2 Revisión de tecnologías existentes

El desarrollo de un sistema domótico requiere de la aplicación de muchos conceptos existentes a un ámbito concreto como es el hogar, o la oficina en el caso de la inmótica.

Por tanto, la domótica puede verse en cierto modo como la aplicación de una serie de tecnologías existentes para la consecución de un fin concreto: el control inteligente de las instalaciones eléctricas de un hogar.

En lo relativo a estas tecnologías podemos distinguir entre aquellas que forman parte del sistema en sí y aquellas que nos permitirán el diseño e implementación de dicho sistema domótico.

El principal objetivo de este apartado es dar una visión general sobre las tecnologías usadas para el desarrollo de un sistema domótico centralizado, objetivo fundamental del presente proyecto. De este modo se pretende aportar información suficiente para comprender el desarrollo realizado así como el conocimiento requerido para evaluar el alcance del proyecto que se ha llevado a cabo.

Además, en este apartado no sólo se hará una revisión de las tecnologías usadas si no que también se ofrecerá una breve descripción de varios protocolos domóticos existentes con el fin de aportar una idea global de la naturaleza del problema abordado en este proyecto.

En este punto cabe hacer una distinción entre el concepto de arquitectura y protocolo domótico. A lo largo de este documento nos referiremos como protocolo domótico a todas las capacidades de comunicación y control de un sistema domótico. Por otro lado, el concepto de arquitectura es global y hará referencia tanto al protocolo de comunicaciones como al resto de funcionalidades y elementos físicos y lógicos del sistema completo.

2.1 Arquitecturas existentes

El diseño de una arquitectura domótica es el objetivo principal planteado para el presente proyecto. Dentro de este objetivo se incluye el diseño de un protocolo de comunicaciones. Dicho diseño constituye una parte fundamental de la arquitectura funcional definida para el sistema domótico centralizado que se ha desarrollado.

En este apartado se va a realizar un repaso teórico de varias arquitecturas existentes en el mercado con el fin de aportar una visión general de los sistemas domóticos que se pueden encontrar funcionando actualmente en muchos lugares.

Algunas de estas arquitecturas ofrecen pocas funcionalidades más allá de las que ofrece un protocolo de comunicaciones, mientras que otras definen una variedad de servicios mucho más amplia. Por esta razón, en este apartado se usará el concepto de "protocolo" tanto para referirse a un caso como al contrario.

2.1.1 X-10

X-10 es uno de los protocolos más antiguos que se usan en aplicaciones domóticas. Fue diseñado en Escocia entre los años 1976 y 1978 con el objetivo de transmitir datos por las líneas de baja tensión a muy baja velocidad (60bps en EEUU y 50bps en Europa) y costes muy bajos. Al usar las líneas de eléctricas de la vivienda, no es necesario instalar nuevos cables para conectar dispositivos.

El protocolo *X-10*, en sí, no es propietario. Cualquier fabricante puede producir dispositivos *X-10* y ofrecerlos en su catálogo, estando obligado a usar los productos del fabricante escocés que diseño esta tecnología. Aunque, al contrario de lo que sucede con la firma *Echelon* y su *NeuronChip* que implementa *Lonworks* (Apartado 2.1.4), los circuitos integrados que implementan el *X-10* tienen un royalty muy bajo (casi simbólico).

Actualmente, existen en Europa tres grandes familias de productos basados en *X-10* compatibles entre sí: *Netzbus*, *Timac* y *Home Systems*.

Gracias a su madurez (más de 20 años en el mercado) y a la tecnología empleada los productos *X-10* tienen un precio muy competitivo de forma que es líder en el mercado norteamericano residencial y de pequeñas empresas (realizadas por los usuarios finales o electricistas sin conocimientos de automatización).

Se puede afirmar que *X-10* es ahora mismo la tecnología más asequible para realizar una instalación domótica de baja complejidad.

Existen tres tipos de dispositivos *X-10*: los que sólo pueden transmitir órdenes, los que sólo pueden recibirlas y los que pueden tanto enviar como recibir instrucciones.

Los transmisores pueden direccionar hasta 256 receptores diferentes. Los receptores vienen dotados de dos pequeños conmutadores giratorios, uno con 16 letras y el otro con 16 números, que permiten asignar a cada receptor una dirección de las 256 posibles. En una misma instalación puede haber varios receptores configurados con la misma dirección, todos realizarán la función preasignada cuando un transmisor envíe una trama con esa dirección. Cualquier dispositivo receptor puede recibir órdenes de diferentes transmisores.

Los dispositivos bidireccionales tienen la capacidad de responder y confirmar la correcta realización de una orden, lo cual puede ser muy útil cuando el sistema *X-10* está conectado a un programa de ordenador que muestre el estado en que se encuentra la instalación domótica de la vivienda.

2.1.1.1 Nivel físico

El protocolo *X-10* hace uso de una modulación muy sencilla en comparación con las que usan otros protocolos de control por ondas portadoras. El transceiver *X-10* detecta los pasos por cero de la onda senoidal de 50Hz de la alimentación eléctrica (60Hz en EEUU) para insertar un instante después una ráfaga muy corta de señal a una frecuencia fija.

Se puede insertar esta señal tanto en el semiciclo positivo como en el negativo de la onda senoidal. La codificación de un bit “uno” o de un bit “cero”, depende de cómo se inyecte esta señal en los dos semiciclos. Un “uno” binario se representa por un pulso de 120KHz durante 1 milisegundo y un “cero” binario se representa por la ausencia de ese pulso de 120 KHz. En un sistema trifásico el pulso de 1 milisegundo se transmite tres veces para que coincida con el paso por el cero en las tres fases.

Por lo tanto, el tiempo de bit coincide con los 20ms que dura el ciclo de la señal, de forma que la velocidad binaria de 50bps viene impuesta por la frecuencia de la red eléctrica en Europa. En Estados Unidos la velocidad binaria es de 60bps.

La transmisión completa de una trama *X-10* necesita once ciclos de corriente. Esta trama se divide en tres campos de información:

- Dos ciclos representan el Código de Inicio.
- Cuatro ciclos representan el Código de Casa (letras A-P).
- Cinco ciclos representan o bien el Código Numérico (1-16) o bien el Código de Función (Encender, Apagar, Aumento de Intensidad, etc.).

Para aumentar la fiabilidad del sistema, esta trama (Código de Inicio, Código de Casa y Código de Función o Numérico) se transmite siempre dos veces, separándolas por tres ciclos completos de corriente. Hay una excepción, en funciones de regulación de intensidad, las cuales se transmiten de forma continuada (por lo menos dos veces) sin separación entre tramas.

2.1.2 EIB

El *European Installation Bus* o *EIB* es un sistema domótico desarrollado bajo los auspicios de la Unión Europea con el objetivo de contrarrestar las importaciones de productos similares que se estaban produciendo desde el mercado japonés y el norteamericano donde estas tecnologías se han desarrollado antes que en Europa.

El objetivo era crear un estándar europeo, con el suficiente número de fabricantes, instaladores y usuarios, que permita comunicarse a todos los dispositivos de una instalación eléctrica como: contadores, equipos de climatización, de custodia y seguridad, de gestión energética o electrodomésticos.

El *EIB* está basado en la estructura de niveles *OSI* y tiene una arquitectura descentralizada. Este estándar europeo define una relación extremo a extremo entre dispositivos que permite distribuir la inteligencia entre los sensores y los actuadores instalados en la vivienda.

La *EIBA* (*EIB Association*) es una asociación de 113 empresas europeas, líderes en el mercado eléctrico, que se unieron en 1990 para impulsar el uso e implantación del sistema domótico *EIB*.

La *EIBA* tiene su sede en Bruselas y sus miembros cubren el 80% de la demanda de equipamiento eléctrico en Europa. Las tareas principales de esta asociación son:

- Fijar las directrices técnicas para el sistema y los productos *EIB*, así como establecer los procedimientos de ensayo y certificación de calidad.
- Distribuir el conocimiento y las experiencias de las empresas que trabajan sobre el *EIB*.
- Encargar a laboratorios de ensayo las pertinentes pruebas de calidad.
- Conceder a los productos *EIB* y a los fabricantes de estos una licencia de marca *EIB* con la que se podrán distribuir los productos.

- Colaborar activamente con otros organismos europeos e internacionales en todas las fases de la normalización y adaptar el sistema *EIB* a las normas vigentes.
- Liderar el proceso de convergencia de los tres buses europeos de más amplia difusión como son el propio *EIB*, el *Batibus* y el *EHS (European Home System*).

Según la *EIBA* hay unos 10 millones de dispositivos *EIB* instalados por todo el mundo, unas 70.000 instalaciones, una gama de 4.500 productos diferentes, 113 empresas asociadas, y 70.000 instaladores cualificados

2.1.2.1 Nivel físico

Aunque en un principio sólo se contempló usar un cable de dos hilos como soporte físico de las comunicaciones, se pretendía que el nivel *EIB.MAC* (*Medium Access Control*) pudiera funcionar sobre los siguientes medios físicos:

- *EIB.TP*: Sobre par trenzado a 9600bps. Además por estos dos hilos se suministra 24 voltios para la alimentación remota de los dispositivos *EIB*. Utiliza la técnica *CSMA* con arbitraje positivo del bus que evita las colisiones evitando así los reintentos de transmisión y maximizando el ancho de banda disponible.
- *EIB.PL*: Corrientes portadoras sobre 230V alternos a 50Hz a 1200/2400bps. Usa la modulación *SFSK* (*Spread Frequency Shift Keying*) similar a la *FSK* pero con las portadoras más separadas. La distancia máxima que se puede lograr sin repetidor es de 600 metros.
- *EIB.net*: Usando el estándar Ethernet a 10Mbps (*IEC 802-2*). Sirve de conexión entre segmentos *EIB* además de permitir la transferencia de telegramas *EIB* a través del protocolo *IP* a viviendas o edificios remotos.
- *EIB.RF*: Radiofrecuencia: usando varias portadoras, se consiguen distancias de hasta 300 metros en campo abierto. Para mayores distancias o edificios con múltiples estancias se pueden usar repetidores.
- *EIB.IR*: Infrarrojo: para el uso con mandos a distancia en salas o salones donde se pretenda controlar los dispositivos EIB instalados.

En la práctica, sólo el par trenzado ha conseguido una implantación masiva mientras que los demás apenas han conseguido una presencia testimonial.

2.1.3 EHS

El estándar *EHS* (*European Home System*) ha sido otro de los intentos que la industria europea (1984), auspiciada por la Comisión Europea, ha realizado para crear una tecnología que permitiera la implantación de la domótica en el mercado residencial de forma masiva. El resultado fue la especificación del *EHS* publicada en 1992. Dicha especificación está basada en una topología de niveles *OSI* (*Open Standard Interconnection*), y se especifican los niveles físico, de enlace de datos, de red y de aplicación.

Desde el inicio de su desarrollo han estado involucrados en él los fabricantes europeos más importantes de electrodomésticos, empresas eléctricas, operadoras de telecomunicaciones y fabricantes de equipamiento eléctrico. El objetivo planteado fue la creación de un protocolo abierto que permitiera cubrir las necesidades de interconexión de los productos de todos estos fabricantes y proveedores de servicios.

Desde su concepción, el objetivo de *EHS* fue cubrir las necesidades de automatización de la mayoría de las viviendas europeas sin la necesidad de usar sistemas más potentes pero también más costosos (como *Lonworks*, *EIB* o *BatiBUS*) debido a la mano de obra especializada que exige su instalación.

El *EHS* viene a cubrir, por prestaciones y objetivos, la parcela que tienen el *CEbus* norteamericano y el *HBS* japonés y rebasa las prestaciones del *X-10* que tanta difusión ha conseguido en EEUU.

La asociación *EHSA* (*EHS Association*) es la encargada de emprender y llevar a cabo diversas iniciativas para aumentar el uso de esta tecnología en las viviendas europeas. Además se ocupa de la evolución y mejora tecnológica del *EHS* y de asegurar la compatibilidad total entre fabricantes de productos con interfaz *EHS*.

Este protocolo está totalmente abierto, esto es, cualquier fabricante asociado a la *EHSA* puede desarrollar sus propios productos y dispositivos que implementen el *EHS*.

El sistema *EHS* posee una filosofía Plug&Play, y aporta las siguientes ventajas a los usuarios finales:

- Compatibilidad total entre dispositivos *EHS*.
- Configuración automática de los dispositivos, movilidad de los mismos (posibilidad de conexión en diferentes emplazamientos) y ampliación sencilla de las instalaciones.
- Capacidad de compartir un mismo medio físico entre diferentes aplicaciones sin que estas interfieran entre ellas.

Cada dispositivo *EHS* tiene asociada una dirección única dentro del mismo segmento de red. Esta dirección, además de identificar unívocamente a un nodo, también lleva asociada información para el encaminado de los telegramas por diferentes segmentos de red *EHS*.

2.1.3.1 Nivel físico

Entre los años 1992 y 1995 la *EHSA* impulsó el desarrollo de componentes electrónicos que implementaran la primera especificación de *EHS*. Como resultado nació un circuito integrado de *ST-Microelectronics* (*ST7537HS1*) que permitía transmitir datos por una canal serie asíncrono a través de las líneas de baja tensión de las viviendas (*powerline communications*). Esta tecnología, basada en modulación *FSK*, consigue velocidades de hasta 2400bps y además también puede utilizar cables de pares trenzados como soporte de la señal.

En la actualidad, se están usando o se están desarrollando los siguientes medios físicos:

- PL-2400: Línea de potencia a 2400bps.
- TP0: Par Trenzado a 4800bps (idéntico al nivel físico del *BatiBUS*).
- TP1: Par Trenzado/Coaxial a 9600bps.
- TP2: Par Trenzado a 64Kbps.
- IR-1200: Infrarrojo a 1200bps.
- RF-1100: Radiofrecuencia a 1100bps.

2.1.4 Lonworks

Echelon presentó la tecnología *Lonworks* en el año 1992. Desde entonces multitud de empresas han usado esta tecnología para implementar redes de control distribuidas y sistemas de automatización. Aunque está diseñada para cubrir los requisitos de la mayoría de las aplicaciones de control, sólo ha tenido éxito de implantación en edificios de oficinas, hoteles o industrias. Pero, debido a su coste, los dispositivos *Lonworks* no han tenido una implantación masiva en los hogares, sobretodo porque existían otras tecnologías de prestaciones similares mucho más económicas.

El éxito que ha tenido *Lonworks* en instalaciones profesionales, en las que importa mucho más la fiabilidad y robustez que el precio, se debe a que desde su

origen ofrece una solución con arquitectura descentralizada, extremo a extremo, que permite distribuir la inteligencia entre los sensores y los actuadores instalados en la vivienda y que cubre desde el nivel físico al nivel de aplicación de la mayoría de los proyectos de redes de control.

Según *Echelon*, su arquitectura es un sistema abierto a cualquier fabricante que quiera usarla sin depender de sistemas propietarios, que permite reducir los costes y aumentar la flexibilidad de la aplicación de control distribuida. Aunque *Echelon* usa el concepto de "sistema abierto", realmente no es una tecnología que pueda implementarse si no es con un circuito integrado registrado por *Echelon*.

Cualquier dispositivo *Lonworks*, o nodo, está basado en un microcontrolador especial llamado *Neuron Chip*. Tanto este circuito integrado como el firmware que implementa el protocolo *LonTalk* fueron desarrollados por *Echelon* en el año 1990.

El *Neuron Chip* tiene un identificador único, el *Neuron ID*, que permite direccionar cualquier nodo de forma unívoca dentro de una red de control *Lonworks*. Este identificador, con un tamaño de 48 bits, se graba en memoria *EEPROM* durante la fabricación del circuito.

Además, tiene un modelo de comunicaciones que es independiente del medio físico sobre el que funciona, esto es, los datos pueden transmitirse sobre cables de par trenzado, ondas portadoras, fibra óptica, radiofrecuencia y cable coaxial, entre otros.

El firmware que implementa el protocolo *LonTalk*, proporciona servicios de transporte y encaminado extremo a extremo. Incluye un sistema operativo que ejecuta y planifica la aplicación distribuida y que maneja las estructuras de datos que se intercambian los nodos.

Estos circuitos se comunican entre sí enviándose telegramas que contienen la dirección de destino, información para el encaminado, datos de control así como los datos de la aplicación del usuario y un *checksum* como código detector de errores.

Todos los intercambios de datos se inician en un *Neuron Chip* y se supervisan en el resto de los circuitos de la red. Un telegrama puede tener hasta 229 bytes de información neta para la aplicación distribuida.

Los datos pueden tener dos formatos; Un mensaje explícito o una variable de red. Los mensajes explícitos son la forma más sencilla de intercambiar datos entre dos aplicaciones residentes en dos *Neuron Chips* del mismo segmento *Lonworks*. Por el contrario, las variables de red proporcionan un modelo estructurado para el intercambio automático de datos distribuidos en un segmento *Lonworks*. Aunque son menos flexibles que los mensajes explícitos, las variables de red evitan que el

programador de la aplicación distribuida tenga que tener en cuenta los detalles de las comunicaciones.

Respecto a los fabricantes, *Echelon* sólo ha concedido licencia a tres fabricantes de semiconductores, los cuales además tienen que pagar un royalty por cada circuito fabricado. Además, el diseño del *Neuron Chip* permanece secreto y ningún otro fabricante, además de estos tres, puede fabricar dicho producto. Por estos motivos, al no existir competencia real y estar la producción controlada por *Echelon*, los precios no se han reducido tanto como para permitir que los nodos *Lonworks* puedan tener un precio realmente competitivo en aplicaciones residenciales. Por lo tanto, aunque *Echelon* califica a su sistema como abierto, en la práctica no es así.

LonMark es una asociación de fabricantes que desarrollan productos o servicios basados en redes de control *Lonworks*. Esta asociación especifica y publica las recomendaciones e implementaciones que mejor se adaptan a cada uno de los dispositivos típicos de las redes de control, para ello se basan en objetos y perfiles funcionales.

Los objetos *LonMark* forman las variables que intercambia la red de control a nivel de aplicación (nivel 7 de la torre *OSI*). Estos objetos describen los formatos de los datos que intercambian los nodos y la semántica que se usa para relacionarlos con otros objetos de la aplicación distribuida. Hay tres objetos que son básicos: el actuador, el sensor y el controlador.

Los perfiles funcionales detallan en profundidad la interfaz de la aplicación distribuida con la red *Lonworks* (variables de red y las propiedades de configuración) y el comportamiento que tendrán las funciones implementadas.

Hay que recalcar que los perfiles funcionales estandarizan las funciones, no los productos. De esta forma se permite que diversos fabricantes ofrezcan el mismo producto a nivel funcional aunque desde el punto de vista hardware no tengan nada que ver un diseño con otro. Lo perfiles *LonMark* aseguran la compatibilidad total entre productos *Lonworks*.

Para no limitar el conjunto de funciones u objetos que un fabricante puede embarcar en un nodo Lonworks, los perfiles funcionales se especifican con un conjunto de objetos o funciones obligatorias además de un conjunto opcional de las mismas. En este punto se debe indicar que aunque existen cientos de productos *Lonworks* no todos tienen la certificación *LonMark*.

2.1.4.1 Nivel físico

El *Neuron Chip* proporciona un puerto específico de cinco pines que puede ser configurado para actuar como interfaz de diversos transceivers de línea y funcionar a

diferentes velocidades binarias. *Lonworks* puede funcionar sobre *RS-485* opto-aislado, acoplado a un cable coaxial o de pares trenzados con un transformador, sobre corrientes portadoras, fibra óptica e incluso radio.

El transceiver es el encargado de adaptar las señales del *Neuron Chip* a los niveles que necesita cada medio físico. De este modo, el nivel físico queda aislado del resto de la jerarquía de niveles, permitiendo toda una serie de posibilidades según el modelo de transceiver elegido.

2.1.5 CEBus

En 1984 varios miembros de la *EIA* norteamericana (*Electronics Industry Association*) llegaron a la conclusión de que existía la necesidad de un bus domótico que aportara más funciones que las que aportaban sistemas de aquella época. Para ello diseñaron y desarrollaron un estándar llamado *CEBus* (*Consumer Electronic Bus*).

En 1992 fue presentada la primera especificación del *CEBus*. Consiste en un protocolo para entornos distribuidos de control, tratándose de una especificación abierta, por lo que cualquier empresa puede implementar este estándar.

En Europa una iniciativa similar en prestaciones, y en el mercado al que va dirigido, es el protocolo *EHS* (*European Home System*) que se vio en el apartado 2.1.3.

La *CIC* (*CEBus Industry Council*) es una asociación de diferentes fabricantes de software y hardware que certifican que los nuevos productos *CEBus* que se lancen al mercado cumplan correctamente con la totalidad de la especificación. Una vez que el producto pase todos los ensayos, el fabricante paga una tasa y es autorizado a poner el logo *CEBus* en ese producto.

Las tramas definidas en *CEBus* pueden tener longitud variable en función de la cantidad de datos que se necesitan transmitir. El tamaño mínimo es 8 bytes y el máximo de aproximadamente 100.

Al igual que los dispositivos *EIB* (Apartado 2.1.2), los nodos *CEBus* tienen grabada una dirección física definida durante el proceso de fabricación, que los identifican de forma unívoca en una instalación domótica. Dicha dirección posee más de 4.000 millones de combinaciones posibles.

Como parte de la especificación *CEBus* se ha definido un lenguaje común para el diseño y especificación de la funcionalidad de un nodo, a este lenguaje se le llamó *CAL* (*Common Application Language*) y es un lenguaje orientado a objetos (estándar *EIA-600*).

CAL es un lenguaje orientado a objetos que permite controlar dispositivos *CEBus* y asignar recursos mediante comandos. Las funciones de asignación de recursos permiten pedir, usar y liberar recursos *CEBus*. Las funciones de control proporcionan la capacidad de enviar comandos *CAL* a dispositivos remotos y responder a dichos comandos.

CAL utiliza el modelo de programación orientada a objetos. Cuando un objeto recibe un mensaje se ejecuta alguno de los métodos disponibles. Un mensaje consiste en un identificador de método seguido de un número variable de parámetros. Cuando se recibe el mensaje, se busca en la lista de métodos cual es el que tiene el identificador y se si se encuentra, se ejecuta. Por ejemplo, si se quiere subir el volumen de la radio en tres unidades, habrá que mandar un mensaje al objeto que controla la radio en cuestión en el que se invoque el método de subir volumen con el parámetro 3.

Los objetos *CAL* no se organizan en jerarquías (no existe el concepto de herencia tal como se entiende en la programación orientada a objetos) sino que el comportamiento depende del contexto en el que se encuentre. Por ejemplo, si tenemos un objeto de control analógico, este se puede usar tanto para representar un control de volumen, como un termostato o un dimmer. La función exacta vendrá determinada por el contexto en el cual es instanciado el objeto.

Los comandos y los informes de estados se transmiten por el canal de control en forma de mensajes. El núcleo de la especificación *CEBus* se centra en definir este canal de control. El formato de los mensajes *CEBus* es independiente del medio físico utilizado. Cada mensaje contiene la dirección de destino del receptor sin ninguna referencia sobre que en medio físico están situados el receptor o el transmisor. De este modo *CEBus* forma una red uniforme a nivel lógico en forma de bus.

CEBus posee una topología flexible. Cualquier dispositivo se puede conectar a cualquier medio siempre que tenga la interfaz adecuada. Para comunicar segmentos de red que tienen diferente medio físico, se utilizan dispositivos llamados “routers”. Estos pueden estar integrados dentro de otro dispositivo con más funcionalidades.

Para facilitar el envío de mensajes a todos los dispositivos estos tienen una dirección de difusión (*broadcast address*). Además, los dispositivos se pueden agrupar y direccionar mediante direcciones de grupo (*group address*). De esta forma se puede mandar un único mensaje a varios dispositivos al mismo tiempo. Un dispositivo puede pertenecer a uno o más grupos.

2.1.5.1 Nivel físico

Se contemplan diversas opciones para que los electrodomésticos y equipos eléctricos puedan comunicarse usando ondas portadoras por líneas de baja tensión,

par trenzado con alimentación remota, cable coaxial, infrarrojo, radiofrecuencia o fibra óptica.

Para la transmisión de datos mediante señales eléctricas portadoras, *CEBus* usa una modulación en espectro expandido. Se transmiten uno o varios bits dentro de una ráfaga de señal con un ancho de banda que va desde los 100KHz hasta los 400KHz. La velocidad media de transmisión es de 7500bps.

En todos los medios físicos, la información de control y datos se transmite a la misma tasa binaria, 8000bps. También se contemplan canales para transmitir audio o vídeo.

2.2 Hardware empleado

En lo referente a las tecnologías empleadas para la realización del sistema domótico centralizado, podemos hacer una distinción entre hardware y software.

En este apartado se describirá todo el hardware que se ha utilizado en el diseño del sistema domótico centralizado. Por consiguiente se tratarán tanto los elementos hardware que forman parte del sistema en sí, como los utilizados para implementar, probar y, en general, desarrollar dicho sistema.

Entre estos elementos hardware se encuentra el elemento más importante de todo el sistema: la placa *NYDIANET*.

También se verán el hardware usado para programar la placa *NYDIANET* y los equipos empleados para la realización de pruebas y ensayos.

2.2.1 Placa *NYDIANET*

La placa *NYDIANET* (Imagen 2.1) es el componente hardware más importante del sistema domótico centralizado desarrollado en este proyecto. Su importancia radica en el hecho de que dicha placa constituye por si misma cada nodo de la red domótica.

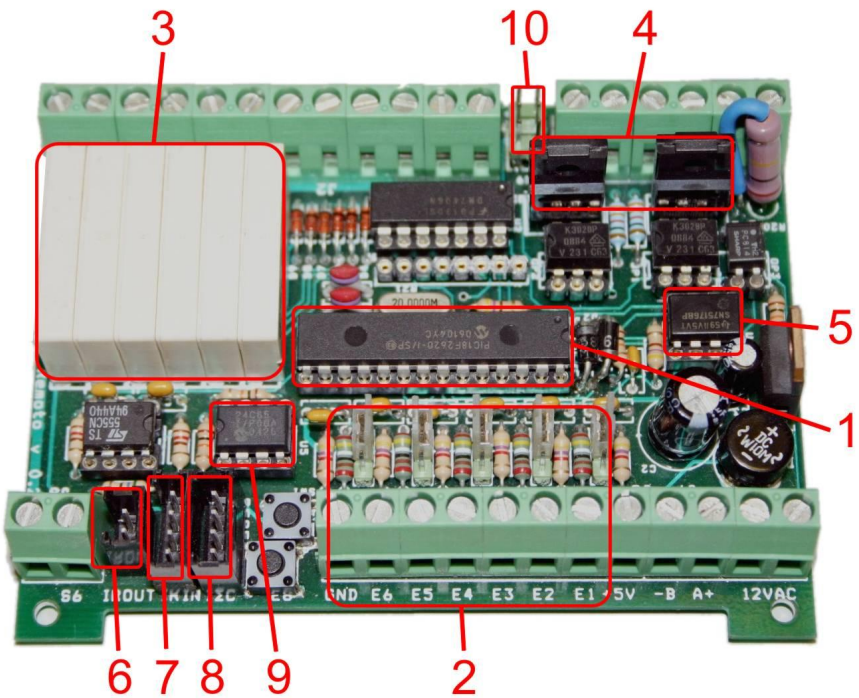


Imagen 2.1 Placa *NYDIANET*

La placa *NYDIANET* consiste a grandes rasgos en un hardware de propósito general que presenta las siguientes características:

- 1: Unidad de procesado basada en microcontrolador *PIC* de *MicroChip*.
- 2: Entradas configurables en varios modos (modos que serán explicados en el apartado 2.2.1.1).
- 3: Salidas digitales mediante relé.
- 4: Salidas reguladas mediante *TRIACS*.
- 5: Comunicación mediante RS-485.
- 6: Conector para instalación de LED emisor de infrarrojos.
- 7: Conector para instalación de detector de infrarrojos.
- 8: Conector para instalación de dispositivos externos I^2C.
- 9: Memoria EEPROM serie externa.
- 10: Conector para programación serie en circuito (*ICSP*).

El diseño físico de la placa nos permite alojarla cómodamente en las cajas de registro de una vivienda. De este modo la placa no solo pasa desapercibida, quedando fuera de la vista de los usuarios del sistema, si no que también puede ser instalada sin la necesidad de alterar las instalaciones existentes en la vivienda.

Para poder disponer de algunas capacidades, como por ejemplo la comunicación entre nodos a través de RS-485, sí será necesario añadir instalaciones a la vivienda. Sin embargo, un gran número de capacidades del sistema no precisa *a priori* de la alteración de las instalaciones eléctricas del hogar.

En el presente apartado haremos hincapié en hacer una descripción funcional de aquellas características fundamentales de la placa que han sido de mayor importancia e influencia a la hora de diseñar y desarrollar el sistema domótico centralizado.

A continuación pasamos a describir de modo detallado algunos de los aspectos que se han considerado más relevantes de la placa *NYDIANET*.

2.2.1.1 Entradas

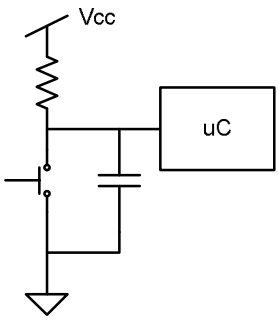
Las entradas de la placa se pueden configurar, a nivel físico, de varias maneras. Esta configuración se hace mediante el uso de jumpers. Algunos de estos modos de funcionamiento requieren del uso de circuitería adicional externa.

Los diferentes modos de configuración de entradas son los siguientes:

- Modo digital simple.
- Modo digital con LED de supervisión.
- Modo digital con detección de corte (supervisado).
- Modo digital en bucle de corriente.
- Modo analógico.

Generalmente el modo de funcionamiento a nivel físico es el **modo digital normal** (Esquema 2.2).

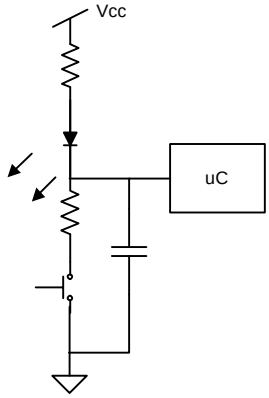
Cuando una entrada se encuentra configurada de este modo, el microcontrolador la percibe como una entrada digital binaria de niveles TTL. El estado de reposo de una entrada digital normal es un cero lógico, que se corresponde con una señal eléctrica de 5 voltios.



Esquema 2.2 Modo digital normal

A este tipo de entradas se deberá conectar un pulsador sin anclaje. Este pulsador permitirá al usuario accionarlo de modo que se genere un pulso de duración igual al tiempo que se esté presionando.

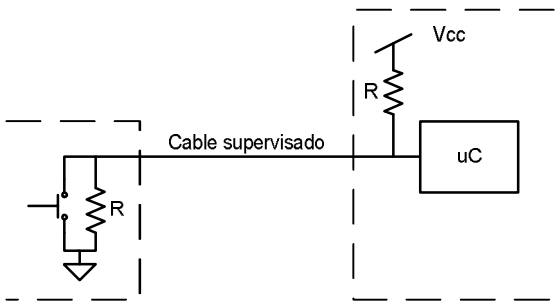
En el **modo digital con LED de supervisión** (Esquema 2.3) se dispone de un LED (que debe ser instalado externamente) cuyo encendido y apagado es controlable desde el microcontrolador.



Esquema 2.3 Modo digital con LED de supervisión.

Dicha configuración aprovecha la capacidad del microcontrolador de cambiar los pines de sus puertos entre los modos de entrada y de salida de forma arbitraria durante la ejecución de su programa, con lo que se consigue multiplexar el control del LED con la captura de los pulsos digitales provenientes del pulsador externo.

El **modo digital con detección de corte** (o modo supervisado) (Esquema 2.4), en cambio, permite la detección del corte del cable que transmite la señal eléctrica desde el pulsador hasta el microcontrolador. De esta forma es posible la implementación de mecanismos de alarma y de seguridad en el hogar cuando la señal de entrada proviene de un sensor de presencia u otro elemento de seguridad pasiva.

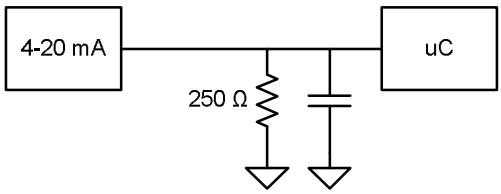


Esquema 2.4 Modo digital con detección de corte

Para ello el microcontrolador configura la entrada correspondiente en modo analógico, permitiendo, de esta forma, diferenciar entre varios niveles de tensión. Según el nivel de tensión detectado se podrán distinguir tres estados: pulsado, no pulsado y cable cortado.

Si el estado de la entrada es “no pulsado” el microcontrolador detecta 2,5 voltios; si, en cambio, el estado es “pulsado” no se detecta voltaje y, por último, si el estado es “cable cortado” se detectan 5 voltios.

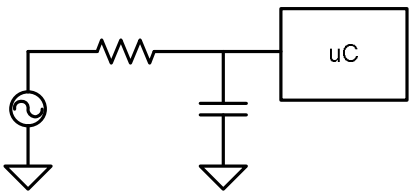
El siguiente modo considerado es el **modo digital en bucle de corriente** (Esquema 2.5). Este modo de funcionamiento es indistinguible del modo digital normal por parte del microcontrolador.



Esquema 2.5 Modo digital en bucle de corriente

Este modo se utilizará en el caso de que la entrada provenga de un dispositivo que disponga de una salida en corriente entre 4 y 20 miliamperios. Mediante una resistencia en paralelo se convierte esos valores de intensidad en valores de tensión que caen en los rangos interpretados como unos y ceros lógicos según el caso.

Finalmente, en el **modo analógico** (Esquema 2.6) las entradas se capturan mediante un conversor de digital a analógico (DAC) a través de una resistencia en serie.



Esquema 2.6 Modo analógico

2.2.1.2 Salidas

A diferencia de las entradas las salidas no son configurables. Se dispone de un número fijo de salidas mediante relés y mediante *TRIAC's*.

Las salidas mediante relé permiten la conmutación eléctrica de dispositivos externos haciendo las veces de interruptor de corte de corriente. Por otro lado, las salidas mediante *TRIAC* permiten tanto el modo conmutado como el modo regulado, en el que el porcentaje de potencia que se hace llegar a la salida es controlable mediante procedimientos software.

El modo regulado mediante *TRIAC* precisa de la sincronización con la red eléctrica externa. Esta sincronización se llevará a cabo a través de un sensor que detecta el paso por cero de la señal de tensión de la red eléctrica. De esta forma el

microcontrolador puede conmutar el *TRIAC* en el momento adecuado para recortar la señal eléctricamente según se desee. Este procedimiento será explicado en detalle en el apartado 4.7.2 de la presente memoria.

2.2.1.3 Microcontrolador

Anteriormente se ha mencionado en repetidas ocasiones al microcontrolador de la placa *NYDIANET*. Dicho microcontrolador es la unidad de procesado de datos de la placa.

En el microcontrolador se encuentra alojado el firmware desarrollado con motivo del presente proyecto. Dicho firmware supone la esencia del sistema domótico centralizado diseñado.

Originalmente la placa contaba con un microcontrolador *PIC16F876* de *MicroChip* (Figura 2.7) que posee las siguientes características:

- Frecuencia de funcionamiento: Hasta 20MHz.
- Memoria de programa FLASH: 8K de palabras de 14 bits.
- Memoria para datos: 368 bytes.
- Memoria EEPROM interna: 256 bytes.
- Fuentes de interrupción: 13.
- Puertos de entrada y salida: 3 (uno de 6 bits y dos de 8 bits).
- Temporizadores: 3 (uno de 16 bits y dos de 8 bits).
- Módulos CCP/PWM: 2.
- Comunicación serie: MSSP y USART.
- Canales analógicos: 5 de 10 bits.
- Juego de instrucciones: 35 instrucciones.

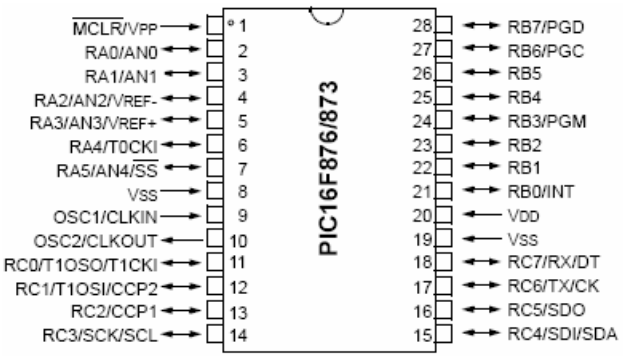


Figura 2.7 Esquema de pines del *PIC16F876*

La implementación de un protocolo de comunicaciones, que, tal como se dijo en la introducción del presente documento, es uno de los objetivos propuestos a la hora de abordar este proyecto, requiere del uso de un gran número de recursos hardware.

Cualquier protocolo que desee ofrecer robustez frente a errores en la transmisión de los datos requiere del uso de mecanismos como el asentimiento, la retransmisión de tramas, la detección de errores o incluso la corrección de los mismos.

Para llevar a cabo todas estas tareas es preciso realizar un gran número de operaciones sobre las unidades de datos del protocolo, que generalmente serán tramas de longitud variable. Por tanto dichas tramas deberán ser almacenadas en memoria para poder ser procesadas y retransmitidas en caso necesario.

Si a todo ello sumamos el hecho de que un protocolo es bidireccional por definición, es decir, que la comunicación debe tener lugar en ambos sentidos entre el emisor y el receptor, las necesidades de almacenamiento se duplican al requerir de memoria suficiente para almacenar las tramas que son recibidas y enviadas de manera separada.

Como se puede observar en la lista de características básicas del microcontrolador *PIC16F876*, este dispone de 368 bytes de memoria de datos. Esta memoria debe ser utilizada para todas las funcionalidades requeridas al sistema domótico, es decir, tanto el protocolo de comunicaciones como el resto de operaciones que este debe realizar.

Uno de los objetivos planteados para este proyecto es la flexibilidad de la aplicación desarrollada. Teniendo esto en cuenta, no podemos restringir de manera significativa algunos parámetros importantes como pudieran ser la longitud de las tramas, la cantidad de tareas ejecutables o la memoria para datos adicionales disponible.

Por todo ello, una de las primeras decisiones tomadas durante el desarrollo del presente proyecto fue la sustitución del microcontrolador por otro modelo superior que permitiese la flexibilidad requerida sin necesidad de alterar el hardware existente con el fin de disponer de los recursos suficientes para el desarrollo del proyecto sin que las limitaciones del hardware fuesen un factor determinante.

El modelo seleccionado es el *PIC18F2620* de *MicroChip* (Figura 2.8). Dicho microcontrolador es eléctricamente y físicamente compatible con el modelo anterior, al poseer el mismo encapsulado y disponer de una distribución del patillaje compatible con el *PIC16F876*.

Estas son sus principales características:

- Frecuencia de funcionamiento: Hasta 40MHz.
- Memoria de programa FLASH: 64K de palabras de 16 bits.
- Memoria para datos: 3986 bytes.
- Memoria EEPROM interna: 1024 bytes.
- Fuentes de interrupción: 19.
- Puertos de entrada y salida: 3 (uno de 6 bits y dos de 8 bits).
- Temporizadores: 4 (uno de 8 bits y tres de 16 bits).
- Módulos CCP/PWM: 2.
- Comunicación serie: MSSP y EUSART.
- Canales analógicos: 10 de 10 bits.
- Juego de instrucciones: 83 instrucciones.

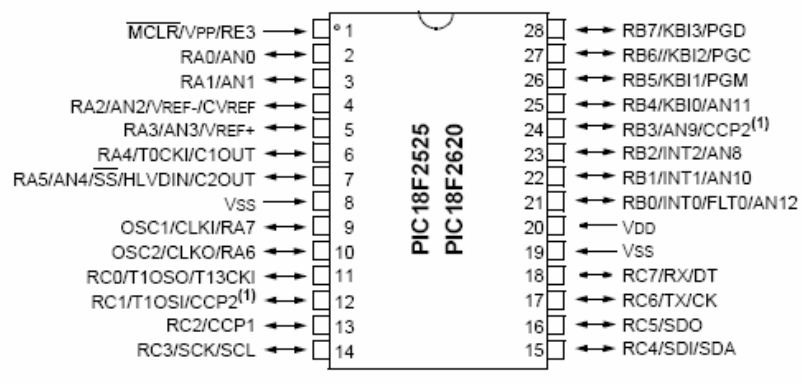


Figura 2.8 Esquema de pines del *PIC18F2620*

Como se puede observar, el nuevo microcontrolador dispone de mayor cantidad de memoria y de recursos, de modo que sus capacidades no sean las que limiten el alcance del proyecto.

2.2.1.4 Transceiver RS-485

La placa *NYDIANET* dispone de un transceiver RS-485 que posibilita su instalación como parte de una red de comunicaciones con topología de bus bidireccional half-duplex.

El estándar RS-485 definirá la mayor parte del nivel físico a falta de algunos parámetros configurables por el usuario. En el apartado 2.2.4 se describirá el estándar

RS-485 de manera global, mientras que en el apartado 4.8.1 se mostrarán los valores escogidos para todos aquellos parámetros configurables por el usuario.

En el Anexo II de la presente memoria se puede encontrar un diagrama general de la placa *NYDIANET*.

2.2.2 Hardware de programación

Para la programación de la placa *NYDIANET* es necesario el uso de un hardware específico para tal fin. Este hardware se compone de los siguientes elementos: De un PC; de un programador, así como de un soporte adecuado para el microcontrolador que se pretende programar. Este soporte será el elemento encargado de alimentar eléctricamente el micro y servir de interfaz entre el módulo programador y este.

En el presente apartado hablaremos del hardware de programación, excluyendo de la descripción al PC, que es un ordenador personal que es necesario para ejecutar el software que permitirá la programación del microcontrolador. Dicho software será considerado en detalle el apartado 2.3.

El programador usado es el *In-Circuit Debugger 2* (en adelante, *ICD2*) de *MicroChip* (Elemento central de la Figura 2.9). Sin embargo, el *ICD2* no sólo permite la programación del microcontrolador, si no que también permite la depuración del código fuente que se esté ejecutando en el propio microcontrolador en lugar de en un emulador o un simulador externo.

Figura 2.9 Esquema de conexionado para programación y depuración

Otra de las ventajas que ofrece este sistema es el hecho de que tanto la programación como la depuración del código bajo desarrollo se llevan a cabo sin la necesidad de extraer el microcontrolador de la placa *NYDIANET*. De esta capacidad se deriva el acrónimo que da nombre al hardware: *In-Circuit Debugger*.

Para ello es necesario que durante el proceso de diseño de la placa se haya tenido esta posibilidad en cuenta y se hayan hecho accesibles los pines adecuados mediante un conector externo que será usado para conectar el *ICD2* a la placa. De este modo, la propia placa *NYDIANET* es el soporte que alimenta al microcontrolador para hacer posible su programación con el *ICD2*.

Posteriormente será necesario elaborar un cable que permita conectar el *ICD2* a la placa. La estructura de este cable será explicada posteriormente.

El *MPLAB ICD2* es un depurador en circuito (*In-Circuit Debugger: ICD*) y un programador serie en circuito (*In-Circuit Serial Programmer: ICSP*) de bajo coste. Su objetivo es servir como ayuda a la evaluación, depuración y programación de software para microcontroladores *PIC* de *MicroChip* en un entorno de laboratorio ofreciendo estas características:

- **Ejecución de código en tiempo real y paso a paso**: Mediante estas características es posible la depuración del código bajo desarrollo de un modo mucho más sencillo y más fiable, puesto que este se está ejecutando en el mismo entorno donde luego funcionará normalmente.
- **Inspección y modificación de breakpoints, registros y variables**: De este modo la depuración es más sencilla y más potente, permitiendo controlar numerosos aspectos del programa bajo supervisión.
- **Depuración en circuito**: Esta capacidad permite no verse obligado a extraer el microcontrolador de la placa donde está alojado para la depuración de la aplicación que se esté desarrollando.
- **Monitorización de alimentación**.
- **LED's de diagnóstico**.
- **Interfaz integrada de desarrollo** (*MPLAB IDE*: Se verá en el apartado 2.3.2).
- **Interfaz RS-232 o USB con un PC**.

Gracias a estas características el *ICD2* nos permitirá:

- Programar microcontroladores de *Microchip* que soporten la programación serie en circuito (*ICSP*).
- Depurar código fuente en la misma aplicación donde se ejecutará.
- Depurar el hardware en tiempo real.

En comparación con otros sistemas como los *ICE* (*In-Circuit Emulator*) este sistema es mucho mas eficiente en cuanto a costes (debido al alto precio de los *ICE*). No obstante, ofrece algunos inconvenientes, como los siguientes:

- El *ICD* requiere del uso exclusivo de parte de los recursos hardware y software del sistema bajo desarrollo.
- El microcontrolador debe tener alimentación, reloj y estar funcionando.
- El *ICD* puede depurar sólo si el sistema está en un estado avanzado de desarrollo hardware.

Un emulador proporciona al desarrollador memoria y un reloj y puede ejecutar código incluso sin estar conectado a una placa de pruebas o a la placa donde se está desarrollando la aplicación. Durante el desarrollo y la depuración de una aplicación de software para microcontroladores, un *ICE* proporciona las herramientas más potentes para conseguir un sistema totalmente funcional, mientras que un *ICD* podría no ser capaz de depurar la aplicación si esta no se ejecuta.

Por otro lado, se puede instalar un conector para la depuración en circuito en la placa desarrollada permitiendo, mediante el uso de un *ICD*, el testeo, depuración y reprogramación del código incluso si la placa ya está siendo producida en serie.

Todo esto hace que el *ICD* tenga las siguientes ventajas frente a un *ICE*:

- Establecer una conexión con la aplicación bajo desarrollo una vez comenzado el proceso de producción no requiere la extracción del microcontrolador para insertar la sonda del *ICE*.
- El *ICD* puede reprogramar el firmware de la aplicación sin más conexiones que la mencionada y para la cual se elaboró un cable específico (que se describirá posteriormente).

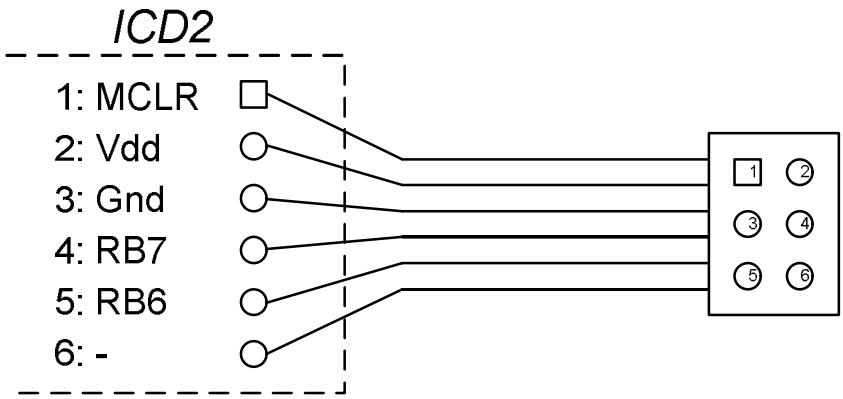
En el caso particular del diseño del sistema domótico centralizado que nos atañe, la placa donde se ejecuta la aplicación ya se encontraba completamente desarrollada, por lo que el uso del *ICD* es lo más recomendado, debido a la dificultad de extraer el microcontrolador de la placa *NYDIANET*.

Además, la extracción continua de elementos delicados como chips e integrados para su programación y depuración puede conllevar a su deterioro físico por mucho cuidado que se ponga durante dicha manipulación, debido a la fragilidad de muchos de sus elementos, especialmente las patillas de conexión.

Todo esto hace del *ICD* una herramienta económica, potente y de fácil uso, por lo que su utilización durante el desarrollo del presente proyecto está justificada.

En lo referente al cable de conexión entre el *ICD2* y la placa *NYDIANET*, como se ha dicho, su objetivo es la conexión del programador con la placa donde se ejecutará la aplicación final, permitiendo así la programación y depuración del microcontrolador.

El cable elaborado (Esquema 2.10) posee un conector *RJ-12* en el lado del *ICD2* y un molex de 6 pines en el lado correspondiente a la placa *NYDIANET*.

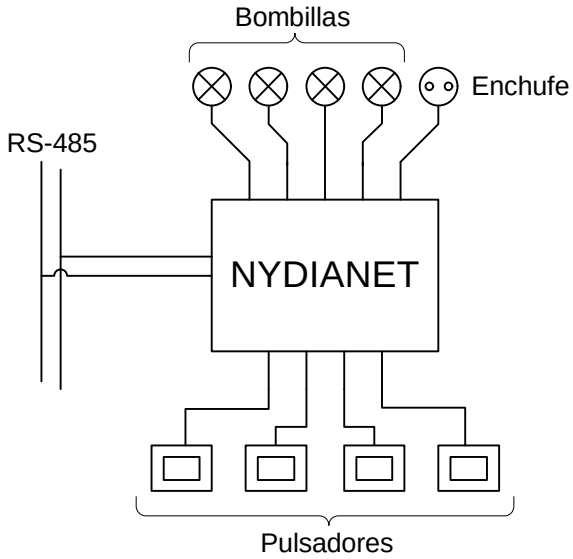


Esquema 2.10 Diagrama del cable de programación.

2.2.3 Panel y montaje de ensayos

Para el desarrollo del proyecto se dispuso de un panel de pruebas que simula una pequeña instalación domótica (Esquema 2.11). Dicho panel consta de los siguientes elementos:

- Dos placas NYDIANET
- Cuatro pulsadores conectados a cada placa a modo de entradas
- Cuatro bombillas conectadas a cada placa. Dos a salidas reguladas mediante *TRIAC* y dos a salidas conmutadas mediante relé.
- Un enchufe hembra externo por cada placa conectado a una salida por relé.
- Un bus RS-485 cableado entre ambas placas más un conversor RS-232 a RS-485 para la conexión de un PC en el bus.
- Un transformador de tensión a 12 voltios para alimentar las placas.
- Un cable para conectar el panel a la red eléctrica.



Esquema 2.11 Diagrama de conexiones básicas de cada placa del panel

Gracias a este panel fue posible la comprobación y verificación de muchas de las funcionalidades implementadas durante el desarrollo del proyecto, facilitando en gran medida la implementación de buena parte de la arquitectura funcional definida.

La topología en bus que proporciona RS-485 hizo posible la conexión al panel de un ordenador personal para la monitorización y depuración del protocolo de comunicaciones. Para realizar esta conexión fue necesario el uso de un conversor de RS-232 (interfaz soportada por el PC) a RS-485.

La inclusión del PC en el bus permitió además la simulación de la existencia de un nodo central en la red domótica.

2.2.4 Nivel físico: RS-485

La interfaz RS485 fue desarrollada (análogamente a la interfaz RS422) para la transmisión en serie de datos de alta velocidad a grandes distancias. Mientras que RS422 sólo permite la conexión unidireccional de hasta 10 receptores en un transmisor, RS485 está concebida como sistema bus bidireccional al que se pueden conectar hasta 32 nodos. Físicamente las dos interfaces sólo se diferencian en pequeños detalles. El bus RS485 puede instalarse tanto en una configuración de 2 hilos como de 4 hilos.

Dado que pueden existir varios transmisores trabajando en una línea común, tiene que garantizarse, con un protocolo, que en todo momento esté activo como máximo un transmisor de datos. Los otros transmisores tienen que encontrarse en ese momento en estado de alta impedancia escuchando los datos que son transmitidos.

La norma RS485 define solamente las especificaciones eléctricas para receptores y transmisores diferenciales en sistemas de bus digitales. La norma ISO 8482 estandariza además adicionalmente la topología de cableado con una longitud máxima de 500 metros. Es decir, que el estándar sólo define el nivel físico del bus. El resto debe ser definido por el diseñador de la aplicación concreta en la que se esté usando el bus RS485.

El bus de 2 hilos RS485 se compone según la Figura 2.12 del cable pareado propio del bus con una longitud máxima de 500m. Los nodos se conectan a este cable a través de una línea adaptadora de máximo 5 metros de largo. La ventaja de la técnica de 2 hilos reside esencialmente en la capacidad multi-master, en donde cualquier participante puede cambiar datos en principio con cualquier otro. El bus de 2 hilos es básicamente apto sólo para comunicaciones half-dúplex. Es decir, puesto que sólo hay disponible un medio de transmisión, nunca puede enviar datos más de un nodo simultáneamente. Sólo después de finalizarse el envío de datos por parte de un nodo, pueden por ejemplo, responder otros. La aplicación más conocida basada en la técnica de 2 hilos es el PROFIBUS.

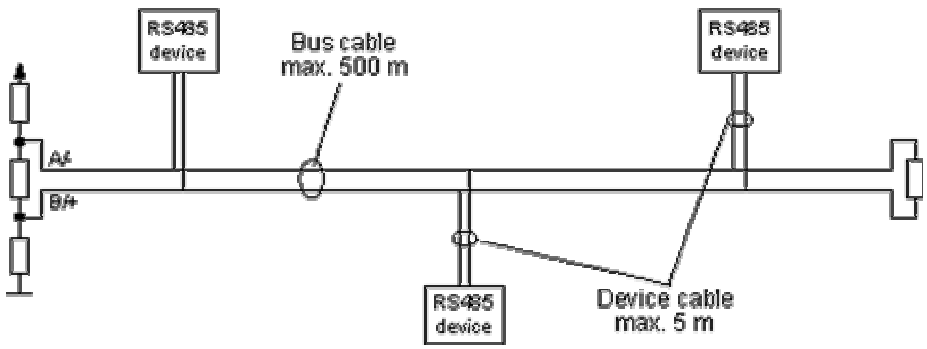


Figura 2.12 Bus de 2 hilos

La técnica de 4 hilos usada por ejemplo por el bus de medición DIN (DIN 66 348) sólo puede ser usada por aplicaciones Master-Slave (Maestro-Esclavo). Conforme a la Figura 2.13 se cablea la salida de datos del Maestro a las entradas de datos de todos los Servidores. Las salidas de datos de los Servidores están conectadas conjuntamente en la entrada de datos del Maestro.

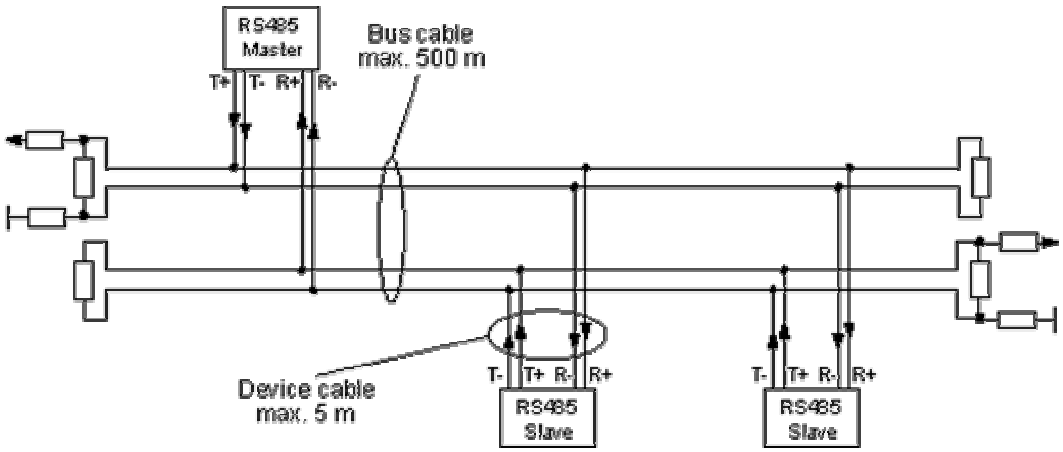


Figura 2.13 Bus de 4 hilos

Los datos en serie, como en interfaces RS422, se transmiten sin referencia de tierra como diferencia de tensión entre dos línea. Para cada señal a transmitir existe un par de conductores que se componen de una línea de señales invertida y otra no invertida. La línea invertida se caracteriza por regla general por el índice "A" o "-", mientras que la línea no invertida lleva "B" o "+". El receptor evalúa solamente la diferencia de tensión existente entre ambas líneas, de modo que los modos comunes de perturbación en la línea de transmisión no falsifican la señal útil. Los transmisores RS485 ponen a disposición bajo carga un nivel de salida de ±2V entre las dos salidas; los módulos de recepción reconocen el nivel de ±200mV como señal válida.

La asignación de tensiones diferenciales con estados lógicos se define del modo siguiente:

- A-B < -0,3 V = MARK = OFF = Lógico 1
- A-B > +0,3V = SPACE = ON = Lógico 0

Usando un método de transmisión simétrico en combinación con cables de pares de baja capacidad y atenuación (pares trenzados) pueden realizarse conexiones muy eficaces cubriendo distancias de hasta 500m con tasas de transmisión altas. El uso de un cable de pares trenzado de alta calidad evita por un lado la diafonía entre las señales transmitidas y por el otro reduce, adicionalmente al efecto del apantallamiento, la sensibilidad de la instalación de transmisión contra señales perturbadoras externas.

En conexiones RS485 es necesario el uso de redes de terminación para forzar el nivel de tensión de “escucha” (cuando no se transmite) en el bus cuando no esté activo ningún transmisor de datos. Esto se suele hacer con resistencias de terminación

cuyo valor recomendado es de 120 ohmios. No obstante, para distancias cortas o cuando la tasa de transmisión es muy baja, estas resistencias son prescindibles.

En la instalación tiene que tenerse cuidado con la polaridad correcta de los pares de cables, puesto que una polaridad falsa lleva a una inversión de las señales de datos. En caso de instalación de nuevos terminales debe tenerse cuidado con este factor.

Las mediciones de diferencia (medición Bus A contra B), especialmente con un osciloscopio, sólo pueden realizarse con un aparato de medición separado galvánicamente del potencial de masa. Si se pone el punto de referencia de la entrada de medición en Masa, esto lleva a un cortocircuito en la medición en un bus RS485.

Tal como se ha visto, en el caso del presente proyecto, cuyo objetivo es, entre otros, establecer un medio de comunicación entre diferentes nodos de una red domótica, el bus RS485 cumple con los requisitos de distancia (500 metros) y número de cargas (32) que se pueden encontrar en una vivienda de dimensiones habituales.

2.3 Software empleado

Un apartado fundamental dentro de las tecnologías empleadas para el diseño y desarrollo de un sistema domótico centralizado es el de las herramientas software utilizadas.

La principal herramienta software es sin duda el compilador de lenguaje C para microcontroladores *PIC*. Gracias a esta herramienta es posible la programación de estos dispositivos mediante el uso de un lenguaje de alto nivel como es el C.

Por otro lado, aunque el compilador posee su propio entorno de desarrollo, la elección del *ICD2* (Apartado 2.2.2) como herramienta hardware hizo aconsejable el uso del software de *MicroChip MPLAB IDE*. La elección de otra herramienta para tal fin hubiera supuesto el desaprovechamiento de buena parte de las capacidades que dicho hardware ofrece.

2.3.1 Compilador

Si se desea realizar la programación de microcontroladores *PIC* en un lenguaje de alto nivel como es el C, es preciso utilizar un *compilador de C*.

Dicho compilador genera ficheros en formato hexadecimal de *Intel*, que es el necesario para programar (utilizando un programador de *PIC* adecuado, como el *ICD2* visto en el apartado 2.2.2) un microcontrolador de 6, 8, 18 ó 40 patillas.

El compilador de C que se ha utilizado es el *PCW* de la casa *CCS Inc*. A su vez, el compilador se integra en un entorno de desarrollo integrado (*IDE*) que nos va a permitir desarrollar todas y cada una de las fases de que se compone un proyecto, desde la edición hasta la compilación pasando por la depuración de errores. La última fase, a excepción de la depuración software y hardware finales, será programar el PIC.

El compilador traduce el código C del archivo fuente (.C) a lenguaje máquina para los microcontroladores *PIC*, generando así un archivo en formato hexadecimal (.HEX). Además de éste, también genera otros seis ficheros que contienen el listado de instrucciones ensamblador equivalente al código escrito y un registro de errores que se hayan generado durante la compilación entre otras cosas.

En este apartado nos centraremos en todos aquellos aspectos de la programación de microcontroladores en C que difieren del estándar ANSI C. Por tanto, no se pretende hacer un resumen del lenguaje C si no que nos centraremos en las directivas de preprocesado y librerías usadas para el desarrollo del presente proyecto.

2.3.1.1 Tipos de datos

Algunos de los tipos de datos usados difieren de los definidos por el estándar ANSI C. Estos son los que se han usado en el presente proyecto:

- **int1**: Entero de un sólo bit.
- **int8**: Entero de 8 bits.
- **int16**: Entero de 16 bits.

2.3.1.2 Directivas de preprocesado utilizadas

La mayoría de las directivas de preprocesado del C estándar son soportadas por el compilador de *CCS*. En este apartado nos centraremos en aquellas que permiten la configuración de diferentes aspectos del funcionamiento hardware del microcontrolador.

#BIT identificador = x.y

Esta directiva creará un identificador "id" que puede utilizarse como cualquier SHORT INT (entero corto; un bit). El identificador referenciará un objeto en la posición de memoria **x** más el bit de desplazamiento **y**.

Ejemplos:

```
#bit tiempo = 3.4
int resultado;
#bit resultado_impar = resultado.0
...
if (resultado_ impar)
...
```

#BYTE Identificador = x

Esta directiva creará un identificador "id" que puede utilizarse como cualquier variable (de tamaño un byte). El identificador referenciará un objeto en la posición de memoria x, donde x puede ser una constante u otro identificador. Si x es otro identificador, entonces éste estará localizado en la misma dirección que el identificador "id".

Ejemplos:

```
#byte status = 3
#byte port_b = 6
struct {
short int r_w;
```

```
short int c_d;
int no_usado : 2;
int dato : 4; } port_a;
#byte port_a = 5
...
port_a.c_d = 1;
```

#LIST

Guarda el código fuente en el archivo .LST (listado de instrucciones de lenguaje ensamblador).

#DEVICE chip

Esta directiva define al compilador la arquitectura hardware utilizada. Esto determina la memoria RAM y ROM así como el juego de instrucciones. Para los chips (microcontroladores, memorias, etc.) con más de 256 bytes de RAM se puede seleccionar entre punteros de 8 o 16 bits. Para usar punteros de 16 bits hay que añadir **=16* después del nombre del chip o en una nueva línea después de la declaración del chip.

Ejemplos:

```
#device PIC16C67 *=16
#device PIC16C74
#device *=16
```

#FUSES opciones

Esta directiva define qué bits de configuración (literalmente “fusibles”) deben activarse en el dispositivo cuando se programe. Esta directiva no afecta a la compilación; sin embargo, esta información se pone en el archivo de salida. La opción SWAP tiene la función especial de intercambiar, los bytes alto y bajo de los datos que no son parte del programa, en el archivo Hex. Esta información es necesaria para algunos programadores de dispositivos. Algunas de las opciones más usadas son:

- LP, XT, HS, RC (para la configuración del reloj).
- WDT, NOWDT (configuración del *Watch Dog Timer*).
- PROTECT, NOPROTECT (protección de la ROM).
- PUT, NOPUT (Power Up Timer).
- BROWNOUT, NOBROWNOUT (reset tras fallo de alimentación).
- SWAP.

Ejemplo:

```
#fuses HS, WDT
```

#INT_xxx

Estas directivas especifican que la función que les sigue es una función de interrupción.

Las funciones de interrupción no pueden tener ningún parámetro. Como es natural, no todas las directivas pueden usarse con todos los dispositivos. Las directivas de este tipo de que disponemos son:

- **#INT_EXT:** Interrupción externa.
- **#INT_RTCC:** Desbordamiento del timer0 (RTCC).
- **#INT_RB:** Cambio en uno de los pines B4, B5, B6 o B7.
- **#INT_AD:** Conversor A/D.
- **#INT_EEPROM:** Escritura en la EEPROM completada.
- **#INT_TIMER1:** Desbordamiento del timer1.
- **#INT_TIMER2:** Desbordamiento del timer2.
- **#INT_CCP1:** Captura de datos por CCP1.
- **#INT_CCP2:** Captura de datos por CCP2.
- **#INT_SSP:** Evento en puerto serie en modo esclavo (SPI, I2C).
- **#INT_PSP:** Evento en puerto paralelo esclavo.
- **#INT_TBE:** Búfer de transmisión vacío.
- **#INT_RDA:** Dato recibido disponible.
- **#INT_ADOF:** Desbordamiento del ADC (*PIC 14000*).
- **#INT_RC:** Cambio en pin del puerto C.
- **#INT_I2C:** Evento I^2C (*PIC 14000*).

El compilador salta a la función de interrupción cuando se detecta una interrupción. Es el propio compilador el encargado de generar el código para guardar y restaurar el estado del procesador.

También es el compilador quien limpiará el flag de la interrupción. Sin embargo, nuestro programa es el encargado de llamar a la *función ENABLE_INTERRUPT()* para activar previamente la interrupción junto con el flag global de interrupciones (*GLOBAL*).

Ejemplo:

```
#int_rda
byte_recibido()
{
 if(kbhit())
 {
   dato = getc();
 }
}
```

#USE DELAY (CLOCK=frecuencia)

Esta directiva indica al compilador la frecuencia del reloj del sistema, en ciclos por segundo, a la vez que habilita el uso de las funciones de retardo *DELAY_MS()* y *DELAY_US()*.

Opcionalmente podemos usar la opción *RESTART_WDT* para que el compilador reinicie el *WDT* durante las funciones de retardo.

Ejemplos:

```
#use delay (clock=20000000)
#use delay (clock=32000, RESTART_WDT)
```

#USE FAST_IO (puerto)

Esta directiva afecta al código que el compilador generará para las instrucciones de entrada y salida. Este método rápido de hacer E/S ocasiona que el compilador realice las operaciones de entrada y salida sin programar el registro de dirección (*TRIS_X*) cada vez que lea o escriba de un puerto. El puerto puede ser A-G según el dispositivo.

Ejemplo:

```
#use fast_io(A)
```

#USE RS232 (BAUD=baudios, XMIT=pin, RCV=pin...)

Esta directiva le dice al compilador la velocidad en baudios y los pines utilizados para la comunicación serie. Esta directiva tiene efecto hasta que se encuentra otra directiva RS232.

La directiva **#USE DELAY** debe aparecer antes de utilizar **#USE RS232**. Esta directiva habilita el uso de funciones tales como *GETCH*, *PUTCHAR* y *PRINTF*. Si la E/S no es estándar es preciso poner las directivas **FIXED_IO** o **FAST_IO** delante de **#USE RS232.**

A continuación se listan algunos de los modificadores que permiten configurar las comunicaciones serie del microcontrolador.

- **RESTART_WDT:** Hace que las funciones de lectura de bytes pongan a cero el *WDT* mientras esperan la recepción de un carácter.
- **INVERT:** Invierte la polaridad de los pines serie (normalmente no es necesario con el convertidor de nivel, como el MAX232). No puede usarse con el SCI interno.
- **PARITY=X:** Define la paridad. X es N (*none*), E (*even*), u O (*odd*).
- **BITS =X:** Define los bits que se transmiten por “byte”. X es 5-9 (no puede usarse 5-7 con el SCI).
- **FLOAT_HIGH:** Se utiliza cuando se usan dispositivos cuyos pines requieren que las salidas sean de colector abierto.
- **ERRORS:** Indica al compilador que guarde los errores recibidos en la variable *RS232_ERRORS* para poder tratarlos cuando se produzcan.
- **BRGH1OK:** Permite velocidades de transmisión bajas en chips que tienen problemas de transmisión. Cuando utilizamos dispositivos con *SCI* y se especifican los pines *SCI*, entonces se usará el *SCI*. Si no se puede alcanzar una tasa de baudios dentro del 3% del valor deseado utilizando la frecuencia de reloj actual, se generará un error.
- **ENABLE=pin:** El pin especificado estará a nivel alto durante la transmisión.

Ejemplo:

```
#use rs232(baud=9600, xmit=PIN_A2, rcv=PIN_A3)
```

#USE RTOS (timer=X, minor_cycle=y...)

Esta directiva de preprocesado permite hacer uso del sistema operativo en tiempo real que el compilador incluye como una funcionalidad opcional. Las características de este sistema se verán en detalle en la sección 2.3.1.4.

#TASK (rate=time, max=time, queue=bytes)

Esta directiva permite indicar al compilador que la función que le sigue es una tarea del sistema operativo en tiempo real. Su sintaxis y descripción se verán en detalle en la sección 2.3.1.4.

2.3.1.3 Funciones utilizadas

A continuación se hará una breve descripción de las funciones específicas de las librerías del compilador de *CCS* que se han utilizado en este proyecto con el fin de facilitar la comprensión del código fuente que aparece en el capítulo 7 del presente documento.

c = fgetc(), c = getc()

Estas funciones esperan un carácter por el pin de recepción del dispositivo RS232 y devuelven el carácter recibido.

Es preciso utilizar la directiva *#USE RS232* antes de la llamada a esta función para que el compilador pueda determinar la velocidad de transmisión y el pin utilizado. La directiva *#USE RS232* permanece efectiva hasta que se encuentre otra que anule la anterior. Los procedimientos de E/S serie exigen incluir *#USE DELAY* para ayudar a sincronizar de forma correcta la velocidad de transmisión.

La variante *fgetc()* permite especificar en qué "canal" se desea realizar la lectura, en caso de haber añadido más de una directiva *#USE RS232* con diferentes etiquetas (mediante la opción *stream = nombre*).

Ejemplo:

```
printf("Continuar (s,n)?");
do {
 respuesta=getc();
} while(respuesta!='s'&& respuesta!='n');
```

fputc (), putc()

Estas funciones envían un carácter a través del pin de transmisión del dispositivo RS232. Es preciso utilizar la directiva *#USE RS232* antes de la llamada a esta función para que el compilador pueda determinar la velocidad de transmisión y la patilla utilizada.

La variante *fputc()* permite especificar en qué "canal" se desea realizar la escritura, en caso de haber añadido más de una directiva *#USE RS232* con diferentes etiquetas (mediante la opción *stream = nombre*).

Ejemplo:

```
if (checksum==0)
 putc(ACK);
else
 putc(NAK);
```

kbhit()

Esta función devuelve *TRUE* si existe un byte listo para ser leído en el búfer de recepción del microcontrolador.

Ejemplo:

```
if (kbhit ())
 keypress = getc();
```

setup_adc(mode)

Esta función permite establecer los parámetros de configuración del conversor analógico/digital. Los modos son los siguientes:

- ADC_OFF
- ADC_CLOCK_DIV_2
- ADC_CLOCK_DIV_8
- ADC_CLOCK_DIV_32
- ADC_CLOCK_INTERNAL

Ejemplo:

```
setup_adc(ADC_CLOCK_INTERNAL);
```

output_high(pin)

Esta función establece el estado del pin indicado a un 1 lógico (nivel alto). El método de acceso de E/S depende de la última directiva *#USE *_IO* utilizada.

Ejemplo:

```
output_high(PIN_A0);
```

output_low(pin)

Esta función establece el estado del pin indicado a un 0 lógico (nivel bajo). El método de acceso de E/S depende de la última directiva *#USE *_IO* utilizada.

Ejemplo:

```
output_low(PIN_A0);
```

set_tris_a(value), set_tris_b(value), set_tris_c(value)

Estas funciones permiten escribir directamente los registros para la configuración de la dirección de los puertos (entrada o salida).

Estas funciones deben usarse cuando se ha hecho uso de la directiva *FAST_IO()* y cuando se accede a los puertos de E/S como si fueran posiciones de memoria, igual que cuando se utiliza una directiva *#BYTE*. Cada bit de *value* representa un pin. Un '1' indica que la patilla es de entrada y un '0' que es de salida.

Ejemplo:

```
set_tris_b(0x0F);
```

enable_interrupts(int_xxx)

Esta función activa la interrupción *int_xxx*. La etiqueta *GLOBAL* habilita todas las interrupciones que estén activadas de forma individual.

Ejemplo:

```
enable_interrupts(INT_RDA);// Quedan habilitadas
                           // estas dos interrupciones,
enable_interrupts(INT_EXT);// pero hasta que no se
                           // habilite GLOBAL, no
                           // podrán activarse
enable_interrupts(GLOBAL); // Ahora sí se pueden
                           // producir las interrupciones
                           // anteriores
```

disable_interrupts(int_xxx)

Esta función desactiva la interrupción *int_xxx*. La etiqueta *GLOBAL* prohíbe todas las interrupciones, aunque estén habilitadas o permitidas.

clear_interrupt(int_xxx)

Limpia el flag de interrupción de la interrupción indicada como parámetro. No se puede usar con el indicador *GLOBAL*.

Ejemplo:

```
clear_interrupt(int_RDA);  //  Limpia  el  flag  de  la
interrupción por dato recibido
```

ext_int_edge(edge)

Esta función determina el flanco de activación de la interrupción externa. El flanco puede ser de subida (L_TO_H) o de bajada (H_TO_L).

Ejemplo:

```
ext_int_edge(L_TO_H);
```

set_timerx(value)

Estas funciones establecen el contador del timer o temporizador correspondiente al valor especificado. *RTCC* y timer 0 son el mismo.

Ejemplo:

```
set_timer1(0); //Reseteo del timer 1
```

setup_timer_x(mode)

Esta función inicializa el timer x. Los valores de *mode* deben ordenarse juntos, tal como se muestra en el ejemplo. Los valores de *mode* son:

Tx_DISABLED, Tx_INTERNAL, Tx_CLK_OUT, Tx_EXTERNAL, Tx_DIV_BY_1, Tx_DIV_BY_4, Tx_DIV_BY_8, etc (según de qué timer se trate)

Ejemplos:

```
setup_timer_1 (T1_DISABLED);
setup_timer_1 (T1_INTERNAL | T1_DIV_BY_4);
setup_timer_1 (T1_INTERVAL | T1_DIV_BY_8);
```

setup_adc_ports(value)

Esta función configura los pines del *ADC* para que sean analógicos, digitales o alguna combinación de ambos. Las combinaciones permitidas varían, dependiendo del dispositivo.

Las constantes usadas también son diferentes para cada integrado. Las constantes *ALL_ANALOG* y *NO_ANALOGS* son válidas para todos los chips.

Algunos otros ejemplos de constantes son:

- *RA0_RA1_RA3_ANALOG:* Esto hace que los pines A0, A1 y A3 sean analógicos y los restantes sean digitales. Los +5v se usan como referencia.
- *RA0_RA1_ANALOG_RA3_REF:* Las patillas A0 y A1 son analógicas; la patilla RA3 se usa como voltaje de referencia y todas las demás patillas son digitales.

Ejemplo:

```
setup_adc_ports( ALL_ANALOG );
```

setup_adc(mode)

Esta función prepara o configura el conversor A/D. Los modos son:

- ADC_OFF
- ADC_CLOCK_DIV_2
- ADC_CLOCK_DIV_8
- ADC_CLOCK_DIV_32
- ADC_CLOCK_INTERNAL

Ejemplo:

```
setup_adc(ADC_CLOCK_INTERNAL);
```

setup_ccpx(mode)

Estas funciones inicializan el módulo de captura, comparación y PWM (CCP). Para acceder a los contadores CCP se utilizan las variables CCP_1 y CCP_2. Los valores para *mode* son:

- CCP_OFF
- CCP_CAPTURE_FE
- CCP_CAPTURE_RE
- CCP_CAPTURE_DIV_4
- CCP_CAPTURE_DIV_16
- CCP_COMPARE_SET_ON_MATCH
- CCP_COMPARE_CLR_ON_MATCH
- CCP_COMPARE_INT
- CCP_COMPARE_RESET_TIMER
- CCP_PWM

read_eeprom(address)

Esta función lee un byte de la dirección *address* de la EEPROM interna del microcontrolador. El rango de direcciones válido depende del dispositivo seleccionado.

Ejemplo:

```
#define LAST_VOLUME 10
volume = read_EEPROM (LAST_VOLUME );
```

write_eeprom(address, value)

Esta función escribe un byte de datos (*value*) en la dirección de memoria EEPROM especificada (*address*). El rango de direcciones depende del dispositivo. Esta función puede tardar varios milisegundos en ejecutarse.

Ejemplo:

```
#define LAST_VOLUME 10
volume++;
write_eeprom(LAST_VOLUME,volume);
```

bit_clear(var,bit)

Esta función pone a '0' el dígito especificado con *bit* dentro del byte o palabra aportado en *var*. El bit menos significativo es el 0.

Esta función es exactamente igual que la siguiente operación binaria: var & = ~(1 << bit).

Ejemplo:

```
int x;
x=5;
bit_clear(x,2); // x = 1
```

bit_set(var,bit)

Esta función pone a '1' el dígito especificado con *bit* dentro del byte o palabra aportado en *var*. El bit menos significativo es el 0.

Esta función es igual que la operación: var | = (1 << bit).

Ejemplo:

```
int x;
x=5;
```

```
bit_set(x,3); // x = 13
```

bit_test(var,bit)

Esta función examina el dígito especificado en *bit* del byte o palabra aportado en *var*. Esta función es igual, aunque mucho más eficaz que esta otra forma: ((var & (1 << bit)) != 0).

Ejemplo:

```
if( bit_test(x,3) || !bit_test (x,1) ){
//o el bit 3 es 1 o el bit 1 es 0
}
```

_mul(x,y)

Esta función realiza una multiplicación optimizada entre dos multiplicandos que pueden de 8 o de 16 bits. Devuelve como resultado un valor de 32 bits si ambos parámetros son de 16 bits.

Para evitar carga de procesado evita la conversión a un tipo de dato de mayor precisión de los parámetros cuando estos son de un tipo diferente al devuelto por la función.

Ejemplo:

```
indice = _mul(ancho_columna, numero_fila);
```

rtos_run()

Ejecuta el despachador de tareas del Sistema Operativo en Tiempo Real (RTOS). Se verá en detalle en el apartado 2.3.1.4.

rtos_yield()

Devuelve el control al despachador de tareas del RTOS. Se verá en detalle en el apartado 2.3.1.4.

2.3.1.4 Sistema operativo en tiempo real

El Sistema Operativo en Tiempo Real de CCS (RTOS) es un sistema cooperativo de pila simple. Permite a los microcontroladores PIC definir y ejecutar regularmente

tareas programadas sin la necesidad del uso de interrupciones. Esto lo lleva a cabo un despachador de tareas (*rtos_run()*) que se implementa en tiempo de compilación a través de las directivas de preprocesado de definición de RTOS, es decir, *#use RTOS()* y *#task()*. Cuando a una tarea debe ejecutarse según su periodicidad, el despachador le da el control del procesador a esa tarea.

Cuando la tarea ha terminado de ejecutarse o no necesita hacer uso del procesador, esta le devuelve el control al despachador el cual le pasará el testigo a la siguiente tarea que deba ejecutarse según lo establecido en la programación de tiempos y periodos de ejecución. A esto se le conoce como multitarea cooperativa.

Esto proporciona al sistema bajo desarrollo la capacidad de programar tareas para que se ejecuten con una periodicidad concreta, además de la posibilidad de cooperar entre ellas mediante el uso de funciones como *rtos_yield()*, *rtos_enable()*, etc.

Un sistema operativo como este permite simplificar en gran medida la programación de tareas concurrentes, especialmente si el número de tareas es elevado. Además, con la posibilidad de programar la frecuencia con la que se ejecuta cada tarea, es muy fácil asignar prioridades a las diferentes funcionalidades que queramos implementar, permitiendo a aquellas que sean más críticas consumir más recursos del procesador.

Otra ventaja es la flexibilidad y sencillez a la hora de añadir tareas nuevas al ciclo de ejecución, capacidad que casa perfectamente con los objetivos del presente proyecto.

A continuación se describen las directivas de preprocesado necesarias para hacer uso del RTOS.

#USE RTOS (timer=X, minor_cycle=y…)

Esta directiva de preprocesado indica al compilador que debe implementar en tiempo de ejecución el RTOS. Los parámetros que se le pasan a esta directiva permiten configurar los siguientes aspectos:

- **Temporizador:** *timer = x*. Indica qué temporizador va a usar el despachador para programar las tareas.
- **Ciclo de ejecución mínimo:** *minor_cycle = time*. Es el máximo tiempo que una tarea podrá ejecutarse. La tasa de ejecución de cada tarea debe ser un múltiplo de esta cantidad. Si se omite este parámetro, el compilador lo calcula.

- **Estadísticas:** mediante la inclusión de la opción *statistics* el sistema operativo hace un seguimiento de los tiempos máximo, mínimo y total consumidos por cada tarea.

#TASK (rate=time, max=time, queue=bytes)

Esta directiva permite indicar al compilador que la función que le sigue es una tarea del sistema operativo en tiempo real. Este tipo de funciones no poseen parámetros de entrada ni devuelven valores como salida. Además, no se pueden invocar como a las funciones normales. Sólo el despachador puede hacerlo. Sus parámetros son los siguientes:

- **Tasa de ejecución:** *rate = time*. Permite indicar cada cuanto tiempo se ejecutará la tarea.
- **Tiempo de ejecución:** *max = time*. Indica el máximo tiempo que podrá emplear la tarea para ejecutarse. Si se omite, el compilador lo calcula.
- **Tamaño de la cola de mensajes:** *queue = bytes*. Indica la cantidad de memoria que se reservará para albergar el búfer de mensajes que las tareas pueden intercambiar entre ellas. Si se omite el compilador lo calcula.

A continuación se muestran las funciones relacionadas con el RTOS:

rtos_run()

Comienza la ejecución del RTOS. Todas las funciones de gestión de tareas son llevadas a cabo por esta función.

rtos_terminate()

Esta función pone fin a la ejecución del RTOS y la devuelve al programa original. Hace las veces de función de retorno de *rtos_run()*.

rtos_enable(task)

Habilita una tarea del RTOS. Una vez que una tarea es habilitada, la función *rtos_run()* la llamará cada vez que deba ejecutarse. El parámetro que se pasa a esta función es el nombre de la tarea a habilitar.

rtos_disable(task)

Desactiva una tarea. Una vez que la tarea está desactivada, la función *rtos_run()* no la invocará hasta que no se use sobre ella la función *rtos_enable()*. El parámetro es el nombre de la función a desactivar.

rtos_msg_poll()

Devuelve *TRUE* si hay datos en la cola de mensajes de una tarea.

rtos_msg_read()

Devuelve el siguiente byte de datos de la cola de mensajes de una tarea.

rtos_msg_send(task,byte)

Envía un byte de datos a la tarea especificada. Los datos se colocan en la cola de recepción de mensajes de la tarea destino.

rtos_yield()

Si se llama dentro de una tarea del RTOS, devuelve el control del programa a la función *rtos_run()*. Al volverse a invocar esa tarea, la ejecución continúa por dicha función. Sería recomendable forzar una llamada a esta función cada vez que se finalice la ejecución de una tarea para asegurar que ninguna tarea monopoliza el tiempo de ejecución por alguna razón no prevista.

rtos_signal(sem)

Incrementa el valor de un semáforo que se usa para conocer la disponibilidad de algún recurso compartido.

rtos_wait(sem)

Espera a que el recurso asociado al semáforo esté libre y entonces decrementa su valor para solicitar el uso del recurso compartido.

rtos_await(expre)

Espera a que la expresión lógica dada sea cierta para poder continuar con la ejecución de la tarea.

rtos_overrun(task)

Devuelve *TRUE* si la tarea ha agotado su tiempo asignado.

rtos_stats(task,stat)

Devuelve la estadística solicitada de la tarea especificada. Las estadísticas incluyen los tiempos mínimo y máximo de ejecución de la tarea y el tiempo total que se ha estado ejecutando.

2.3.2 Entorno de desarrollo

MPLAB IDE es un software que se ejecuta en un PC para desarrollar aplicaciones para microcontroladores de MicroChip. Se le denomina como *Entorno de Desarrollo Integral* o *IDE* (*Integrated Development Environment*) porque proporciona un entorno único donde desarrollar código para ese tipo de plataformas.

MPLAB IDE contiene todos los componentes necesarios para diseñar y desarrollar aplicaciones para sistemas embebidos.

Las tareas más habituales para llevar a cabo esta labor son:

- Creación del diseño a alto nivel: A partir de las especificaciones y funcionalidades requeridas se debe decidir qué dispositivo se va a utilizar y a partir de esa decisión realizar el diseño de todo el hardware que dependa de ello.
- Compilación del código: Con el uso del compilador adecuado se procederá a la generación del firmware que se cargará en el microcontrolador. En el presente proyecto, dicho compilador es una herramienta externa al entorno de desarrollo (véase apartado 2.3.1). No obstante, a través de la instalación de un *plug-in*, es posible integrarlo en el entorno de desarrollo de forma totalmente transparente al usuario.
- Programación del microcontrolador: Con el uso del hardware específico adecuado se realiza la programación del integrado.
- Depuración del código: La depuración del código se lleva a cabo tanto simulando como mediante el uso del módulo *ICD2* de *MicroChip* visto en el apartado 2.2.2.

MPLAB IDE proporciona herramientas para realizar las siguientes tareas:

- Edición de código: Posee un editor de código que reconoce la sintaxis de algunos lenguajes de programación. No obstante, no reconoce la sintaxis del lenguaje usado en el presente proyecto: CCS C. Sí reconoce el lenguaje ANSI C, de modo que la compatibilidad es limitada.

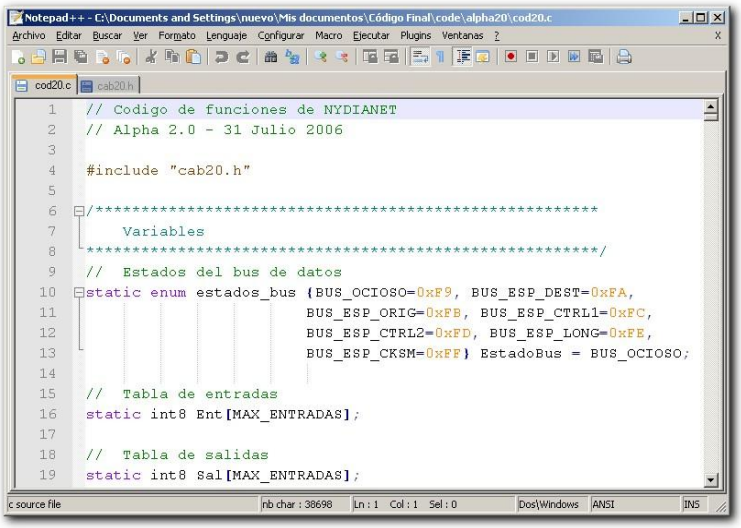
- Gestión de proyectos: Permite gestionar de manera sencilla los archivos que forman parte del proyecto en que se esté trabajando. Tanto ficheros de código y de cabecera como archivos de texto, imágenes, etc.
- Monitorización del proceso de programación: Este software es la interfaz visible entre el usuario y el programador *ICD2*. A través de él es posible interactuar con el módulo de programación así como recibir el estado de dicho hardware y los errores que se produzcan durante el proceso de programación.
- Monitorización del proceso de depuración: Durante la fase de depuración de código In-Circuit *MPLAB IDE* permite la visualización de los registros internos del microcontrolador, monitorizar variables dentro de nuestro código e incluso, gracias al plug-in antes mencionado, la ejecución paso a paso del código diseñado visualizándolo por parte del usuario. Igualmente permite la inserción de breakpoints y toda una serie de facilidades que hacen al proceso de programación y depuración indistinguible del que se seguiría cuando de una aplicación orientada a un PC se trata.

Gracias a todo esto, *MPLAB IDE* es una potente herramienta que permite acelerar el proceso de diseño y depuración de aplicaciones para microcontroladores *PIC*.

A modo de ejemplo se mostrarán los pasos que se deben seguir para la creación, compilación y depuración de una aplicación para microcontroladores de *MicroChip*:

Paso previo: Antes de poder trabajar con el compilador de *CCS* en el entorno de desarrollo MPLAB IDE es necesario asegurarse de tener instalados en el ordenador los siguientes elementos: Compilador de *CCS*, entorno de desarrollo *MPLAB IDE* y *plug-in* del compilador para poder integrarlo en MPLAB IDE. El *plug-in* podemos descargarlo de la página web del fabricante del compilador (www.ccsinfo.com)

Paso 1: Escribir el código: Lo más sencillo es escribir el código usando cualquier software de edición a gusto de cada usuario. El entorno de desarrollo posee herramientas de edición de código, pero con capacidades limitadas. Durante el presente proyecto se ha usado el software libre "*Notepad ++*" (Captura 2.14) para realizar esta labor.

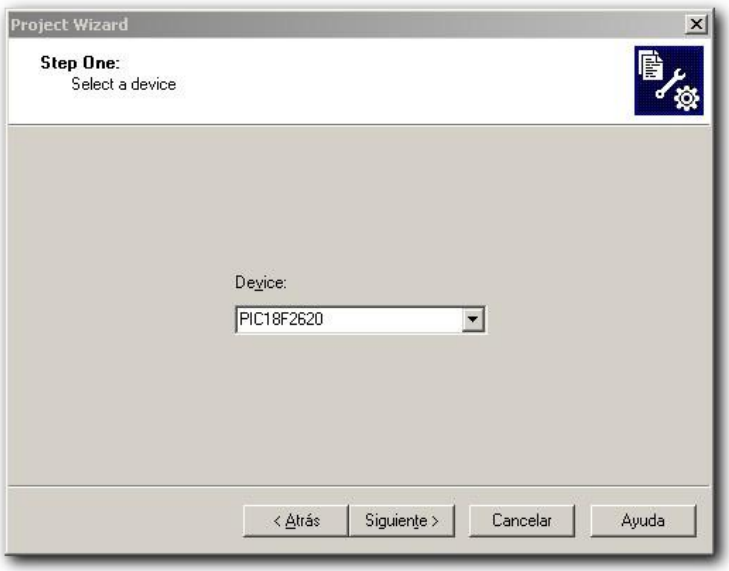


Captura 2.14 Editor de código fuente

Paso 2: Crear el proyecto de *MPLAB*: Lo más sencillo para llevar a cabo este paso es usar el asistente de creación de proyectos (Captura 2.15).

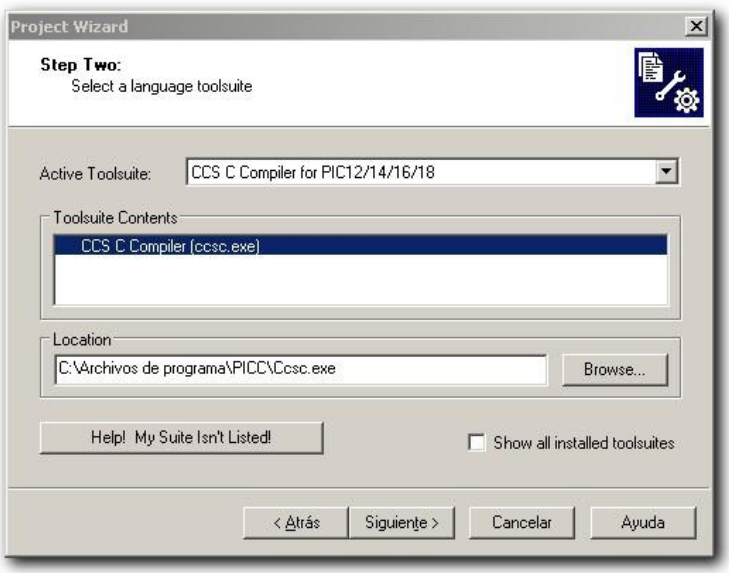
Captura 2.15 Asistente de creación de proyectos: Saludo Inicial

En el primer paso podremos seleccionar de manera rápida y sencilla el dispositivo para el que se va a desarrollar la aplicación (Captura 2.16).



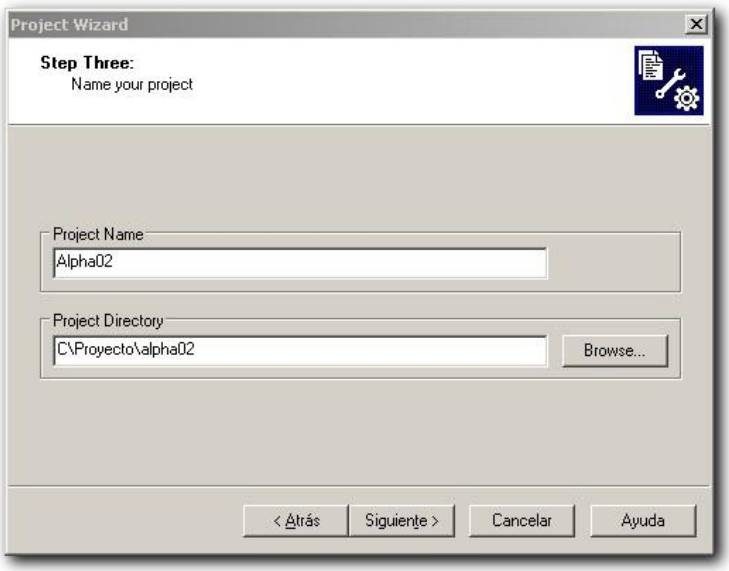
Captura 2.16 Selección de dispositivo

Seleccionamos el compilador que se va a utilizar (Captura 2.17). Gracias al *plug-in* instalado podremos seleccionar el compilador de *CCS* en este paso.



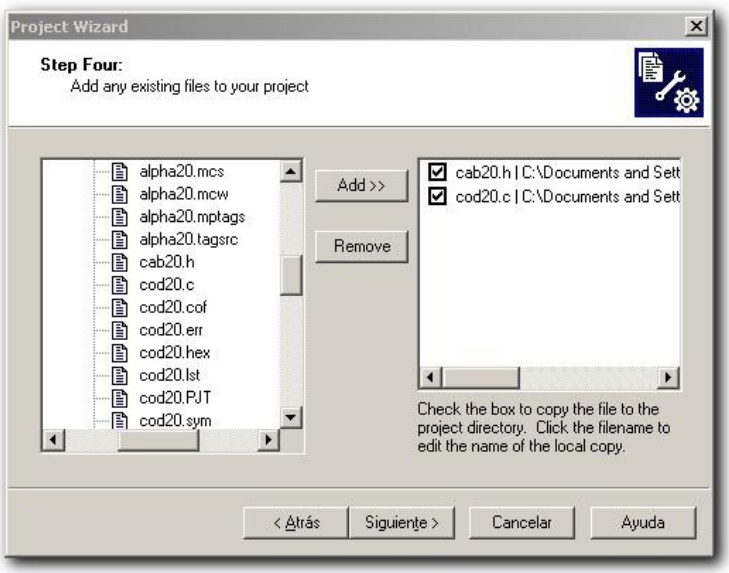
Captura 2.17 Selección de compilador

Después elegimos el nombre del proyecto y su localización en nuestro PC (Captura 2.18).



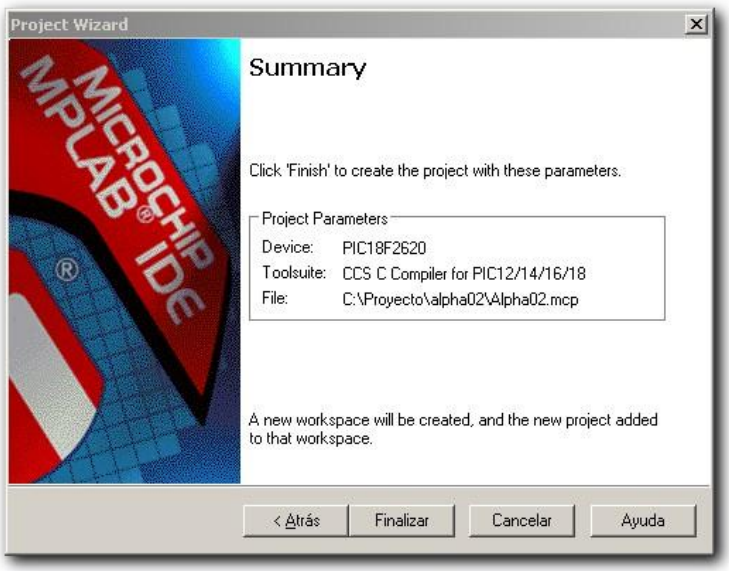
Captura 2.18 Selección de nombre y localización del proyecto

Finalmente, los ficheros que queremos incluir inicialmente en el proyecto (Captura 2.19).



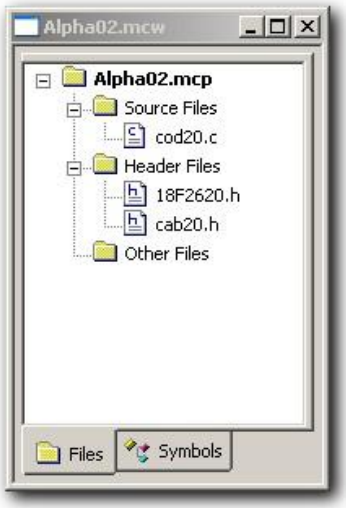
Captura 2.19 Selección de los ficheros a incluir en el proyecto

El asistente mostrará un resumen de las opciones escogidas a modo de colofón (Captura 2.20).



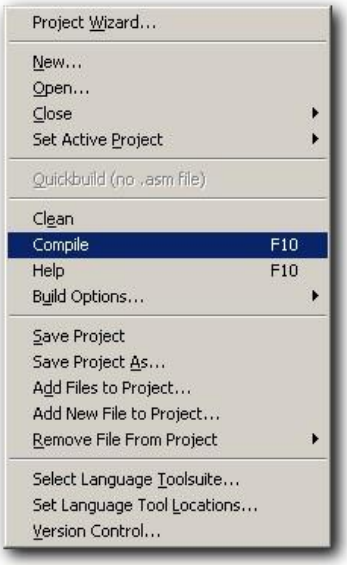
Captura 2.20 Resumen del asistente

Tras esto, en la ventana principal del entorno de desarrollo podremos acceder al visor de ficheros del proyecto, que permite gestionar rápidamente los ficheros que componen el proyecto actual (Captura 2.21).



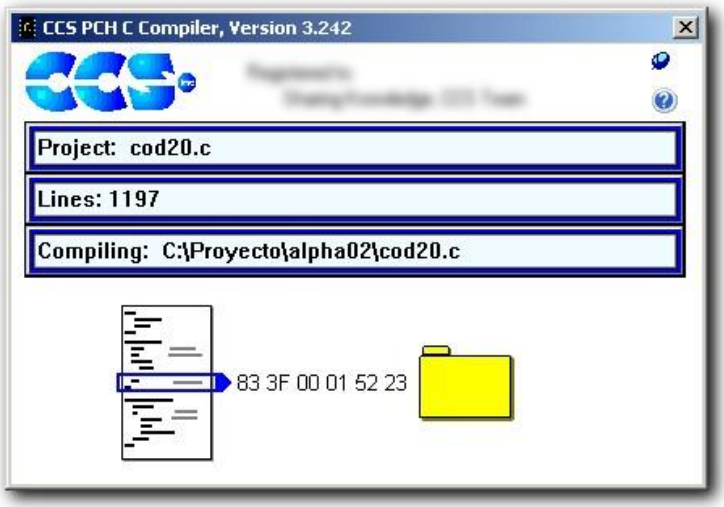
Captura 2.21 Visor de proyecto

Paso 3: Compilación: Para compilar el proyecto actual debemos ir al menú *Project* y seleccionar *Compile* (Captura 2.22).



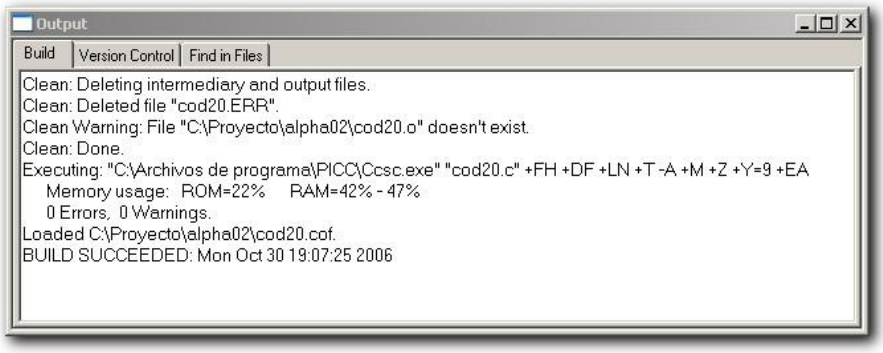
Captura 2.22 Menú para compilación

De este modo *MPLAB IDE* invocará al compilador de CCS con los parámetros que configuremos para llevar a cabo la compilación del código (Captura 2.23).



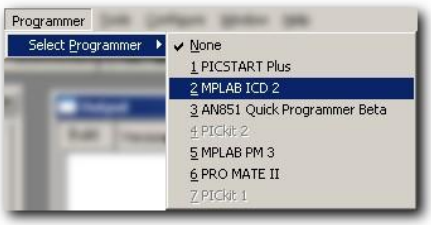
Captura 2.23 Ventana emergente del compilador

Durante este proceso, el entorno de desarrollo mostrará los resultados intermedios que se vayan obteniendo en la ventana *Output* (Captura 2.24).

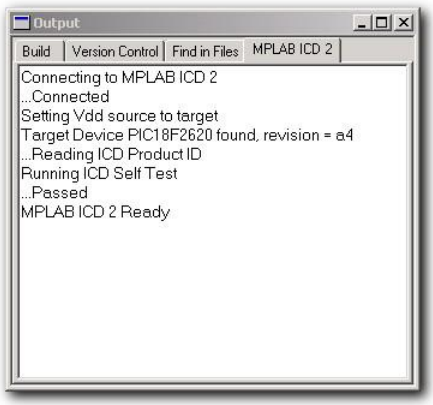


Captura 2.24 Informe de resultado de la compilación

Paso 4: Programación: Para la programación del micro debemos seleccionar el programador que vayamos a utilizar, que en nuestro es caso el *ICD2* (Captura 2.25) y establecer una conexión con la placa donde se ejecutará el software (Captura 2.26).



Captura 2.25 Menú de selección de programador

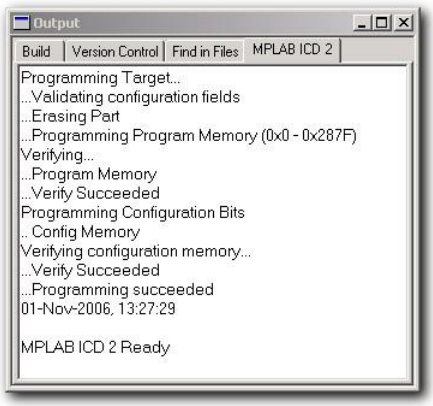


Captura 2.26 Confirmación de estado operativo del *ICD2*

Una vez realizado este paso programamos el micro accionando el icono de programación de la barra de tareas del *ICD2* (Captura 2.27).

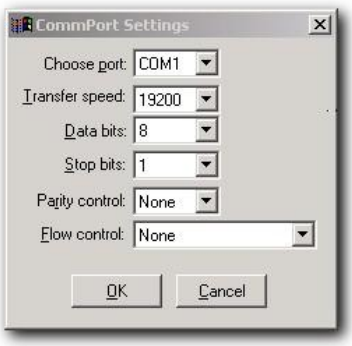
Captura 2.27 Barra de tareas del *ICD2*

Conforme se lleva a cabo la programación del micro, en la ventana *Output* se mostrarán los pasos intermedios y el resultado alcanzado (Captura 2.28).

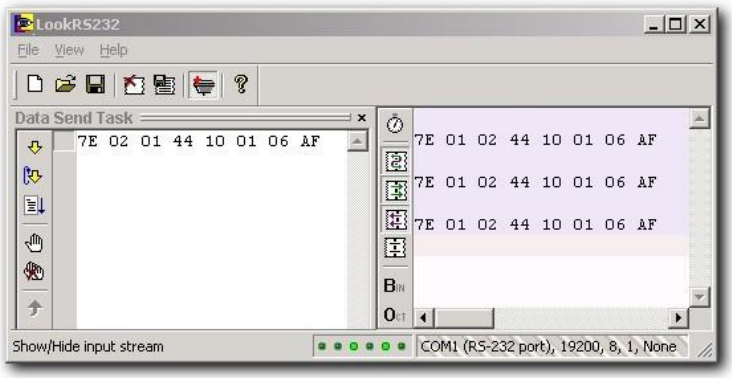


Captura 2.28 Monitorización del proceso de grabación del programa en el micro

Paso 5: Depuración: Para depurar el código debemos ejecutarlo. Existen muchas opciones a la hora de depurar, desde la ejecución paso a paso hasta la inserción de breakpoints o la lectura de la EEPROM interna del micro. Otro software que permitió la depuración de las funcionalidades de comunicación del sistema domótico fue el software de monitorización del puerto serie LookRS232. Mediante dicho programa, tras establecer una conexión con el puerto serie (Captura 2.29), fue posible usar el PC a modo de monitor de la red e incluso emular la existencia de un nodo central que enviase tramas a través del bus RS-485 (Captura 2.30).



Captura 2.29 Menú de configuración de conexión



Captura 2.30 Software para monitorización de la red