mantiene en sus valores predeterminados, ya que no necesita empezar con ningún pin configurado de algún modo especial. Con estos cambios, se logrará que el coordinador y el nodo estén en la misma red PAN y se comuniquen mutuamente.

3.6. Programa controlador en C++

Una vez configurados los XBee y montado el circuito general, es el momento de implementar el programa que se encargará de controlar la red sensorial. Tal y como está planteado el sistema, el coordinador estará conectado a un PC mediante el módulo de comunicación, de forma que el ordenador actúe de controlador y, mediante los datos recibidos del nodo remoto, tome la decisión de enviar uno u otro comando a la red.

El lenguaje de programación en el que se creó el programa es C++, por sus funcionalidades en cuanto a tratamiento de objetos y mejoras respecto al C estándar, permitiendo manejar ciertos elementos de la comunicación como clases con atributos. El entorno de programación fue un sistema operativo GNU/Linux, que incluye las librerías necesarias para abordar este problema sin la necesidad de añadir ningún programa adicional. Adicionalmente, para administrar las distintas versiones del programa, seguir su histórico y mantener copias de seguridad de todos los cambios, se utilizó el sistema de control de versiones GIT y la plataforma GitHub [12] como repositorio.

3.6.1. Descripción de tramas API empleadas

Antes de describir el programa en C++, es conveniente describir en detalle las tramas que se manejan en el programa, tanto recibidas como enviadas. A la hora de trabajar con estas tramas, se ha empleado como ayuda, además de los correspondientes documentos y manuales ya citados anteriormente y recogidos en la bibliografía, una herramienta on-line de Digi [13] que permite construir o interpretar tramas API según la ID de trama, de forma interactiva y con texto de ayuda sobre el significado de cada fragmento del mensaje, además de calcular la longitud y el CRC y crear el paquete en texto plano para poder ser copiado e introducido en el X-CTU.

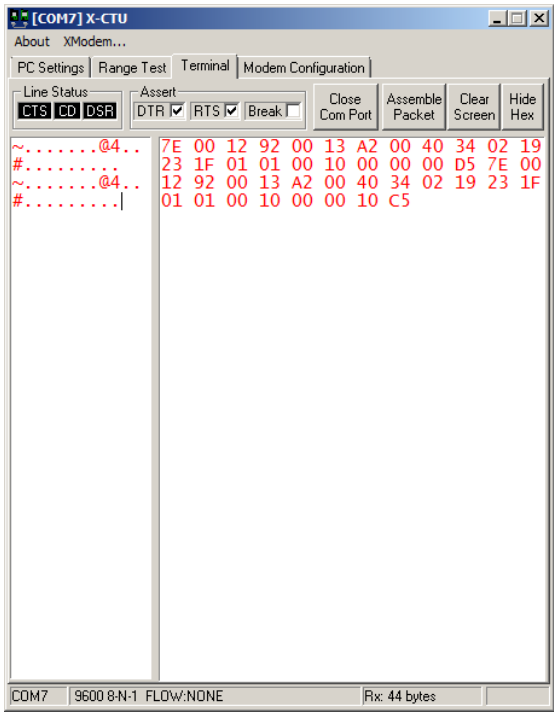

Figura 36. Mensajes de estado recibidos por el coordinador.

3.6.1.1. Mensajes de estado

En el ejemplo mostrado en la Figura 36, se pueden observar dos mensajes de estado distintos recibidos en el coordinador tras una pulsación del botón conectado al pin 11 del nodo remoto. El primero de ellos indica el cambio de la entrada digital a estado lógico bajo al pulsarse el botón, mientras que el segundo indica el retorno al estado lógico alto cuando se suelta el botón. En concreto, la estructura de los mensajes de estado es la que se muestra en la Tabla 10. La única diferencia entre ambos es el valor de los tres últimos bytes.

Tabla 10. Estructura de mensajes de estado

Bytes	Contenido (hexadecimal)	Significado
1	7E	Encabezado.
2 – 3	00 12	Longitud del mensaje.
4	92	ID de trama correspondiente al mensaje de muestreo de datos.
5 – 12	00 13 A2 00 40 34 02 19	Dirección de 64 bits del nodo remoto.
13 – 14	23 1F	Dirección de 16 bits del nodo remoto en la PAN.
15	01	Opciones de recepción. 0x01 indica paquete reconocido. Hay otros valores posibles y combinaciones de los mismos.
16	01	Número de muestras, siempre viene fijado a 1.
17 – 18	00 10	Máscara de bits de entradas digitales monitorizadas (ver Tabla 9). En este caso, se indica la monitorización de la entrada digital 4.
19	00	Máscara de bits de entradas analógicas monitorizadas. En este caso no se monitoriza ninguna, por lo que está a cero.
20 – 21	00 00 / 00 10	Estado de las entradas digitales. Estos bytes se interpretan del mismo modo que la máscara de bits de entradas monitorizadas (Tabla 9). En este caso, en el primer mensaje indica que todas están a cero, mientras que en el segundo que la entrada digital 4 está a 1.
22	D5 / C5	CRC. El primer valor es el CRC del primer mensaje, el segundo valor el del segundo.

3.6.1.2. Comando AT remoto

Cuando se recibe una trama con ID 0x92 como la anterior, el programa procederá a enviar al XBee coordinador una trama del tipo comando AT remoto, cuya ID es 0x17. Este mensaje hará que el coordinador envíe al nodo remoto la orden de que configure su salida digital x-ésima como salida digital a nivel bajo (ATDx 4) o a nivel alto (ATDx 5) en función de lo que haya sucedido. La estructura de un mensaje de este tipo es como se recoge en la Tabla 11.

Tabla 11. Estructura de trama de comando AT remoto

Bytes	Contenido (hexadecimal)	Significado
1	7E	Encabezado.
2 – 3	00 10	Longitud del mensaje.
4	17	ID de trama correspondiente a la petición de comando AT remoto.
5	00	Si está a cero, no se está solicitando un mensaje de confirmación por parte del nodo remoto, y si está a 1, el nodo responderá con una trama de ID 0x97 (*Remote Command Response*). Esto se emplea para uno de los modos de funcionamiento del programa, que utiliza esta respuesta para encender un LED de confirmación.
6 – 13	00 13 A2 00 40 34 02 19	Dirección de 64 bits del nodo remoto.
14 – 15	FF FE	Dirección de 16 bits del nodo remoto en la PAN. El valor FF FE indica que esta dirección es desconocida.
16	02	Aplicar los cambios al nodo remoto. Si no se asigna valor (0x00), se debe enviar el comando AC, o la secuencia de comandos WR + FR si se desean conservar los cambios tras reinicios.
17 – 18	44 30	Valores hexadecimales correspondientes a los caracteres ASCII que identifican el comando. En este caso, 44 30 equivale a los caracteres D0.
19	05	Parámetro asociado al comando anterior.
20	2B	CRC.

3.6.1.3. Respuesta a comando AT remoto

Si el byte 5 de respuesta de la trama anterior está a 1, el nodo remoto enviará al coordinador un mensaje de respuesta con la ID 0x97. Este mensaje tiene una estructura muy similar al mensaje enviado, como se observa en la Figura 37.

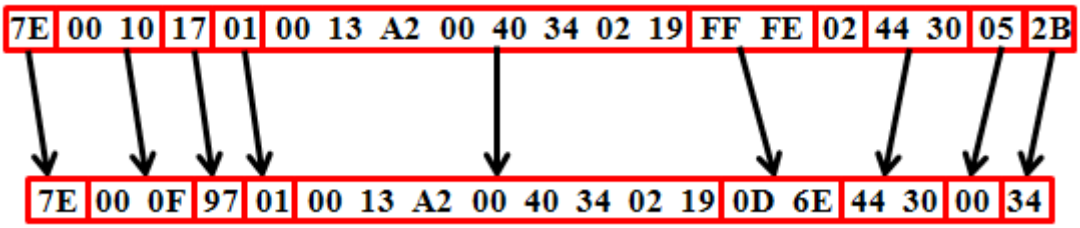

Figura 37. Comparativa entre el mensaje de comando AT remoto y la respuesta a dicha trama. Los valores están en hexadecimal.

Las diferencias principales, además de la ID, la longitud y el CRC, son que la respuesta tendrá la dirección completa del nodo remoto, no necesita el byte de confirmación de cambios y el parámetro del comando AT estará vacío.

3.6.1.4. Comando AT

Otro mensaje que se emplea en el programa es la trama de ID 0x08 (comando AT local). Simplemente incluye el comando AT y si se desea confirmación o no. En este caso, se emplea para encender o apagar el LED del coordinador como confirmación de que el nodo remoto tiene el interruptor cerrado.

3.6.2. Definición de clases y métodos

Para realizar el tratamiento completo de los paquetes de datos y la conexión del dispositivo con el ordenador, se han creado una serie de archivos en C++ que implementan diferentes funcionalidades necesarias. Para empezar, un corto archivo llamado `defines.hpp` incluye la definición formal de una variable de tipo byte, el cual, al tratarse de la unidad básica de datos con la que se trabajará, se emplea en todos los demás archivos. Para volcar estos bytes por pantalla, los archivos `VectorByte.*` implementan una sobrecarga del operador `"<<"` empleado con los operadores y comandos de la librería `iostream`, que permitirá que si se desea mostrar en pantalla el contenido

de una variable de tipo byte o un vector de bytes, el contenido de la misma se represente siguiendo un formato fácil de interpretar similar al de escritura manual.

La conexión a los dispositivos se administra mediante una clase abstracta llamada `Conexion` definida en el archivo `Conexion.hpp`. En ella, se especifican de forma virtual los métodos de lectura, lectura con *timeout* y escritura de datos, así como los de apertura y cierre de la conexión, de forma que en otros archivos más específicos se definan completamente los mismos. Los métodos de lectura y escritura utilizan variables de tipo byte o vectores de bytes, para lo cual es necesario emplear la definición formal aportada en el `defines.hpp`. La particularización de estos métodos de lectura y escritura se realiza en los archivos `Serial_Conexion.*`, una clase hija de `Conexion` que hereda todos sus métodos y los define, además de implementar métodos nuevos que permitan cambiar la velocidad de transferencia en bps. Siguiendo la estructura típica de un programa en C++, la declaración de los métodos se lleva a cabo en el archivo `Serial_Conexion.hpp`, mientras que la definición y el funcionamiento de los mismos se incluye en el archivo `Serial_Conexion.cpp`.

Creada la conexión, el siguiente paso para implementar la comunicación con el dispositivo es crear una clase que permita interpretar y tratar las tramas que utiliza el modo API en que está configurado el coordinador. Dicha clase se ha bautizado como `Dialogo_API`, definida en los archivos `Dialogo_API.*`. Entre sus métodos se incluyen el cálculo y comprobación del CRC de los mensajes, la construcción de paquetes de datos, la recepción de paquetes o distintos métodos de comunicación clasificados por el ID de trama. No obstante, esta clase únicamente muestra por pantalla la información de los paquetes recibidos. Para implementar la lógica de control deseada, se ha creado la subclase `dialogoVentilador`, heredada de `Dialogo_API`, en la cual la única modificación realizada es eliminar el *timeout* o tiempo máximo de espera a la recepción de paquetes, para poder esperar indefinidamente el mensaje recibido tras la pulsación del botón del nodo remoto. Por ello, los archivos `dialogoVentilador.*` sólo incluyen este método de lectura de los paquetes recibidos, ya que los demás se emplean literalmente de la clase madre. Los métodos que contiene, si encuentran el paquete adecuado, enviarán el comando para modificar la salida correspondiente. El nombre de estos archivos modificados tiene su origen en que inicialmente el programa controlaba el ventilador de ordenador, aunque su funcionamiento es extensible a otros sistemas.

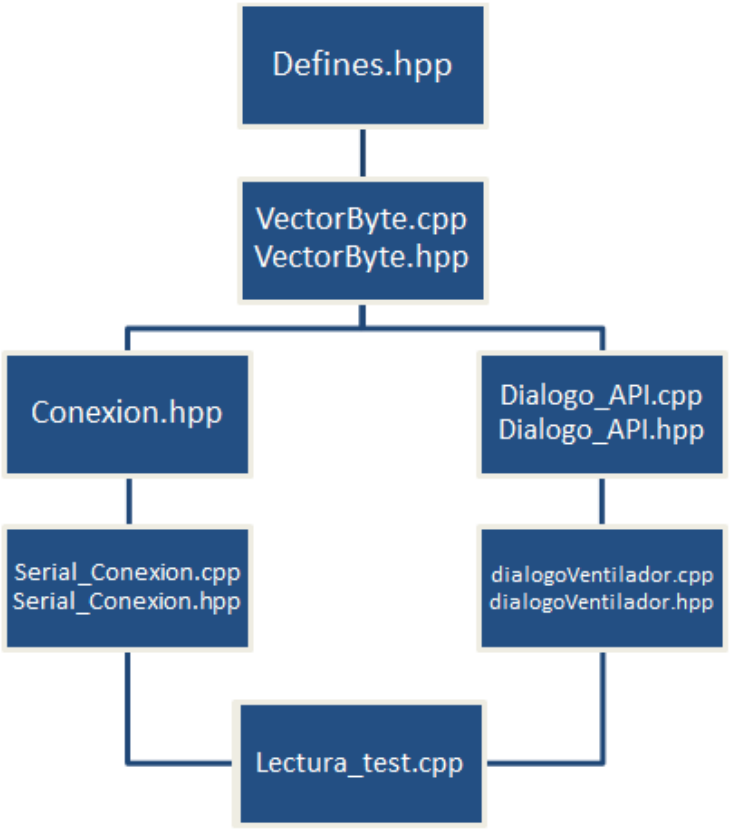

Figura 38. Estructura jerárquica de dependencias del código.

3.6.3. Programa de control

Ahora que todos los elementos de la comunicación están definidos, se crea el programa que recibe, administra e interpreta los mensajes y envía respuestas en función de los sucesos que hayan ocurrido. Para ello, cuenta con tres modos de funcionamiento similares: un primer modo en el que una pulsación del botón pondrá a 1 la salida digital del XBee, mientras que una segunda pulsación lo regresará al valor lógico 0; un segundo modo en el que al presionar el botón, además de poner a 1 la salida digital, comience una cuenta atrás de 10 segundos, tras la cual automáticamente se volverá al estado lógico bajo; y un tercer modo idéntico al primero pero que incluye el encendido de un LED de confirmación en el coordinador, de forma que remotamente se pueda saber el estado de la salida digital. Dicho programa se llama `Lectura_test.cpp`, y al ser un elemento principal del sistema domótico se describirá más en detalle a continuación. En cualquier caso, todo el código de los archivos antes mencionados se incluirá íntegramente como un anexo del presente informe.

En primer lugar, se incluyen las librerías y dependencias que requiere el programa para su funcionamiento.

```
#include <stdlib.h>
#include <iostream>
#include <iomanip>
#include <unistd.h>
#include <string>
#include "defines.hpp"
#include "Serial_Conexion.hpp"
#include "dialogoVentilador.hpp"
/* Macro para crear vector a partir de array de bytes*/
#define BVEC(v) VectorByte( v, v + sizeof(v)/sizeof(byte) )
```

La librería `stdlib.h` define varias funciones de propósito general, como el manejo de memoria dinámica. `iostream` se encarga de definir los objetos de flujo de entrada y salida estándar, permitiendo imprimir en pantalla mensajes o capturar datos introducidos por el usuario mediante el teclado, mientras que `iomanip` proporciona manipuladores paramétricos como `setfill`, que permite fijar el parámetro que empleará la función `setw` para volcar por pantalla un carácter (el especificado en `setfill`) tantas veces como se defina en `setw`. Por otro lado, la librería `unistd.h` incluye la definición de constantes y tipos simbólicos diversos, y declara varias funciones diversas. Entre ellas, se encuentra la función `sleep` para mantener el sistema a la espera durante un cierto tiempo. Esta librería es exclusiva de sistemas UNIX, por lo que el programa no funcionará en sistemas operativos que no sean GNU/Linux. Por último, se incluye la librería `string` para poder administrar cadenas de caracteres.

A continuación, se incluyen los tres otros archivos que definen y describen las clases o métodos que este código va a emplear, ya sea la conexión serial (`Serial_Conexion.hpp`) o el manejo de las tramas del modo API del coordinador (`dialogoVentilador.hpp`). Al realizar la construcción del programa ejecutable, estos programas solicitarán a su vez sus archivos .cpp y sus dependencias propias, de forma que al final en el archivo ejecutable está incluido todo lo necesario. Por último, se declara una macro que permite construir los vectores de bytes a partir de variables de tipo byte individuales.

La siguiente función, declarada antes de la función principal, se encarga de preguntar al usuario del programa si desea cerrarlo o volver a ejecutar algún modo del mismo.

```
// Función para hacer la pregunta de salida al final de la ejecución de cada modo del
// programa
int preguntaSalir() {
   // Creamos las variables fin (la que se devolverá), finC para leer la entrada del
   // usuario y OK para
   // mantenernos en el bucle de petición de mensaje hasta que este sea S o N.
   int fin = 0;
   std::string finC, si = "S", no = "N";
   int OK = 0;
   // El bucle pedirá la entrada de usuario. Si esta es S la función devolverá un 1, el
   // cual hará que en
   // el main se fuerce el apagado del programa; si es N, la función devolverá un 0 y
   //  por tanto el programa seguirá ejecutándose.
   do{
   std::cout << "¿Desea salir? S/N" << std::endl;
   std::cin >> finC;
   if (finC.compare(si) != 0 && finC.compare(no) != 0) {
     std::cout << "ERROR; Escribe S para salir o N para volver a ejecutar" <<std::endl;
     OK = 0;
     }
   else if (finC.compare(si) == 0) { fin = 1; OK = 1; }
   else if (finC.compare(no) == 0) { fin = 0; OK = 1; }
   }while( OK != 1);
   return fin;
}
```

La función imprimirá por pantalla el mensaje solicitando la introducción de una S mayúscula para salir del programa, o bien de una N mayúscula para regresar al inicio del programa, permitiendo al usuario volver a ejecutar un modo del mismo. En caso de que la introducción del usuario no se corresponda con cualquiera de estas dos letras, el programa informará de este dato al usuario y volverá a solicitar la introducción. Para esto, es útil el método `compare()` incluido en la librería `string`, así como la propia clase `string`. Inicialmente se había implementado usando simplemente variables del tipo carácter (`char`), pero se cambió a cadenas de caracteres al comprobar que la introducción (voluntaria o no) de más de un carácter por parte del usuario llevaba al programa a un error de desbordamiento. Por otro lado, el motivo de implementar la pregunta en una función aparte radica en el hecho de que se realiza más de una vez en el código, al menos una vez por cada modo, permitiendo por tanto la modularidad y facilidad de modificación de la misma.

Seguidamente, se introduce la función principal del programa. Tras las comprobaciones iniciales y el procesamiento de los parámetros de entrada de la misma, y una vez definido el puerto serial y la velocidad de transmisión, se procede a abrir una conexión serial con estos parámetros.

```
// Abrimos una nueva conexión serial
Serial_Conexion *serc = new Serial_Conexion();
std::cout<<"Usando puerto "<<puerto<<" con velocidad "<<baudios<<std::endl;
serc->open_port( puerto, baudios);

dialogoVentilador dip = dialogoVentilador(serc);
```

Por pantalla se mostrará un mensaje que informe del puerto empleado y de la velocidad de transmisión. Se crea un puntero que apunte a una clase de conexión serial creada dinámicamente, aprovechando así las virtudes de C++. Cuando el puerto esté abierto, se envía la información mediante el puntero a la clase que administra la comunicación en modo API.

Antes de introducir el bucle principal del programa, se declaran las variables comunes a todo el mismo:

```
// Estos son los mensajes de salida: el mensaje A encenderá el ventilador, y el B lo
// apagará.
byte mensalA[] = { 0x17, 0x00, 0x00, 0x13, 0xA2, 0x00, 0x40, 0x34, 0x02, 0x19, 0xFF,
0xFE, 0x02, 0x44, 0x30, 0x05 };
VectorByte mensalence = BVEC(mensalA);

byte mensalB[] = { 0x17, 0x00, 0x00, 0x13, 0xA2, 0x00, 0x40, 0x34, 0x02, 0x19, 0xFF,
0xFE, 0x02, 0x44, 0x30, 0x04 };
VectorByte mensalapa = BVEC(mensalB);

// Estas variables sirven para salir de los distintos bucles que hay dentro del main
// (es el caso de exit o fin), para indicar si ha ocurrido un error y el botón no
// respondía al apretarlo (error) y para escoger el modo del programa (modo).
int exit = 0;
bool error = false;
int modo = 0;
int fin = 0;
```

En primer lugar se declaran los mensajes que se enviarán al nodo remoto cuando se reciba el mensaje de estado al pulsar el botón en el mismo. Se puede comprobar que se trata de un array de bytes que constituyen los bytes de la trama API que se envía al coordinador, array que luego se transforma en un vector de bytes empleando la macro definida en el encabezado del programa. En ambos casos, tanto para el mensaje que enciende el ventilador como para el que lo apaga, se sigue la estructura anteriormente mencionada, sin incluir el encabezado 0x7E, la longitud del mensaje o el CRC, pues todo esto es incorporado al mensaje mediante los métodos definidos en `Dialogo_API`. Las demás variables declaradas, como se recoge en el comentario, sirven para salir de los distintos bucles, informar de un error de comunicación o indicar el modo a probar.

Se inicia finalmente el bucle que realizará recursivamente los modos seleccionados hasta que se introduzca la opción de salida del programa. La descripción de las distintas operaciones realizadas en el bucle se incluye en forma de comentarios dentro del código.

```
do {
   // Se pregunta qué modo se quiere probar.
   std::cout << "Escoge el modo del programa:\n1) Presionar dos veces para apagar\n2)
   Apagado tras 10 segundos\n3) Modo 1 con LED de estado de ventilador en coordinador" <<
   std::endl;
   std::cin >> modo;
   // Por si acaso, se vuelve a anular la variable fin por si tras ejecutarse un modo
   // esta ha cambiado de valor.
   fin = 0;
   // Selector de modo
   switch(modo) {
       case 1: {
          std::cout << "Modo 1: presionar dos veces para apagar" <<std::endl;
          do{
              // Se espera un tiempo para asegurar que el botón funciona.
              std::cout << "Espera..." << std::endl;
              sleep(2);
              std::cout << "Presiona el botón... " << std::endl;

              // Se lee el mensaje de entrada y se guarda en la primera variable de
              // entrada.
              VectorByte *menent1 = dip.recibePaquete();
              // Para imprimir el mensaje recibido en pantalla, descomentar la siguiente
              // línea.
              // std::cout << *menent1 << std::endl;

              // Se comprueba si el mensaje no es el esperado, en cuyo caso se sale del
              // programa.
              if( menent1->at(17) != 0x00) {
                  std::cout << std::endl << "El botón no está funcionando correctamente"
                  << std::endl;
                  exit = 2;
                  error = true;
                  break;
              }

              // Si el mensaje recibido es correcto y se está en la primera pulsación, se
              // enciende el ventilador.
              else if( menent1->at(17) == 0x00 && exit == 0) {
                  std::cout << "OK" << std::endl;
                  dip.enviaPaquete(mensalence);
              }

              // Si el mensaje recibido es correcto y se está en la segunda pulsación, se
              // apaga el ventilador.
              else if( menent1->at(17) == 0x00 && exit == 1) {
                  std::cout << "OK" << std::endl;
```

```
            dip.enviaPaquete(mensalapa);
        }

        // Al soltar el botón se recibe otro mensaje, el cual se guardará en otra
        // variable para evitar que este interfiera con el funcionamiento
        // del programa.
        std::cout << "Suelta el botón... " << std::endl;

        VectorByte *menent2 = dip.recibePaquete();
        // Para imprimir en pantalla este segundo mensaje, descomentar la
        // siguiente línea.
        // std::cout << *menent2 << std::endl;

        // Se comprueba si el mensaje recibido es el esperado, en caso contrario
        // se sale del programa.
        if( menent2->at(17) != 0x10) {
            std::cout << std::endl << "El botón no está funcionando correctamente"
            << std::endl;
            exit = 2;
            error = true;
        }

        else {
            std::cout << "OK" << std::endl;

            // Si los mensajes fueron correctos, se borran de la memoria para
            // tenerlos limpios en la siguiente ejecución del programa.

            menent1->clear();
            menent2->clear();

            // Se esperan 3 segundos si estamos en la primera iteración, si no
            // se espera un segundo antes de salir.
            if(exit < 1) sleep(3);
            else sleep(1);

            // Sea la iteración que sea, se incrementa en uno la variable exit.
            // Cuando exit sea igual a 2 significará que hemos encendido y
            // apagado el ventilador y se saldrá del bucle.
            exit++;
        }
    }while(exit < 2);

    // Se pregunta si se quiere salir en caso de que no se haya salido por un
    // error. Si hubo un error, se envía un mensaje de apagado del ventilador
    // y se termina el programa.
    if(error == false) fin = preguntaSalir();
    if(error == true) {
    fin = 1;
    dip.enviaPaquete(mensalapa);
}
```

```
    break;
}

    case 2: {
        std::cout << "Modo 2: apagado tras 10 segundos" <<std::endl;
        std::cout << "Espera..." << std::endl;
        sleep(2);
        std::cout << "Presiona el botón... " << std::endl;

        // Como en el caso 1, se guarda el primer mensaje en menent1 y se comprueba
        // que este funcione correctamente. En caso contrario, imprime un mensaje
        // de error y se termina el programa.
        VectorByte *menent1 = dip.recibePaquete();

        if( menent1->at(17) != 0x00) {
            std::cout << std::endl << "El botón no está funcionando correctamente" <<
            std::endl;
            fin = 1;
            error = true;
            break;
        }

        else {
            std::cout << "OK, puedes soltarlo" << std::endl;
            dip.enviaPaquete(mensalence);
        }

        // Aunque no se utiliza, guardamos el otro mensaje de cambio de estado para
        // que no interfiera en caso de que volvamos a querer ejecutar el caso 2.
        // Del mismo modo, se comprueba que este sea correcto por si está otra vez
        // funcionando el botón mal.
        VectorByte *menent2 = dip.recibePaquete();

        if( menent2->at(17) != 0x10) {
            dip.enviaPaquete(mensalapa);
            std::cout << "El botón no está funcionando correctamente" << std::endl;
            fin = 1;
            error = true;
            break;
        }
        // Si todo está correcto, se esperarán 10 segundos y se enviará el mensaje
        // de apagado.
        else {
            sleep(10);
            dip.enviaPaquete(mensalapa);
            std::cout << "Ventilador apagado tras 10 segundos" <<std::endl;

            // Se borran los mensajes de entrada para volverlos a usar en otras
            // iteraciones.

            menent1->clear();
            menent2->clear();
```

```
        }
    fin = preguntaSalir();
    break;

}

    case 3: {
        std::cout << "Modo 3: Modo 1 con LED de estado de ventilador en coordinador"
        <<std::endl;
        std::cout << "En este modo, cuando el ventilador esté funcionando se encenderá un
        LED verde en el coordinador." <<std::endl;

        // Los mensal*b son iguales a los del modo 1, con la diferencia de que se
        // solicita respuesta al nodo. mensalD y mensalE son los mensajes que
        // encenderán y apagarán el led del coordinador.
        byte mensalAb[] = { 0x17, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x40, 0x34, 0x02, 0x19,
        0xFF, 0xFE, 0x02, 0x44, 0x30, 0x05 };
        VectorByte mensalenceLED = BVEC(mensalAb);

        byte mensalBb[] = { 0x17, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x40, 0x34, 0x02, 0x19,
        0xFF, 0xFE, 0x02, 0x44, 0x30, 0x04 };
        VectorByte mensalapaLED = BVEC(mensalBb);

        byte mensalD[] = { 0x08, 0x00, 0x44, 0x34, 0x05 };
        VectorByte mensalLEDon = BVEC(mensalD);

        byte mensalE[] = { 0x08, 0x00, 0x44, 0x34, 0x04 };
        VectorByte mensalLEDoff = BVEC(mensalE);

        VectorByte *menent3;

        do{

            std::cout << "Espera..." << std::endl;
            sleep(2);
            std::cout << "Presiona el botón... " << std::endl;

            VectorByte *menent1 = dip.recibePaquete();
            // std::cout << *menent1 << std::endl;

            if( menent1->at(17) != 0x00) {
                std::cout << std::endl << "El botón no está funcionando correctamente" <<
                std::endl;
                exit = 2;
                error = true;
                break;
            }

            else if( menent1->at(17) == 0x00 && exit == 0) {
                std::cout << "OK" << std::endl;
                dip.enviaPaquete(mensalenceLED);
```

```
            // Se recibe el mensaje de confirmación de arranque y se enciende el LED.
            menent3 = dip.recibePaquete();
            //  std::cout << *menent3 << std::endl;
            // Se comprueba que tenga la estructura esperada.
            if (menent3->at(12) == 0x44 && menent3->at(13) == 0x30)
            dip.enviaPaquete(mensalLEDon);

        else {
            std::cout << "Ha ocurrido un error, saliendo del programa" << std::endl;
            exit = 2;
            error = true;
            break;
        }
    }

    else if( menent1->at(17) == 0x00 && exit == 1) {
        std::cout << "OK" << std::endl;
        dip.enviaPaquete(mensalapaLED);

        // Se recibe el mensaje de confirmación de parada y se apaga el LED.
        menent3 = dip.recibePaquete();
        // std::cout << *menent3 << std::endl;
        // Se comprueba que tenga la estructura esperada.
        if (menent3->at(12) == 0x44 && menent3->at(13) == 0x30)
        dip.enviaPaquete(mensalLEDoff);

     else {
       std::cout << "Ha ocurrido un error, saliendo del programa" << std::endl;
       exit = 2;
       error = true;
       break;
     }
   }

   std::cout << "Suelta el botón... " << std::endl;

   VectorByte *menent2 = dip.recibePaquete();
   // std::cout << *menent2 << std::endl;

   if( menent2->at(17) != 0x10) {
        std::cout << std::endl << "El botón no está funcionando correctamente" <<
        std::endl;
        exit = 2;
        error = true;
   }


   else {
        std::cout << "OK" << std::endl;

       menent1->clear();
       menent2->clear();
```

```
          menent3->clear();

          if(exit < 1) sleep(3);
          else sleep(1);

          exit++;
       }
    }while(exit < 2);

    // En caso de error se apagan el ventilador y el LED del coordinador.
    if(error == false) fin = preguntaSalir();
    if(error == true) {
      fin = 1;
      dip.enviaPaquete(mensalapa);
      dip.enviaPaquete(mensalLEDoff);
    }

    break;
}

    default:

       std::cout << "Modo no implementado" << std::endl;

       fin = preguntaSalir();

}

}while(fin != 1);


if(error == false) std::cout << "Todo OK, fin del programa" <<std::endl;
// Al finalizar el programa se apaga el ventilador por si ha quedado encendido
// accidentalmente.
if(fin == 1) dip.enviaPaquete(mensalapa);
```

Algunas de las funciones de este bucle fueron añadidas a medida que se probaba el sistema y se comprobaban los errores que iban surgiendo. Así, se necesitó almacenar los dos mensajes de estado en variables distintas porque causaban problemas de lectura, y además dichas variables debían limpiarse al final de cada iteración porque si no el programa interpretaba que ya había recibido mensajes y los daba por erróneos. Otra funcionalidad que se añadió fue el mensaje de error en caso de que fallase la comunicación, ya que en ocasiones el botón no producía ningún mensaje de estado. Esto puede ser debido a que los dispositivos carecen de antena y su rango de alcance se ha visto seriamente mermado por ello, o bien puede ser un problema a la hora de realizar la comunicación serial. En cualquier caso, si no se comporta como se espera, el mensaje de error hará salir al sistema del bucle automáticamente.

Relacionado con este último problema, una solución que lo subsana temporalmente consiste en enviar el comando AT de reinicio de software al nodo. Más concretamente, mediante una trama 0x17, enviar el comando FR, que permite ejecutar dicho reinicio. El comando no necesita parámetros, así que el mensaje tiene un byte menos de longitud.

```
// Esto permitirá enviar al nodo un mensaje AT de reinicio de software para solucionar
// fallos en caso de que se hubiesen presentado en el programa.
if(error == true) {
    std::string confirmaReset, si = "S", no = "N";
    int OK = 0;

    byte mensalC[] = { 0x17, 0x01, 0x00, 0x13, 0xA2, 0x00, 0x40, 0x34, 0x02, 0x19, 0xFF,
     0xFE, 0x02, 0x46, 0x52 };
    VectorByte mensalreset = BVEC(mensalC);

    do{
       std::cout << "¿Resetear nodo? S/N" <<std::endl;
       std::cin >> confirmaReset;
       if (confirmaReset.compare(si) != 0 && confirmaReset.compare(no) != 0) {
          std::cout << "ERROR; Escribe S para confirmar reseteo o N para no hacer nada"
          <<std::endl;
          OK = 0;
       }
       else if (confirmaReset.compare(si) == 0) {
          dip.enviaPaquete(mensalreset);
          std::cout << "Nodo reseteado, espera unos segundos a que se conecte"
             <<std::endl;
          OK = 1;
       }
       else if (confirmaReset.compare(no) == 0) OK = 1;
     }while( OK != 1);
   }
}
```

4. Resultados

En los apartados anteriores se ha expuesto todo el hardware y software elaborado para la consecución de los objetivos propuestos. Tras realizar el montaje en la placa de puntos de todo el sistema, se procede a realizar pruebas de funcionamiento con distintos actuadores y en distintas condiciones. Los resultados de estas pruebas se describen a continuación.

4.1. Medidas de tiempos

En primer lugar, se conectaron a la alimentación los dos módulos básicos. Se situaron a una distancia de pocos centímetros y se midió cuánto tiempo tardan en enlazarse. Desde apagados, o desde un reinicio mediante el pulsador correspondiente, se ha medido un tiempo medio de 3.5 s si no ocurre ningún problema. Al ocurrir el enlace y estar establecida la red, ambos módulos encenderán sus LED verdes conectados al pin RSSI, lo cual indica la transmisión de paquetes de datos por radiofrecuencia. Este diodo también se encenderá sólo para el coordinador, cuando éste reciba un paquete de datos del estado de las entradas del nodo remoto. El tiempo que toma el módulo coordinador en apagar este LED va desde los 3.8 s cuando se establece el enlace entre coordinador y nodo, hasta los 4.15 s cuando recibe un paquete de estado. El nodo remoto tarda un poco más en apagar su diodo al enlazarse, unos 5 s, ya que recibe más datos que el coordinador. Es importante destacar que, aunque el diodo tarda varios segundos en apagarse, la transmisión de datos es mucho más rápida. De hecho, se ha comprobado que en condiciones de funcionamiento correcto, estos mensajes se reciben o envían inmediatamente.

El resto de medidas, como el tiempo que transcurre desde que se suelta el botón de reinicio hasta que se enciende el dispositivo o el tiempo que toma el nodo remoto en aplicar el comando AT remoto del coordinador han resultado muy pequeños, por lo que se consideran despreciables.

4.2. Medida del rango efectivo

Se ha comprobado el rango a partir del cual el coordinador es incapaz de recibir los mensajes de estado del nodo remoto. Dado que carecen de antena, esta distancia se ha visto reducida a 1.43 m en línea recta, aproximadamente.

4.3. Medida del consumo de corriente

Se ha medido la corriente que circula por el módulo cuando está encendido, quedando este valor estable en torno a los 80 mA. El incremento frente a los 40 mA que consumiría el XBee puede deberse a los componentes adicionales del circuito, como el regulador de tensión o los diodos LED.

4.4. Prueba con ventilador de ordenador de 12 V

Sin el módulo de relé, se conecta un ventilador de ordenador de 12 V a las entradas V+ y V- del módulo interruptor. La alimentación, en los otros dos pines del conector de tornillo, es de 15 V. El coordinador se conecta al PC con sistema operativo GNU/Linux mediante el módulo de comunicación con el MAX3232, y se alimenta del ordenador empleando un cable USB. El nodo remoto recibe la alimentación de una fuente de laboratorio ajustada a 5 V.

Cuando reciben alimentación, los diodos LED verdes conectados a los pines DOUT de ambas radios se encenderán inmediatamente. En el caso del coordinador, como también recibe datos seriales provenientes del ordenador mediante el módulo de comunicación, también tendrá encendido el diodo LED amarillo conectado al pin DIN. Tras 3.5 segundos, el enlace se establece y el nodo remoto se une a la red del coordinador, pudiendo desde ese momento enviar paquetes de datos desde el nodo remoto con cada pulsación del botón. Durante todo este proceso, el ventilador debe estar detenido.

Desde el ordenador, se ejecuta el programa Lectura_test, y se prueba el funcionamiento de los tres modos del programa. En el primero, cuando aparece el mensaje en pantalla que pide la pulsación del botón, y si este se presiona, el programa lee el paquete de datos producido, y si es correcto se almacena y se muestra por pantalla un OK. Seguidamente pide al usuario que suelte el botón, dando otro OK si el mensaje producido es correcto. En caso de que alguno de los dos mensajes no fuese el esperado, se pasa al tratamiento de errores y se ofrece la posibilidad de reiniciar el software del nodo remoto. Si ambos mensajes eran correctos, el programa envía al coordinador la trama del comando AT remoto que ordena al nodo remoto a fijar a 1 el estado lógico de su salida. Esto hace que el interruptor se cierre, encendiendo el ventilador. Tras esto, mientras el programa siga ejecutándose, el usuario puede volver a presionar el botón para pararlo, haciéndose el mismo tratamiento de los mensajes y enviándose un comando AT remoto para fijar la salida a 0. Finalmente, el programa pregunta si se desea ejecutar otro modo o si desea salir.

El segundo modo pide del mismo modo la pulsación del botón, pero a diferencia del anterior, no pide pulsaciones posteriores. El programa inicia una cuenta atrás de diez segundos, tras lo cual envía automáticamente el comando AT remoto de apagado. El tercer modo, por otro lado, se comporta igual que el primer modo, con la diferencia de que en esta ocasión se solicita un mensaje de confirmación de recepción al nodo remoto. Cuando este mensaje se recibe, se interpreta y si es válido se envía al coordinador un comando AT local (trama 0x08) que encenderá el LED verde de confirmación. Esto permite saber cómo está el ventilador sin tener contacto visual directo con él.

4.5. Prueba con motor de corriente continua

Para comprobar que el sistema permite el manejo de aparataje de mayor potencia y utilidad para una instalación domótica, como es el caso de un motor de DC (que por ejemplo podría encargarse de subir o bajar una persiana), se incorpora tras el módulo interruptor el módulo de relé, preparado para administrar y soportar corrientes más elevadas. A dicho módulo se conecta un motor de corriente continua de hasta 24 V, siendo esta la mayor tensión que se introduciría en el interruptor. El funcionamiento del sistema es el mismo que con el ventilador, observándose que efectivamente el motor arranca y funciona a una velocidad adecuada cuando se oprime el botón.

5. Conclusiones y líneas abiertas

5.1. Conclusiones

Se ha desarrollado una red inalámbrica basada en el protocolo ZigBee, implementada en dispositivos XBee S2 del fabricante Digi, como modelo de una red domótica genérica de bajo consumo.

Se ha estudiado en profundidad el protocolo empleado y sus funcionalidades, y en base a ello se ha desarrollado un programa para el control centralizado de la red.

Se han diseñado e implementado circuitos de alimentación, comunicación, sensado y actuación para los dispositivos XBee S2 elegidos, de acuerdo con sus especificaciones y requerimientos.

Se ha probado este prototipo de red con un sensor digital y actuadores para distintos consumos y necesidades.

5.2. Conclusions

A wireless network based on the ZigBee protocol has been developed, implemented in the XBee S2 modules from Digi, as a model of a generic and low-consumption home automation network.

The protocol and its capabilities have been studied, and based on them, a program for the centralized control of the network has been developed.

The energy supply, communication, sensing and actuation circuits for the chosen XBee S2 modules have been designed and implemented, according to its specifications and requirements.

This network prototype has been tested by using a digital sensor and actuators for different energy consumptions and necesities.

5.3. Líneas abiertas

Por falta de recursos y tiempo, quedó pendiente realizar pruebas con más nodos, tanto routers como dispositivos finales, utilizando también el modo hibernación. Además, también queadaron sin realizarse pruebas con sensores analógicos, e implementar un mayor número de sensores digitales y

analógicos. También, comprobar y probar el funcionamiento de triacs para cargas de mayor demanda energética o de corriente alterna.

Una línea de trabajo adicional es estudiar la funcionalidad y capacidad de las salidas PWM, aunque actualmente los módulos no son capaces de trabajar con este tipo de señales.

En cuanto al software, una modificación interesante es añadir modularidad y abstracción de los más bajos niveles, permitiendo una reconfiguración flexible de sensores y actuadores. En ese sentido, también es interesante estudiar la posibilidad de crear una interfaz de usuario que permita elegir el tipo de lógica a seguir, y que proporcione una fácil reconfiguración de la red.

Posteriormente, se puede estudiar la integración en el sistema conocido como “el Internet de las cosas”, una red paralela a Internet y sin acceso o control humano en el que los aparatos se comunican entre sí y comparten datos, además de tener una inteligencia que permita al sistema deducir qué desea el usuario. Una utilidad práctica de este sistema es la asistencia a personas discapacitadas o de avanzada edad.

6. Bibliografía

[1] R. Faludi, Building Wireless Sensor Networks, O'Reilly, 2011.

[2] S. C. Ergen, «ZigBee/IEEE 802.15. 4 Summary,» vol. 10, 2004.

[3] Digi International Inc., «Product Manual v1.x.4x,» -, Minnetonka, 2012.

[4] ST Microelectronics, «Hoja de datos LM317T,» 2003. [En línea]. Available: http://pdf.datasheetcatalog.com/datasheet/stmicroelectronics/2154.pdf. [Último acceso: 4 Julio 2014].

[5] SparkFun, «Esquemático Serial Explorer XBee,» 2009. [En línea]. Available: https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Serial-Explorer-v12.pdf. [Último acceso: 4 Julio 2014].

[6] SparkFun, «Placa de interfaz XBee 2mm - 2,54 mm,» [En línea]. Available: https://www.sparkfun.com/products/8276. [Último acceso: 4 Julio 2014].

[7] Tunnelsup, «Guía rápida XBee S2,» 2012. [En línea]. Available: http://www.tunnelsup.com/images/XBee-Quick-Reference-Guide.pdf. [Último acceso: 4 Julio 2014].

[8] ZigBee Alliance, «Página web oficial de la alianza ZigBee,» 2014. [En línea]. Available: www.zigbee.org. [Último acceso: 4 Julio 2014].

[9] Wikipedia, «Comparativa de módulos 802.15.4,» 2014. [En línea]. Available: en.wikipedia.org/wiki/Comparison_of_802.15.4_radio_modules. [Último acceso: 4 Julio 2014].

[10] L. Frenzel y Electronicdesign, «Alternativas al protocolo ZigBee,» 2006. [En línea]. Available: http://electronicdesign.com/communications/comparing-four-short-range-wireless-options-monitoring-and-control. [Último acceso: 4 Julio 2014].

[11] V. C. Gungor y G. P. Hancke, «Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches,» IEEE TRANSACTIONS ON INDUSTRIAL

ELECTRONICS, vol. 56, nº 10, pp. 4258-4265, 2009.

[12] GitHub Inc., «Web de la plataforma GitHub,» 2014. [En línea]. Available: www.github.com. [Último acceso: 4 Julio 2014].

[13] Digi International Inc, «Herramienta online de construcción de tramas API,» [En línea]. Available: http://ftp1.digi.com/support/utilities/digi_apiframes2.htm. [Último acceso: 4 Julio 2014].

Anexos

A continuación se incluyen los anexos del trabajo. Estos anexos consisten en los códigos fuente en C++ empleados para la elaboración del software de control.

Índice de anexos

```
#ifndef _CONEXION_HPP_
#define  _CONEXION_HPP_

#include <string>

#include "defines.hpp"
#include "VectorByte.hpp"

/** Clase abstrata para conexión genérica */
class Conexion {

public:
  /** Constructor */
  Conexion() { abierto = false; }

  /** Destructor cierra puerto si está abierto */
  ~Conexion() {   if(abierto) this->close_port(); }

  /** Implementa el read bloqueante añadiendo bytes al vector
   * \return si positivo, número de bytes leidos, <0 error
   */
  virtual int read_port(VectorByte &bufer /**< [out] Donde almacenar los bytes leidos */
  , const unsigned int nbytes /** [in] Número de bytes a leer */
  ) = 0;

  /** Idem que read pero con time out indicado, añade los bytes al vector
   * \return si positivo, número de bytes leidos, <0 error o timeout
   */
  virtual int read_timeout_port(VectorByte &bufer /**< [out] Donde almacenar los bytes
  leidos */
                      , const unsigned int nbytes /** [in] Número de bytes a leer */
                      , const unsigned long miliTO = 1000 /** [in] Número máximo de
                      milisegundos a esperar (1000)*/
                      ) = 0;

  /** Trata de enviar bytes  por la serial
   * \return el número de bytes enviados, <0 si error
   */
  virtual int write_port(const VectorByte &buf
                , const int num /** [in] si se desea menos de la longitud de buf */
                ) = 0;

  /** Trata de enviar todos los bytes del vector
   * \return el número de bytes enviados, <0 si error
   */
  virtual int write_port(const VectorByte &buf  ) = 0;

  bool get_open() { return abierto; }

  /** Cierra la conexion al puerto serial */
  virtual bool close_port() { return true; };



protected:

  /** Si el puerto ha sido abierto */
```

```
  bool abierto;

};

#endif /* del ifndef _CONEXION_HPP_ */
```

```
/** Fichero con las definiciones comunes a toda la aplicación */
#ifndef _DEFINES_HPP_
#define  _DEFINES_HPP_

typedef unsigned char byte;

#endif /* del #ifndef _DEFINES_HPP_ */
```

```
#ifndef _DIALOGO_API_HPP_
#define  _DIALOGO_API_HPP_

#include "defines.hpp"
#include "VectorByte.hpp"
#include "Conexion.hpp"

/** Clase que implementa el diálogo API con XBee */
class Dialogo_API {

public:
  /** Constructor */
  Dialogo_API(Conexion *cp /**< [in] puntero a Conexion a utilizar */
              ): ptc(cp), frameIdAT(0) {};
  /** Destructor */
  ~Dialogo_API() {};

  /** Comprueba la validez del CRC del frame. Devuelve true si correcta */
  static bool compruebaCRC( const VectorByte &frame );

  /** Método que, dado un VectorByte devuelve entero al agrupar bytes en rango.
   * Considera MSB primeros. Si fuera del rango devuelve -1.
   * TODO debería estar en la "clase" VectorByte :-(
   */
  static long vecB2long ( const VectorByte* paquete
    , unsigned int ini /*< [in] indice del elemento por el que empezar */
    , unsigned int fin = 100000 /*< [in] indice del elemento por el que terminar, si es
    >size todo el vector */
  );

  /** Método auxiliar que descompone el valor pasado y lo inserta delante de la posión
   * indicada (o al final la posición es mayor que la longitud).
   * Con numbytes se indica nº de bytes a ocupar, si negativo los mínimos
   * Si valor negativo y numbytes también, no inserta nada.
   * TODO debería estar en la "clase" VectorByte :-(
   */
  static void long2vecB (VectorByte* paquete /*< [in] el VectorByte donde insertar el entero*/
  , long int valor /*< [in] el entero a insertar */
  , int numbytes = -1 /*< [in] el número de bytes que debe ocupar, <0 los mínimos */
  , unsigned int pos = 100000 /*< [in] la posición, dentro del vector, donde insertarlo */
  );

  /** Envía el paquete por la conexion creando el frame necesario antes del envío
    El frame añade cabezera, tamaño y CRC
    */
  void enviaPaquete( const VectorByte &paquete, bool debug = false);

  /** Lee de la conexion hasta completar frame válido, extrae el paquete y lo devuelve */
  VectorByte* recibePaquete(const bool debug = false);

  /** Método que envia cmd AT local a través del API usando API 0x08.
   * Devuelve el id del paquete enviado.
   */
  byte comandoATlocal(const std::string cmd, bool debug = false);

  /** Método que envia cmd AT local a través del API usando API 0x08.
   * El valor del comando se pasa como un entero
```

```
   * Devuelve el id del paquete enviado.
   */
  byte comandoATlocal( const std::string cmdOri, long int valor, bool debug = false );

  /** Método que envia cmd AT remoto a través del API usando API 0x17.
   * Devuelve el id del paquete enviado.
   */
  byte comandoATremoto(const std::string cmd
  , long int dest64 /*< [in] dirección 64 bits del nodo destino, si no se sabe poner -1 */
  , int dest16 /*< [in] dirección 16 bits del nodo destino, si no sabe poner -1 */
  , bool debug = false);

  /** Método que envia cmd AT remoto a través del API usando API 0x17.
   * El valor del comando se pasa como un entero
   * Devuelve el id del paquete enviado.
   */
  byte comandoATremoto( const std::string cmdOri
  , long int dest64 /*< [in] dirección 64 bits del nodo destino, si no se sabe poner -1 */
  , int dest16 /*< [in] dirección 16 bits del nodo destino, si no sabe poner -1 */
  , long int valor /*< [in] valor del parámetro del comando AT */
  , bool debug = false );

  //TODO entender el otro 0x17 de broadcast


  /** Método que transmite los datos a través del API usando API 0x10.
   * Devuelve el id del paquete enviado.
   */
  byte trasnmisionRemota( long int dest64 /*< [in] dirección 64 bits del nodo destino, si no
  se sabe poner -1 */
  , int dest16 /*< [in] dirección 16 bits del nodo destino, si no sabe poner -1 */
  , const std::string &datos /*< [in] datos a transmitir */
  , byte hops = 0 /*< [in] maximo número de saltos 0 => máximos permitidos (10) */
  , bool multicast = false
  , bool debug = false );

  /** Reparte un paquete (recibido) invocando el método correspondiente.
   * Descifra la estructura interna e invoca el método con los parámetros
   * desmenuzados
   */
  void reparteMensaje( const VectorByte &frame, bool debug = false );

  /** Reparte un paquete (recibido) invocando el método correspondiente
   */
  void reparteMensaje( const VectorByte *frame, bool debug = false );


protected:
  /** Mantiene Conexion recibida en el constructor */
  Conexion *ptc;

  /** Mantienen el ultimo Id utilizado */
  byte frameIdAT;

  /** Devuelve cadena de caracteres correspondiente al valor de status */
  std::string status2string( byte status );
```

```
  /** Método para gestionar paquetes de respuesta AT local (0x88)
   * Las clases hija podrán modificar su código para implementar otra funcionalidad
   * En esta clase sólo saca por salida estandar los datos
   */
  void respuestaATlocal(byte frameId /**< [in] id del paquete*/
  , std::string comandoAT /**< [in] string con los 2 caracteres del comando AT */
  , byte status /**< [in] status */
  , int value /**< [in] valor del comando <0 si no lo hay */
  , const VectorByte *paquete /**< [in] paquete entero */
  );

  /** Método para gestionar paquetes de respuesta AT remota (0x97)
   * Las clases hija podrán modificar su código para implementar otra funcionalidad
   * En esta clase sólo saca por salida estandar los datos
   */
  void respuestaATremota(byte frameId /**< [in] id del paquete*/
  , long addr64 /**< [in] dirección larga remota */
  , int addr16 /**< [in] dirección corta remota */
  , std::string comandoAT /**< [in] string con los 2 caracteres del comando AT */
  , byte status /**< [in] status */
  , int value /**< [in] valor del comando <0 si no lo hay */
  , const VectorByte *paquete /**< [in] paquete entero */
  );

  /** Método auxilar de los de comandoATx para preparar el strig del comando AT.
   * Pasa a mayusculoas, quita AT inicial, devuelve el valor del comando (-1 si no hay)
   */
  long int trataCmdAT(std::string &cmd);
};

#endif /* del  #ifndef _DIALOGO_API_HPP_ */
```

```
#include "Dialogo_API.hpp"

#include <iostream>
#include <iomanip>
#include <algorithm>

/** Comprueba la validez del CRC del frame. Devuelve true si correcta */
bool Dialogo_API::compruebaCRC( const VectorByte &frame ) {
  if( frame.size()<4 ) //No tiene tamaño mínimo
    return false;
  unsigned int lon = frame[1]*256 + frame[2];
  if( frame.size()< (lon+4) ) //no tiene tamaño según logitud indicada
    return false;
  byte crc = 0;
  for( unsigned int i=3; i<3+lon+1; i++)
    crc += frame[i];
  return (crc == 0xFF);
}

void Dialogo_API::long2vecB (VectorByte* paquete, long int valor, int numbytes, unsigned int
pos ){
  if ( numbytes<0 && valor<0 )
    return;
  //descomponemos value en bytes
  long int diff  = ( pos < paquete->size() ) ? pos : paquete->size();
  int bytea = 0; //numero de byte actual
  do {
    bytea++;
    VectorByte::iterator pos = paquete->begin() + diff;
    paquete->insert( pos, (valor & 0xFF) );
    valor >>= 8;
  } while ( ( (numbytes>0) && (bytea<numbytes) ) || ( (numbytes<0) && (valor>0) ) );

}

/** Envía el paquete por la conexion creando el frame necesario antes del envío
  El frame añade cabezera, tamaño y CRC
  */
void Dialogo_API::enviaPaquete( const VectorByte &paquete, bool debug){

  VectorByte frame;
  frame.push_back(0x7E); //cabezera
  unsigned int lenPq = paquete.size();
  frame.push_back( (lenPq & 0xFF00) >> 8 ); //largo MSB
  frame.push_back( lenPq & 0xFF ); //largo LSB
  //copiamos el paquete en el frame
  frame.insert( frame.end(), paquete.begin(), paquete.end() );
  byte crc = 0;
  for( unsigned int i=0; i<paquete.size(); i++)
    crc += paquete[i];
  crc = 0xFF - crc;
  frame.push_back( crc );
  if( debug ){
    std::cout << "[Dialogo_API::enviaPaquete] Paquete a enviar"
      << frame << std::endl;
  }
  //enviamos el frame
```

```
  //TODO controlar el tamaño de lo enviado, generar excepción
  ptc->write_port( frame );
}

/** Lee de la conexion hasta completar frame válido, extrae el paquete y lo devuelve */
VectorByte* Dialogo_API::recibePaquete(const bool debug) {
  VectorByte frame;

  while( true ) {
    frame.clear();
    int leidos=ptc->read_timeout_port(frame, 1);
    if( leidos!= 1 ) {
      std::cerr << "Se ha producido timeout esperando cabecera" << std::endl;
      continue;
    }
    if( frame[0] == 0x7E) break;
    std::cerr << frame[0] << " fuera del paquete"
              << std::dec << std::endl;
  }
  //Tenemos la cabecera procedemos a leer la  longitud
  while( ptc->read_timeout_port(frame, 1) < 1 )
    std::cerr << "Se ha producido timeout esperando longitud MSB" << std::endl;
  //Tenemos la cabecera procedemos a leer la  longitud
  while( ptc->read_timeout_port(frame, 1) < 1 )
    std::cerr << "Se ha producido timeout esperando longitud LSB" << std::endl;
  unsigned int lon = frame[1]*256 + frame[2];

  //Ya tenemos longitud del paquete, procedemos a leerlo
  for(unsigned int i=0; i<lon; i++)
    while( ptc->read_timeout_port(frame, 1) < 1 )
      std::cerr << "Se ha producido timeout Byte del paquete" << std::endl;
  //Leemos CRC
  while( ptc->read_timeout_port(frame, 1) < 1 )
    std::cerr << "Se ha producido timeout esperando CRC" << std::endl;

  if( !compruebaCRC(frame) ){
    std::cerr << "El CRC del paquete recibido no es correcto. Descartamos" << std::endl;
    return new VectorByte();
  }
  if( debug ){
    std::cout << "[Dialogo_API::recibePaquete] Paquete recibido "
      << frame << std::endl;
  }
  VectorByte *paquete = new VectorByte(frame.begin()+3, frame.end()-1);
  return paquete;
}

/** Método auxilar de los de comandoATx para preparar el strig del comando AT.
 * Pasa a mayusculoas, quita AT inicial, devuelve el valor del comando (-1 si no hay)
 */
long int Dialogo_API::trataCmdAT(std::string &cmd) {

  //pasamos a mayusculas
  std::transform(cmd.begin(), cmd.end(), cmd.begin(), ::toupper);

  //borramos \r final
  if( cmd[ cmd.size()-1 ] == '\r' ) cmd.erase( cmd.end()-1 );
```

```
  //borramos AT inicial
  if( cmd[0]=='A' && cmd[1]=='T') cmd.erase(0, 2);

  //determinamos el valor, si lo hay
  long int value = -1;
  if ( cmd.size() > 2) {   //añadimos los parámetros
    //convertimos los dígitos hexadecimales a un entero
    value = 0;
    for (unsigned int i=2; i<cmd.size(); i++)
      value = (value << 4 ) + (cmd[i]-(cmd[i]>'9'?'A'-10:'0'));
  }

  return value;
}

byte Dialogo_API::comandoATlocal(const std::string cmdOri, bool debug) {
  std::string cmd = cmdOri;
  long int value = trataCmdAT(cmd);

  return comandoATlocal(cmd, value, debug);
}

byte Dialogo_API::comandoATlocal(const std::string cmdOri, long int valor, bool debug) {
  std::string cmd = cmdOri;
  trataCmdAT(cmd);  //no tenemos en cuenta value devuelto

  //Montamos el paquete de tipo 0x08
  VectorByte paquete;
  paquete.push_back(0x08);
  byte Id = ++frameIdAT;
  paquete.push_back( Id );
  //añadimos las dos letras del comando AT
  paquete.push_back( cmd[0] );
  paquete.push_back( cmd[1] );

  long2vecB( &paquete, valor);

  if( debug ) {
    std::cout << "[Dialogo_API::comandoAT] " << cmd << " con valor 0x" << std::hex << valor
    << std::dec
      <<" Paquete a enviar" << paquete << std::endl;
  }
  enviaPaquete(paquete, debug);

  return Id;
}

byte Dialogo_API::comandoATremoto(const std::string cmdOri, long int dest64, int dest16, bool
 debug) {
  std::string cmd = cmdOri;
  long int value = trataCmdAT(cmd);

  return comandoATremoto(cmd, dest64, dest16, value, debug);

}
```

```
byte Dialogo_API::comandoATremoto( const std::string cmdOri, long int dest64, int dest16
  , long int valor, bool debug ) {
  //comprobamos que haya alguna dirección valida
  if( (dest64<0) && (dest16<0) ) {
    std::cerr << "Las dos direcciones no pueden ser desconocidas" << std::endl;
    //TODO lanzar excepción
    return 0;
  }
  if( dest16<0 ) {
    //la dirección 15 es desconocida
    dest16 = 0xFFFE; //ponemos el valor comodín
  }
  std::string cmd = cmdOri;
  trataCmdAT(cmd);  //ignoramos el valor devuelto

  //Montamos el paquete de tipo 0x17
  VectorByte paquete;
  paquete.push_back(0x17);

  //añadimos Id
  byte Id = ++frameIdAT;
  paquete.push_back( Id );

  //añadimos dirección 64
  long2vecB( &paquete, dest64, 8);

  //añadimos dirección 16
  long2vecB( &paquete, dest16, 2);

  //añadimos modo de aplicación
  paquete.push_back(0x02); //TODO parametro para aplicación no inmediata

  //añadimos las dos letras del comando AT
  paquete.push_back( cmd[0] );
  paquete.push_back( cmd[1] );

  long2vecB( &paquete, valor);

  if( debug ) {
    std::cout << "[Dialogo_API::comandoATremoto] " << cmd << " Paquete a enviar"
    << paquete << std::endl;
  }
  enviaPaquete(paquete, debug);

  return Id;
}

byte Dialogo_API::trasnmisionRemota( long int dest64, int dest16
, const std::string &datos, byte hops, bool multicast, bool debug ) {
  //comprobamos que haya alguna dirección valida
  if( (dest64<0) && (dest16<0) ) {
    std::cerr << "Las dos direcciones no pueden ser desconocidas" << std::endl;
    //TODO lanzar excepción
    return 0;
  }
  if( dest16<0 ) {
    //la dirección 15 es desconocida
```

```
    dest16 = 0xFFFE; //ponemos el valor comodín
  }

  //Montamos el paquete de tipo 0x10
  VectorByte paquete;
  paquete.push_back(0x10);

  //añadimos Id
  byte Id = ++frameIdAT;
  paquete.push_back( Id );

  //añadimos dirección 64
  long2vecB( &paquete, dest64, 8);

  //añadimos dirección 16
  long2vecB( &paquete, dest16, 2);

  //Broadcast radius
  paquete.push_back(hops);

  //Multicast
  if( multicast ) paquete.push_back(0x08);
  else paquete.push_back(0x00);

  //los datos
  for(unsigned int i = 0; i<datos.size(); i++)
    paquete.push_back( datos[i] );

  if( debug ) {
    std::cout << "[Dialogo_API::trasnmisionRemota] Paquete a enviar: "
    << paquete << std::endl;
  }
  enviaPaquete(paquete, debug);

  return Id;

}



std::string Dialogo_API::status2string( byte status ) {
  switch( status ) {
    case 0: return std::string("OK");
    case 1: return std::string("ERROR");
    case 2: return std::string("Invalid Command");
    case 3: return std::string("Invalid Parameter");
    default: return std::string("STATUS NO VALIDO");
  }
}

void Dialogo_API::reparteMensaje( const VectorByte &paquete, bool debug) {
  reparteMensaje( &paquete, debug );
}
void Dialogo_API::reparteMensaje( const VectorByte *paquete, bool debug) {
  if( debug )
    std::cout << "[Dialogo_API::reparteMensaje] Repartiendo el paquete " << *paquete << std::
    endl;
```

```
  byte apiId = paquete->at(0);
  switch( apiId ) {
    case 0x88: {
      if( paquete->size() < 5 ) {
        std::cerr << "No tiene el tamaño necesario " << paquete->size() << "<5" << std::endl;
        break;
      }
      byte frameId = paquete->at(1);
      std::string comandoAT;
      comandoAT += (char)paquete->at(2);
      comandoAT += (char)paquete->at(3);
      byte status = paquete->at(4);
      //miramos si hay parametros y los presentamos
      int value = vecB2long( paquete, 5);
      respuestaATlocal(frameId, comandoAT, status, value, paquete);
      break; }

    case 0x97: {
      if( paquete->size() < 15 ) {
        std::cerr << " 0x97: No tiene el tamaño necesario " << paquete->size() << "<5" << std
        ::endl;
        break;
      }
      byte frameId = paquete->at(1);
      long addr64 = vecB2long( paquete, 2, 9);
      int addr16 = vecB2long( paquete, 10, 11);
      std::string comandoAT;
      comandoAT += (char)paquete->at(12);
      comandoAT += (char)paquete->at(13);
      byte status = paquete->at(14);
      //miramos si hay parametros y los presentamos
      int value = vecB2long( paquete, 15);
      respuestaATremota(frameId, addr64, addr16, comandoAT, status, value, paquete);
      break; }

    default:
      std::cout << "Id desconocido " << apiId << std::endl;
      break;
  }
}

/** Método que, dado un VectorByte devuelve entero al agrupar bytes en rango.
 * Considera MSB primeros. Si fuera del rango devuelve -1.
 */
long Dialogo_API::vecB2long ( const VectorByte* paquete, unsigned int ini, unsigned int fin) {
  int value = -1;
  if( paquete->size() > ini ) {
    value = 0;
    for( unsigned int i=ini; (i<paquete->size()) && (i<=fin); i++)
      value = (value << 8) + paquete->at(i);
  }
  return value;
}


void Dialogo_API::respuestaATlocal(byte frameId, std::string comandoAT, byte status
  , int value, const VectorByte *paquete) {
```

```
  std::cout << "Respuesta 0x88: comando AT local:"
    << "Id=" << frameId << " Comando AT: -" << comandoAT <<"- Estado: "
    << status2string(status);
  if( value >= 0)
    std::cout << " Valor =  0x" << std::hex << value << " (" << std::dec << value <<")" ;
  std::cout << std::endl;

}

void Dialogo_API::respuestaATremota(byte frameId, long addr64, int addr16
  , std::string comandoAT, byte status
  , int value, const VectorByte *paquete) {

  std::cout << "Respuesta 0x97: comando AT remoto desde "
  << std::hex << "0x" << addr64 << " (0x" << addr16 << ") "
  << "Id=" << frameId << " Comando AT: -" << comandoAT <<"- Estado: "
  << status2string(status);
  if( value >= 0)
    std::cout << " Valor =  0x" << std::hex << value << " (" << std::dec << value <<")" ;
  std::cout << std::endl;

}
```

```
// #ifndef _DIALOGOVENTILADOR_HPP_
// #define  _DIALOGOVENTILADOR_HPP_

#include "Dialogo_API.hpp"

class dialogoVentilador : public Dialogo_API {

public:

  dialogoVentilador(Conexion *cp): Dialogo_API(cp) {};
  ~dialogoVentilador() {};

  VectorByte* recibePaquete(const bool debug = false);

};

// #endif
```

```
#include "dialogoVentilador.hpp"

#include <iostream>
#include <iomanip>
#include <algorithm>


VectorByte* dialogoVentilador::recibePaquete(const bool debug) {
  VectorByte frame;

  while( true ) {
    frame.clear();
    int leidos=ptc->read_port(frame, 1);
//     if( leidos!= 1 ) {
//       std::cerr << "Se ha producido timeout esperando cabecera" << std::endl;
//       continue;
//     }
    if( frame[0] == 0x7E) break;
    std::cerr << frame[0] << " fuera del paquete"
              << std::dec << std::endl;
  }
  //Tenemos la cabecera procedemos a leer la  longitud
  while( ptc->read_timeout_port(frame, 1) < 1 )
    std::cerr << "Se ha producido timeout esperando longitud MSB" << std::endl;
  //Tenemos la cabecera procedemos a leer la  longitud
  while( ptc->read_timeout_port(frame, 1) < 1 )
    std::cerr << "Se ha producido timeout esperando longitud LSB" << std::endl;
  unsigned int lon = frame[1]*256 + frame[2];

  //Ya tenemos longitud del paquete, procedemos a leerlo
  for(unsigned int i=0; i<lon; i++)
    while( ptc->read_timeout_port(frame, 1) < 1 )
      std::cerr << "Se ha producido timeout Byte del paquete" << std::endl;
  //Leemos CRC
  while( ptc->read_timeout_port(frame, 1) < 1 )
    std::cerr << "Se ha producido timeout esperando CRC" << std::endl;

  if( !compruebaCRC(frame) ){
    std::cerr << "El CRC del paquete recibido no es correcto. Descartamos" << std::endl;
    return new VectorByte();
  }
  if( debug ){
    std::cout << "[dialogoVentilador::recibePaquete] Paquete recibido "
      << frame << std::endl;
  }
  VectorByte *paquete = new VectorByte(frame.begin()+3, frame.end()-1);
  return paquete;
}
```

```
#ifndef _SERIAL_CONEXION_HPP_
#define  _SERIAL_CONEXION_HPP_

#include <termios.h>
#include <string>

#include "defines.hpp"
#include "VectorByte.hpp"
#include "Conexion.hpp"


/** Clase para la gestion de la comunicación a través de la serial */
class Serial_Conexion : public Conexion {

public:
  /** Constructor */
  Serial_Conexion() {};

  /** Destructor */
  ~Serial_Conexion() {};

  /** Inicializa la conexión al puerto
   * \return si se pudo abrir el puerto correctamente
   */
  bool open_port(const std::string &NombrePuerto /**< [in] nombre del puerto a abrir */
                , const int baudrate /**< [in] Velocidad inicial */
                );

  /** Implementa el read bloqueante añadiendo bytes al vector
   * \return si positivo, número de bytes leidos, <0 error
   */
  int read_port(VectorByte &bufer /**< [out] Donde almacenar los bytes leidos */
  , const unsigned int nbytes /** [in] Número de bytes a leer */
  );

  /** Idem que read pero con time out indicado, añade los bytes al vector
   * \return si positivo, número de bytes leidos, <0 error o timeout
   */
  int read_timeout_port(VectorByte &bufer /**< [out] Donde almacenar los bytes leidos */
                      , const unsigned int nbytes /** [in] Número de bytes a leer */
                      , const unsigned long miliTO = 1000 /** [in] Número máximo de
                      milisegundos a esperar (1000)*/
                      );

  /** Trata de enviar bytes  por la serial
   * \return el número de bytes enviados, <0 si error
   */
  int write_port(const VectorByte &buf
                , const int num /** [in] si se desea menos de la longitud de buf */
                );

  /** Trata de enviar todos los bytes del vector
   * \return el número de bytes enviados, <0 si error
   */
  int write_port(const VectorByte &buf  );

  bool get_open() { return abierto; }
```

```
  /** Cierra la conexion al puerto serial */
  bool close_port();

  /** Cambia la velocidad de la comunicación serial */
  bool set_baudrate(const int baudrate /**< [in] velocidad deseada */
                    );


private:
  /** Descriptor de archivo de la serial*/
  int fdSer;

  /** Guardarán las configuración del puerto*/
  struct termios oldtio,newtio;


  bool baudrate2define(int baudrate,tcflag_t &bitsSpeed);


};

#endif /* del ifndef _SERIAL_CONEXION_HPP_ */
```

```cpp
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/time.h>
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
#include <strings.h>

#include "Serial_Conexion.hpp"


bool Serial_Conexion::baudrate2define(int baudrate,tcflag_t &bitsSpeed) {
  switch (baudrate) {
    // case 0: bitsSpeed=B0; return true;
    case 50: bitsSpeed=B50; return true;
    case 75: bitsSpeed=B75; return true;
    case 110: bitsSpeed=B110; return true;
    case 134: bitsSpeed=B134; return true;
    case 150: bitsSpeed=B150; return true;
    case 200: bitsSpeed=B200; return true;
    case 300: bitsSpeed=B300; return true;
    case 600: bitsSpeed=B600; return true;
    case 1200: bitsSpeed=B1200; return true;
    case 1800: bitsSpeed=B1800; return true;
    case 2400: bitsSpeed=B2400; return true;
    case 4800: bitsSpeed=B4800; return true;
    case 9600: bitsSpeed=B9600; return true;
    case 19200: bitsSpeed=B19200; return true;
    case 38400: bitsSpeed=B38400; return true;
    case 57600: bitsSpeed=B57600; return true;
    case 115200: bitsSpeed=B115200; return true;
    case 230400: bitsSpeed=B230400; return true;
  }
  return false;
}

bool Serial_Conexion::open_port(const std::string &NombrePuerto, const int baudrate) {

  if(abierto)
    return false;
  fdSer = open(NombrePuerto.c_str(), O_RDWR | O_NOCTTY );
  if (fdSer <0) {
    perror(NombrePuerto.c_str());
    return false;
  }

  tcgetattr(fdSer,&oldtio); /* salva configuracion actual del puerto  */

  /* limpiamos struct para recibir los nuevos parametros del puerto */
  bzero((void *)&newtio, sizeof(newtio));

  /*
   *      BAUDRATE: Fija la tasa bps. Podria tambien usar cfsetispeed y cfsetospeed.
   *      CRTSCTS : control de flujo de salida por hardware (usado solo si el cable
   *      tiene todas las lineas necesarias Vea sect. 7 de Serial-HOWTO)
```

```
   *       CS8     : 8n1 (8bit,no paridad,1 bit de parada)
   *       CLOCAL  : conexion local, sin control de modem
   *       CREAD   : activa recepcion de caracteres
   */

  if(! baudrate2define(baudrate,newtio.c_cflag)) {
    close(fdSer);
    return false;
  }

  newtio.c_cflag = newtio.c_cflag | CS8 | CLOCAL | CREAD;

  /*
   *       IGNPAR  : ignora los bytes con error de paridad
   *       ICRNL   : mapea CR a NL (en otro caso una entrada CR del otro ordenador
   *       no terminaria la entrada) en otro caso hace un dispositivo en bruto
   *       (sin otro proceso de entrada)
   */
  newtio.c_iflag = IGNPAR;

  newtio.c_oflag = 0;

  /* pone el modo entrada (no-canonico, sin eco,...) */

  newtio.c_lflag = 0;

  newtio.c_cc[VTIME]    = 0;   /* temporizador entre caracter, máximo 100 ms */
  newtio.c_cc[VMIN]     = 1;   /* bloquea lectura hasta recibir 1 chars  */

  tcflush(fdSer, TCIFLUSH);
  tcsetattr(fdSer,TCSANOW,&newtio);

  abierto=true;
  return true;
}


bool Serial_Conexion::set_baudrate(const int baudrate) {
  tcflag_t BRdefine;
  if(!abierto)
    return false;
  if(! baudrate2define(baudrate,BRdefine)) {
    return false;
  }
  cfsetispeed(&newtio,BRdefine);
  cfsetospeed(&newtio,BRdefine);

  /*  tcflush(fdSer, TCIFLUSH); */
  if(tcsetattr(fdSer,TCSANOW,&newtio)<0) {
    perror("Al establecer la velocidad");
    return false;
  }
  return true;
}

int Serial_Conexion::read_port(VectorByte &bufer, const unsigned int nbytes) {
  byte memoria[nbytes];
```

```
  int rec = read(fdSer, memoria, nbytes);
  if ( rec>0 ) //tenemos algo, rellenamos el vector
    for( int i=0; i<rec; i++)
      bufer.push_back(memoria[i]);
  return rec;
}


/** Idem que read pero con time out indicado en miliTO
 */
int Serial_Conexion::read_timeout_port(VectorByte &bufer, const unsigned int nbytes
                                       , const unsigned long miliTO) {
  if(miliTO>0) {
    fd_set readfs;
    int r;
    struct timeval tv;

    FD_ZERO(&readfs);
    FD_SET(fdSer,&readfs);

    tv.tv_sec=miliTO/1000; /*nos quedamos con los segundos*/
    tv.tv_usec=(miliTO%1000)*1000; /*nos quedamos con los mili.sg y pasamos a micro.sg*/

    do {
      r=select(fdSer+1,&readfs,NULL,NULL,&tv);
    }  while(r==-1 && errno == EINTR);
    if(r!=1)
      return 0; /*ha habido timeout o otro error*/
  }
  /*Debe haber algo disponible o se ha pasado time out 0*/
  byte memoria[nbytes];
  int rec = read(fdSer, memoria, nbytes);
  if ( rec>0 ) //tenemos algo, rellenamos el vector
    for( int i=0; i<rec; i++)
      bufer.push_back(memoria[i]);
  return rec;
}

int  Serial_Conexion::write_port(const VectorByte &buf) {
  return write_port(buf, buf.size() );
}

int  Serial_Conexion::write_port(const VectorByte &buf, const int num) {
  if (!abierto)
    return -1;
  int largo = buf.size();
  if (num < largo )
    largo = num;
  if (largo==0)
    return 0;
  byte memoria[ largo ];
  for(int i=0; i<largo; i++)
    memoria[i] = buf[i];
  return write(fdSer, memoria, largo );
}

bool Serial_Conexion::close_port() {
```

```
  if(abierto)
    if(close(fdSer) < 0)
      return false;
    return true;
}
```

```cpp
#ifndef VECTORBYTE_HPP
#define VECTORBYTE_HPP

#include <vector>
#include <iostream>
#include "defines.hpp"

typedef std::vector<byte> VectorByte;

std::ostream& operator<<(std::ostream &os, const VectorByte &v);

std::ostream& operator<<(std::ostream &os, const byte &b);

#endif // VECTORBYTE_HPP
```

```cpp
#include "VectorByte.hpp"
#include <iomanip>

std::ostream& operator<<(std::ostream &os, const byte &b) {
  os << std::setfill('0') << std::hex << "0x"
    << std::setw(2) << (int) b << std::dec;
  return os;
}


std::ostream& operator<<(std::ostream &os, const VectorByte &v) {
  os << "[("  << v.size() << ")" ;
  for ( unsigned int ba = 0; ba < v.size(); ba++)
    os << ' ' << v[ba];
  os << "] " << std::dec;
  return os;
}
```