Universidad del País Vasco
Euskal Herriko Unibertsitatea

Facultad de Informática

Grado de Ingeniería Informática

▪ Proyecto Fin de Grado ▪

Ingeniería de Computadores

Reconversión de Protocolo de Comunicaciones MIWI™ XC16 en CCS

Autor: Odei Ayastuy

Directores del proyecto: Gonzalo Álvarez e Iván Piquer

Junio 2016

informatika fakultatea
facultad de informática

RESUMEN

Los objetivos principales de este proyecto son dos: el primero de ellos es migrar el protocolo Microchip Wireless (en adelante MiWi™) escrita para el compilador XC16 al lenguaje C orientado al compilador CCS. El segundo objetivo es crear *templates* para cada uno de los dispositivos y características exigidas por la empresa *Atelei Engineering*.

La elección de realizarlo en el lenguaje C orientado al compilador CCS se justifica en su fácil utilización en los microcontroladores PIC de Microchip, ya que a la hora de programar microcontroladores PIC el compilador CCS nos ofrece soluciones más sencillas y eficientes que el compilador XC16.

En este proyecto se ha realizado la migración solamente para para el *transceiver* MRF89XAM8A. Para ello se ha hecho uso de diferente hardware (placas, sniffer, debugger, etc.) como software (entorno de desarrollo, terminal, analizador lógico, etc.).

Para ello se ha realizado un análisis del protocolo y su código, se ha realizado la migración del código y su verificación. Por último se han creado los *templates* y una aplicación real y se ha documentado todo el proyecto.

Aunque tanto los dos objetivos principales como los objetivos transversales se hayan cumplido quedan algunas líneas interesantes en las que se puede trabajar en un futuro. Por ejemplo, hacer la migración para que el protocolo funcione con otros *transceivers*.

Indice del contenido

Lista de Figuras y Tablas

FIGURAS

TABLAS

Agradecimientos

La realización del presente proyecto habría sido imposible sin la firme colaboración de *Atelei Engineering*. Deseo agradecer, especialmente, a Iván Piquer, Iñigo Olaso y a David Ruiz por su constante entusiasmo, apoyo y dedicación a lo largo de todos estos meses.

El proyecto se ha elaborado bajo la dirección de Gonzalo Álvarez profesor de la Facultad de Informática de la Universidad del País Vasco, al que me gustaría reconocer por el trabajo y tiempo invertidos.

A mi familia y mis amigos/as por escucharme y alentarme, aún sin saber de lo que hablaba.

A todos y a todas vosotras, gracias.

1

INTRODUCCIÓN

1.1 Visión General

Después de la aparición de dispositivos electrónicos inteligentes, en los últimos años se están produciendo algunos cambios significativos en el mundo de la electrónica y la informática, y el concepto de *domótica* o *automatización del hogar* cada vez es más popular.

En rasgos generales podemos definir la domótica como la disciplina que se ocupa de la aplicación de la electrónica y de la informática en la vivienda dotándola de aplicaciones electrónicas o informáticas con objeto de mejorar las condiciones de habitabilidad. Es decir, la domótica se refiere a la utilización de un conjunto de tecnologías para controlar y automatizar de forma inteligente una vivienda. Esto permite que haya un mejor uso de la energía, así como la inclusión de una mayor seguridad, comodidad y comunicación entre el usuario/a y todos los sistemas de su hogar. Al hablar de domótica nos referimos a la automatización y control de aparatos y sistemas de instalaciones eléctricas y electrotécnicas (iluminación, puertas, persianas, etc.) de forma centralizada y/o remota.

Pero para lograr la automatización del hogar se trabaja con sistemas complejos con una gran variedad de elementos conectados entre sí, siendo imprescindible una organización rigurosa del sistema para que en su conjunto pueda funcionar correctamente, se deben definir unas reglas de automatización y de comunicación de manera que los dispositivos de percepción (sensores) comuniquen el estado actual de varios aspectos (actuadores) para poder llevar a cabo el principal objetivo de la domótica. Además, debe haber una interfaz para que el usuario pueda personalizar el sistema inteligente a su antojo. Todo esto debe convivir simultáneamente para que el usuario logre un sistema integral y de fácil implementación. [1]

Para implementar las reglas de comunicación mencionadas anteriormente existen diferentes tecnologías. En este proyecto se hará uso del protocolo de comunicación MiWi™ por ser el utilizado por la empresa de *Atelei Engineering*.

MiWi™ es un protocolo de comunicación inalámbrica para redes de área personal basada en el estándar IEEE 802.15.4 por la empresa Microchip Technology. Al ser un protocolo de Microchip está diseñado para ser empleado con sus componentes. Es por ello que está diseñado para funcionar con todos los *transceivers* de microchip que operan en las bandas ISM de 2.4 GHz y 868/915 MHz. También es transportable entre las distintas familias de MCU.

1.2 Objetivos

Este proyecto fue propuesto por la empresa *Atelei Engineering*. *Atelei Engineering* se dedica al diseño y desarrollo de productos electrónicos desde su conceptualización hasta su fabricación. Son diseñadores y fabricantes *OEM* (Original Equipment Manufacturer) de dispositivos wireless para control y acceso.

Los dispositivos fabricados por *Atelei Engineering*, forman una red inalámbrica invisible, que permite que éstos se comuniquen entre sí mediante mensajes. De esta manera es posible configurar la red para que, ante ciertos eventos en un dispositivo, cualquiera que se encuentre en la misma red, esté o no alejado de él, pueda llevar a cabo acciones.

Entre las familias de dispositivos fabricados y utilizados por la empresa se encuentran los denominados controladores. Podemos definir los controladores como un sistema de control

que tiene que permitir integrar productos de terceros, de forma que estos se integren en la instalación como un elemento más.

Los controladores se comunican entre ellos a través de una red, ésta red está creada en base al protocolo MiWi™. Este protocolo de red solamente está escrito en el lenguaje C orientado al compilador XC16. Dado que la citada empresa a la hora de programar PICs utiliza el compilador CCS el objetivo principal de este proyecto es migrar el código del protocolo para que pueda funcionar correctamente con el compilador CCS y la creación de *templates* de proyecto para cada uno de los protocolos y dispositivos. El desarrollo de éste conllevara objetivos y competencias transversales:

- Conocer el funcionamiento de una empresa y trabajar en ella.
- Comprender a fondo el protocolo de comunicación inalámbrica MiWi™.
- Dominar el lenguaje de programación C orientado al compilador CCS y XC16.
- Profundizar en la programación de los microcontroladores PIC (SPI, UART, Interrupciones, etc.).
- Profundizar en el software MPLAB X.

2

PLANIFICACIÓN Y GESTIÓN DEL PROYECTO

2.1 Alcance

Como se ha señalado anteriormente, el objetivo final de éste proyecto es migrar el protocolo MiWi™ escrito en C orientado al compilador XC16 al C orientado al compilador CCS V. 5.051 y crear *templates* de proyecto para cada uno de los protocolos y dispositivos. Para llevar a cabo la migración se consideró pertinente omitir las características no requeridas por *Atelei Engineering;* dadas las particularidades y restricciones (especialmente temporales) de un Proyecto Fin de Grado.

La migración partirá desde un ejemplo MiWi™ de *Microchip Libraries for Aplications* y perseguirá el objetivo de lograr un funcionamiento correcto con el compilador CCS; cumpliendo con las características solicitadas por *Atelei Engineering.* Dichas características son las siguientes:

- El protocolo MiWi™ para crear y gestionar la red ofrece tres protocolos de pila (se explicarán en el *apartado 3*). Nuestra migración tendrá que funcionar con los tres protocolos
- Se tendrá que emplear el *transceiver* MRF89XAM8A.
- Los paquetes enviados entre dos dispositivos deberán tener la opción de ir encriptados.
- Cualquier dispositivo final tendrá que tener la capacidad dormirse para ahorrar energía. Al despertar deberá recibir los mensajes que no haya recibido mientras estaba dormido.
- Si en un momento dado la red se cae, al volver a iniciar el protocolo, los dispositivos deberán tener la opción de recuperar el estado de la red anterior.

2.2 Estructura de desglose del trabajo

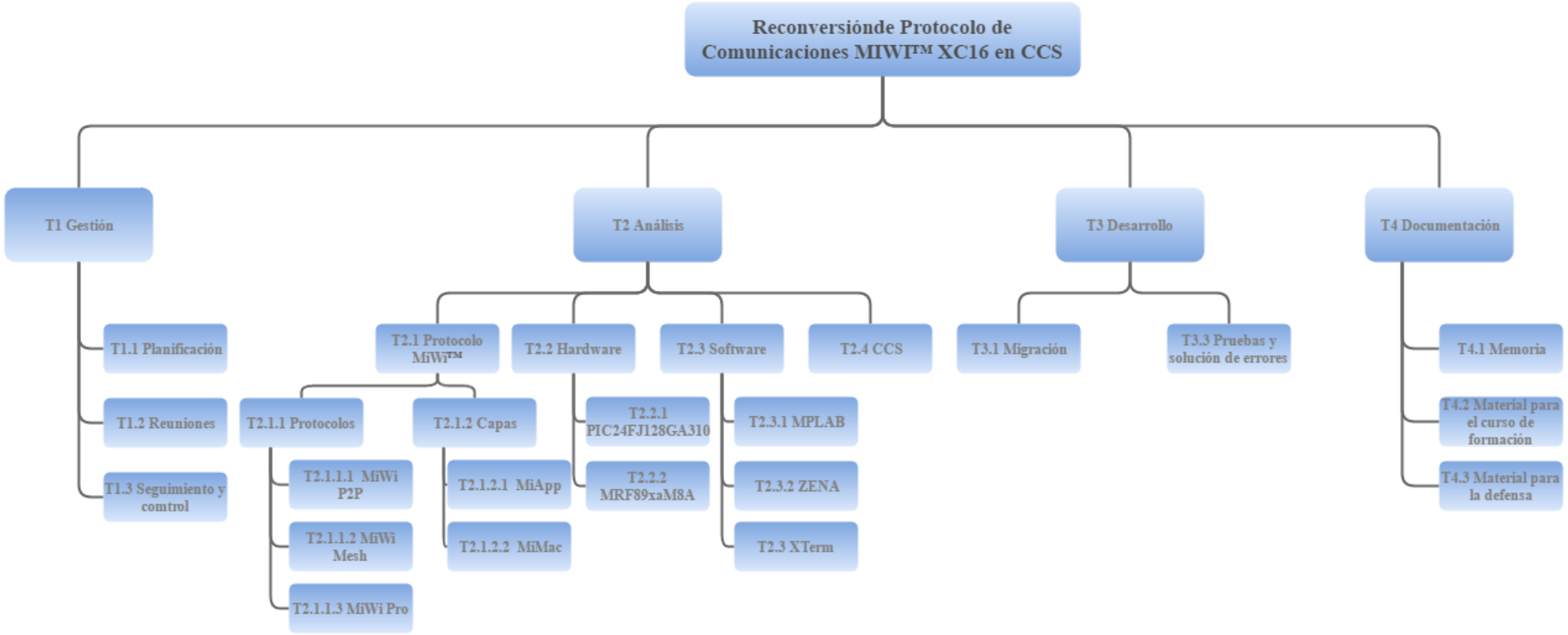


Ilustración 1: Estructura de desglose de trabajo

Para facilitar el desarrollo, la redacción y la futura lectura-compresión del proyecto se decidió dividirlo en cuatro bloques generales:

- **Gestión del Proyecto**: corresponde a la planificación, puesta en marcha y seguimiento y control del proyecto. También se refiere a las reuniones, tanto con los profesionales de *Atelei Engineering*, como con el profesor Gonzalo Álvarez.
- **Análisis del proyecto:** hace referencia a la parte más teórica del proyecto. A lo largo de este bloque se analizara: el protocolo MiWi™ en profundidad, el hardware en el que deberá funcionar el protocolo, el software utilizado, etc.
- **Desarrollo del proyecto:** en este bloque se pondrá en práctica todo lo analizado y se creará el producto.
- **Documentación:** éste último bloque lo conformaran la memoria, el material para el curso de formación en la empresa y el material de la defensa.

2.3 Tareas y sus características

- T1 Gestión
  - T1.1 Planificación
    - T1.1.1 Definición del alcance del proyecto con la empresa.
    - T1.1.2 Definición de la estructura de desglose de trabajo.
    - T1.1.3 Definición de las tareas.
    - T1.1.4 Identificar y definir los entregables del proyecto.
    - T1.1.5 Planificar el tiempo
      - Hacer el diagrama Gantt
      - Estimar la dedicación
    - T1.1.6 Hacer el plan de calidad
    - T1.1.7 Definir la metodología de trabajo que se vaya a utilizar para llevar a cabo el proyecto.
    - T1.1.8 Definir un plan de comunicación con la empresa.
    - T1.1.9 Hacer un plan de riesgo tanto para el producto como para el proyecto.
- T1.2 Reuniones
  - T1.2.1 Preparar las reuniones y llevarlas a cabo
- T1.3 Seguimiento y control
  - T1.3.1 Sacar las horas reales invertidas.
  - T1.3.2 Identificar las variaciones.
  - T1.3.3 Sacar las conclusiones de las variaciones entre las horas estimadas y las horas invertidas
- T2 Análisis
  - T2.1 Protocolo MiWi™
    - T2.1.1 Protocolos
      - T2.1.1.1 MiWi P2P
      - T2.1.1.2 MiWi Mesh
      - T2.1.1.3 MiWi Pro
    - T2.1.2 Capas
      - T2.1.2.1 MiApp
      - T2.1.2.2 MiMac
  - T2.2 Hardware

        - T2.2.1 PIC24F128GA310
        - T2.2.2 MRF89xaM8A
    - T2.3 Software
        - T2.3.1 MPLAB X
            - T2.3.1.1 Instalación
            - T2.3.1.1 Funcionalidades
        - T2.3.2 Wireless Development Studio
            - T2.3.2.1 Instalación
            - T2.3.2.2 Funcionalidades
        - T2.3.3 XTerm
            - T2.3.3.1 Instalación
            - T2.3.3.2 Funcionalidades
    - T2.4 CCS
            - T2.4.1 Variables
            - T2.4.2 Configuración del hardware
            - T2.4.3 Funciones
            - T2.4.4 Temporizaciones
            - T2.4.5 Interrupciones
            - T2.4.6 Bibliotecas
            - T2.4.7 Directivas
- T3 Desarrollo
    - T3.1 Migración
    - T3.2 Pruebas y solución de errores
    - T3.3 Creación de *templates*
- T4 Documentación.
        - T4.1 Memoria
        - T4.3 Material para el curso de formación
        - T4.4 Material para la defensa.

2.4 Características e identificación de los entregables

- El producto: el producto lo componen, por un lado, todo el hardware prestado por la empresa para llevar a cabo el proyecto y, por otro lado, la implementación del protocolo inalámbrico MiWi™ que no deberá dar errores al compilar con CCS y a su vez deberá funcionar correctamente.
- La memoria: es el documento en el que queda recogido los resultados y el desarrollo del proyecto. Este documento deberá cumplir todos los requisitos impuestos por la universidad. Este documento se tendrá que subir a la plataforma digital ADDI para el día 24 de junio de 2016.
- Material para el curso de formación en la empresa: no todos los profesionales de *Atelei Engineering* han trabajado con el protocolo MiWi™, por este motivo, se decidió por petición del tutor de la empresa ofrecer un curso de formación sobre dicho

protocolo. El material utilizado será un ejemplo con apoyo visual en diapositivas complementado con una explicación del protocolo, sus características y su modo de utilización. El curso de formación se realizará el 6 de julio de 2016.

- El material para la defensa: la defensa del proyecto se llevara a cabo entre el día 8 de julio de 2016 y 12 de julio de 2016. En la defensa, se explicara cómo se ha llevado a cabo el proyecto y lo que se ha hecho de forma sencilla y clara. La defensa será pública ante un tribunal y la explicación se apoyara en diapositivas.

2.5 Planificación del tiempo

2.5.1 Diagrama de Gantt

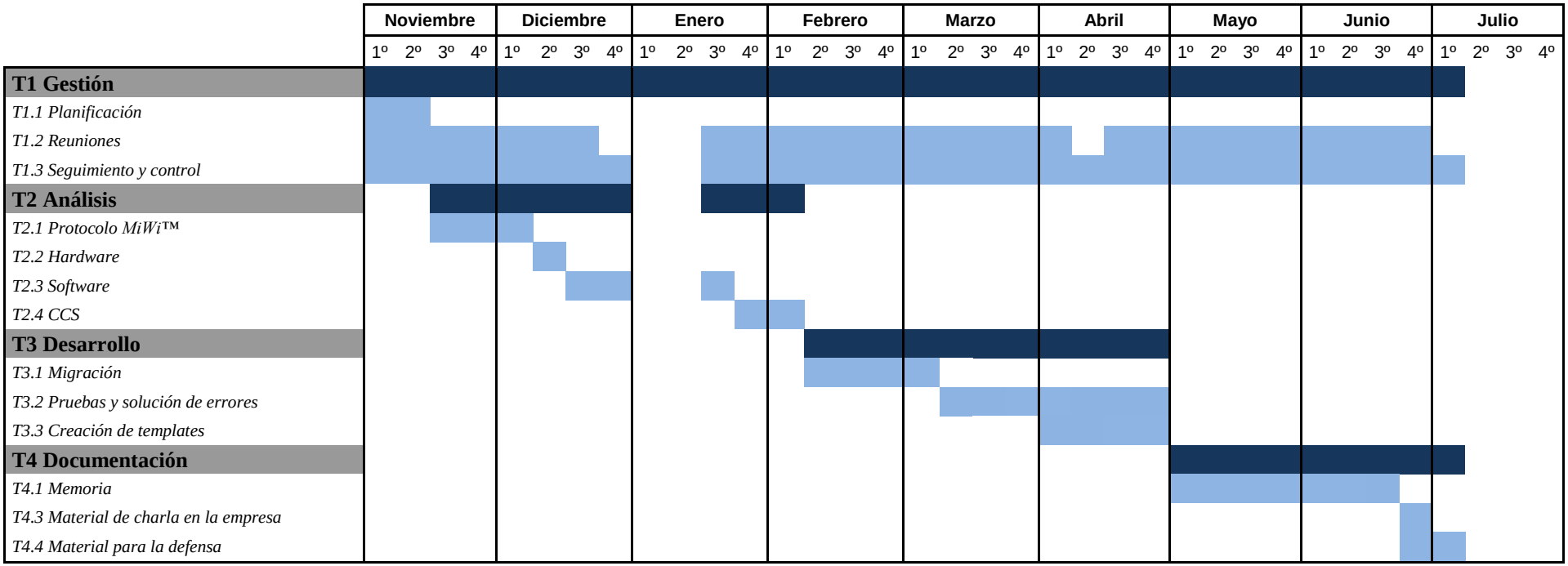


Tabla 1: Diagrama de Gantt

Según el diagrama de Gannt (véase la *tabla 1*) para elaborar la planificación del tiempo se ha decidido dividir el proceso en 4 etapas relacionadas pero diferenciadas; Gestión, análisis, desarrollo y documentación.

La única etapa transversal a todo el proyecto es la **gestión del proyecto**, por razones obvias. Las dos primeras semanas de esta etapa se espera que se dediquen a la planificación, es decir, se decidirá la estrategia para concluir satisfactoriamente el proyecto. Las reuniones con el tutor de la empresa están previstas con una frecuencia semanal (a excepción de épocas de exámenes y festividades) y el seguimiento y control se ejecutará todas las semanas que haya previstas tareas.

El objetivo del **análisis o la recogida de información** es examinar y comprender los diferentes elementos clave en el desarrollo del programa; Protocolo MiWi™, Hardware, Software y CCS. Para ello se estima una duración temporal de 9 semanas. Durante las dos primeras se recopilará información sobre los diferentes protocolos de pila que tiene el protocolo MiWi™ (P2P, Mesh y Pro). Se continuará analizando las capas del protocolo MiWi™ (MiApp y MiMac) y, para seguir con el análisis, se espera dedicar una semana a la documentación sobre el microchip (PIC24FJ128GA310) y el *transceiver* (MRF89xaM8A). Después, se proseguirá con la instalación y la comprensión de las utilidades del software que se va a emplear para programar (MPLAB X) y para las pruebas (XTerm y Wireless Development Studio). Para terminar, y dado que uno de los objetivos del proyecto es migrar el protocolo al C orientado a CCS, se espera obtener la mayor documentación posible acerca de variables, interrupciones, configuración del hardware, temporización, etc.

La tercera etapa se ha nombrado **desarrollo** y se puede definir como la fase en la que se pondrá en práctica todo lo concretado en la etapa anterior para lograr el desarrollo del producto. Para ello, se estiman 11 semanas: 4 para migrar y las 7 restantes para la creación de *templates* y la realización de pruebas y búsqueda de posibles errores y soluciones.

Finalmente, la cuarta y última etapa se refiere al desarrollo de la **documentación** (una vez ya esté finalizado el producto). La mayor parte de ese periodo se dedicará a hacer la memoria, también se espera preparar el material para el curso de formación de la empresa y la defensa. Aunque estas tareas se realizaran en la última parte, durante todo el transcurso del proyecto se irá apuntando las decisiones tomadas, los problemas encontrados y cambios importantes realizados en la migración.

2.5.2 Dedicación

	Dedicación estimada(horas)	Dedicación real(horas)	Dedicación estimada (%)	Dedicación real (%)
T1 Gestión	**60**	**60**	**16,67**	**16,62**
T1.1 Planificación	*25*	*15*	*6,94*	*4,16*
T1.2 Reuniones	*20*	*25*	*5,56*	*6,93*
T1.3 Seguimiento y control	*15*	*20*	*4,17*	*5,54*
T2 Análisis	**55**	**52**	**15,28**	**14,40**
T2.1 Protocolo MiWi	*28*	*37*	*7,78*	*10,25*
T2.1.1 Protocolos	*16*	*19*	*4,44*	*5,26*
T2.1.1.1 MiWi P2P	4	4	*1,11*	*1,11*
T2.1.1.2 MiWi Mesh	6	7	*1,67*	*1,94*
T2.1.1.3 MiWi Pro	6	8	*1,67*	*2,22*
T2.1.2 Capas	*12*	*18*	*3,33*	*4,99*
T2.1.2.1 MiApp	6	8	*1,67*	*2,22*
T2.1.2.2 MiMac	6	10	*1,67*	*2,77*
T2.2 Hardware	*12*	*9*	*3,33*	*2,49*
T2.2.1 PIC24FJ128	6	6	*1,67*	*1,66*
T2.2.2 MRF89xaM8A	*6*	*3*	*1,67*	*0,83*
T2.3 Software	*10*	*6*	*2,78*	*1,66*
T2.3.1 MPLAB	8	5	*2,22*	*1,39*
T2.3.2 ZENA	*2*	*1*	*0,56*	*0,28*
T2.3.3 Xterm	0.5	0.5	*0.14*	*0.14*
T2.4 CCS	*18*	*20*	*5,00*	*5,54*
T3 Desarrollo	**150**	**160**	**41,67**	**44,32**
T3.1 Migración	60	50	16,67	13,85
T3.2 Pruebas y solución de errores	60	50	16,67	13,85
T3.3 Creación de templates	30	40	8,33	11,08
T3.3 Aplicación real	0	20	0	5,54
T4 Documentación	**95**	**89**	**26,39**	**24,65**
T4.1 Memoria	*75*	*75*	*20,83*	*20,78*
T4.3 Material para el curso de formación	*10*	*8*	*2,78*	*2,22*
T4.4 Material para la defensa	*10*	*6*	*2,78*	*1,66*
TOTAL	360	361	100,00	100,00

Tabla 2: Variación de la dedicación entre la estimada y la real

En la *tabla 2* se pueden observar las horas estimadas en la planificación y las horas realmente invertidas durante el proyecto. Al realizar la planificación no se planifico hacer una aplicación real pero viendo que al finalizar la fase de desarrollo se habían invertido 10 horas menos de las previstas se estudió la posibilidad de realizar dicha aplicación y se llevó a cabo.

Como es visible en el diagrama de anillos (véase *ilustración 2*) la mayor parte del tiempo se ha invertido en el desarrollo del producto tal como se había planificado. Seguido por documentación y la gestión. Por último, la fase que menos inversión temporal ha requerido ha sido el análisis.

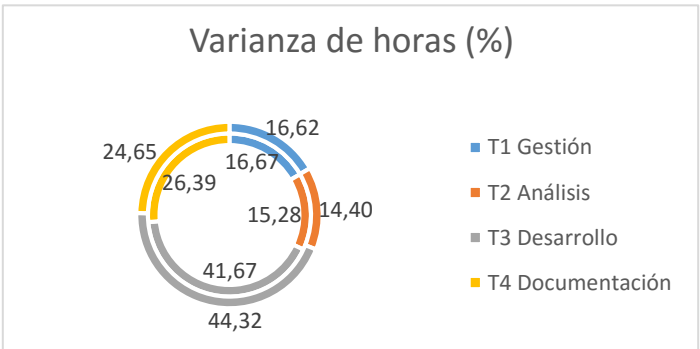
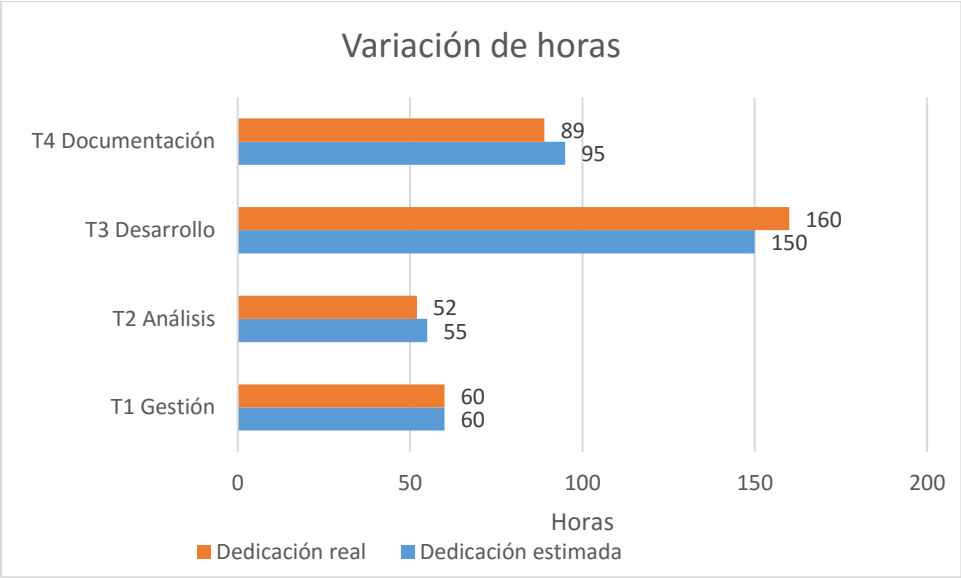


Ilustración 2: Diagrama de anillos de variación de tiempo en porcentajes.

Al acabar el proyecto se ha visto que las horas invertidas a cada fase del proyecto han sido muy parecidas a las estimadas. La mayor desviación entre las horas estimadas e invertidas se ha dado en la fase de desarrollo por la realización de la aplicación real. Las otras fases en cambio se han desarrollado aproximadamente en las horas estimadas.



2.6 Plan de calidad

En este apartado se definirá el plan de calidad del proyecto, es decir, el nivel elemental que deberá tener el producto para poder considerar que está finalizado con éxito. Los criterios a tener en cuenta serán los siguientes (puntualizamos que van acorde con las *características necesarias* mencionadas en el alcance):

- **Debe funcionar con el transmisor MRF89XAM8A.**

  La empresa *Atelei Engineering* utiliza el *transceiver* MRF89XAM8A para emitir o recibir señales analógicas, por tanto, el producto a desarrollar deberá funcionar correctamente con dicho *transceiver*.

- **Deben funcionar los tres protocolos de pila que tiene el protocolo MiWi™.**

  El protocolo MiWi™ como se explicara más adelante en el *punto 3.4* contiene estos tres protocolos de pila: MiWi P2P, MiWi Mesh y MiWi Pro. El producto deberá funcionar con los tres.

- **Los mensajes podrán ir encriptados.**

  Los mensajes de datos que se envían entre los nodos deberán tener la opción de ir encriptados. El protocolo tiene estos siete modos de encriptación y deberá funcionar con todos:

  - AES-CTR
  - AES-CCM-64
  - AES-CCM-32

- AES-CCM-16
- AES-CBC-MAC-64
- AES-CBC-MAC-32
- AES-CBC-MAC-16

- **Los nodos finales podrán dormirse.**

  Este protocolo tiene la característica de que los nodos finales RFD (*Reduced Function Device*) se puedan dormir para gastar el mínimo de energía posible. Nuestro producto también deberá incorporar esta característica.

- **Los nodos podrán cargar la red anterior.**

  Al inicializar el protocolo MiWi™, éste tiene la característica de poder recuperar los datos de una red anterior almacenada en memoria no volátil. Nuestro producto también deberá tener dicha característica.

2.7 Metodología de trabajo

En cuanto al desarrollo del Proyecto, se va a implementar la siguiente metodología de trabajo. Inicialmente se empezará elaborando la **planificación**. Dado que se desconoce el tiempo exacto de duración, las estimaciones temporales se van a basar en la experiencia previa de la empresa *Atelei Engineering.*

Seguidamente se continuará con la fase de **análisis**. Una vez iniciada dicha fase se registrará diariamente las horas invertidas en el proyecto. Si hay variaciones considerables entre las horas estimadas y realmente invertidas se intentara encontrar la razón. En esta fase también se redactará una parte de la bibliografía. Por último, se instalará y se estudiará el funcionamiento del software que se vaya a utilizar.

Después del análisis se empezará con el **desarrollo** del proyecto. Se continuará realizando un registro esta vez más amplio ya que abordará los posibles problemas que vayan surgiendo y las soluciones encontradas.

Una vez se cumpla con el alcance del proyecto, se comenzará con la redacción de la memoria. Para terminar se elaborara el material para el curso de formación en *Atelei Engineering* y el material para la defensa.

2.8 Plan de comunicación

El plan de comunicaciones con los directores del proyecto se diferencia en dos: el plan de comunicación con el director de la empresa y plan de comunicación con el director de la universidad.

2.8.1 Plan de comunicación con el director de la empresa

El proyecto se llevara a cabo desde casa pero la comunicación con la empresa deberá ser fluida hasta la finalización del producto, especialmente durante las etapas de gestión, análisis y desarrollo. Para ello, se concretarán reuniones semanales en la empresa y se definirán medios de comunicación alternativos (como video llamada o e-mail).

2.8.2 Plan de comunicación con el director de la universidad

Una vez terminado el producto se establecerá la comunicación con el director de la universidad para llevar a cabo la documentación. Se estima que con el director de la universidad se efectuaran reuniones semanales con el objetivo de dirigir la memoria. Por supuesto dependiendo de las circunstancias y los problemas que puedan ir surgiendo.

2.9 Plan de riesgo

Para asegurar que el resultado del proyecto sea exitoso, es conveniente hacer un plan de riesgos identificando cada uno de los riesgos que pueda haber que hagan cambiar el resultado del proyecto.

Dichos riesgos se han diferenciado en dos subgrupos; por un lado los **riesgos del producto** que son los riesgos que puedan hacer que el producto no funcione adecuadamente y, por otro lado, los **riesgos del proyecto** que son los riesgos de que el producto no esté finalizado el día de la entrega.

2.9.1 Producto

- **Rotura de algún cable o soldadura**

  Si alguna soldadura o algún cable se rompiera tendríamos los siguientes escenarios:

  - **El cable que conecta el Pickit™ 3 con uno de los microcontroladores.**

    En este caso no podríamos cargar el programa en el microcontrolador y tampoco podríamos alimentarlo de energía. Estos cables son cables comunes y podrían ser facilitados por la empresa sin ningún inconveniente.

    Probabilidad de que pase: Pequeña.

    Influencia: Pequeña.

  - **El cable que conecta el Pickit™ 3 con el ordenador.**

    Estaríamos en la misma circunstancia que en el punto anterior. El cable este es USB-Mini USB, es muy común y por lo que resultaría sencillo conseguirlo de nuevo.

    Probabilidad de que pase: Pequeña.

    Influencia: Pequeña.

- **Los cables o soldaduras del UART**

  Si estos cables o las soldaduras se rompiesen, no podría mandar ningún mensaje al terminal para debugearlo. Los cables son cables corrientes y la soldadura puede ser realizada en la empresa, por tanto, no supondría una gran pérdida de tiempo.

  Probabilidad de que pase: Grande.

  Influencia: Pequeña.

- **Rotura de Pickit™ 3.**

  En este caso no podríamos cargar el programa en el microcontrolador y tampoco podríamos alimentarlo de energía. La empresa dispone de varios módulos y también tiene un ICD3, que también nos serviría con lo cual no supondría un grave problema.

  Probabilidad de que pase: Pequeña.

  Influencia: Pequeña.

- **Rotura de Zena.**

  El Zena es el sniffer que se utilizará para captar los mensajes que se envían entre los nodos. En caso de que se rompiese no podría captar mensajes y por consiguiente no se podrían llevar a cabo las pruebas. No obstante la empresa cuenta con varios Zena por lo que sería un problema de fácil solución.

  Probabilidad de que pase: Pequeña.

  Influencia: Pequeña.

- **Rotura del microcontrolador PIC24FJ128GA310**

  Si los cables de alimentación se introdujeran del revés, se rompería el microcontrolador y habría que remplazarlo. Como en los casos anteriores la empresa dispone del material extra.

  Probabilidad de que pase: Pequeña.

  Influencia: Pequeña.

- **Rotura del microcontrolador PIC24FJ32GA102**

  Al igual que con el microcontrolador PIC24FJGA310 si los cables de alimentación se introdujeran del revés, se rompería el microcontrolador y habría que remplazarlo. Como en los casos anteriores la empresa dispone del material extra.

  Probabilidad de que pase: Pequeña.

  Influencia: Pequeña.

- **Rotura del alimentador de energia**

  Dos de los nodos se alimentan mediante un alimentador de energía. Si se rompiese, ésta no podría alimentar dicho nodo. La empresa también cuenta con material para sustituirlo.

  Probabilidad de que pase: Pequeña.

Influencia: Pequeña.

2.9.2 Proyecto

- **Perdida del código o parte del código desarrollado**

  Perder el código del programa migrado, sería un grave error ya que atrasaría mucho el proyecto. Para que esto no ocurra siempre tendremos una copia de seguridad en la carpeta de Dropbox compartida con la empresa.

  Probabilidad de que pase: Pequeña.

  Influencia: Grande.

- **Perdida de la documentación desarrollada**

  Perder la documentación de la memoria sería un grave error ya que esto también atrasaría mucho el proyecto. Para que esto no ocurra siempre tendremos una copia de seguridad en la carpeta de Dropbox personal.

  Probabilidad de que pase: Pequeña.

  Influencia: Grande.

- **Incompatibilidad a la hora de hacer reuniones**

  Durante el desarrollo del proyecto pueden surgir incompatibilidades horarias tanto con el director universitario como con el director de la empresa. Por este motivo las reuniones se acordaran previamente con suficiente antelación.

  Probabilidad de que pase: Pequeña.

  Influencia: Medio.

3

PROTOCOLO MIWI™

3.1 Introducción MiWi™

MiWi™ es un protocolo de red, propiedad del fabricante Microchip. El código fuente se suministra con la condición de que se utilice con los módulos de Microchip. El protocolo está basado en el estándar IEEE 802.15.4 enfocado a redes de área personal que tengan baja tasa de transferencia de datos y bajo coste.

Se estructura en 3 capas principales:

- Capa de aplicación
- Protocolos de pila.
- Drivers de los transmisores o capa MAC.

Para realizar la unión entre la capa de aplicación y el protocolo de pila, MiWi™ define la capa de interfaz MiApp; y para la unión entre el protocolo de pila y los transceptores de Radiofrecuencia de microchip, MiWi™ define la capa de interfaz MiMac.

Cada capa tiene su archivo de configuración correspondiente. Estos archivos han sido desarrollados para facilitar el cambio de protocolo de pila o de transceptor.

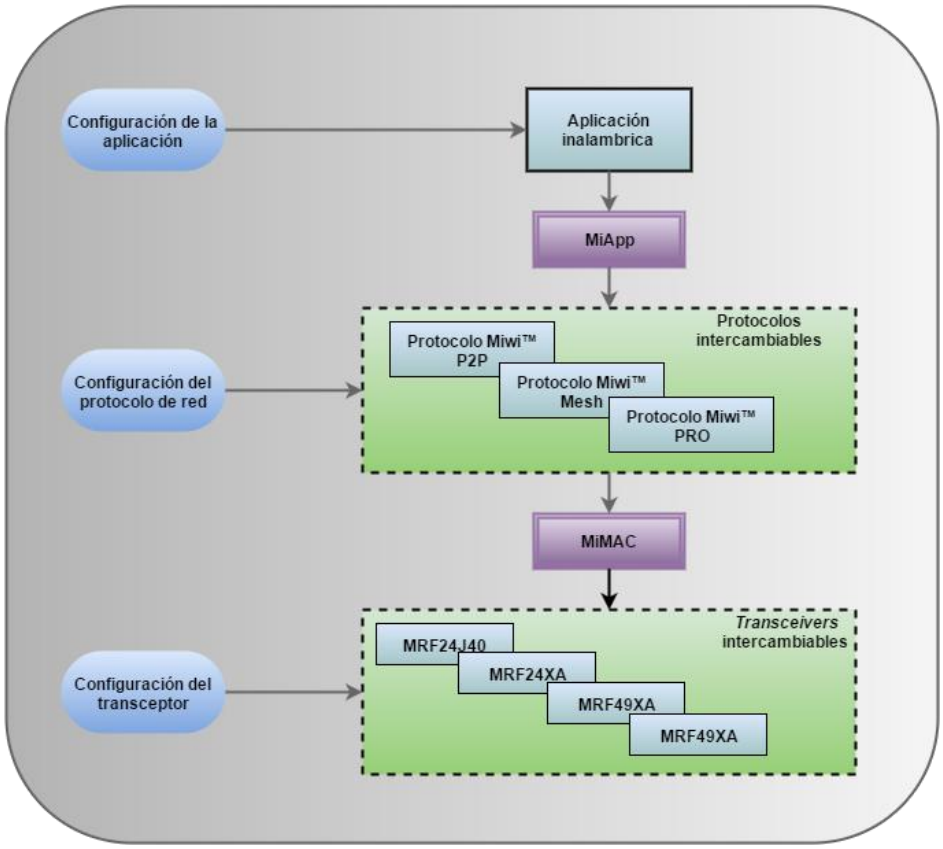


Ilustración 3: Arquitectura del protocolo Miwi

3.2 Tipos de dispositivos

Según la funcionalidad que tengan los dispositivos en la red, el estándar IEEE 802.15.4 define dos tipos de dispositivos:

- **Full Function Device (FFD):**

  Estos dispositivos suelen tener fuente de energía continua para que el *transceiver* pueda estar encendido de forma continua. Por ello, tienen la funcionalidad completa y pueden desempeñar diferentes roles como: coordinador PAN (*Personal Area Network*) iniciando y controlando la red, como enrutador o también como dispositivo final.

- **Reduced function Device(RFD)**

  Al contrario que FFD estos dispositivos disponen de las mínimas funcionalidades con el objetivo de conseguir bajo consumo. Estos dispositivos solo pueden tener el rol de nodo final y solo pueden asociarse a dispositivos FFD. Los recursos (potencia y memoria) de estos dispositivos son mínimos y generalmente tienen el *transceiver* apagado cuando están inactivos. Cuando el *transceiver* está apagado se dice que el dispositivo está dormido. Cuando esto pasa, el coordinador al que está conectado tiene que almacenar todos los mensajes que van dirigidos a éste dispositivo para que al despertar se los envíe.

Según el rol que desempeñan dentro de la red, el protocolo MiWi™ define tres tipos de dispositivos:

- **Coordinador PAN:**

  Debe ser un dispositivo FFD. Hay uno solo por cada red. Es el encargado de crear la red y de guardar las direcciones de los demás dispositivos.

- **Coordinador:**

  Estos dispositivos también son FFD. Son opcionales y sirven para aumentar el rango de la red permitiendo así que más nodos puedan unirse. Puede realizar tareas de control y/o monitorización.

- **Nodo final:**

  Pueden ser FFD o RFD y suelen desempeñar tareas de control y/o monitorización.

La pila de protocolo es la responsable de asignar a cada dispositivo su rol.

3.3 Topología de red

En este protocolo existen cuatro tipos de topologías que se basan en el estándar IEEE 802.15.4:

3.3.1 Topología en estrella:

Esta topología está formada por un único coordinador que es, además el coordinador PAN y mínimo un nodo final. Todos los dispositivos están conectados únicamente con el coordinador PAN y por este motivo si un nodo final quiere comunicarse con otro la comunicación se deberá de realizar mediante el coordinador PAN.

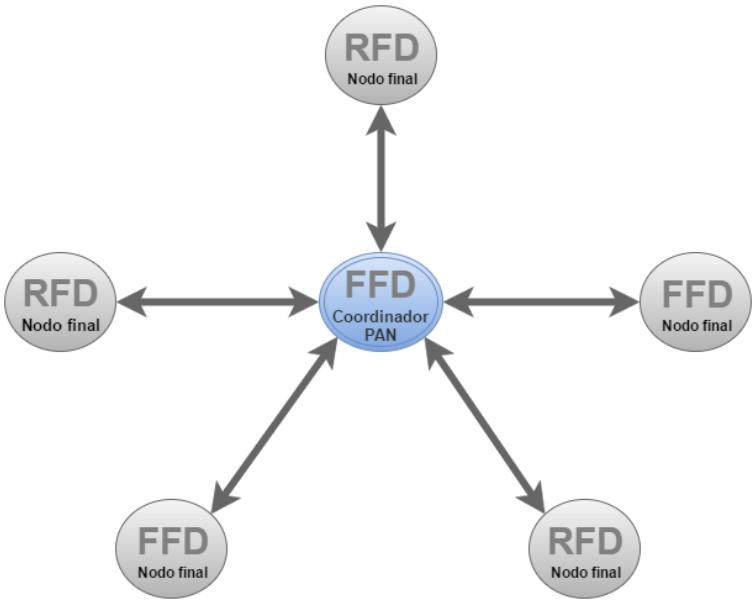


Ilustración 4: Topología en estrella

3.3.2 Topología en árbol:

Hay un coordinador PAN que será la raíz, a este coordinador se unirán otros coordinadores que formaran las ramas. Y finalmente, habrá nodos finales conectados a los coordinadores y estos serán las hojas.

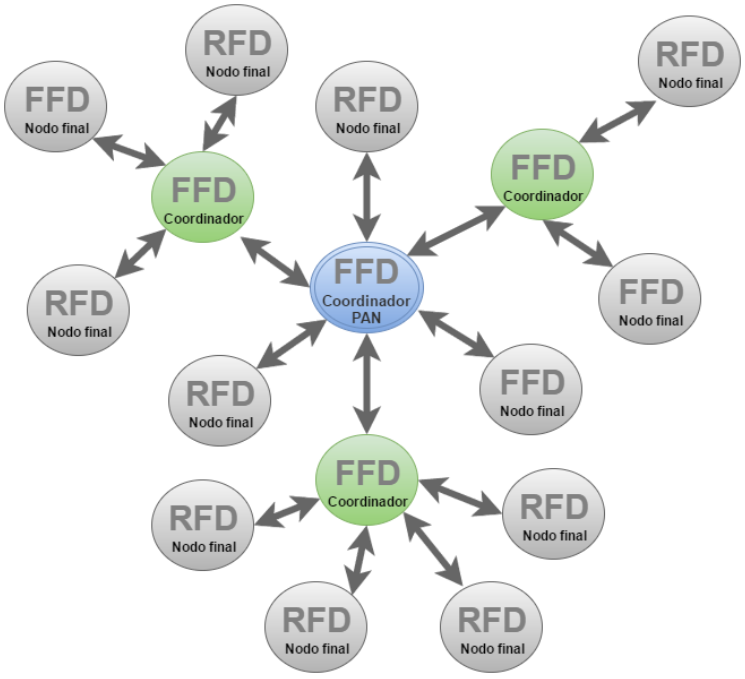


Ilustración 5: Topología en árbol

3.3.3 Topología en malla:

Es similar a la topología en árbol. Se diferencian únicamente en que en esta topología los coordinadores pueden comunicarse entre ellos sin necesidad de seguir la estructura de árbol. Los nodos finales en cambio se tienen que comunicarse con algún coordinador.

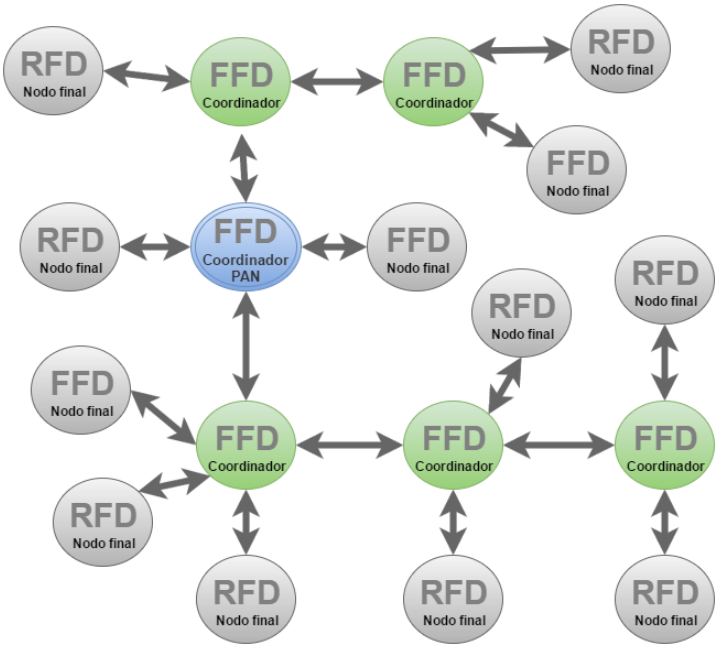


Ilustración 6: Topología en malla

3.3.4 Topología P2P:

Es la topología más sencilla. Solamente se conecta un dispositivo con otro. No se distinguen maestros y esclavos. En este tipo de red cada dispositivo es capaz de formar múltiples enlaces directos a otros dispositivos (no necesariamente el Coordinador PAN). El coordinador del PAN será un dispositivo FFD y los dispositivos terminales podrán ser tanto FFD como RFDs. No obstante, en esta topología los dispositivos finales solo pueden conectarse a FFD.

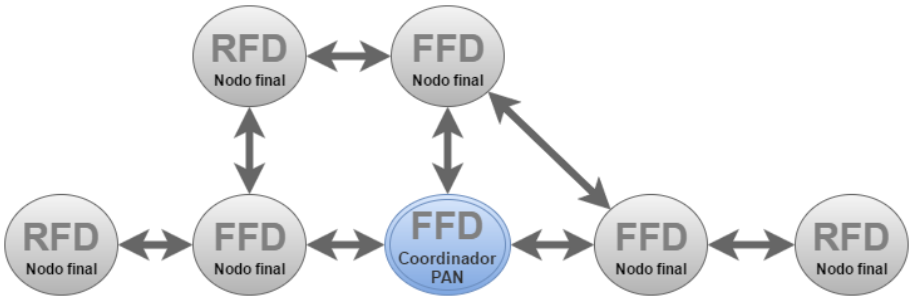


Ilustración 7: Topología P2P

3.4 Protocolos inalámbricos

El protocolo MiWi™ soporta tres diferentes protocolos: MiWi P2P, MiWi Mesh y MiWi Pro. Los tres protocolos pueden enviar un mensaje desde el origen al destino pero la diferencia recae en cuantos saltos puede dar un mensaje antes de llegar al destino.

3.4.1 Protocolo MiWi P2P

Este es el protocolo más simple soportado. Solo es capaz de crear la red de topología estrella o P2P. No tiene capacidad de enrutamiento por tanto, el mensaje solo se puede mandar a los dispositivos que estén al alcance de radio. La ventaja que tiene este protocolo es su simplicidad, pero si se necesita una distancia de comunicación mayor el mensaje se tendrá que enrutar y en ese caso se tendrá que descartar este protocolo.

3.4.2 Protocolo MiWi Mesh

El protocolo MiWi Mesh está diseñado para aumentar la cobertura de la red proporcionando la opción de enrutamiento. Soporta 8 coordinadores incluyendo el coordinador PAN. Además soporta la realización de 4 saltos desde un nodo final a otro, o dos desde el coordinador PAN a cualquier nodo final. Para simplificar el enrutamiento el coordinador en esta red solo se puede asociar al coordinador PAN y nunca a otro coordinador. Cuando un coordinador a la hora de asociarse no tiene en su radio al coordinador PAN o ya existen 8 coordinadores, automáticamente se convierte en un nodo final. Aunque este protocolo puede acoger más dispositivos y más territorio es obvia su limitación ya que solo puede dar 4 saltos. Mediante este protocolo se puede crear la red de topología árbol y estrella.

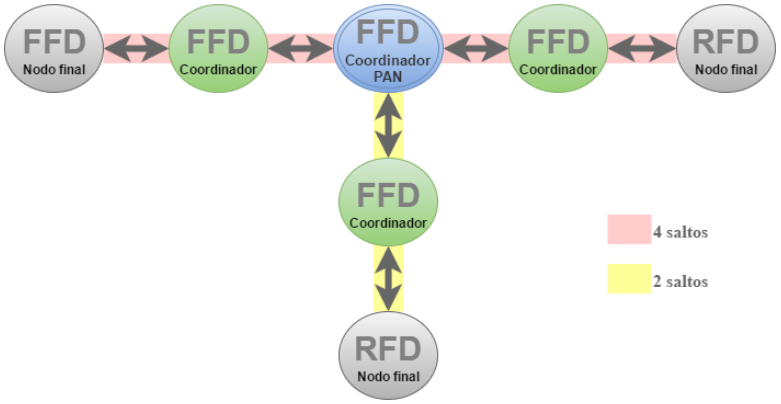


Ilustración 8: Protocolo MiWi Mesh

3.4.3 Protocolo MiWi Pro

Este protocolo tiene el enrutamiento mejorado y puede soportar hasta 64 coordinadores. Además un coordinador se puede asociar a otro coordinador sin que este tenga que ser coordinador PAN. Cuando la red se forme de forma lineal, un mensaje puede llegar a realizar 65 saltos desde un nodo final a otro, o 64 desde el coordinador PAN al dispositivo final. Mediante este protocolo se puede crear la red de topología malla, árbol y estrella.

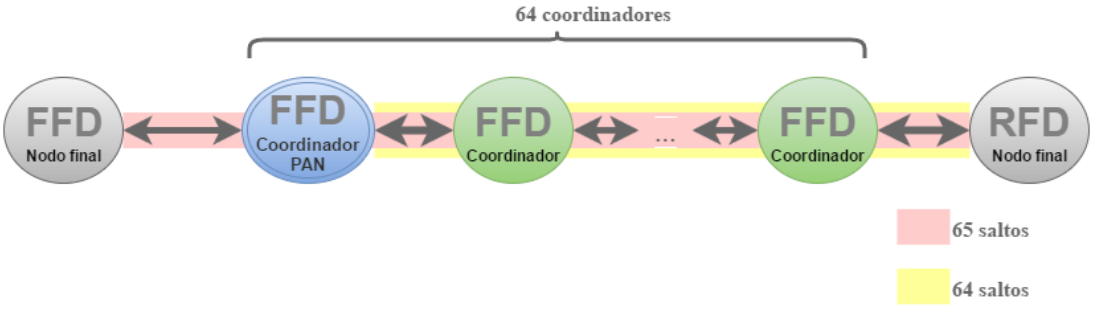


Ilustración 9: Protocolo MiWi Pro.

2.4.4 Resumen de protocolos

Protocolo	n° de saltos	Topologías posibles
MiWi P2P	1	P2P
MiWi Mesh	4	Árbol y estrella
MiWi Pro	65	Malla, árbol y estrella

Tabla 3: Resumen de las topologías

3.5 Control de acceso al medio

Cualquier red que esté implementada bajo el estándar IEEE 802.15.4 es una red de acceso múltiple, por ello, todos los nodos de la red pueden acceder por igual de condiciones al medio de comunicación.

A la hora de acceder a la red se puede hacer con baliza o sin ella, pero el protocolo de pila solo soporta el acceso al medio de transmisión sin baliza. Esto quiere decir que cualquier nodo tiene la posibilidad de transmitir cuando el canal este libre y no tiene que esperar a la baliza. Como cualquier dispositivo tiene la posibilidad de transmitir cuando el canal está libre puede haber colisiones y por este motivo el protocolo MiWi™ utiliza la detección CSMA-CA antes de transmitir un mensaje.

3.6 Formación de la red

A la hora de formar la red el proceso de asociación o *handshaking* es diferente en el protocolo MiWi P2P y MiWi Mesh y MiWi Pro.

3.6.1 Formación de la red con el protocolo de pila MiWi P2P

El proceso de asociación en el protocolo MiWi P2P es muy sencillo. El proceso es el siguiente:

1. El dispositivo que busca conexión manda un comando de petición de conexión P2P (*P2P Connection Request*). Este mensaje será un mensaje broadcast.
2. Cualquier dispositivo al alcance de la radio contesta con un comando de respuesta de conexión P2P que finaliza la conexión (*P2P Connection Response*). Este mensaje será un mensaje unicast.

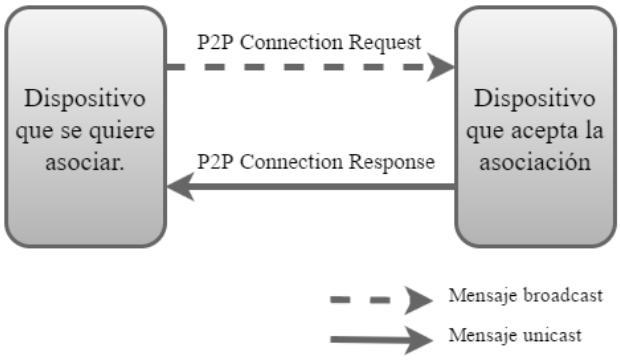


Ilustración 10: Proceso de asociación en el protocolo de pila MiWi P2P

3.6.2 Formación de la red con el protocolo de pila MiWi Mesh y MiWi Pro

El proceso de asociación en los protocolos MiWi Mesh y MiWi Pro es más complicado. Se parece al *handshaking* típico de IEEE 802.15.4. Este proceso es el siguiente:

1. El dispositivo que busca conexión envía una petición de baliza (*beacon request*). Este mensaje será un mensaje broadcast.
2. Todos los dispositivos que tengan la capacidad de conectarse con otros y que hayan recibido la petición le enviarán una baliza (*beacon*). Este mensaje también será un mensaje broadcast.
3. El dispositivo inicial recoge todas las balizas que ha recibido durante cierto periodo de tiempo y después decide qué baliza usar para establecer la conexión y manda un comando de petición de asociación (*Association Request*). Este mensaje será unicast.
4. El dispositivo del otro lado de la conexión manda una respuesta de asociación (*Association Response*).

En estos protocolos de pila un dispositivo solo se puede unirse a un solo coordinador. Escoger el coordinador requiere: primero, enumerar los posibles coordinadores; y segundo, escoger el más adecuado como su coordinador.

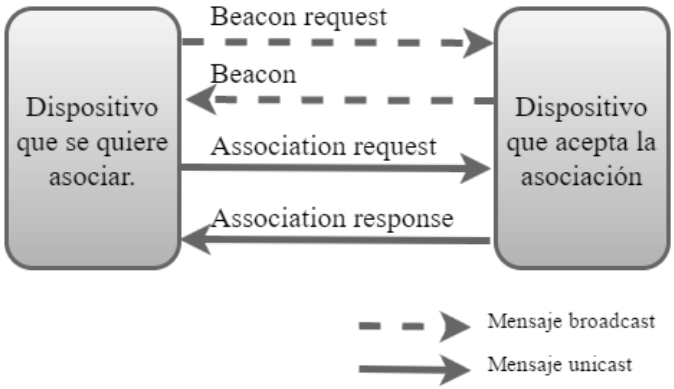


Ilustración 11: Proceso de asociación en los protocolos de pila MiWi Mesh y MiWi Pro

3.7 Direccionamiento

Para organizar las direcciones de los nodos, el protocolo MiWi™ utiliza el sistema especificado en el estándar IEEE 802.15.4. La especificación define tres tipos diferentes de direcciones:

- **Extended Organizationally Unique Identifier (EUI):**

  Es un número de 8 bytes único en el mundo. Cada dispositivo que cumpla las especificaciones de IEEE 802.15.4 debe tener la dirección única EUI. Los 3 primeros bytes hay que comprar a IEEE y los 5 últimos están disponibles para el usuario. Para los propietarios de los *transceiver* RF subGHz las direcciones EUI pueden tener entre 2 y 8 bytes.

- **PAN Identifier (PANID):**

  Es una dirección de 16 bits que define a un grupo de nodos. Todos los nodos dentro del grupo, es decir dentro de una misma PAN comparten el PANID. El PANID de un dispositivo se define de forma manual en el archivo de configuración *config_MiApp.h.*

- **Short Address(SA):**

  A esta dirección también se le llama como la dirección del dispositivo. Es una dirección de 16 bits que le asigna el padre del nodo. Esta dirección es única en la red (dentro de la misma PAN) y se emplea para direccionar y comunicarse. IEEE 802.15.4 especifica que el coordinador siempre tiene la SA 0000h. Un dispositivo adquiere un SA cuando decide unirse a la red.

A la hora de enviar un mensaje, como dirección de destino se puede poner tanto la SA como la EUI. Como la SA se asigna en tiempo de ejecución normalmente se utilizaran las direcciones EUI para el envío de mensajes.

3.7.1 SA en el protocolo MiWi Mesh y MiWi P2P

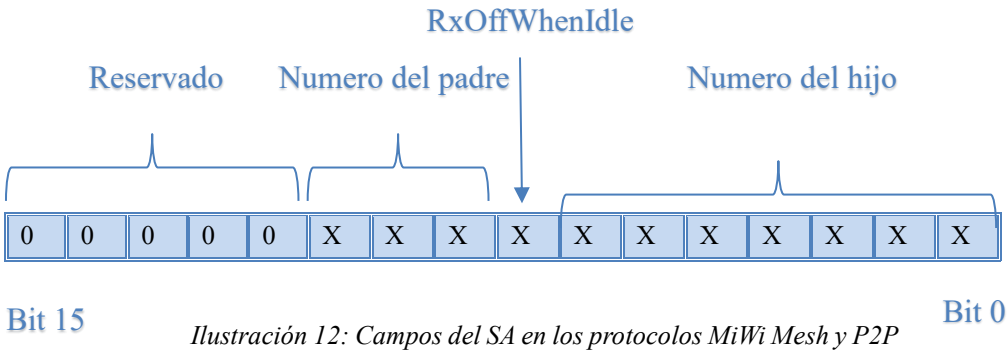


Ilustración 12: Campos del SA en los protocolos MiWi Mesh y P2P

Los bits 15-10 están reservadas y por lo tanto siempre serán 0.

El número del padre (bits 10-8) es único para cada coordinador incluyendo el coordinador PAN. Éste número es el que limita el número de coordinadores en una red. Como son solo 3 bits puede haber 8 coordinadores como mucho.

El RxOffWhenIdle (bit 7) es la propiedad inversa al definido en el IEEE 802.15.4 RxOnWhenIdle. Cuando este bit está activado indica que el dispositivo va a apagar su *transceiver* y, por tanto, no podrá recibir ningún mensaje; a esto se le llama dormir. Cuando esto ocurre, los mensajes se mandaran al padre del nodo que esté durmiendo y este las almacenara en el buffer. Cuando se despierta el nodo (cuando enciende su *transceiver*) le enviara los mensajes almacenados.

El número de hijo de cualquier coordinador es 00h. Esto indica que están operando como coordinador. Otros valores para este campo están definidos por el tipo de nodo y su función en el PAN. La *ilustración 13* explica el cómo se definen los SAs.

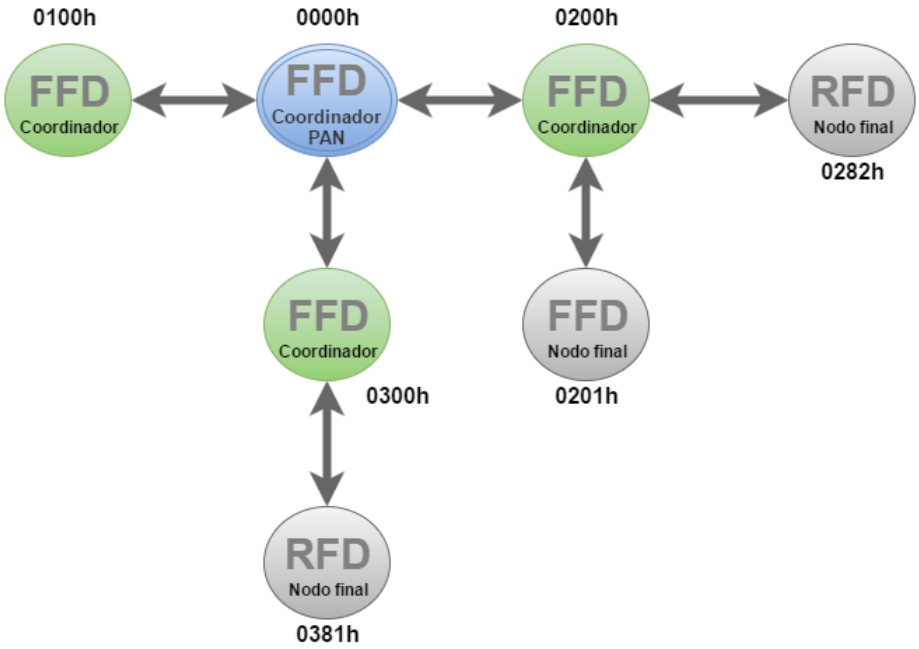


Ilustración 13: Direccionamiento

3.7.2 En el protocolo MiWi Pro

En el protocolo MiWi Pro el direccionamiento es muy parecido al de los protocolos MiWi Mesh y MiWi P2P (*véase punto 3.7.1*), el único campo que cambia es el número de padre. Este protocolo puede definir hasta 64 coordinadores por lo tanto son necesarios 6 bits para poder numerarlos.

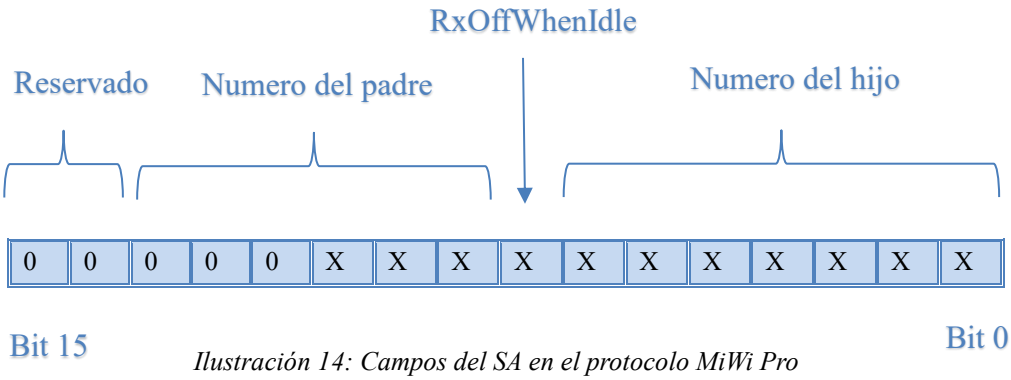


Ilustración 14: Campos del SA en el protocolo MiWi Pro

3.8 Comunicación entre nodos

Una vez creada la red, la próxima gran tarea consiste en cómo enviar un mensaje entre los nodos. Para ello, cualquier nodo o dispositivo perteneciente a la red, emplea la SA. La SA le indica la posición del nodo destino y como hacerle llegar el paquete.

3.8.1 Formato de los paquetes

Para un transceptor RF compatible con IEEE 802.15.4, el protocolo MiWi™ utiliza la SA de 16 bits para transmitir y recibir mensajes (siempre y cuando sea posible). Los paquetes son construidos en base a la Sección 7.2 de la especificación IEEE 802.15.4. Sin embargo, los transceptores RF subGHz usan direcciones EUI en la capa MAC. [2]

Por encima de esta capa reside la cabecera del protocolo que contiene la información necesaria para el enrutamiento de paquetes.

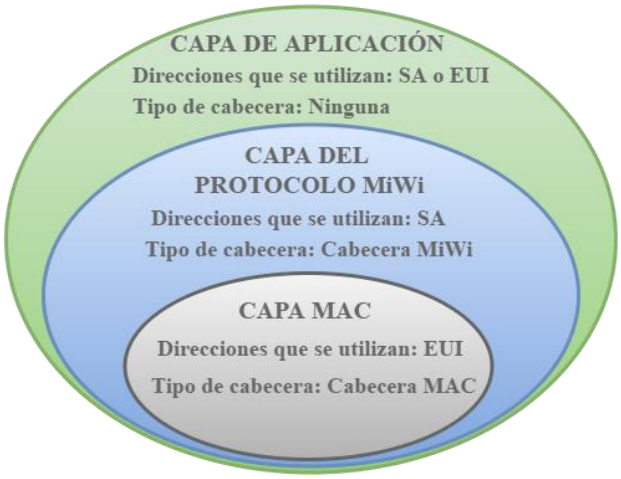


Ilustración 15: Direcciones que se utilizan en cada capa y tipo de cabecera

El formato de la cabecera del protocolo (cabecera MiWi) está integrada de los siguientes componentes (*véase ilustración 16*):

- **Hops:** El número de saltos que el paquete puede ser transmitido. Si es 00h es que no puede ser retransmitido (1 byte).
- **Frame Control:** Define el comportamiento del paquete como por ejemplo si el paquete de datos esta encriptado o se requiere autorización de capas superiores para recibirlo (1 byte).
- **Dest PANID:** El PANID de la red en la que se encuentra el nodo destino (2 bytes).
- **Dest SA:** SA del nodo destino (2 bytes).
- **Source PANID:** El PANID de la red en la que se encuentra el nodo que transmite el mensaje (2 bytes).
- **Source SA:** SA del nodo que transmite (2 bytes).
- **Sequence Number:** Una secuencia numérica que puede ser empleada para determinar el estado del paquete en su viaje a través de la red (1 byte).
- **Report type:** Es el tipo de informe. MiWi™ utiliza informes para él envió de información del protocolo como puede ser ACK de la capa de aplicación. El protocolo permite la implementación de hasta 256 tipos de informes. El tipo 00h está reservado al protocolo, los demás están libres para el uso del usuario.
- **Report id:** Es el identificador del informe. Cada tipo puede tener 256 informes diferentes, este parámetro es el que identifica el informe dentro del tipo.

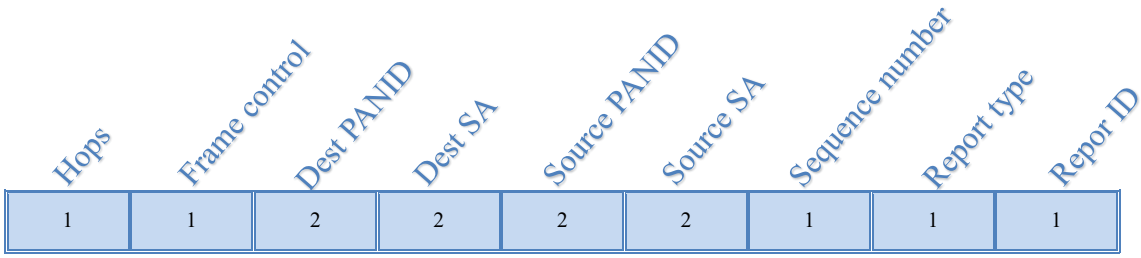


Ilustración 16: Cabecera MiWi

Leyenda: Los números son la longitud de cada componente en bytes.

El frame control está integrado por los siguientes componentes:

- **Bit7-bit3**: Reservado.
- **Bit2**: Si este bit está activado, el nodo que transmite le pide el ACK del nivel superior al nodo final.
- **Bit1**: Bit intra clúster, está reservado y hay que mantener a '1'.
- **Bit0**: Si este bit está activado, el paquete de datos esta encriptado a nivel de aplicación.

0	0	0	0	0	X	1	0
					ACKREQ	INTRCLST	ENCRYPT
Bit 7							Bit 0

Ilustración 17: Campos del frame control

3.8.2 Enrutamiento en el protocolo MiWi Mesh

El enrutamiento en una red sin cables suele ser una tarea muy compleja. El protocolo MiWi™ resuelve este problema utilizando la asignación de direcciones para indicar el padre del dispositivo con el que se quiere comunicar y usando servicios IEEE 802.15.4 para ayudar a intercambiar y retransmitir información en la red.

Una tarea de los algoritmos de enrutamiento es determinar el siguiente paso para cualquier paquete saliente. El protocolo MiWi™ utiliza el mecanismo de unión IEEE 802.15.4 para descubrir dichos caminos.

Cuando un nodo quiere unirse a una red, primero tiene que mandar un *beacon request*. Todos los coordinadores que reciben este mensaje *beacon request* responden con un paquete beacon informando los vecinos que tiene en la red.

En el protocolo MiWi Mesh le añaden tres bytes de información adicional al *beacon* (la respuesta de los coordinadores):

- **Protocol ID (1 byte):**

  Este contenido ayuda a distinguir las redes de protocolo MiWi™ y otros de IEEE 802.15.4 que están operando en la misma radio. El *protocol ID* siempre tendrá que ser 4Dh.

- **Version number (1 byte):**

  El número de la versión de la especificación.

- **Local coordinators (1 byte):**

  Este campo es un mapa de bits que indica qué coordinadores son actualmente visibles por el coordinador que está enviando el *beacon*. Cada posición de bit representa directamente a uno de los coordinadores de 8 posibles. El bit 0 es 0000h (el coordinador PAN). Bit 1 indica que este coordinador puede hablar directamente a 0100h, y así sucesivamente. A través de este campo todos los coordinadores de la red aprenderán sobre diferentes rutas posibles a todos los nodos.

```
Protocol ID: 0x4d
version: 65
Known Coordinators: 0b00000001
```

Ilustración 18: Información adicional del beacon

En la *ilustración 18* se puede observar un ejemplo de coordinadores que conoce el nodo que manda el mensaje *beacon*. En este caso el mensaje *beacon* manda el dispositivo coordinador PAN y él es el único dispositivo de la red por tanto solo conoce a sí mismo.

Una vez sepamos los coordinadores vecinos y a su vez que pueden ver dichos vecinos el enrutamiento se hace de la manera que se ve en la *ilustración 19*.

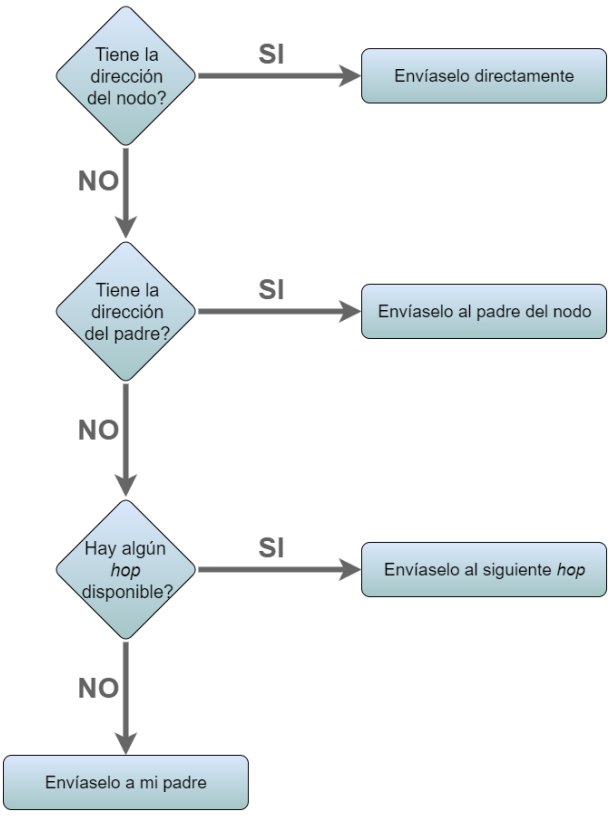


Ilustración 19: Enrutamiento en el protocolo MiWi Mesh

3.8.3 Enrutamiento en el protocolo MiWi Pro

En el protocolo MiWi Pro, al igual que en el protocolo MiWi Mesh el mensaje siempre puede ser distribuido por el padre del destino. Aplicando este hecho el mecanismo se centra en enviar

el mensaje hasta el mismo nodo en caso de que el destinatario sea un coordinador. En cambio, si el destinatario es un nodo final se enrutará hasta el coordinador que sea su padre.

Existen dos posibilidades de enrutar hasta el destino en el protocolo MiWi Pro:

- **Enrutamiento árbol:**

Enrutamiento árbol es el mecanismo donde el mensaje se envía a través de la relación de padres e hijos. Para utilizar este mecanismo es necesario que todos los nodos conozcan la topología de la red. El protocolo MiWi Pro hace el seguimiento de la topología permitiendo solo al coordinador PAN asignar SAs a los nodos que quieran asociarse. De esta manera el coordinador PAN conocerá toda la topología de la red. El coordinador PAN después distribuirá dicha información a todos los coordinadores que tengan la capacidad de enrutamiento.

Para guardar los recursos del sistema y enviar la topología de la red fácilmente, el protocolo MiWi Pro utiliza tablas de árbol de familia (véase *tabla 4*).

El byte mayor de la dirección del coordinador	El byte mayor de la dirección del padre del coordinador	Descripción
0	0	Coordinador PAN 0x0000 no tiene padre, por lo tanto, se apunta a sí mismo como padre
1	0	El padre del coordinado 0x0100 es 0x0000
2	1	El padre del coordinado 0x0200 es 0x0100
3	1	El padre del coordinado 0x0300 es 0x0100
4	3	El padre del coordinado 0x0400 es 0x0300
…	..	
n-2	1	El padre del coordinador 0x(n-2)00 es 0x0100
n-1	0xFF	El coordinador 0x(n-1)00 aún no se ha unido a la red por lo tanto su padre apunta al coordinador inexistente 0xFF

Tabla 4: Tabla del árbol de familia

Cada vez que un coordinador intenta unirse a la red el padre del coordinador tiene la responsabilidad de avisar al coordinador PAN. Éste le asignará un SA al coordinador que se quiera unir y tras esto actualizará la tabla del árbol de familia y la distribuirá entre los coordinadores que tengan la capacidad de enrutar.

- **Enrutamiento Malla:**

Para poder usar el enrutamiento Malla todos los nodos que tengan la capacidad de enrutar tienen que conocer tanto sus vecinos como los vecinos de sus vecinos. Para lograr esto cada coordinador tiene que intercambiar la tabla de vecinos al unirse a la red. La tabla de vecinos de cada coordinador utiliza un bit para indicar si puede contactar directamente con el otro coordinador. Por ejemplo, si el bit 9 de la tabla de vecinos esta activada quiere decir que el coordinador 0x0900 estará accesible directamente.

El mecanismo de enrutamiento de MiWi Pro es una combinación de los dos métodos de enrutamiento. Básicamente consiste en que el coordinador puede mirar dos pasos para ver si el enrutamiento malla es posible al nodo destino o al árbol de familia del nodo destino. Si el enrutamiento malla no fuera posible mandará el mensaje por el árbol de la familia para ver si alguno de la familia tiene enrutamiento malla hacia el destino o a la familia del destino (*véase la ilustración 20*).

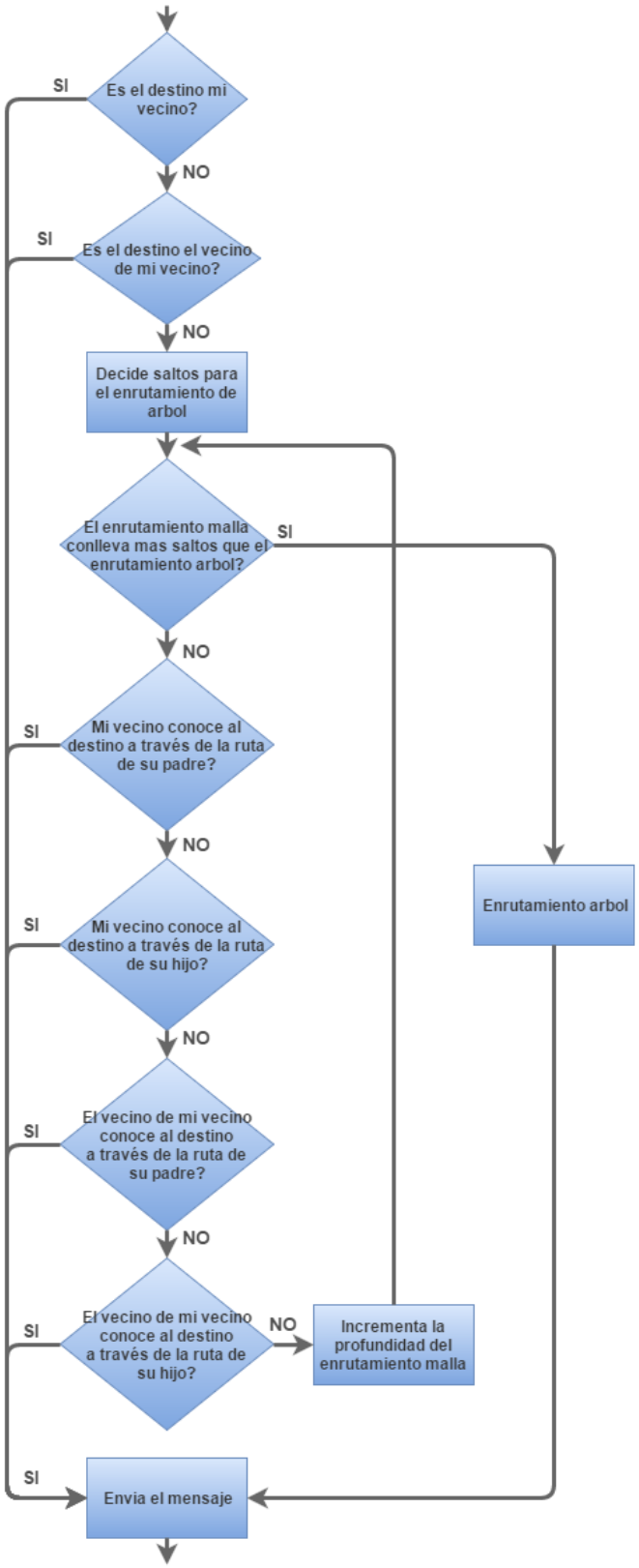


Ilustración 20: Mecanismo de enrutamiento del protocolo MiWi Pro

3.9 Mensajes broadcast

Cuando un coordinador de la red recibe un mensaje broadcast tiene que retransmitir como broadcast otra vez siempre y cuando el contador de saltos (*hops*) no sea igual a cero. Después de retransmitir, tienen que procesar dicho mensaje.

Los mensajes broadcast tienen que tener el bit de ACK request desactivada tanto en la cabecera MiWi como en la cabecera MAC.

3.10 Mensajes multicast

Multicast es un método de comunicación que une un grupo de dispositivos como destino. En el protocolo MiWi P2P y MiWi Mesh no se pueden mandar mensajes multicast.

En el protocolo MiWi Pro en cambio, hay dos posibles grupos al que se pueden enviar un mensaje multicast. Uno de los grupos forman todos los dispositivos FFD (*SA=0xFFFE*) y el otro todos los coordinadores (*SA=0xFFFD*).

3.11 Seguridad del protocolo MiWi™

El protocolo MiWi™ sigue la definición de seguridad MAC especificada en el IEEE 802.15.4. Los siete modos de seguridad definidas en la especificación están incluidos en el protocolo.

Existen tres grupos:

- **AES-CTR:** Este modo codifica el contenido del mensaje de un paquete de tal manera que el atacante no entenderá su contenido sin una clave. Este modo de seguridad no puede verificar la trama del contenido de la cabecera ni la dirección del emisor del paquete.
- **AES-CBC-MAC:** Este modo asegura la integridad del paquete. El código de integridad del mensaje (Message Integrity Code, MIC) que se manda adjuntado al mensaje verifica que ni el contenido ni la cabecera del paquete han sido manipulados durante la transmisión. Cada modo tiene su tamaño de MIC y cuanto más grande sea esté más protección le proporciona al paquete. Estos modos no dejan la opción de cifrar el contenido del mensaje.
- **AES-CCM:** Este modo codifica y asegura la integridad del paquete.

Modo de seguridad		Servicio de seguridad			MIC	PAYLOAD
Identificador	Nombre	Control de Acceso	Encriptación de datos	Integridad del paquete		
01h	AES-CTR	✓	✓		0	24(+11 P2P)
02h	AES-CCM-64	✓	✓	✓	8	16(+11 P2P)
03h	AES-CCM-32	✓	✓	✓	4	20(+11 P2P)
04h	AES-CCM-16	✓	✓	✓	2	22(+11 P2P)
05h	AES-CBC-MAC-64	✓		✓	8	16(+11 P2P)
06h	AES-CBC-MAC-32	✓		✓	4	20(+11 P2P)
07h	AES-CBC-MAC-16	✓		✓	2	22(+11 P2P)

Tabla 5: Modos de seguridad

Al habilitar la seguridad, el bit del campo *frame control* se pone a uno y la pila añade tres nuevos elementos en la cabecera del paquete justo antes del contenido del paquete:

- El campo Frame Counter (4 bytes)
- El campo Long Source Address (8 bytes)
- El campo Key Sequence Number (1 byte)

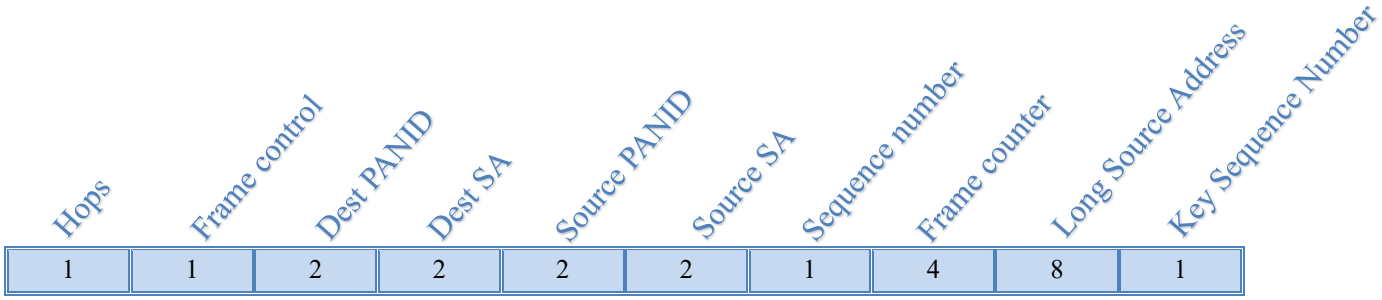


Ilustración 21: Cabecera Miwi con seguridad

Leyenda: Los números son la longitud de cada componente en bytes.

Si se habilita la seguridad se necesitan entre 13 y 21 bytes adicionales para la cabecera auxiliar y el MIC. La seguridad del protocolo MiWi™ además comprueba el contador de trama (*frame counter*) para evitar ataques de repetición.

La seguridad del protocolo MiWi™ asume que toda la información de seguridad, incluyendo la clave de seguridad de 16 bytes, el número de secuencia de clave y el modo de seguridad, son preconfigurados en la pila. La información de seguridad se almacena en la memoria del programa y no se puede modificar durante la ejecución.

4

TECNOLOGIA UTILIZADA

En este apartado se mostrara en detalle el hardware y el software utilizado a la hora de desarrollar el proyecto.

4.1 Hardware

A la hora de desarrollar el proyecto se ha utilizado diferente hardware.

Se han empleado tres placas idénticas: dos de ellas se alimentan por la corriente mediante una fuente de energía y la otra se alimenta desde el ordenador mediante el PICKit3.

Las tres placas llevan el microcontrolador PIC24FJ128GA310.

Las tres placas necesitan estar conectadas con un *transceiver* para poder emitir y recibir mensajes. El *transceiver* que se ha utilizado ha sido el MRF89XAM8A.

Una de ellas se ha conectado el USB to Serial Bridge Controller para poder imprimir mensajes en la pantalla del ordenador; utilizando para ello el UART.

Para programar y depurar el citado microcontrolador se han utilizado dos debuggers. Inicialmente se usó el debugger ICD3. Pasado un mes, y dado que la empresa requería el único debugger ICD3 que disponen, se sustituyó por el PICKit3.

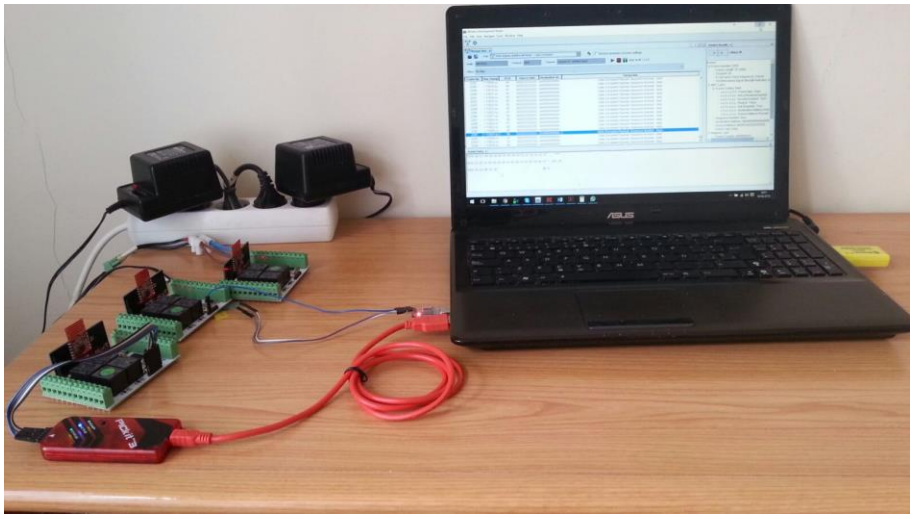


Ilustración 22: Mesa de trabajo

Las conexiones con el ICD3 son las siguientes. Para conectar el ICD3 con el ordenador necesitamos un cable que tenga en un lado USB tipo A macho y en el otro USB tipo B macho. Para conectar el ICD3 con la placa en cambio se necesita un cable que tenga en un lado RJ11 macho y en el otro ICSP hembra.

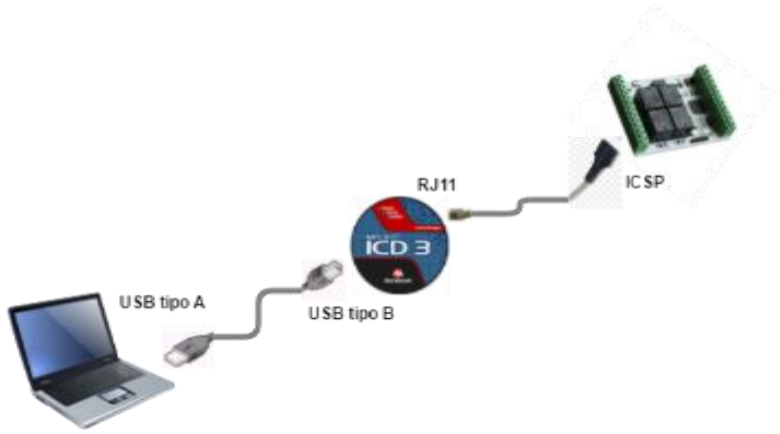


Ilustración 24: Diagrama de conexión para la programación del microcontrolador utilizando el debugger ICD

Las conexiones con el PICKit3 son diferentes a las de ICD3. Para conectarse el PICKit3 con el ordenador se precisa de un cable que tenga en un lado USB tipo A macho y en el otro mini USB macho. Para conectar el PICKit3 con la placa, en cambio, se necesita un cable que tenga en un lado ICSP macho y en el otro ICSP hembra.

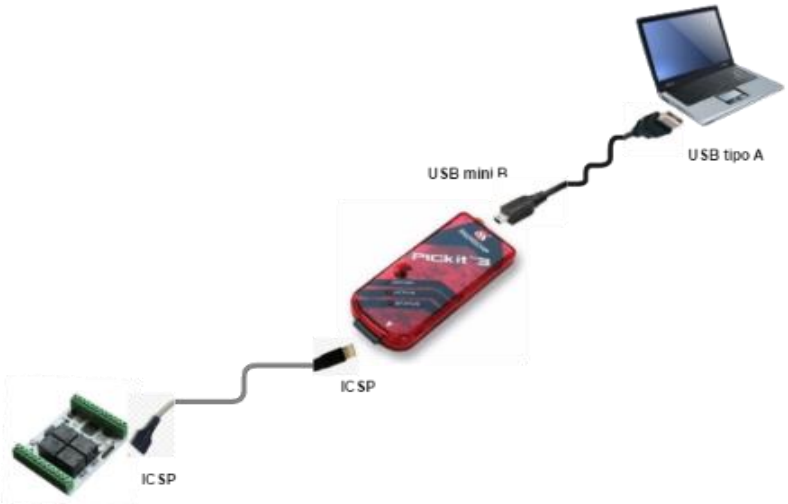


Ilustración 23: Diagrama de conexión para la programación del microcontrolador utilizando el debugger ICD

Para terminar cabe destacar que a la hora de hacer las pruebas hemos usado el sniffer Zena. Dicho sniffer captura todas las tramas enviadas entre los dispositivos y los enseña mediante la aplicación software Wireless Development Studio (WDS) V.3.2 en la pantalla.

Ilustración 25: Sniffer Zena

Una vez terminado el producto y hechas las pruebas de verificación se ha realizado una aplicación práctica. Para ello se han utilizado dos placas que tienen el microcontrolador PIC24FJ32GA102 y el *transceiver* MRF89XAM8A.

A continuación se detallan las características de cada hardware.

4.1.1 PIC24FJ128GA310

Es un microcontrolador PIC de 16 bits con las siguientes características:

- Velocidad de CPU: 16 MIPS, 32MHz.
- 64 KB de memoria de programa.
- 2 Módulos SPI.
- 4 Módulos de UART.
- 7 módulos de capturas de entradas.
- 5 timers de 16 bits con Prescaler. Estos timers se pueden emparejar consiguiendo timers de 32 bits.
- 5 fuentes de interrupción externas.
- Opción de ahorrar energía con la característica *DEEP-SLEEP* que hace que duerma la MCU. Se puede despertar mediante la interrupción externa. En este modo la corriente teórica es solamente de 40nA.

Ilustración 26: Microcontrolador PIC24FJ128GA310

4.1.2 PIC24FJ32GA102

Es un microcontrolador PIC de 16 bits con las siguientes características:

- Velocidad de CPU: 16 MIPS, 32MHz.
- 32 KB de memoria de programa.
- 2 Módulos SPI.
- 2 Módulos de UART.
- 5 módulos de capturas de entradas.
- 5 timers de 16 bits con Prescaler. Estos timers se pueden emparejar consiguiendo timers de 32 bits.
- 5 fuentes de interrupción externas.
- Opción de ahorrar energía con la característica DEEP-SLEEP que hace que duerma la MCU. Se puede despertar mediante la interrupción externa. En este modo la corriente teórica es solamente de 40nA.

Ilustración 27: Microcontrolador PIC24FJ32GA102

4.1.3 placa DOOR CONTROLLER PHY#1

La placa de desarrollo utilizada para la migración ha sido una placa creada por la empresa *Atelei Engineering*. Las características destacables relacionadas con el proyecto son las siguientes:

- Microcontrolador PIC24FJ128GA310.
- 10 leds.
- Una memoria EEPROM de 4KB.
- UART.
- Un SPI que une el microcontrolador con el EEPROM y el transceiver.

Ilustración 28: Placa de desarrollo DOOR CONTROLLER PHY#1

4.1.4 placa RSSI METER 868MHz

La placa de desarrollo utilizada para llevar a cabo la aplicación real ha sido una placa creada por la empresa *Atelei Engineering*. Las características destacables relacionadas con el proyecto son las siguientes:

- Microcontrolador PIC24FJ32GA102.
- Transceiver MRF89XSM8A.
- 8 leds.
- Un SPI que une el microcontrolador con el transceiver.

Ilustración 29: Placa de desarrollo RSSI METER 868MHz

Ilustración 30: Placa de desarrollo RSSI METER 868MHz

4.1.3 ICD3

Las especificaciones técnicas del módulo son las siguientes:

- Emulación de baja tensión (de 2,0 a 5,5 voltios).
- Ejecución en tiempo real.

4.1.4 PICKit3

Las especificaciones técnicas del módulo son las siguientes:

- Emulación de baja tensión (de 2,0 a 6,0 voltios).
- Ejecución en tiempo real.

4.1.5 MRF89XSM8A

Las especificaciones técnicas del módulo son las siguientes:

- Es compatible con los protocolos inalámbricos Sub-GHz del propietario. Por lo tanto es compatible con MiWi™.
- Tiene una interfaz SPI con interrupciones.
- Es compatible con los PIC PIC24.
- Consumo muy reducido.
  - Escuchando: 3mA
  - Enviando: 25 mA +10 dBm
  - Dormido: 0.1 µA
- Tasa de datos de 40 kbps.
- Buffer de transmisión y recepción de 64 bytes.
- Opera en las bandas de frecuencia europeas 863-870MHz

Ilustración 31: MRF89XAM8A

4.1.6 USB to Serial Bridge Controller

Las especificaciones técnicas del módulo son las siguientes:

- Lleva el transceptor USB v1.1 y el regulador 5V-3.3V.
- Soporta la interfaz en serie RS232 con la velocidad programable desde 75 bps hasta 12 Mbps.

4.2 Software

Al desarrollar la migración se ha seguido el desarrollo circular que se puede observar en la *ilustración 32*.

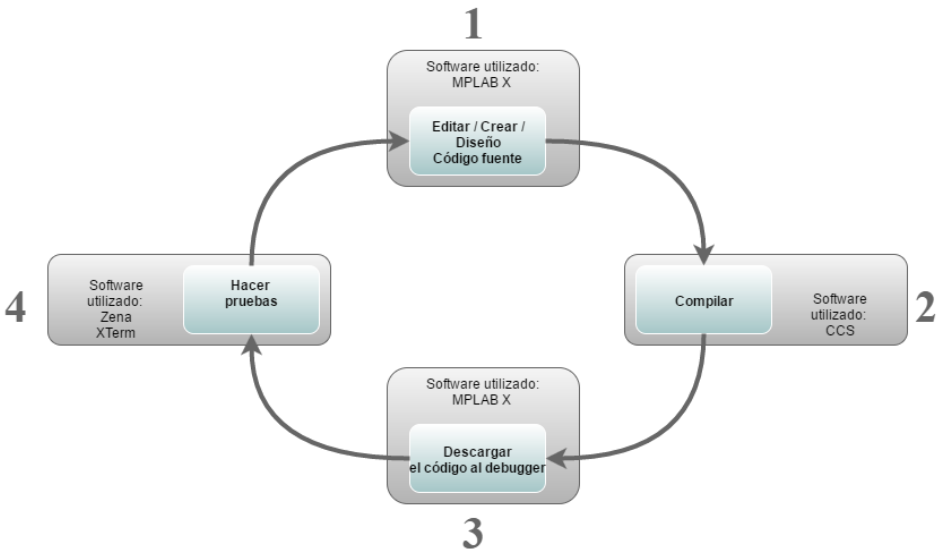


Ilustración 32: Ciclo de desarrollo

En cada ciclo de desarrollo se ha utilizado un software. Para la migración se ha utilizado el MPLAB X. Una vez que la migración se ha realizado, se ha compilado un proyecto con el

compilador CCS y el ejecutable resultante se ha cargado en el microcontrolador (PIC) y se ha procedido a realizar las pruebas.

En la fase de pruebas se han utilizado dos aplicaciones software. La primera de ellas ha sido *Wireless Development Studio*. Este software se ha utilizado para ver los paquetes que detectaba el sniffer. El segundo ha sido *RealTerm*, que se ha utilizado para imprimir mensajes en la pantalla a la hora de buscar las causas de los errores. También se ha hecho uso del analizador lógico *Saleae LLC* a la hora de solucionar errores.

Durante todo el transcurso del proyecto se ha utilizado la plataforma *Dropbox* para hacer las copias de seguridad tanto del código como de documentos.

Respecto a las aplicaciones software utilizadas para la comunicación con la empresa han sido *Skype* y *TeamViewer*.

4.2.1 MPLAB X IDE

A la hora de desarrollar el proyecto, se ha empleado MPLAB X V.3.15 desarrollada por la empresa Microchip y basado en netbeans de Oracle.

El IDE MPLAB X no lleva de serie la opción de compilar el lenguaje C orientado a CCS. Para ello tienes que agregar el compilador CCS. Hay que tener en cuenta que se necesita una licencia para poder trabajar con dicho compilador. [3] [4]

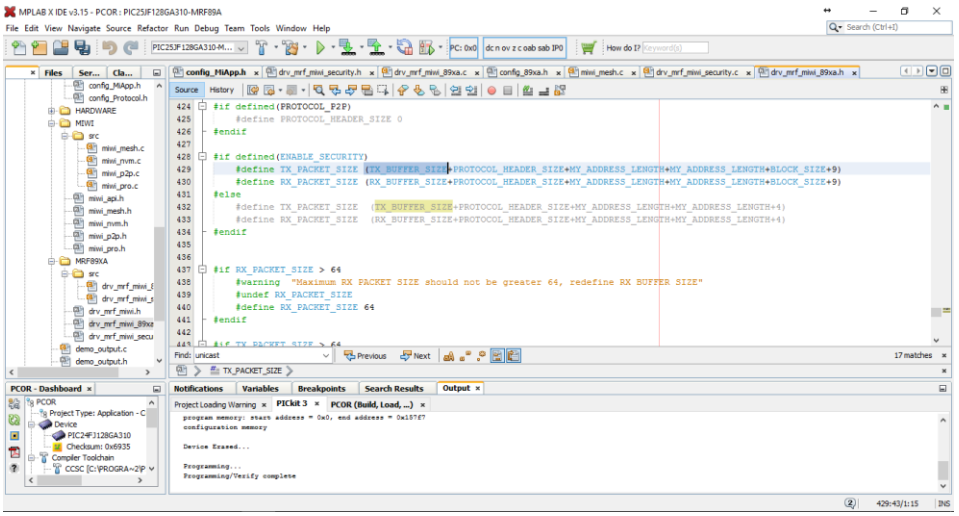


Ilustración 33: MPLAB X IDE

4.2.2 Compilador CCS

CCS es un compilador de C programado y comercializado por la empresa CCS Inc. para las aplicaciones integradas que utilizan dispositivos PIC MCU y PIC24/dsPIC DSC.

Este compilador tiene una optimización muy avanzada. Además como la sintaxis de las funciones es igual para todos los dispositivos la migración a otro microcontrolador se hace de un modo muy sencillo. También dispone de muchos ejemplos de programas y bibliotecas de dispositivos. [5]

4.2.3 Wireless Development Studio

El WDS es una interfaz gráfica de usuario basada en Java que permite el desarrollo rápido y fácil de aplicaciones inalámbricas basadas en los protocolos MiWi ™. Cuenta con un sniffer

para para el seguimiento, la depuración y la recopilación de información del protocolo MiWi™.

También dispone de un configurador con una interfaz gráfica de usuario que permite la personalización y configuración sencilla de redes inalámbricas.

En el proyecto solamente se utilizara para capturar tramas y así hacer un seguimiento y comprobar el correcto funcionamiento del protocolo. [6] [7]

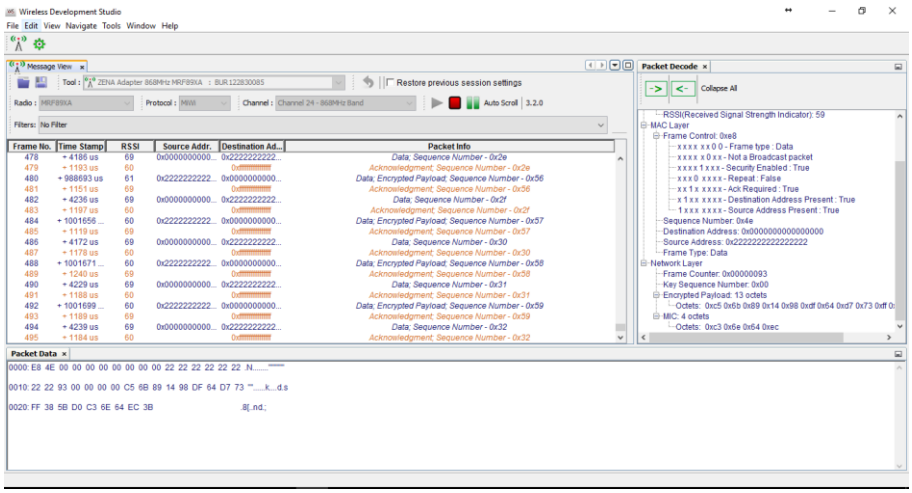


Ilustración 34: WDS

4.2.4 RealTerm

RealTerm es un programa de terminal especialmente diseñado para capturar, controlar y depurar secuencias de datos binarios y otras secuencias de datos difíciles. [8]

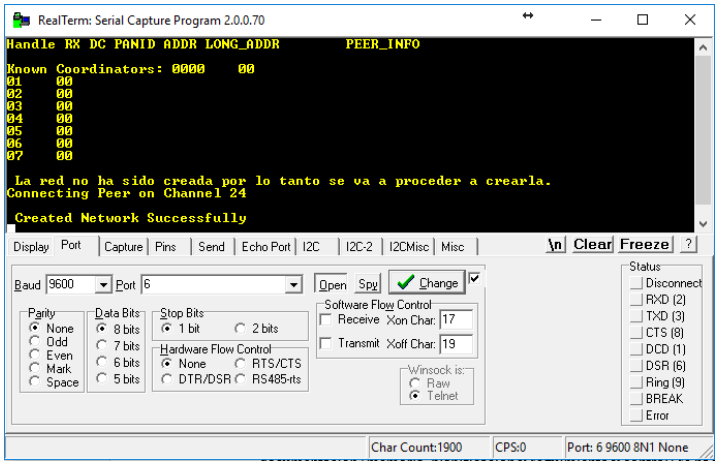


Ilustración 35: RealTerm

4.2.5 Saleae LLC

Es un analizador lógico fabricado por la empresa Saleae. Permite grabar la señal, decodificar según el protocolo que se esté utilizando, exportar a un fichero, etc. [9]

4.2.6 Dropbox

Dropbox es un servicio de alojamiento de archivos multiplataforma en la nube, operado por la compañía Dropbox. El servicio permite a los usuarios almacenar y sincronizar archivos en línea y entre ordenadores y compartir archivos y carpetas con otros usuarios. [10]

4.2.7 Skype

Es un software que permite comunicaciones de texto, voz y vídeo sobre Internet (VoIP). [11]

4.2.8 TeamViewer 11

Es un software de escritorio remoto. Entre sus funciones están: compartir y controlar escritorios, reuniones en línea, videoconferencias y transferencia de archivos entre ordenadores. [12]

5

MIGRACIÓN

5.1 Estructura del Proyecto

A la hora de migrar el protocolo MiWi™, lo primero que se ha llevado a cabo ha sido la correcta estructuración de los archivos, ya que éstos se encuentran en *Microchip Libraries for Aplications* (MLA de aquí en adelante) de forma desordenada.

Con motivo de dicha organización, se ha decidido clasificarlos en las siguientes carpetas:

- HARDWARE: Dentro de esta carpeta se almacenaran aquellos archivos que controlan el hardware utilizado. Los archivos *.c* se almacenaran en la carpeta *src* ubicada dentro de ésta carpeta.
    - *console.h* y *console.c:* en estos archivos se definen y se implementan las funciones que controlan el terminal.
    - *spi.h* y *spi.c:* en estos archivos se definen y se implementan las funciones que controlan el SPI.
    - *eeprom.h* y *eeprom.c:* en estos archivos se definen y se implementan las funciones que controlan la EEPROM.
    - *symbol.h* y *symbol.c:* en el protocolo MiWi™ el tiempo transcurrido se cuenta en ticks. Estos archivos son los que definen el tipo de dato *MIWI_TICK* y funciones para controlar el periodo transcurrido.
    - *system.h* y *system.c:* en estos archivos se definen los tipos de datos para el driver y también la función que inicializa el sistema.

Estos archivos en MLA se encuentran en el directorio: *mla\v2015_08_10\apps\miwi\miwi_p2p\simple_example\firmware\src\system_config*

- MIWI: Dentro de esta carpeta se almacenan los archivos que controlan el protocolo inalámbrico MiWi™. Los archivos *.c* se almacenaran en la carpeta *src* ubicada dentro de esta carpeta.
    - *miwi_p2p.h* y *miwi_p2p.c:* en estos archivos se definen y se implementan todos los tipos de datos y todas las funciones que necesita el protocolo de pila MiWi P2P.
    - *miwi_mesh.h* y *miwi_mesh.c:* en estos archivos se definen y se implementan todos los tipos de datos y todas las funciones que necesita el protocolo de pila MiWi Mesh.
    - *miwi_pro.h* y *miwi_pro.c:* en estos archivos se definen y se implementan todos los tipos de datos y todas las funciones que necesita el protocolo de pila MiWi Pro.
    - *miwi_nvm.h* y *miwi_nvm.c:* en estos archivos se definen estructuras de datos y se implementan funciones necesarias para utilizar la memoria no volátil.
    - *miwi_api.h:* en este archivo se definen estructuras de datos y funciones comunes en los tres protocolos de pila.

Estos archivos en MLA se encuentran en el directorio: *mla\v2015_08_10\framework\miwi*

- CONFIGURACIÓN: Dentro de esta carpeta se almacenan los tres archivos de configuración del protocolo MiWi™.
    - *miwi_config.h:* es el archivo de configuración de la aplicación.

- *Config_protocol.h:* es el archivo de configuración de los protocolos de pila. Cada protocolo de pila puede tener un archivo de configuración o puede que los tres protocolos tenga un único archivo. En nuestro caso la configuración de los tres protocolos se define en este único archivo.
- *config_89xa.h:* es el archivo de configuración del transmisor.

Estos archivos en MLA se encuentran en el directorio:
mla\v2015_08_10\apps\miwi\miwi_p2p\simple_example\firmware\src\system_config

- MRF89XA: Dentro de esta carpeta se almacenan los archivos que controlan el transmisor. Los archivos *.c* se almacenaran en la carpeta *src* ubicada dentro de esta carpeta.
  - *drv_mrf_miwi.h:* en este archivo se definen estructuras de datos y funciones comunes en los transceptores soportados por el protocolo.
  - *drf_mrf_miwi_89xs.h* y *drf_mrf_miwi_89xs.c:* son los archivos donde se definen y se implementan las funciones de la capa MAC. Estas funciones están diseñadas únicamente para el transceptor MRF89XA.
  - *drf_mrf_miwi_security.h* y *drf_mrf_miwi_security.c:* son los archivos donde se definen y se implementan las funciones de seguridad.

Estos archivos en MLA se encuentran en el directorio: *mla\v2015_08_10\framework\driver\mrf_miwi*

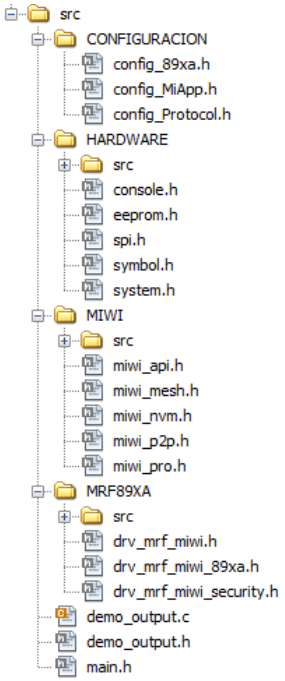


Ilustración 36: Estructura del proyecto

5.2 Bibliotecas

Una biblioteca es un conjunto de funciones compiladas, que pueden ser utilizados por el/la programador/a para realizar determinadas operaciones. El compilador CCS ofrece una serie de bibliotecas, siendo estas utilizadas para la realización del proyecto.

Las variables de tamaño fijo (*uint8_t*, *uint16_t*, *uint32_t*, etc.) el compilador XC16 las reconoce en cambio el compilador CCS no. Por esta razón se ha tenido que utilizar la biblioteca *stdint.h.* También se consideró la opción de definirlas en la cabecera del archivo principal o cambiar los tipos uno a uno; no obstante se decidió emplear las bibliotecas por ser la vía más práctica y respetuosa con el código original.

Por esta misma razón y ya que los booleanos y la variable NULL no se definen igual en ambos compiladores, se ha hecho uso de las bibliotecas *stdbool.h* y *stddef.h.*

5.3 Directivas

Las directivas son instrucciones dirigidas al compilador y no al microcontrolador. Las directivas se diferencian de las instrucciones en que:

- Se suelen ubicar en los ficheros de cabecera del programa (.h)
- Todas comienzan con el símbolo del numeral #

El orden de las directivas es crucial; primero siempre se le indica al compilador cual es el PIC que se va a emplear y luego cual es la frecuencia del oscilador. Posteriormente, se le puede agregar el resto de directivas. Si no se siguiera este proceso habría probabilidad de error en el proceso de compilación especialmente cuando se usa alguna función propia del compilador para manejar algún periférico y cuando esta misma necesita saber la frecuencia del oscilador, por ejemplo *spi_read( )* o *spi_write( ).* Todas estas directivas se agregan en el archivo *main.h*.

5.3.1 Directivas agregadas

En este punto aparecerán solamente las directivas generales. Las directivas de cada periférico se explicaran en los puntos donde se desarrollan las configuraciones de dichos periféricos.

Como se ha explicado anteriormente, la primera directiva a introducir es el PIC que se vaya a usar. En nuestro caso el PIC usado es el PIC24FJ128GA310.

También habrá que agregar la directiva del debugger que se vaya a usar. Sin que se agregue esta directiva sería imposible debuggear el programa.

Una vez agregada la directiva del microcontrolador ya se puede hacer uso de las funciones que ofrece CCS para dicho microcontrolador. Algunas funciones no tendrán un funcionamiento adecuado puesto que aún falta definir la frecuencia del reloj. Para ello, se utiliza la directiva *#use delay( ).* A esta directiva hay que pasarle tanto el tipo (Interna, oscilador, cristal o RC) como la frecuencia del reloj. En nuestro caso, se empleó un oscilador interno con la frecuencia de 32MHz.

Una vez que el compilador sabe que microcontrolador se va a utilizar y la frecuencia del reloj se pueden activar o desactivar ciertos bits de configuración. Para ello se usan las directivas *fuses* que indican al compilador si se usa o no cierto bit. Este bit normalmente está

relacionado con la activación o no de cierto dispositivo como puede ser el dispositivo Watch Dog Timer (resetea el microcontrolador ante un bloqueo del mismo).

Las etiquetas usadas para activar o desactivar esos bits de configuración, están incluidas en el archivo de cabecera del microcontrolador y deben ser consultadas siempre, ya que éstas suelen cambiar entre versiones del compilador o tipos de microcontroladores.

Al poner la palabra NO a la etiqueta se informa al compilador de que el dispositivo en cuestión está desactivado, mientras que escribiendo solo la etiqueta activamos el dispositivo. Por otra parte, la activación o desactivación de los distintos bits de configuración se puede realizar en varias líneas o en una sola línea separando cada etiqueta con comas.

Para el desarrollo de este proyecto se han utilizado las siguientes directivas *#fuses*:

- #FUSES NOWDT: Con esta directiva desactivaremos el Watchdog Timer. El WDT lo que hace fundamentalmente es resetear el micro tras un periodo de tiempo determinado.
- #FUSES NOBROWNOUT: Con esta directiva desactivaremos el BrownOut. El BrownOut resetea el PIC si la alimentación VCC baja de un cierto valor.

```
#include <24FJ128GA310.h>

#device ICD=TRUE
#device ICSP=2

#use delay(internal=32MHz)

#FUSES NOWDT                    //No Watch Dog Timer
#FUSES NOBROWNOUT               //No brownout reset
```

5.4 Serial Peripheral Interface (SPI)

El Bus SPI es un estándar de comunicaciones que se utiliza para la transferencia de paquetes de 8 bits entre circuitos integrados. Las comunicaciones son síncronas y full-duplex, es decir, son reguladas por un reloj y la comunicación es bidireccional.

El bus tiene tres líneas: una línea es la del reloj (en adelante SCK), otra la de los datos entrantes (en adelante SDI) y la siguiente la de los datos salientes (en adelante SDO).

Los dispositivos conectados al Bus, en el caso de este proyecto, el microchip, la memoria EEPROM y el *transceiver*, son definidos como maestro o esclavos. Un maestro es el dispositivo que inicia la transferencia de datos sobre el bus y genera las señales de reloj y control. Un esclavo, en cambio, es el dispositivo controlado por el maestro. Cada esclavo es controlado a través de una línea Chip Select (en adelante CS). Solamente puede estar activado un esclavo en cada momento; por tanto, la línea CS de un esclavo estará activado cuando se esté comunicando con el maestro y todas las otras líneas CS estarán desactivadas. Cada esclavo es seleccionado por un nivel lógico bajo (‘0’).

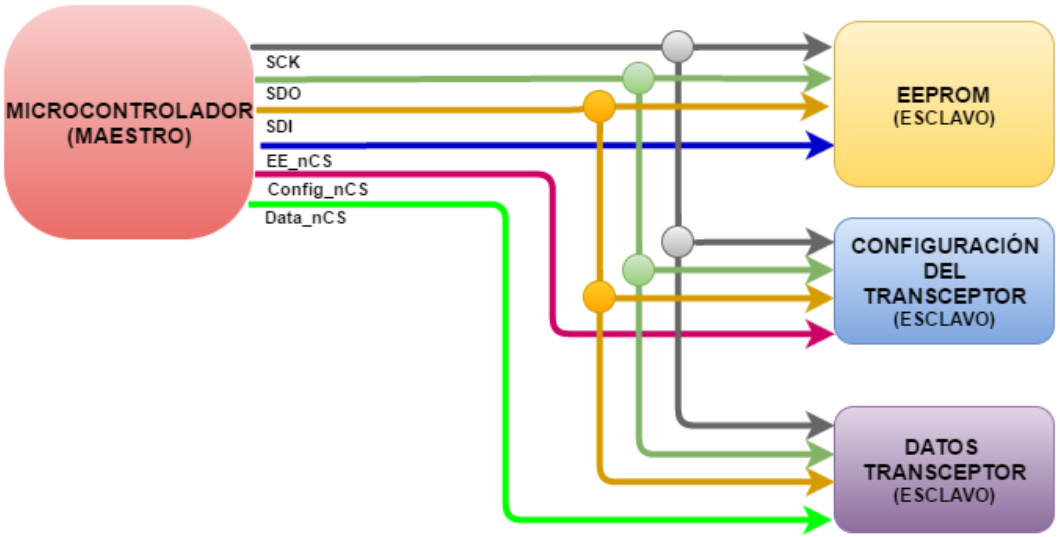


Ilustración 37: Comunicación SPI

El protocolo MiWi™ ofrece dos opciones a la hora de utilizar el bus SPI: La primera opción es utilizar dos buses SPI, una para la memoria EEPROM y la otra para el *transceiver*. La otra opción es que compartan el bus la memoria y el *transceiver*. Por otro lado el PIC utilizado también tiene la opción de controlar dos buses SPI ya que dispone de dos dispositivos maestros. La placa, en cambio, solo dispone de un solo bus en consecuencia el protocolo en esta placa solo podrá funcionar con un bus SPI. Aunque se tenga que compartir el bus SPI por las limitaciones de la placa, se implementara la estructura del segundo para que, en caso de incorporar otro bus a la placa, pueda ser configurado fácilmente.

Como tenemos un solo bus SPI, habrá un solo maestro comunicando con tres esclavos (un esclavo en cada momento). Cuando se quiera comunicar con la memoria, ya sea para leer o escribir activaremos su línea CS (*EE_nCS*). En cambio si se quisiera configurar el *transceiver* ya sea para poner en modo emisor o receptor se activara la línea *Config_nCS*. Por ultimo si se quisiera recibir o enviar datos desde el *transceiver* se activara la línea *Data_nCS*.

5.4.1 Configuración

La configuración del SPI mediante la programación C orientada al compilador CCS resulta sencilla puesto que se elabora mediante funciones de la biblioteca *PIC24FJ128GA310.h*. En este documento solo aparece la configuración del SPI1 ya que son casi idénticos.

La configuración tanto del SPI1 como del SPI2 se hará en el archivo *main.h* y se implementara en dos pasos. El primero de ellos será la asignación de un PIN a cada línea descrita en la *ilustración 37* a excepción de las líneas CS. El segundo consistirá en definir el comportamiento del mismo.

La asignación de los PINs se hará de esta forma:

```
#pin_select SCK1OUT=PIN_SCK
#pin_select SDI1=PIN_SDI
#pin_select SDO1=PIN_SDO
```

Seguidamente se definirá el comportamiento del mismo a través de la directiva *#use SPI*. A esta directiva se le pasaran los siguientes parámetros:

- MASTER o SLAVE: Este parámetro configura el rol que desempeña el dispositivo SPI del microcontrolador.
- BAUD: Este parámetro configura la velocidad de comunicación.
- MODE: El modo en el que opera el bus SPI. Hay cuatro modos (véase la *ilustración 38*).

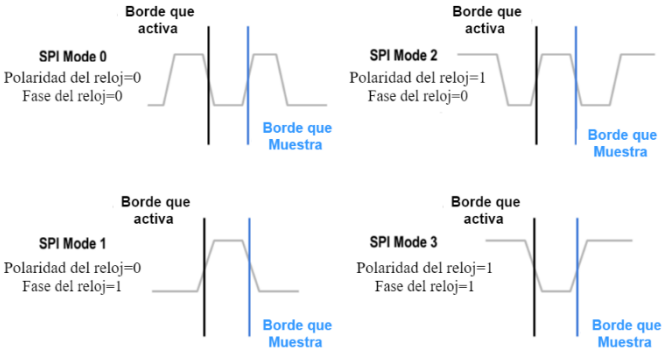


Ilustración 38: Modos SPI

- BITS: Este parámetro configura cuantos bits se enviaran en cada transferencia.
- STREAM: Especifica un nombre para este SPI.

```
#use spi(MASTER,BAUD=BAUD_RATE, MODE=X, BITS=Y, stream=SPI_PORT1)
```

5.4.2 Utilización

Para utilizar el SPI, primeramente hay que seleccionar el esclavo con que se quiere comunicar. Este paso se tendrá que hacer manualmente. No hay que olvidar volver a desactivar después de hacer la operación. Estos cambios se han tenido que realizar en los archivos *miwi_nvm,c* y *drv_mrf_miwi_89xa.c*.

```
output_low(CS);// Se activa el CS. CS=Data_nCS, Config_nCS, EE_nCS.
//Cualquier operación de SPI (Escribir, leer).
output_high(CS);//Se desactiva CS
```

Una vez seleccionado el esclavo, se podrá enviar información al esclavo o recibir información de él. Las bibliotecas de CCS ofrecen funciones para hacer dicha escritura o lectura. En el caso de la escritura ofrece la función *spi_write(X)* y para la lectura *spi_read(Y)*. En las lecturas del bus SPI siempre hay que escribir algo en el bus para que podamos leer y, por ello, le pasamos el parámetro *Y*.

Después de llegar a este punto tenemos dos opciones de migración. La primera es cambiar todas las llamadas a la escritura y lectura del bus (por las funciones descritas en el párrafo anterior). La segunda opción es modificar las funciones de lectura y escritura implementadas en *spi.c* y llamar a las funciones de escritura y lectura de CCS dentro de estas funciones. Para el desarrollo de este proyecto se ha optado por la segunda opción puesto que es la más asequible y respetuosa con el código original ya que solo se tendrá que modificar el archivo *spi.c*.

5.5 Universal Asynchronous Receiver-Transmitter (UART)

A la hora de llevar a cabo las pruebas será conveniente imprimir información en el terminal para verificar que ésta es apropiada y, en caso de que no sea así, buscar los posibles errores. Para ello se ha utilizado el Transmisor-Receptor Asíncrono Universal (en adelante UART).

El UART es una simple y útil interfaz de comunicación de serie que permite la comunicación del microchip con dispositivos periféricos, tales como computadoras personales. Es un sistema full-duplex asíncrono. Las unidades de datos básicos transferidos son bytes.

A la hora de enviar información se envían tramas de datos de 8-X bits. Primero se envía un bit de inicio (el bit '0'), después se envía el dato (normalmente es de un byte) seguido por un bit de paridad opcional (normalmente no se utiliza) y uno o dos bits de parada (el bit '1').

Cuando la línea está libre, está a nivel lógico alto. La velocidad a la que se envían los bits se llama velocidad de transmisión. Las velocidades de transmisión más comunes son 9600, 19200, 38400, 115200.

Por este motivo, a la hora de configurar el UART será necesario conocer los PINs de lectura y escritura, la velocidad de transmisión o la tasa de datos, el tipo de paridad y el número de bits de parada.

5.5.1 Configuración

Al igual que la configuración del SPI, la configuración del UART resulta sencilla pues se hace a través de funciones ofrecidas por las bibliotecas de CCS.

La configuración del UART se hará en el archivo *main.h* como se detalla a continuación:

Primero se asignara los PINs del microcontrolador a la línea de lectura y escritura.

```
#pin_select U1TX=PIN_TX
#pin_select U1RX=PIN_RX
```

Una vez asignados los PINs se efectuara la configuración. Hay que tener cuidado ya que esta directiva tiene que ir siempre después de la directiva *#use delay*.

```
#use rs232(UART1, baud=BAUD_RATE, stream=salida)
```

En cuanto a la paridad y el número de bits de parada se utilizara el que está por defecto. Por lo tanto, solo se tendrá que pasar a la directiva qué UART queremos utilizar, la velocidad de transmisión y el nombre.

5.5.2 Utilización

Cuando este todo configurado y listo para ser utilizado la biblioteca CCS ofrece las funciones *printf( )*, *putc( )* y *getc( )*.

Al igual que en la utilización del SPI tenemos dos opciones de migración. La primera es cambiar todas las llamadas a la escritura y lectura. La otra es modificar solamente las funciones *CONSOLE_PutString( )*, *CONSOLE_Put( )* y *CONSOLE_Get()*. Se ha optado por la segunda opción, por ser más respetuosa con el código existente y resultar más sencilla a la hora de elaborar el proyecto; ya que solo se tendrá que modificar el archivo *console.c*.

5.6 Temporización

En el protocolo MiWi™ las temporizaciones se cuentan en *ticks*. Esta unidad de tiempo está definida en el estándar 802.15.4. Un tick es equivalente a un símbolo (*Simbol time*) o a 16 us.

La estructura del tick (*MIWI_TICK*) está formada con 4 bytes y es capaz de contar hasta 19 horas.

Tiene la siguiente estructura:

-El valor entero (4 bytes).

-El valor entero dividido en dos palabras de 16 bytes *(wordTICK.w1 y wordTICK.w2*).

-El valor entero dividido en bytes (*byteTICK.b1, byteTICK.b2, byteTICK.b3, byteTICK.b4*).

El microcontrolador que se ha empleado tiene cinco timers de 16 bytes por lo que para poder contar los 32 bytes del tick es necesario utilizar dos. En el software ofrecido por microchip este proceso se realiza con dos timers (TIMER2 y TIMER3). Al contrario, en la programación C orientada al compilador CCS no es necesario configurar los dos; se puede configurar directamente como si fuera uno solo de 32 bytes utilizando las funciones de las bibliotecas que nos ofrece.

La función mencionada anteriormente es *#use timer()* a esta función se le pasará tanto la duración de un tick (*TICK=16us*) como el tamaño (*bits=32*) del mismo. También se tendrá que pasar el parámetro NOISR para desactivar la interrupción por el desbordamiento del timer.

Una vez configurado para coger el valor del tick del momento hemos llamado a la función *get_tick()* y el valor que nos da hemos metido en la estructura del tick.

Para introducir en la estructura, hemos tenido que realizar diferentes operaciones para separarlo en palabras o en bytes. Las operaciones realizadas han sido desplazamientos (>>) y selecciones (&FFFF para las palabras y &FF para bytes).

5.7 Interrupciones

Las capturas de entradas son un método de gestionar las entradas en un sistema embebido. Cómo el mismo nombre indica capturará los eventos que sucedan en el PIN de entrada correspondiente. La captura podrá saltar una interrupción, en caso de que esto ocurra, el microcontrolador pausara la tarea que estaba realizando en ese momento y pasara a ejecutar la rutina de la interrupción que ha saltado.

Por un lado, los eventos podrán ser tanto subidas como bajadas de los pines correspondientes a las capturas, además, podrá capturar cada subida o bajada del flanco, cada 4 subidas de flanco o cada 16 subidas. Por otro lado, las interrupciones podrán saltar en cada captura, cada dos capturas, cada tres capturas o cada cuatro capturas.

El protocolo MiWi™ tiene que capturar dos eventos. El primero es cuando llega algún mensaje al *transceiver*; este evento tiene que hacer saltar la interrupción y pasar a gestionar el mensaje recibido. El segundo evento será al enviar el mensaje; este evento también hará saltar la interrupción. Cuando salte, dejara la tarea que está realizando y pasara a gestionar la interrupción.

Para llevar a cabo este procedimiento primeramente se tendrán que configurar bien las capturas y después se gestionarán las interrupciones saltadas por las citadas capturas.

5.7.1 Configuración

La configuración de las capturas en el lenguaje C orientado al compilador CCS se hace en dos pasos. El primer paso es asignar los PINs de entrada en la que estarán esperando a los eventos. El segundo configurar las capturas.

La asignación de los PINs se hace de la manera descrita en los puntos anteriores.

```
#pin_select IC1=IRQ1_INT_PIN
#pin_select IC2=IRQ0_INT_PIN
```

A continuación se detalla cómo se realiza la configuración:

- La interrupción saltara en flanco ascendente (Rising Edge).
- Las interrupciones saltaran en cada cambio de estado en el pin asociado.
- El timer a tener en cuenta será el timer del sistema.

```
setup_capture(1, CAPTURE_RE | INTERRUPT_EVERY_CAPTURE |
CAPTURE_SYSTEM_CLOCK | CAPTURE_SYNCHRONIZE | CAPTURE_TRIG_SYNC_IC1);

    setup_capture(2, CAPTURE_RE | INTERRUPT_EVERY_CAPTURE |
CAPTURE_SYSTEM_CLOCK | CAPTURE_SYNCHRONIZE | CAPTURE_TRIG_SYNC_IC2);
```

Una vez configuradas las dos capturas se habilitarán las interrupciones de dichas capturas. Para que las interrupciones puedan saltar hay que habilitar todas las interrupciones.

```
enable_interrupts(INT_IC1);
enable_interrupts(INT_IC2);
enable_interrupts(INTR_GLOBAL);
```

5.7.2 Utilización

Una vez que las interrupciones estén configuradas y que cada evento dispare la interrupción se proseguirá con la gestión de las interrupciones. Para la gestión de interrupciones las bibliotecas de CCS ofrecen estas funciones.

- *interrupt_active(Interrupción):* Esta función dice si el flag de la interrupción está activado, es decir, dice si ha saltado o no la interrupción. Si devuelve un 1 es que ha saltado y si devuelve un 0 es que no.
- *clear_interrupts(Interrupción):* Al acabar de ejecutar la rutina de la interrupción hay que limpiar el flag para que no vuelva a entrar directamente en la interrupción. Esta función es la encargada de limpiar el flag. Llamar a esta función tiene el mismo efecto que hacer la operación *IF=0* en el lenguaje C orientado al compilador XC16.
- *enable_interrupts(Interrupción):* Esta función habilita la interrupción. Llamar a esta función tiene el mismo efecto que hacer la operación *IE=1* en el lenguaje C orientado al compilador XC16.
- *disable_interrupts(Interrupción):* Esta función deshabilita la interrupción. Llamar a esta función tiene el mismo efecto que hacer la operación *IE=0* en el lenguaje C orientado al compilador XC16.
- *Interrupt_enabled(Interrupción):* Esta función dice si la interrupción está activada o no. Devolverá un 1 si la interrupción esta activada y 0 si no lo está.

A la hora de hacer la migración se han realizado estos cambios:

Las operaciones *PHY_IRQ0 = 0;* y *PHY_IRQ1 = 0;* limpian el flag de la respectiva interrupción por lo tanto se le ha llamado a la función *clear_interrupts()*.

Las operaciones *PHY_IRQ0_En = 0;* y *PHY_IRQ1_En = 0;* deshabilitan la respectiva interrupción por lo tanto se le ha llamado a la función *disable_interrupts( )*.

Las operaciones *PHY_IRQ0_En = 1;* y *PHY_IRQ1_En = 1;* habilitan la respectiva interrupción por lo tanto se le ha llamado a la función *enable_interrupts( )*.

A la hora de preguntar por el flag de la interrupción o si esta activada la interrupción hay dos opciones. La primera opción es cambiar todas las veces que aparecen *PHY_IRQ0* y *PHY_IRQ1* por la función *interrupt_active()* y todas las veces que aparecen *PHY_IRQ0_En* y *PHY_IRQ0_En* por la función *interrupt_enabled( )*. La segunda opción es utilizar un operador condicional.

En este proyecto se decidió optar por la segunda opción por ser la más sencilla y respetuosa con el código original. A continuación se detalla el aspecto final:

```
#define PHY_IRQ0        (interrupt_active(Interrupción)?1:0)
#define PHY_IRQ0_En     (interrupt_enabled(Interrupción)?1:0)
```

6

MEJORAS EN EL PROTOCOLO LLEVADAS A CABO

6.1 Enviar mas de un mensaje indirecto por cada mensaje "*data request*"

El protocolo MiWi™ posee la opción de que un dispositivo RFD se pueda dormir. Si dicho dispositivo se duerme, el padre de éste tiene que ser capaz de almacenar los mensajes indirectos destinados a él. Al despertar, este dispositivo enviara un mensaje *data request* al cual el padre responderá con el ACK de la capa MAC (si esta activada) y un mensaje almacenado. Para que el dispositivo RFD pueda recibir un nuevo mensaje almacenado deberá enviar otro mensaje *data request*.

La empresa *Atelei Engineering* estaba especialmente interesada en que cualquier dispositivo RFD tuviera la capacidad de dormirse, no obstante, no acababan de parecer convencidos con su funcionamiento. La intención era que al despertarse el dispositivo mandara el mensaje *data request* y que su padre le enviara todos los mensajes almacenados seguidos uno tras otro.

Dado que el funcionamiento del protocolo era diferente al funcionamiento que la empresa deseaba se han debido modificar los tres protocolos de pila. Los cambios realizados en los protocolos de pila MiWi Mesh y MiWi Pro han sido los mismos ya que tienen igual modo de gestionar los mensajes almacenados. En cambio, en el protocolo de pila MiWi P2P ha sido necesario realizar otros cambios, ya que éste gestiona de manera diferente los mensajes almacenados.

6.1.1 Cambios en los protocolos de pila MiWi Mesh y MiWi Pro

En los protocolos de pila MiWi Mesh y MiWi Pro se ha modificado la función que manda los mensajes almacenados (*sendIndirectPacket()*) en los ficheros tanto *miwi_mesh.c* como *miwi_pro.c*. Las modificaciones que se han realizado son las siguientes:

1. En el protocolo ofrecido por Microchip después de enviar un mensaje llama al comando *return* y, por tanto, después de enviar un mensaje abandona esta función.

En el protocolo que se ha modificado se ha decidido eliminar dicha llamada y, en consecuencia, al enviar un mensaje no abandona la función y pasa a verificar si hay algún mensaje almacenado válido más.

2. En caso de que encuentre algún mensaje válido más tendrá que escribir en el buffer de transmisión, y enviar dicho mensaje. Por este motivo, después de cada envío de mensaje se ha ordenado que limpie el buffer de transmisión.
3. Después de analizar todo el buffer de los mensajes almacenados debe abandonar la función, por tanto, se ha llamado al comando *retun*.

El tamaño del buffer de los mensajes indirectos se define en el archivo de configuración *config_protocol.h* del coordinador.

Hay que tener en cuenta que todos los mensajes se mandan prácticamente a la vez; por esta razón en la capa MAC del dispositivo que se duerme habrá que prestar especial atención al buffer de mensajes recibidos. Por defecto tiene un buffer (*BANK_SIZE*) de dos mensajes. En caso de que el tamaño del buffer de los mensajes indirectos del coordinador sea mayor de dos habrá que aumentar el tamaño del buffer de mensajes recibidos del nodo final (en el archivo de configuración *config_89xa.h*), pues, en caso contrario, los mensajes que no quepan serán eliminados.

En caso de que el ACK de la capa de aplicación o el ACK de la capa MAC están activadas este buffer de mensajes recibidos también se tendrá que aumentar en el coordinador.

Por lo tanto hay que prestar especial atención al parámetro *INDIRECT_MESSAGE_SIZE* del coordinador y al parámetro *BANK_SIZE* del nodo final. En caso de que este activado algún ACK también habrá que prestar especial atención al parámetro *BANK_SIZE* del coordinador.

6.1.2 Cambios en el protocolo de pila P2P

En el protocolo de pila MiWi P2P se tuvo que modificar la función que gestiona los mensajes recibidos (*P2PTasks()*) en el fichero *miwi_p2p.c.* En dicha función solamente se cambió el apartado que gestiona los mensajes recibidos que sean *data request* (*CMD_DATA_REQUEST* o *CMD_MAC_DATA_REQUEST*). Las modificaciones realizadas han sido las siguientes:

1. En el protocolo ofrecido por Microchip después de enviar un mensaje éste iba al apartado de finalización de envío de mensajes almacenados (*END_OF_SENDING_INDIRECT_MESSAGE*). En ese apartado elimina el mensaje *data request* y sale de esta función.

En el protocolo modificado al enviar un mensaje no termina con la gestión de mensajes almacenados; sino que pasa a verificar si hay algún mensaje almacenado válido más.

2. Si encuentra algún mensaje válido tendrá que escribir en el buffer de transmisión y enviar dicho mensaje. Por ello después de cada envío de mensaje se tendrá que limpiar buffer de transmisión.
3. Después de analizar todo el buffer de los mensajes almacenados deberá salir de la función. Justo después está el apartado de finalización de envío de mensajes almacenados conque no hace falta llamar a la misma.

Como se ha explicado en el punto anterior, hay que tener en cuenta que todos los mensajes se mandan prácticamente a la vez y que, por este motivo, en la capa MAC del dispositivo que se duerme habrá que prestar especial y cuidadosa atención al buffer de mensajes recibidos. En caso de que el ACK de la capa MAC esté activada este buffer de mensajes recibidos también se tendrá que aumentar en el coordinador.

7

PRUEBAS REALIZADAS

A la hora de llevar a cabo las pruebas se utilizaran tres placas DOOR CONTROLLER PHY#1 cada uno conectado a un transceiver MRF89XAM8A.

Las direcciones EUI que se le asignaran a cada controlador serán de 8 bytes.

La red que se formara estará compuesta por un PAN coordinador (en adelante PCOR), un coordinador (en adelante COR) y un dispositivo final (en adelante ED). El PANID de la red será 0000.

La dirección EUI del dispositivo PCOR será 0000000000000000, el del dispositivo COR 1111111111111111 y el del dispositivo ED 2222222222222222.

Todo esto se tendrá que configurar en el archivo de configuración *config_MiApp.h* junto a otras configuraciones que se detallaran más adelante.

Una vez configurado hay que hacer una aplicación de prueba. Todas las aplicaciones que vayan a utilizar este protocolo siguen más o menos esta estructura aunque con algunas derivaciones:

Ilustración 39: La red utilizada para llevar a cabo las pruebas

Ilustración 40: Diagrama de flujo del sistema

En este apartado de la memoria se va a ir creando una aplicación paso a paso hasta tener todo el flujo del sistema implementado. Una vez hecho esto se probaran diferentes características mediante ejemplos cotidianos.

7.1 Formación de la red

Antes de formar la red hay que configurar el protocolo de pila que se vaya a utilizar. Esta configuración se introducirá en el archivo de configuración *config_MiApp.h.*

Ilustración 41: Formación de la red

Una vez que las configuraciones estén confeccionadas se deberá programar la primera parte de la aplicación. Para la realización de esta prueba es imprescindible programar también la parte de datos recibidos ya que, de algún modo, tendrá que contestar a las peticiones de baliza. El procedimiento de lectura de paquetes recibidos se desarrollará en el *punto 7.2*.

El bloque de la configuración será el mismo para los tres dispositivos. Dentro de este bloque se tomarán dos acciones. La primera; la inicialización del protocolo y la segunda; el

establecimiento del canal. Si se quisiera recuperar el estado de la red anterior no se tendrá que establecer el canal.

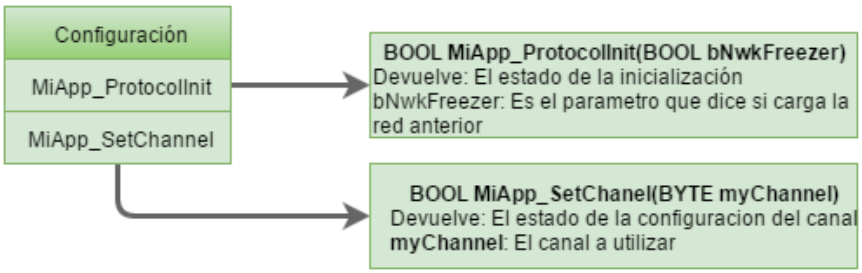


Ilustración 42: Configuración

Una vez acabado el bloque de la configuración, se proseguirá con la implementación de las instrucciones necesarias para obtener una conexión. Dentro de este bloque también se tomaran dos acciones. La primera, seleccionar el modo de conexión. La segunda será diferente según el tipo de dispositivo. El dispositivo PCOR creará la red y en cambio los dispositivos COR y ED establecerán conexión con el dispositivo PCOR. Si se quiere recuperar la red anterior no se tendrá que utilizar este bloque.

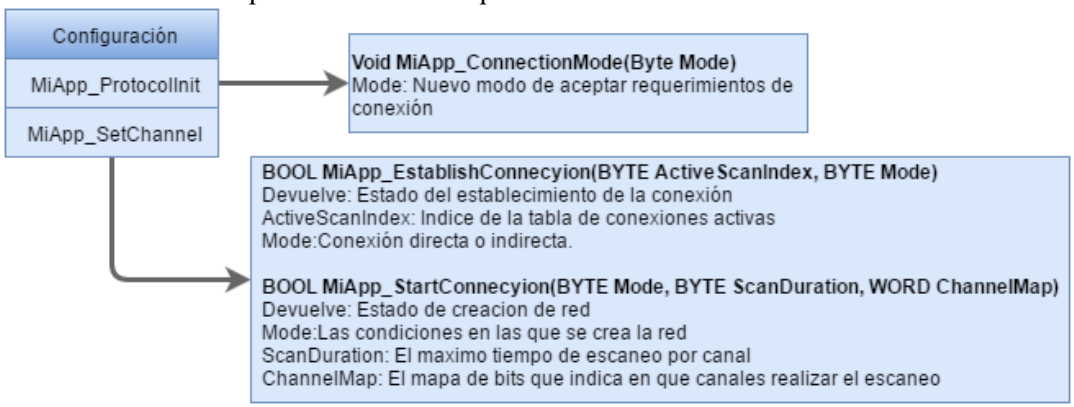


Ilustración 43: Obtener la conexión

Una vez implementada la aplicación hasta este punto sólo falta poner un bucle infinito y mirar si hay algún mensaje recibido continuamente. Después, ya se podrán realizar las pruebas de la formación de la red. Los resultados obtenidos se desarrollan a continuación:

7.1.1 MiWi P2P

Al activar el dispositivo PCOR no se verá nada ya que lo único que hace es crear la red y para ello no hay ningún intercambio de mensajes.

Después, al activar el dispositivo COR(véase la *ilustración 44*), el dispositivo COR empieza enviando unas peticiones de conexión (#4) a la que le responde el dispositivo PCOR (#5). En la respuesta le manda el SA. El dispositivo COR al recibir la respuesta le envía un ACK de la capa MAC (#6).

Después el dispositivo ED se activa y empieza a enviar unos mensajes de petición de conexión (#7). El dispositivo COR al recibir la petición enviara una respuesta con la SA asignada (#8). A esta respuesta el dispositivo ED responderá con un ACK de la capa MAC (#9).

4	+ 395856 us	54	0x1111111111...	0xffffffffffff	MiWi P2P Command : P2P Connection Request
5	+ 3000 us	69	0x0000000000...	0x1111111111...	MiWi P2P Command : P2P Connection Response
6	+ 1212 us	54		0xffffffffffff	Acknowledgment; Sequence Number - 0x60
7	+ 2532585 ...	63	0x2222222222...	0xffffffffffff	MiWi P2P Command : P2P Connection Request
8	+ 3018 us	53	0x1111111111...	0x2222222222...	MiWi P2P Command : P2P Connection Response
9	+ 1196 us	63		0xffffffffffff	Acknowledgment; Sequence Number - 0xa0

Ilustración 44: Formación de una red P2P

7.1.2 MiWi Mesh

Con el protocolo MiWi Mesh al activar el dispositivo PCOR, tampoco se verá nada ya que lo único que hace es crear la red y para ello no hay ningún intercambio de paquetes.

Al activar el dispositivo COR, en cambio, empezará a enviar peticiones de baliza (#3781). El dispositivo PCOR al recibir una de ellas, y si aún la tabla de conexiones no la tiene llena, le responderá con una respuesta de baliza (#3782).

Después el dispositivo COR enviará una petición de asociación (#3783) que será respondida con una respuesta de asociación en la que se le asignará una dirección (#3784) (véase las *ilustraciones 46 y 47*).

Todos los mensajes unicast (#3783, #3785, #3789, #3791) necesitaran un ACK de la capa MAC (#3784, #3786, #3790, #3792).

3781	+ 197559 us	51	0x1111111111...	0xffffffffffff	MiWi Command : Beacon Request
3782	+ 4604 us	69	0x0000000000...	0xffffffffffff	Beacon
3783	+ 193762 us	52	0x1111111111...	0x0000000000...	MiWi Command : Association Request
3784	+ 1213 us	69		0xffffffffffff	Acknowledgment; Sequence Number - 0xa0
3785	+ 4401 us	68	0x0000000000...	0x1111111111...	MiWi Command : Association Response
3786	+ 2730 us	51		0xffffffffffff	Acknowledgment; Sequence Number - 0x3c
3787	+ 2494239 ...	61	0x2222222222...	0xffffffffffff	MiWi Command : Beacon Request
3788	+ 6408 us	60	0x1111111111...	0xffffffffffff	Beacon
3789	+ 191288 us	63	0x2222222222...	0x1111111111...	MiWi Command : Association Request
3790	+ 1205 us	60		0xffffffffffff	Acknowledgment; Sequence Number - 0x8b
3791	+ 3821 us	60	0x1111111111...	0x2222222222...	MiWi Command : Association Response
3792	+ 1246 us	63		0xffffffffffff	Acknowledgment; Sequence Number - 0xa2

Ilustración 45: Formación de una red con el protocolo MiWi Mesh

Command Payload
- Frame Type: MiWi Command Frame
- Command Packet Type: Association Response
- Assigned Short Address: 0x0100
- Association Status: 0x00 - Association Successful

Ilustración 47: Asignación del SA al dispositivo COR

Command Payload
- Frame Type: MiWi Command Frame
- Command Packet Type: Association Response
- Assigned Short Address: 0x0181
- Association Status: 0x00 - Association Successful

Ilustración 46: Asignación del SA al dispositivo ED

7.1.3 MiWi Pro

En el protocolo de pila MiWi Pro el proceso de *HandShaking* es el mismo que el del protocolo de pila MiWi Mesh. La única variación es que tras haberse conectado dos coordinadores, en este caso el dispositivo COR y el dispositivo PCOR, se intercambian las tablas de enrutamiento y el árbol de familia.

40	+ 199479 us	58	0x1111111111...	0xffffffffffffffff	MiWi Command : Beacon Request
41	+ 22219 us	69	0x0000000000...	0xffffffffffffffff	Beacon
42	+ 176127 us	59	0x1111111111...	0x0000000000...	MiWi Command : Association Request
43	+ 1198 us	69		0xffffffffffffffff	Acknowledgment; Sequence Number - 0x40
44	+ 3165 us	68	0x0000000000...	0x1111111111...	MiWi Command : Association Response
45	+ 1194 us	58		0xffffffffffffffff	Acknowledgment; Sequence Number - 0x25
46	+ 201583 us	60	0x1111111111...	0xffffffffffffffff	Data; Sequence Number - 0x41
47	+ 20821 us	69	0x0000000000...	0xffffffffffffffff	Data; Sequence Number - 0x26
48	+ 201512 us	59	0x1111111111...	0xffffffffffffffff	Data; Sequence Number - 0x42
49	+ 7283 us	69	0x0000000000...	0xffffffffffffffff	Data; Sequence Number - 0x27
50	+ 1508398 ...	60	0x2222222222...	0xffffffffffffffff	MiWi Command : Beacon Request
51	+ 5713 us	54	0x1111111111...	0xffffffffffffffff	Beacon
52	+ 192026 us	60	0x2222222222...	0x1111111111...	MiWi Command : Association Request
53	+ 1247 us	54		0xffffffffffffffff	Acknowledgment; Sequence Number - 0x63
54	+ 3164 us	54	0x1111111111...	0x2222222222...	MiWi Command : Association Response
55	+ 1237 us	60		0xffffffffffffffff	Acknowledgment; Sequence Number - 0x44

Ilustración 48: Creación de la red con el protocolo MiWi PRO

Los dos primeros mensajes de datos son informes del árbol de familia y los dos siguientes son informes de la tabla de enrutamiento (véase las *ilustraciones 30 y 31*).

Report Type: 0x00
Report ID: 0xa3

Ilustración 50: Identificador del informe de árbol de familia

Report Type: 0x00
Report ID: 0xa4

Ilustración 49: Identificador de tabla de enrutamiento

7.2 Intercambio de mensajes

Una vez creada la red seguiremos programando nuestra aplicación. Para ello se tendrán que recibir y enviar mensajes. En primer lugar hay que configurar el buffer de transmisión o escritura y el buffer de recepción o lectura. Esta configuración se hace en el archivo de configuración *config_MiApp.h*. Hay que prestar cuidadosa atención porque los *transceivers* tienen su límite de buffer. Los buffers del *transceiver* que se ha utilizado tienen un tamaño de 64 bytes, por lo tanto, el buffer de aplicación tiene que ser más pequeño que el citado sin olvidar que las cabeceras también suman en esos 64 bytes.

Una vez configurados los buffers el siguiente paso es consultar repetitivamente si existe un mensaje disponible para ser recibido. Este paso, como se ha mencionado en el punto anterior, es obligatorio a la hora de formar la red porque tiene que recibir las peticiones.

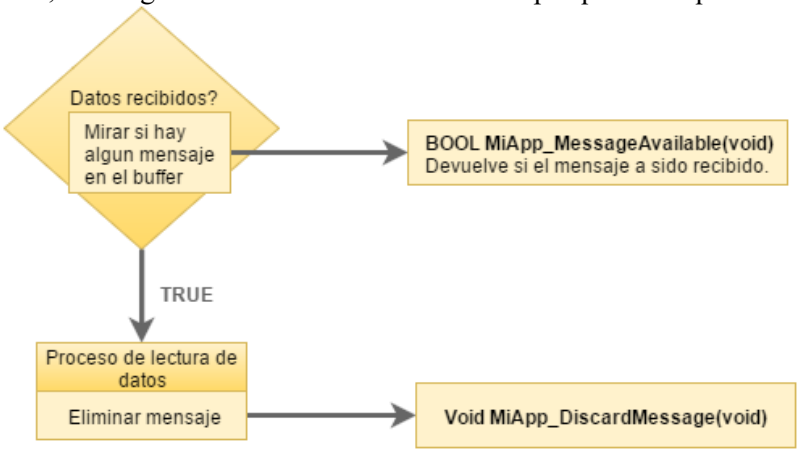


Ilustración 51: Gestión de mensajes recibidos

La aplicación estará en un bucle infinito comprobando si existe algún mensaje. Si existe, esté será recibido y procesado. Una vez finalizado esto, se descartará el dato para realizar las tareas indispensables para preparar el buffer para el próximo mensaje.

Si no hay datos a recibir se consultara si se necesita transmitir datos, de ser así se implementara la secuencia según aparece en la *ilustración 52*.

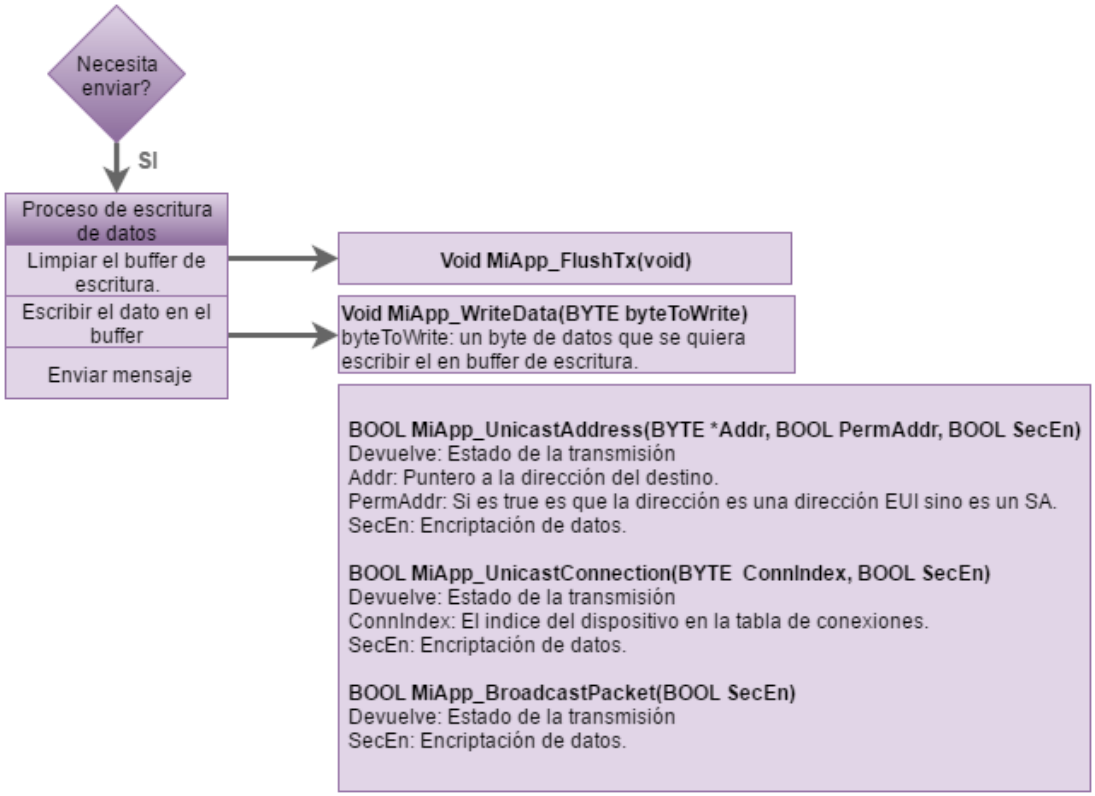


Ilustración 52: Envió de un mensaje

Lo primero que se hace para enviar un mensaje es una limpieza del buffer de escritura o de transmisión, esto reinicia el índice del vector donde se almacenan los datos.

Luego se escriben los bytes a enviar, un byte cada vez que se llame a la instrucción. Cada vez que se escribe un byte en el buffer el índice se incrementa para escribir el siguiente dato sin sobrescribir este. Este proceso se repite hasta completar él envió total de la información.

Por último se envía el mensaje. Se puede enviar un mensaje broadcast, un mensaje multicast o un mensaje unicast. Los tres tipos de mensaje pueden ir encriptados o no.

7.2.1 MiWi P2P

En el protocolo MiWi P2P no es posible enrutar mensajes por este motivo solamente se podrá probar el envío de mensajes desde el dispositivo PCOR al dispositivo COR (#272) y desde el dispositivo ED al dispositivo COR (#274).

272	+ 35225 us	60	0x2222222222...	0x1111111111...	*Data; Sequence Number - 0xb9*
273	+ 1249 us	40		0xffffffffffffffff	*Acknowledgment; Sequence Number - 0xb9*
274	+ 13342 us	40	0x1111111111...	0x2222222222...	*Data; Sequence Number - 0xb9*
275	+ 1200 us	60		0xffffffffffffffff	*Acknowledgment; Sequence Number - 0xb9*

Ilustración 53: Intercambio de mensajes entre el dispositivo COR y el dispositivo ED

265	+ 35188 us	69	0x0000000000...	0x1111111111...	Data; Sequence Number - 0x85
266	+ 1203 us	62		0xffffffffffff	Acknowledgment; Sequence Number - 0x85
267	+ 13351 us	62	0x1111111111...	0x0000000000...	Data; Sequence Number - 0xb7
268	+ 1252 us	69		0xffffffffffff	Acknowledgment; Sequence Number - 0xb7

Ilustración 54: Intercambio de mensajes entre el dispositivo COR y el dispositivo PCOR

7.2.2 MiWi Mesh y MiWi Pro

En el Protocolo MiWi Mesh y MiWi Pro el intercambio de mensajes es el mismo. Además como ambos protocolos pueden enrutar se hará la prueba para comprobar que el dispositivo ED puede intercambiar mensajes con el dispositivo PCOR. El flujo de mensajes entre PCOR y COR es el mismo que el de la *ilustración 54* y entre COR y ED es el mismo que el de la *ilustración 53*.

En la *ilustración 55* se puede observar el flujo de mensajes que intercambian los tres dispositivos para transmitir un mensaje desde el dispositivo ED hasta el dispositivo PCOR (#4939 y #4941) y devolver el mismo mensaje desde el dispositivo PCOR hasta el dispositivo ED (#4943 y #4945).

4939	+ 987343 us	54	0x22222222222...	0x11111111111...	Data; Sequence Number - 0xcf
4940	+ 1212 us	53		0xffffffffffff	Acknowledgment; Sequence Number - 0xcf
4941	+ 3808 us	53	0x11111111111...	0x00000000000...	Data; Sequence Number - 0xe2
4942	+ 1207 us	69		0xffffffffffff	Acknowledgment; Sequence Number - 0xe2
4943	+ 11902 us	69	0x00000000000...	0x11111111111...	Data; Sequence Number - 0x8f
4944	+ 1203 us	52		0xffffffffffff	Acknowledgment; Sequence Number - 0x8f
4945	+ 3762 us	52	0x11111111111...	0x22222222222...	Data; Sequence Number - 0xe3
4946	+ 1189 us	54		0xffffffffffff	Acknowledgment; Sequence Number - 0xe3

Ilustración 55: Intercambio de mensajes entre PCOR y ED

En el protocolo MiWi Mesh y MiWi Pro, como hay conexiones indirectas (*Sockets*), se puede activar el ACK de la capa de aplicación. Este ACK será enrutada hasta el origen del mensaje recibido dando los saltos que haga falta. En cambio, los ACK de la capa MAC solo podrán hacer un salto.

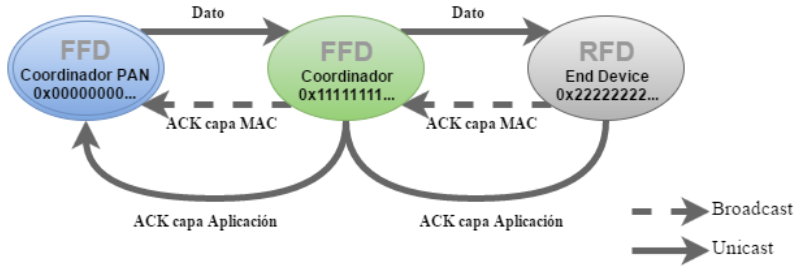


Ilustración 56: ACK de la capa de aplicación y ACK de la capa MAC

Si activamos el ACK de la capa de aplicación (en el archivo de configuración *config_protocol.h*) el flujo de mensajes será diferente (véase *ilustración 58*). Inicialmente el dispositivo ED enviará un mensaje (#21) y al llegar éste al dispositivo COR será respondido con un ACK de la capa MAC (#22). A continuación como este dispositivo no es el destino, éste enrutará el mensaje (#23). El dispositivo PCOR le responderá con un mensaje ACK de la capa MAC (#24). El dispositivo PCOR analizará el mensaje y verá que es para él, por este motivo no tendrá que enrutar. Si el ACK de la capa de aplicación está activado se dará cuenta que necesita mandar un ACK de la capa de aplicación y enviara este ACK (#25). Para saber

que este mensaje es el ACK de la capa de aplicación habrá que fijarse en el identificador del informe. El informe 0x30 es el ACK de la capa de aplicación.

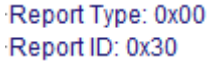


Ilustración 57: El report del mensaje ACK de la capa de aplicación

El dispositivo COR al recibir el citado mensaje enviará un ACK de la capa MAC (#26) y enrutará) este ACK de la capa de aplicación al destino (#27). El dispositivo ED al recibir el mensaje enviará un ACK de la capa MAC (#28) y dará por concluido el envío del mensaje.

En este ejemplo PCOR al recibir un mensaje envía el mismo mensaje al origen del mensaje recibido, por tanto después se repite el flujo de los mensajes pero en este caso enviando el mensaje de datos desde el dispositivo PCOR al dispositivo ED y respondiendo con el ACK de la capa de aplicación desde el dispositivo ED al dispositivo PCOR.

21	+ 54697 us	69	0x22222222222... 0x11111111111...	Data; Sequence Number - 0xba
22	+ 1194 us	65	0xffffffffffff	Acknowledgment; Sequence Number - 0xba
23	+ 3808 us	66	0x11111111111... 0x00000000000...	Data; Sequence Number - 0xf1
24	+ 1207 us	48	0xffffffffffff	Acknowledgment; Sequence Number - 0xf1
25	+ 3810 us	48	0x00000000000... 0x11111111111...	Data; Sequence Number - 0x74
26	+ 1184 us	66	0xffffffffffff	Acknowledgment; Sequence Number - 0x74
27	+ 3809 us	65	0x11111111111... 0x22222222222...	Data; Sequence Number - 0xf2
28	+ 1208 us	69	0xffffffffffff	Acknowledgment; Sequence Number - 0xf2
29	+ 6838 us	48	0x00000000000... 0x11111111111...	Data; Sequence Number - 0x75
30	+ 1194 us	65	0xffffffffffff	Acknowledgment; Sequence Number - 0x75
31	+ 3808 us	66	0x11111111111... 0x22222222222...	Data; Sequence Number - 0xf3
32	+ 1179 us	68	0xffffffffffff	Acknowledgment; Sequence Number - 0xf3
33	+ 974440 us	63	0x22222222222... 0x11111111111...	Data; Sequence Number - 0xbb
34	+ 1194 us	60	0xffffffffffff	Acknowledgment; Sequence Number - 0xbb
35	+ 3772 us	60	0x11111111111... 0x00000000000...	Data; Sequence Number - 0xf4
36	+ 1191 us	38	0xffffffffffff	Acknowledgment; Sequence Number - 0xf4

Ilustración 58: Intercambio de mensajes entre PCOR y ED con el ACK de la capa de aplicación habilitada

7.2.3 Mensajes broadcast

El protocolo también ofrece la opción de enviar mensajes broadcast. Un mensaje broadcast no está dirigido a nadie por esta razón no se necesitara el ACK ni de la capa MAC ni de la capa de aplicación.

Los coordinadores que reciban un mensaje broadcast tendrán que volver a enviar para que les llegue a sus hijos.

3374	+ 585667 us	66	0x2222222222...	0xffffffffffff	Data; Sequence Number - 0xe6
3375	+ 4750 us	54	0x0000000000...	0xffffffffffff	Data; Sequence Number - 0x22
3376	+ 4862 us	64	0x1111111111...	0xffffffffffff	Data; Sequence Number - 0x1a

Ilustración 59: Envió de un mensaje broadcast y su difusión

7.2.4 Mensajes multicast

Solamente el protocolo MiWi Pro tiene opción de enviar mensajes multicast. Estos mensajes van dirigidos a un grupo concreto de dispositivos. En el protocolo MiWi Pro solamente existen dos grupos de dispositivos a las que se le pueden enviar mensajes multicast, estos son los siguientes:

- Los coordinadores: este grupo lo forman todos los coordinadores de la red. Tienen la SA reservada 0xFFFE.

- Los dispositivos FFD: este grupo lo forman todos los dispositivos FFD de la red. Tienen la SA reservada 0xFFFD.

Para enviar un mensaje multicast se utiliza la función *MiApp_UnicastAddress()* pero en vez de introducir una dirección normal se debe introducir una de las direcciones reservadas para multicast.

80	+ 3128 us	63	0x0000000000...	0xffffffffffffffff	Data; Sequence Number - 0x12
81	+ 5259 us	64	0x2222222222...	0xffffffffffffffff	Data; Sequence Number - 0xd7
82	+ 54637 us	39	0x1111111111...	0xffffffffffffffff	Data; Sequence Number - 0xaa

Ilustración 60: Envió de un mensaje multicast y su difusión

Estos mensajes son iguales que un mensaje broadcast, la única diferencia que hay es la SA de destino. El broadcast tiene 0xffff mientras que este tiene 0xfffd (véase la *ilustración 61*).

Destination Short Address: 0xfffd

Ilustración 61: Dirección del destino del mensaje multicast

7.2.5 Mensajes encriptados

Para poder enviar mensajes encriptados hay que configurar la seguridad del protocolo. El primer paso es habilitar y eso se hace en el archivo de configuración *config_MiApp.h*. Una vez que este habilitada la seguridad hay que configurar los parámetros de seguridad. Estos parámetros se sitúan en el archivo de configuración *config_89xa.h.* Hay que tener cuidado porque al activar el modo de seguridad disminuye el tamaño máximo del buffer de la aplicación.

Los parámetros a modificar son la clave de seguridad, el identificador de la clave de seguridad y el modo de seguridad.

Una vez hechas las configuraciones, en la aplicación a la hora de enviar el mensaje se le pasara como parámetro de *SecEn* true.

3359	+ 596132 us	69	0x2222222222...	0x1111111111...	Data; Encrypted Payload; Sequence Number - 0xdf
3360	+ 1187 us	65		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xdf

Ilustración 62: Mensaje unicast encriptado

3384	+ 596025 us	68	0x2222222222...	0xffffffffffffffff	Data; Encrypted Payload; Sequence Number - 0xea
3385	+ 3937 us	54	0x0000000000...	0xffffffffffffffff	Data; Encrypted Payload; Sequence Number - 0x26
3386	+ 5845 us	65	0x1111111111...	0xffffffffffffffff	Data; Encrypted Payload; Sequence Number - 0x1e

Ilustración 63: Mensaje broadcast encriptado

7.2.6 Política de retransmisiones

Un dispositivo al enviar un mensaje, si este mensaje falla por no recibir el ACK de la capa MAC a tiempo, no dejara el error directamente en manos de la aplicación, intentara retransmitir unas cuantas veces antes de pasar el error a la capa de aplicación. Cuantas veces se tiene que retransmitir el paquete define el parámetro *RETRANSMISSION_TIMES* del archivo de configuración *config_89xa.h.* Hay que tener en cuenta que para que se pueda retransmitir también hay que habilitar las retransmisiones (*ENABLE_RETRANSMISSION*) en el mismo archivo de configuración.

7.3 Comunicación entre dos dispositivos al desactivar el coordinador PAN

El dispositivo PCOR en la comunicación entre el dispositivo COR y el dispositivo ED no tiene nada que ver. Él solo crea la red y luego ya son los otros dos dispositivos los que se comunican entre ellos. Si en un momento dado el dispositivo PCOR se apaga la red debería seguir funcionando igual y, la comunicación entre el dispositivo COR y el dispositivo ED debería de continuar.

Para realizar esta prueba se ha utilizado la misma aplicación que en el punto anterior y el intercambio de mensajes ha sido entre COR y ED. En un momento dado se ha apagado el dispositivo PCOR y se ha visto que el flujo de los mensajes es el mismo que la de *ilustración* 53.

7.4 Comprobar la comunicación con coordinador PAN al reactivarse

Para que un dispositivo, después de haberse caído de la red, al volver a activarse se una a la red sin que haga el *HandShaking* necesita tener activada la característica *NetworkFreezer*. Esta característica se activa en el archivo de configuración *config_MiApp.h*. Si el protocolo escogido es el MiWi Pro, esta característica es obligatoria.

Una vez activado el *NetworkFreezer* hay que configurar bien el uso de la memoria no volátil en el archivo *miwi_nvm.h*. Para el desarrollo de este proyecto la memoria utilizada ha sido una memoria externa de 4KB.

Cuando se den por finalizadas todas las configuraciones en la implementación de la aplicación, en la parte de configurar la red, a la función *MiApp_protocolInit()* se le pasará como parámetro un *TRUE*. Si quisiéramos resetear la memoria le tendríamos que pasar *FALSE*.

Cuando esté recuperada he estado anterior de la red se mirará si la tabla de conexiones está vacía, si es así, se hará todo el proceso de establecer la red, ya que el dispositivo no ha estado conectado a ninguna red anteriormente. En caso de que no sea así se saltara directamente al bucle infinito.

En la *ilustración 64* se puede ver como en un momento dado el dispositivo PCOR cae y el dispositivo COR no consigue enviar el mensaje.

8397	+ 979240 us	60	0x22222222222...	0x11111111111...	Data; Sequence Number - 0xfc
8398	+ 1189 us	59		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xfc
8399	+ 3798 us	59	0x11111111111...	0x00000000000...	Data; Sequence Number - 0xa2
8400	+ 13543 us	58	0x11111111111...	0x00000000000...	Data; Sequence Number - 0xa2
8401	+ 13515 us	59	0x11111111111...	0x00000000000...	Data; Sequence Number - 0xa2
8402	+ 13518 us	58	0x11111111111...	0x00000000000...	Data; Sequence Number - 0xa2
8403	+ 13460 us	59	0x11111111111...	0x00000000000...	Data; Sequence Number - 0xa2
8404	+ 13524 us	58	0x11111111111...	0x00000000000...	Data; Sequence Number - 0xa2
8405	+ 13517 us	59	0x11111111111...	0x00000000000...	Data; Sequence Number - 0xa2
8406	+ 13517 us	59	0x11111111111...	0x00000000000...	Data; Sequence Number - 0xa2
8407	+ 40533 us	59	0x11111111111...	0x00000000000...	Data; Sequence Number - 0xa2

Ilustración 64: El dispositivo PCOR desaparece

Después de un momento dado el dispositivo PCOR reaparecerá y sin hacer el *HandShaking* se unirá a la red (véase la *ilustración 65).*

8441	+ 13516 us	59	0x11111111111... 0x00000000000...	Data; Sequence Number - 0xa2
8442	+ 13500 us	60	0x11111111111... 0x00000000000...	Data; Sequence Number - 0xa2
8443	+ 13511 us	63	0x11111111111... 0x00000000000...	Data; Sequence Number - 0xa2
8444	+ 1282 us	69	0xffffffffffff	Acknowledgment; Sequence Number - 0xa2
8445	+ 262188 us	69	0x00000000000... 0x11111111111...	Data; Sequence Number - 0x7c
8446	+ 1128 us	56	0xffffffffffff	Acknowledgment; Sequence Number - 0x7c
8447	+ 3805 us	56	0x11111111111... 0x22222222222...	Data; Sequence Number - 0xa3
8448	+ 1193 us	58	0xffffffffffff	Acknowledgment; Sequence Number - 0xa3

Ilustración 65: El dispositivo PCOR reaparece

7.5 Al desactivar el padre asociarse a otro

Cuando no hay ningún mensaje recibido y tampoco se tiene que enviar ningún dato, se puede mirar si el dispositivo sigue conectado. Si sigue conectado volverá a mirar si hay algún mensaje en el buffer de mensajes recibidos pero si no sigue conectado se tendrá que volver a conectar.

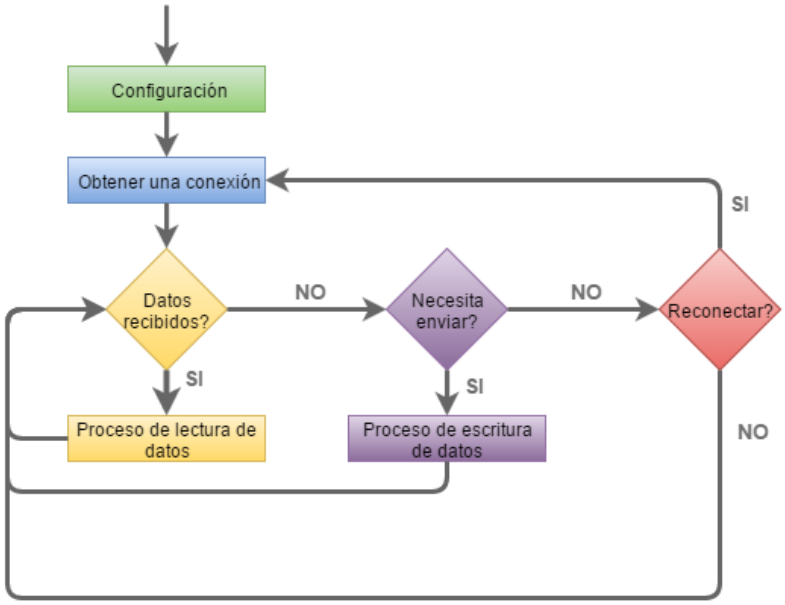


Ilustración 66: Diagrama de flujo con la reconexión

Para ver si sigue conectado no hay ninguna opción que no sea enviar un mensaje al padre y ver si lo recibe. Si cada vez que no haya recibido ningún mensaje envía un mensaje a su padre, se puede comprobar con esos mensajes enviados. Si no se deberá enviar un mensaje con un dato aleatorio y ver si le llega bien.

7.5.1 MiWi P2P

En la *ilustración 67* se puede observar como en un momento dado el dispositivo COR (que es el padre del dispositivo ED) se desconecta. Tras intentarlo unas cuantas veces pasará el error de envío a la aplicación del dispositivo ED. El dispositivo ED al recibir el error del envío intentara asociarse a otro coordinador.

20	+ 997338 us	59	0x2222222222...	0x1111111111...	Data; Sequence Number - 0xbd
21	+ 1240 us	54		0xffffffffffff	Acknowledgment; Sequence Number - 0xbd
22	+ 13352 us	52	0x1111111111...	0x2222222222...	Data; Sequence Number - 0xa8
23	+ 1161 us	59		0xffffffffffff	Acknowledgment; Sequence Number - 0xa8
24	+ 997371 us	59	0x2222222222...	0x1111111111...	Data; Sequence Number - 0xbe
25	+ 12592 us	59	0x2222222222...	0x1111111111...	Data; Sequence Number - 0xbe
26	+ 12626 us	59	0x2222222222...	0x1111111111...	Data; Sequence Number - 0xbe

Ilustración 67: Se ha desconectado el dispositivo COR

276	+ 12604 us	59	0x2222222222...	0x1111111111...	Data; Sequence Number - 0xbe
277	+ 12638 us	59	0x2222222222...	0x1111111111...	Data; Sequence Number - 0xbe
278	+ 12599 us	59	0x2222222222...	0x1111111111...	Data; Sequence Number - 0xbe
279	+ 20002 us	59	0x2222222222...	0xffffffffffff	MiWi P2P Command : P2P Connection Request
280	+ 13048 us	67	0x0000000000...	0x2222222222...	MiWi P2P Command : P2P Connection Response
281	+ 1196 us	59		0xffffffffffff	Acknowledgment; Sequence Number - 0x60

Ilustración 68: El dispositivo ED se asocia al dispositivo PCOR

7.5.2 MiWi Mesh y MiWi Pro

Con los protocolos MiWi Mesh y MiWi Pro pasa exactamente lo mismo (véase las *ilustraciones 69 y 70*).

2570	+ 987548 us	55	0x22222222222...	0x11111111111...	Data; Sequence Number - 0xbe
2571	+ 1138 us	58		0xffffffffffff	Acknowledgment; Sequence Number - 0xbe
2572	+ 3808 us	58	0x11111111111...	0x00000000000...	Data; Sequence Number - 0xc5
2573	+ 1187 us	69		0xffffffffffff	Acknowledgment; Sequence Number - 0xc5
2574	+ 11853 us	69	0x00000000000...	0x11111111111...	Data; Sequence Number - 0x80
2575	+ 1198 us	57		0xffffffffffff	Acknowledgment; Sequence Number - 0x80
2576	+ 3807 us	57	0x11111111111...	0x22222222222...	Data; Sequence Number - 0xc6
2577	+ 1186 us	55		0xffffffffffff	Acknowledgment; Sequence Number - 0xc6
2578	+ 987553 us	56	0x22222222222...	0x11111111111...	Data; Sequence Number - 0xbf
2579	+ 13510 us	56	0x22222222222...	0x11111111111...	Data; Sequence Number - 0xbf
2580	+ 13539 us	56	0x22222222222...	0x11111111111...	Data; Sequence Number - 0xbf
2581	+ 13519 us	56	0x22222222222...	0x11111111111...	Data; Sequence Number - 0xbf

Ilustración 69: Se ha desconectado el dispositivo COR

2830	+ 13529 us	56	0x22222222222...	0x11111111111...	Data; Sequence Number - 0xbf
2831	+ 13522 us	56	0x22222222222...	0x11111111111...	Data; Sequence Number - 0xbf
2832	+ 13522 us	55	0x22222222222...	0x11111111111...	Data; Sequence Number - 0xbf
2833	+ 19766 us	56	0x22222222222...	0xffffffffffff	MiWi Command : Beacon Request
2834	+ 5091 us	69	0x00000000000...	0xffffffffffff	Beacon
2835	+ 192630 us	56	0x22222222222...	0x00000000000...	MiWi Command : Association Request
2836	+ 1298 us	69		0xffffffffffff	Acknowledgment; Sequence Number - 0xc1
2837	+ 4574 us	69	0x00000000000...	0x22222222222...	MiWi Command : Association Response
2838	+ 1149 us	55		0xffffffffffff	Acknowledgment; Sequence Number - 0x82
2839	+ 9849 us	55	0x22222222222...	0x00000000000...	Data; Sequence Number - 0xc2
2840	+ 1202 us	69		0xffffffffffff	Acknowledgment; Sequence Number - 0xc2
2841	+ 11901 us	69	0x00000000000...	0x22222222222...	Data; Sequence Number - 0x83
2842	+ 1212 us	56		0xffffffffffff	Acknowledgment; Sequence Number - 0x83
2843	+ 984283 us	57	0x22222222222...	0x00000000000...	Data; Sequence Number - 0xc3
2844	+ 1277 us	69		0xffffffffffff	Acknowledgment; Sequence Number - 0xc3
2845	+ 11828 us	69	0x00000000000...	0x22222222222...	Data; Sequence Number - 0x84
2846	+ 1183 us	57		0xffffffffffff	Acknowledgment; Sequence Number - 0x84

Ilustración 70: El dispositivo ED se asocia al dispositivo PCOR

7.6 Configurar un minimo de cobertura para la comunicación

Cuando se quiera instalar la red hay que posicionar los dispositivos. A la hora de realizar esta prueba los dispositivos se posicionaran de esta manera: el dispositivo PCOR estará al lado del ordenador, el dispositivo ED estará lejos y el dispositivo COR lo colocaremos entre éstos.

Al hacer este tipo de redes a veces suele pasar que el dispositivo PCOR esté al alcance del dispositivo ED pero con muy poca cobertura. Como hay poca cobertura es probable que la comunicación entre ellos sea mala por ello es más conveniente que se conecten a través del dispositivo COR.

Para hacer posible eso, el protocolo MiWi Pro define el parámetro *COMM_RSSI_THRESHOLD* en el archivo de configuración *config_protocol.h*. Este parámetro es un número entre 0 y 70 y establece el mínimo de cobertura que exige un dispositivo para responder a los mensajes. Solamente está disponible para el protocolo MiWi Pro.

Por tanto si aumentamos este parámetro en el dispositivo PCOR no responderá a los beacon request que envía el dispositivo ED.

Para hacer esta prueba se le ha puesto el valor 40 al parámetro *COMM_RSSI_THRESHOL* del dispositivo PCOR. Esto quiere decir que todos los mensajes que llegan al dispositivo PCOR serán rechazados si tienen una cobertura menor que 40.

7.7 Almacenamiento de mensajes indirectos dirigidos a un dispositivo RFD

Cuando un dispositivo RFD esté dormido, todos los mensajes dirigidos hacia éste dispositivo tienen que ser bufferizados por el padre de este. El protocolo MiWi™ a estos mensajes le llama mensajes indirectos (i*ndirect messages*) y para que se almacenen y se envíen correctamente una vez que haya despertado el dispositivo RFD hay que definir una serie de parámetros.

El protocolo MiWi™ en realidad solo envía un mensaje almacenado por cada mensaje *data request*. Esto aumenta el tráfico y tiempo, por ello, en este proyecto se ha mejorado la gestión de dichos mensajes para que cuando el dispositivo COR reciba el mensaje *data request* envié todos los mensajes almacenados a la vez. Esta prueba se ha realizado con la mejora mencionada incorporada que se explica en el *punto 6*.

Los parámetros a definir son los siguientes:

En el dispositivo RFD se tiene que activar la característica de dormir. Este dispositivo al despertar enviara un mensaje *data request* y recibirá todos los mensajes indirectos a la vez, por este motivo el buffer de mensajes recibidos de la capa MAC tiene que ser mínimo del mismo tamaño del buffer de los mensajes indirectos que puede guardar su padre. Esto se configurará con el parámetro *BANK_SIZE* del archivo de configuración *config_89xa.h*. Una vez que hayamos configurado este dispositivo pasaremos a implementar la aplicación.

La aplicación será como la descrita en la *ilustración 71*. El dispositivo mirara si hay mensajes recibidos, de ser así procesara dichos mensajes. De lo contrario pasara a mirar si necesita enviar algún dato. Si tuviera que enviar, haría todo el proceso de transmisión de mensaje y después pasaría al proceso de dormirse. Si no tuviera que enviar ningún mensaje pasaría directamente al proceso de dormirse. Al despertar mandaría un mensaje *data request* y volvería a mirar si tiene algún mensaje recibido.

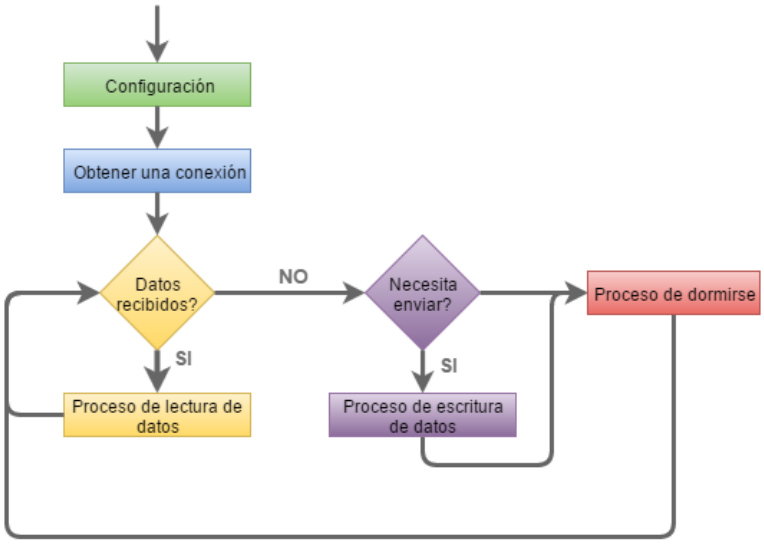


Ilustración 71: Diagrama de flujo de la aplicación de un dispositivo RFD

El proceso de dormirse es muy sencillo. Primero apaga el *transceiver,* una vez que este apagada espera el tiempo acordado y después enciende el transceiver y envía un mensaje de petición de datos.

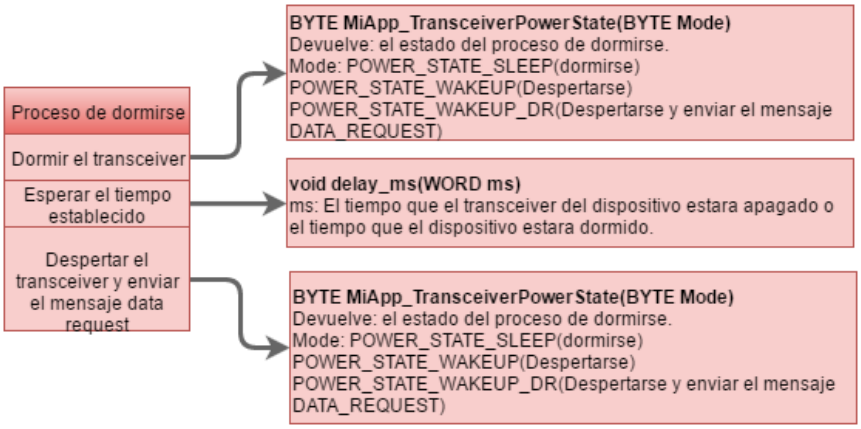


Ilustración 72: El proceso de dormirse un dispositivo RFD

Una vez terminado con el dispositivo RFD hay que configurar su padre. En su padre solamente se cambiara la configuración, la aplicación se dejara igual.

En este dispositivo se activaran los mensajes indirectos (*ENABLE_INDIRECT_MESSAGES*) y cada cuanto tiempo se despertara su hijo (*RFD_WAKEUP_INTERVAL*). Esto se hace en el archivo de configuración *config_MiApp.h.* Después de hacer esas configuraciones habrá que configurar el protocolo. En la configuración del protocolo se tendrá que establecer el tamaño del buffer de los mensajes indirectos (*INDIRECT_MESSAGES_SIZE*). Si el ACK de la capa de aplicación o de la capa MAC

estuviera activado también se tendrá que prestar atención al parámetro *BANK_SIZE* en el archivo de configuración *config_89xa.h*.

Después de hacer todas las configuraciones pasamos a hacer la prueba. En la prueba se verá como primero el dispositivo ED envía un mensaje *data request* a su padre (#1995), en este caso al dispositivo COR. Este al recibir, envía los 4 mensajes que tenía almacenadas en el buffer de mensajes indirectos (#1997, #1999, #2001, #2003). El dispositivo ED después de recibir todos estos mensajes y gestionarlos apagara el transceiver para ahorrar batería y después de un rato volverá a encender y enviara otro mensaje *data request* (#2005). Si al recibir dicho mensaje el buffer de mensajes indirectos está vacía le enviara solamente un mensaje vacío (#2007).

1995	+ 1605201 ...	54	0x2222222222...	0x1111111111...	Command : Data Request
1996	+ 1178 us	60		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xbd
1997	+ 3795 us	60	0x1111111111...	0x2222222222...	Data; Sequence Number - 0xc4
1998	+ 1232 us	54		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xc4
1999	+ 3799 us	60	0x1111111111...	0x2222222222...	Data; Sequence Number - 0xc5
2000	+ 1191 us	54		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xc5
2001	+ 3761 us	60	0x1111111111...	0x2222222222...	Data; Sequence Number - 0xc6
2002	+ 1194 us	54		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xc6
2003	+ 2659 us	60	0x1111111111...	0x2222222222...	Data; Sequence Number - 0xc7
2004	+ 1236 us	54		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xc7
2005	+ 1608052 ...	54	0x2222222222...	0x1111111111...	Command : Data Request
2006	+ 1218 us	60		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xbe
2007	+ 2661 us	60	0x1111111111...	0x2222222222...	Data; Sequence Number - 0xc8
2008	+ 1228 us	54		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xc8
2009	+ 1606366 ...	55	0x2222222222...	0x1111111111...	Command : Data Request
2010	+ 1240 us	60		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xbf

Ilustración 73: Mensajes indirectos

7.8 Desconectarse de la red

Si un dispositivo (no hay restricción de rol) en un momento dado quiere desconectar una conexión puede hacerlo mediante la function *MiApp_RemoveConexion()*.Si a esta función le pasamos el parámetro *0xFF* desconectara todas las conexiones y por lo tanto se desconectara de la red en la que estaba.

7.8.1 MiWi P2P

En el protocolo de pila MiWi P2P se intercambian dos mensajes (véase la *ilustración 74*). Primero, un dispositivo envía una petición de eliminar la conexión (#12). Después el dispositivo que reciba dicho mensaje le enviara la respuesta (#14)

12	+ 35085 us	54	0x22222222222...	0x11111111111...	MiWi P2P Command : P2P Connection Removal Request
13	+ 1205 us	69		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xbb
14	+ 2835 us	69	0x11111111111...	0x22222222222...	MiWi P2P Command : P2P Connection Removal Response
15	+ 1168 us	54		0xffffffffffffffff	Acknowledgment; Sequence Number - 0xc1

Ilustración 74: Eliminar conexión en el protocolo de pila P2P

7.8.2 MiWi Mesh y MiWi Pro

La misma operación se hace con un solo mensaje en los protocolo de pila MiWi Mesh y MiWi Pro. El dispositivo que quiera desconectarse envía una notificación de que se va a desconectar (#83) y espera al ACK de la capa MAC. El dispositivo que recibe la notificación envía el ACK

de la capa MAC (#84) y borra dicho dispositivo de la tabla de conexiones. El dispositivo que inicio la desconexión al recibir el ACK borra la conexión.

Ilustración 75: Eliminar conexión en los protocolo de pila Mesh y Pro

8

PROBLEMAS ENCONTRADOS Y SOLUCIONDOS

Al llevar a cabo el proyecto, se ha tenido que afrontar varios problemas. Algunos de ellos han sido por culpa del software, otros por culpa de hardware y otros por errores humanos. En este apartado se explican los problemas que han ido surgiendo durante el transcurso del proyecto, la metodología que se ha utilizado para afrontar dichos problemas y las soluciones que se han encontrado para cada uno de ellos.

8.1 Error del linker

Al principio todos los archivos de cabecera se ubicaban en la carpeta *header files* de MPLAB y los archivos fuente en la carpeta *source files*. Pero, con esta organización, daba error del linker al compilar ya que el compilador CCS no admite esa distribución. Con motivo de solucionar el problema se movió el archivo principal (*main.c*) a la carpeta *source files* y todos los de más archivos a *important files* dando la estructura explicada en el *punto 5.1*.

8.2 Controladores de los USB

8.2.1 ZENA

El principal problema que se encontró al utilizar ZENA fue que no se localizaba el controlador, por lo que su funcionamiento resultaba imposible. Además, una vez localizado al intentar instalarlo Windows no permitía la instalación por que el controlador no estaba firmado y Windows 10, por defecto, solo admite controladores firmados. Para solucionar el problema se siguió el proceso detallado a continuación:

1. Reiniciar el ordenador de forma avanzada
   Inicio / Configuración / Actualización y seguridad / Recuperación / Reiniciar ahora
2. Desactivar la opción de que solo se puedan instalar controladores firmados.
   Solución de problemas / Opciones avanzadas / Configuración de inicio / Deshabilitar el uso obligatorio de controladores firmados
3. Reiniciar.
4. Instalar el controlador.
   Configuración / Dispositivos / Dispositivos conectados / Administrador de dispositivos
5. Encontrar el dispositivo y dar al botón derecho.
   Actualizar software del controlador / Buscar software del controlador en el equipo
6. Poner la ubicación del controlador e instalar.

8.2.2 USB to Serial Bridge

Al iniciar el software XTerm no detectaba el puerto en el que se hallaba éste USB, y esto se debía a una incorrecta instalación del controlador. El problema residía en que el controlador tiene una versión por defecto del 2015 que no funciona y se debe instalar el de 2008. Para ello se siguió el proceso detallado a continuación:

Configuración / Dispositivos / Dispositivos conectados / Administrador de dispositivos

En este punto hay que localizar Prolific USB-to-Serial Comm Port y dar al botón derecho.

Actualizar software del controlador / Buscar software del controlador en el equipo / elegir en una lista de controladores de dispositivo en el equipo

En este punto ya se puede seleccionar la versión de 2008.

8.3 No se veía nada en WDS

Una vez conseguida la correcta compilación con el compilador CCS, se empezó con las pruebas. Para hacer las pruebas lo primero que se llevó a cabo fue la apertura del software WDS y configurar para capturar todas las tramas que emiten los dispositivos.

Al ejecutar las pruebas el sniffer no capturaba ninguna trama y se empezó a diagnosticar la causa de este problema. Se supuso que el error podría estar causado por una de estas dos opciones:

- Las Interrupciones:

Al investigar si el error podía provenir de las interrupciones se comprobó que estas no saltaban. Tras contactar con motivo de dicho error a *Atelei Engineering SLA,* éstos recomendaron crear un nuevo proyecto y que en ese proyecto se configuraran de nuevo y correctamente las interrupciones. Una vez que las interrupciones funcionaran bien en este nuevo proyecto el tutor sugirió que fueran migradas al proyecto principal asegurando, así, que las interrupciones funcionan.

Se siguieron las recomendaciones del tutor de la empresa creando un nuevo proyecto y elaborando un programa para que saltara la interrupción al presionar un pulsador. El error radicaba en que en CCS es necesario, antes de la rutina de la interrupción, introducir *#INT_interrupcion* y esto no se estaba realizando.

Cuando se consiguió, se migró al programa principal asegurando, así, que las interrupciones no eran el foco del error. Se decidió ponerlo a prueba de nuevo pero esta vez tampoco emitían ninguna trama, por lo que se supuso que debía ser algún error en el dispositivo SPI.

- El dispositivo SPI:

Para ver si el problema era del dispositivo SPI se siguieron los mismos pasos que con las interrupciones. Al no conseguir ningún avance se contactó con *Atelei Engineering SLA* y se analizó el error con la ayuda de un analizador lógico comprobando si los cronogramas eran correctas.

Se constató que, efectivamente el cronograma salía mal indicando un error en la configuración. Se dedicó un gran periodo de tiempo indagando cual podría ser el fallo puesto que aparentemente todo parecía correcto. Finalmente el tutor de empresa detectó que el error residía en que los PINs asignados se hallaban intercambiados, es decir, a la línea SDI se le asignaba el PIN SDO y al revés.

Con esta información se cambiaron los PINs y se llevaron a cabo de nuevo las pruebas. Esta vez el error se había solucionado y el dispositivo ED mandaba el mensaje de petición de asociación.

8.4 No se veía nada en el terminal

A la hora de hacer las pruebas es muy conveniente imprimir la información en un terminal para poder debuggear. Con este cometido en el desarrollo de este proyecto se ha utilizado el UART.

Al probar no se podía ver nada en el terminal, por este motivo se analizó la configuración para encontrar el error.

Primeramente se aseguró que los PINs establecidos para la escritura y para la lectura estaban correctamente asignados. Después de realizar dicha comprobación se creó un proyecto nuevo y se comprobó, esta vez, en ese nuevo proyecto. Entonces se constató que con la misma configuración, sin haber cambiado nada, se podían imprimir mensajes en la pantalla; lo que confirmaba que la configuración era correcta.

Finalmente se verificó que el error era que el proyecto que estaba introduciendo en el dispositivo tenía configurado el UART pero no tenía ningún mensaje para imprimir en el terminal. En cambio, el otro proyecto abierto poseía información para imprimir en el terminal pero no tenía configurado el UART. Al no compilar este segundo proyecto no aparecía ningún error.

8.5 No se conectaban bien

Una vez se hubo conseguido el envío de mensajes y ver la información en el terminal para poder debuggear se prosiguió con la *prueba 1*. Cuando se analizó el resultado obtenido por el sniffer se constató que no se conectaban los dispositivos.

El dispositivo COR le enviaba un mensaje de petición de baliza (*beacon request*) al dispositivo PCOR. Este al recibir la petición le mandaba un ACK y la baliza (*beacon*). Después el dispositivo COR le respondía con un ACK y la petición de asociación (*association request*). Acto seguido, el dispositivo PCOR le respondía con un ACK y la respuesta de la asociación (*association response*).

Hasta este punto el flujo de mensajes era correcto. El error vino cuando el dispositivo COR no respondía al mensaje de respuesta de asociación con el respectivo ACK.

Analizando por el terminal, se comprobó que el dispositivo COR no recibía a tiempo el ACK de la petición de asociación. La causa del problema podía estar originada en:

- Que el ACK mandado por el dispositivo PCOR estuviera mal y, por este motivo, no detectara que es un mensaje ACK. Con el objetivo de asegurar que el mensaje era correcto se comprobaron en el software WDS todos los campos del mensaje (*Sequence number, frame type…*) y se verificó que el ACK enviado por el dispositivo PCOR era correcto. Con esta verificación se demostró que el error no era del dispositivo PCOR sino que era del dispositivo COR.
- Que el ACK llegara tarde al dispositivo COR. Al analizar si llegaba tarde el mensaje se comprobó que éste era la causa del error pero éste, a su vez, podría estar causado por varios errores:
    - Que las temporizaciones estuvieran mal y por este motivo contara mal el tiempo transcurrido desde el envío del mensaje de petición de asociación

provocando que salga del bucle de espera antes de tiempo. Se verificó que este no era el origen del error.

- Por último, solo quedaba la opción de que la interrupción no saltara correctamente. Al hacer varias pruebas se cotejó que después de salir de la función que manda el mensaje (*txPacket()*) acto seguido saltaba la interrupción. Tras analizar más profundamente se comprobó que las configuraciones de las interrupciones estaban elaboradas correctamente como se constató en el *problema 7.3* pero que el compilador desactivaba estas interrupciones en la función mencionada por las llamadas recursivas; ya que en la rutina de la interrupción volvía a llamar a la función de envío de mensajes para enviar el ACK de la capa MAC cuando hiciera falta.

La única solución, por tanto, era engañar al compilador y para ello se duplicó la función y se le asignó otro nombre (*txPacket2*). Seguidamente se modificó en la rutina de la interrupción el nombre a esa función y se le asignó el nombre de la copia.

8.6 Error al escribir o leer la EEPROM

Al activar la característica de guardar el estado de la red actual en la EEPROM y recuperar la misma al volver a iniciar el programa este procedimiento no se llevaba a cabo de forma correcta. No era posible saber si el error era por un motivo de escritura o de lectura errónea; por no tener ninguna variable que comprobara el estado de la EEPROM.

Como no era posible verificar cuál de las dos operaciones era la que la causante del mal funcionamiento en el programa se optó por comprobar ambas. Con este propósito se utilizó el analizador lógico.

Primeramente se constató que escribía de forma apropiada y, para ello, se verificó que la sucesión de pasos que seguía eran correctos: [13]

1. Tiene que leer el estado de la EEPROM.
2. Debe poner el estado en modo de escritura.
3. Escribe la dirección en la cual se escribirá la información que se quiera guardar.
4. Almacena la información que se quiera guardar.

Al ver que se daban todos los pasos correctamente se confirmó que se almacenaba bien la información, por tanto, el error no era de la escritura.

Consecuentemente, se pasó a constatar que la lectura era correcta y, por este motivo, y como se hizo anteriormente se verificó que todas las operaciones que hacia eran correctas:

1. Debe leer el estado de la EEPROM.
2. Pone el estado de la EEPROM en modo de escritura.
3. Escribe la dirección en la que esta guardada la información que se quiere leer.
4. Lee la información almacenada.

Al analizar el cronograma fue evidente que el primer paso no se daba y que, como resultado no leía bien la información almacenada.

Después de detectar y solucionar el problema de lectura y de escritura de la EEPROM se llevaron a cabo las pruebas. Se observó que a veces fallaba y cómo el fallo no podía ser causa de la EEPROM, por lo anteriormente expuesto, tenía que ser del *transceiver* ya que comparten la SPI.

Al hacer pruebas con varios *transceivers* se percibió que con unos funcionaba y con otros no y que sin el *transceiver* escribía y leía correctamente la EEPROM. Por la citada razón, el problema era del hardware de los *transceivers*.

9

MEDIDOR DE COBERTURA

A la hora de instalar los dispositivos para formar una red, es muy importante saber que cobertura hay entre los dispositivos. Entre dos dispositivos que vayan a formar la red tiene que haber suficiente cobertura para que el flujo de los mensajes sea correcto; por este motivo antes de colocar el dispositivo hay que verificar que existe suficiente cobertura entre dos dispositivos asociados.

Para la realización de esta aplicación se han utilizado dos placas *RSSI METTER 868MHz*. Una de las placas dispondrá de 8 leds y dos botones. La otra en cambio tendrá un led y un botón.

En el protocolo MiWi™ la cobertura se mide con el valor *Received Signal Strength Indicator* (en adelante RSSI). El RSSI es una escala de referencia para medir el nivel de potencia de las señales recibidas por un dispositivo en las redes inalámbricas. La escala de RSSI en el protocolo MiWi™ barema desde 0 a 70 siendo 70 la cobertura óptima.

La cobertura en este ejemplo se plasmara mediante 7 leds y en un solo dispositivo. Para ello la escala la dividiremos en 7:

- Si el valor RSSI entre dos dispositivos es entre 0-10, lo redondearemos a 10 y se encenderán los leds hasta ese valor. En este caso solamente ese.
- Si el valor RSSI entre dos dispositivos es entre 10-20, lo redondearemos a 20 y se encenderán los leds hasta ese valor.
- Si el valor RSSI entre dos dispositivos es entre 20-30, lo redondearemos a 30 y se encenderán los leds hasta ese valor.
- Si el valor RSSI entre dos dispositivos es entre 30-40, lo redondearemos a 40 y se encenderán los leds hasta ese valor.
- Si el valor RSSI entre dos dispositivos es entre 40-50, lo redondearemos a 50 y se encenderán los leds hasta ese valor.
- Si el valor RSSI entre dos dispositivos es entre 50-60, lo redondearemos a 60 y se encenderán los leds hasta ese valor.
- Si el valor RSSI entre dos dispositivos es entre 60-70, lo redondearemos a 70 y se encenderán los leds hasta ese valor. En este caso los 7 leds.

El dispositivo que tendrá los leds del ecualizador será el dispositivo MASTER, este será el encargado de llevar las riendas del test. El otro dispositivo será el dispositivo SLAVE.

Por otro lado, ambos dispositivos tendrán un led que indicará el proceso de asociación. Cuando los dispositivos se encuentren en el proceso de asociación, este led parpadeará; una vez que se asocien este led quedara encendido hasta que no haya cobertura entre ellos o uno de ellos se desconecte.

Respecto a los botones, ambos dispositivos tendrán el botón *power*. Éste servirá para dormir o despertar el microcontrolador (no el *transceiver*). Al pulsar este botón, si el microcontrolador está dormido, se despertará. En cambio, si el microcontrolador se encuentra despierto dejará de hacer la tarea que está realizando y se dormirá.

Ilustración 76: El dispositivo MASTER

El dispositivo MASTER además del botón *power* tiene el botón *test*. Éste servirá para empezar o terminar el test. Al despertar el microcontrolador, el dispositivo MASTER se encontrará esperando a pulsar el botón *test*. Al pulsar dicho botón empezará a realizar el test. Una vez que ya queramos parar el test de cobertura tendremos que volver a pulsar el botón *test*.

Si el dispositivo MASTER está esperando a que se pulse el botón *test* más de un minuto, éste se dormirá. Hay que tener en cuenta que si un dispositivo se duerme el otro se dará cuenta que no hay conexión y pasara a intentar asociarse.

Si el dispositivo SLAVE está conectado a la red creada por el dispositivo MASTER y no recibe mensajes durante un minuto este dispositivo también se dormirá.

Al empezar el test el dispositivo MASTER enviara 5 mensajes al dispositivo SLAVE. Cuando el dispositivo SLAVE reciba los mensajes enviará los mismos al dispositivo MASTER. El dispositivo MASTER al recibirlos comparará si son iguales que los enviados, si así fuese, calculará la cobertura y volverá a hacer el test.

Ilustración 77: El dispositivo SLAVE

Si el dispositivo MASTER al enviar un mensaje falla esto querrá decir que ha perdido la conexión con el dispositivo SLAVE e intentará volver a conectarse. Si el envío de los 5 paquetes se realizara correctamente esperará un rato a las respuestas, si llegan las 5 respuestas calculara la media de los valores RSSI y volverá a realizar el test. En cambio, si transcurre el tiempo y no ha recibido respuesta alguna la media será 0 y volverá a realizar el test.

9.1 Diagrama de flujo del dispositivo MASTER

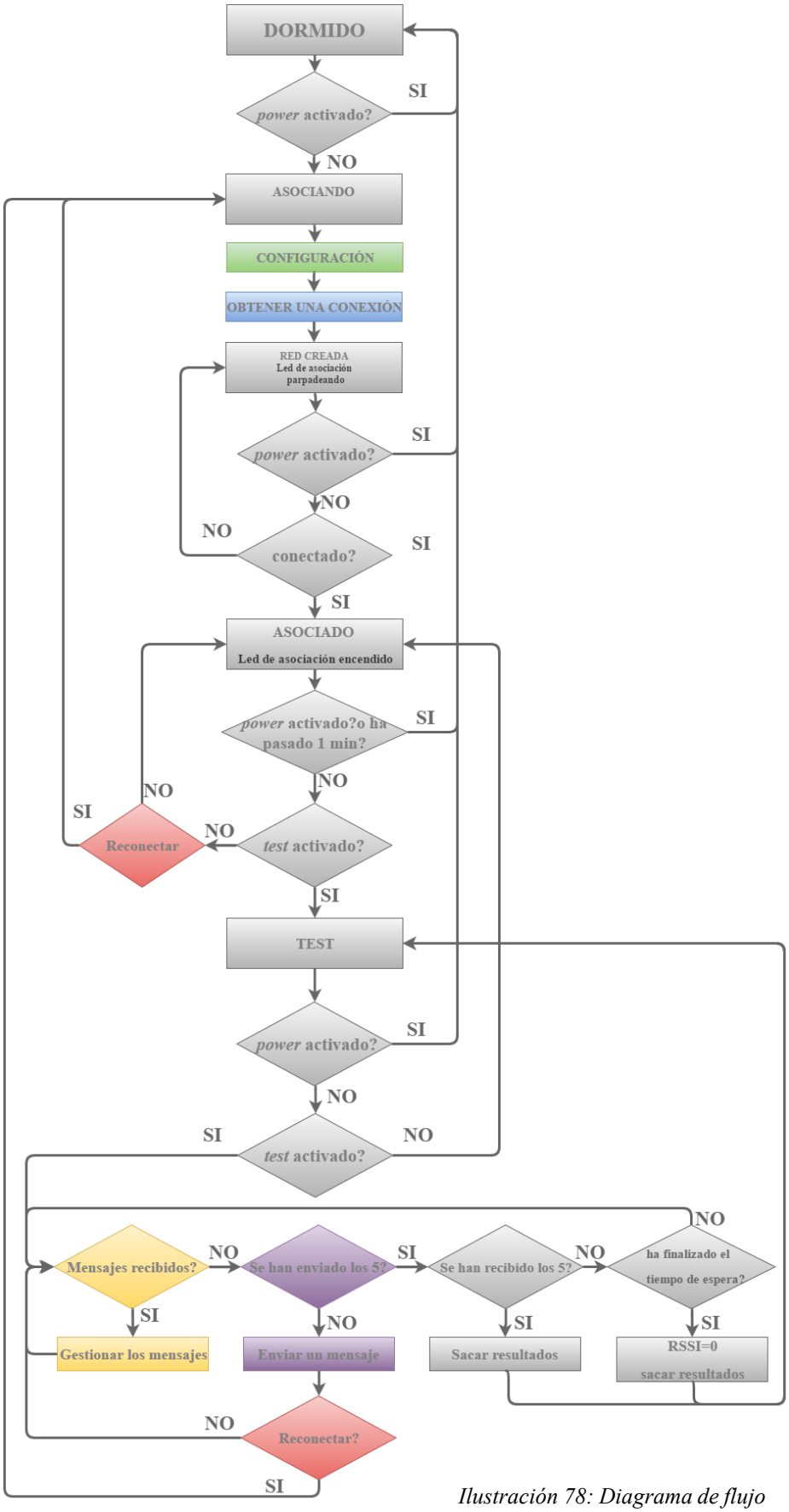


Ilustración 78: Diagrama de flujo del dispositivo MASTER

9.2 Diagrama de flujo del dispositivo SLAVE

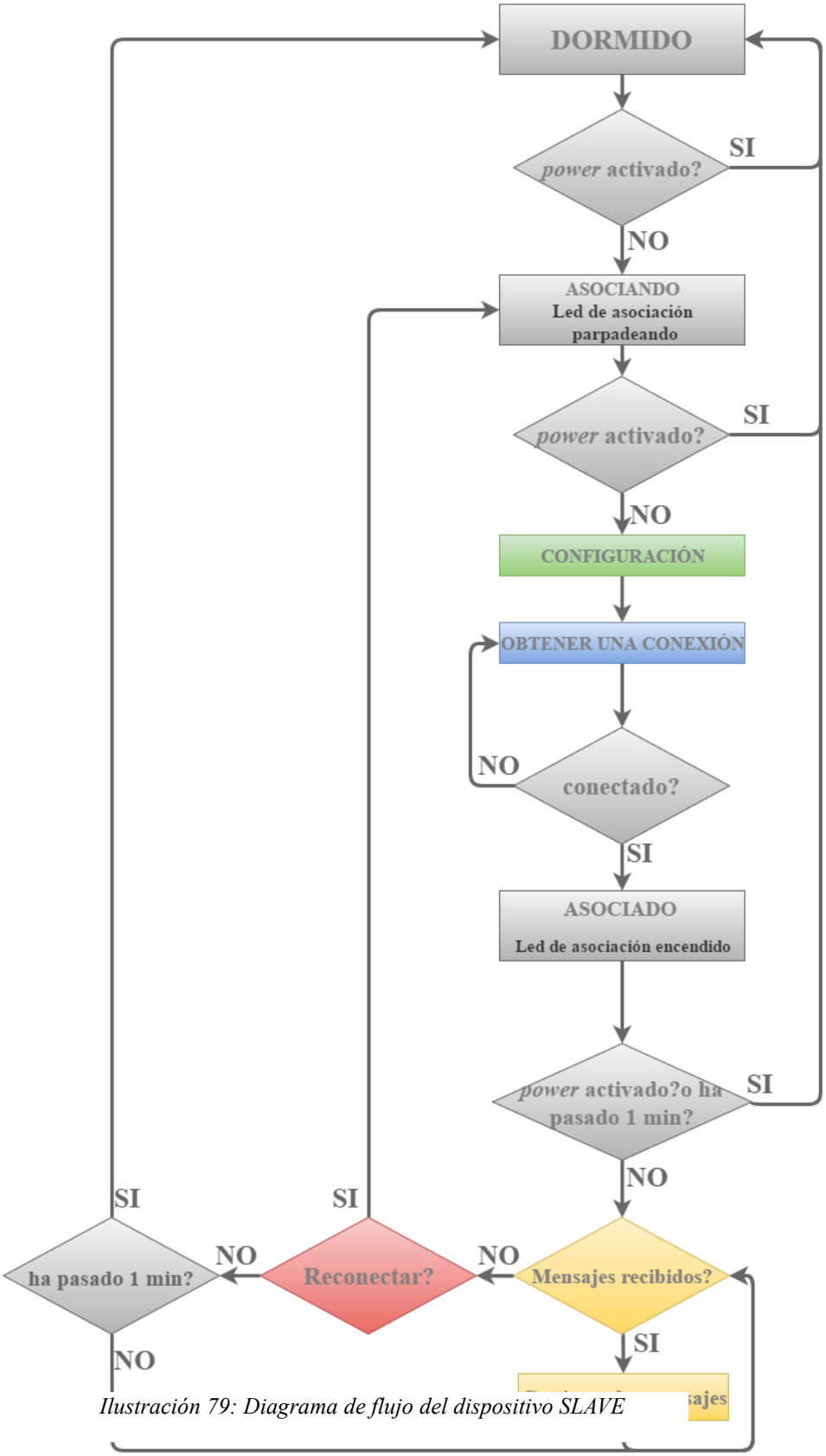


Ilustración 79: Diagrama de flujo del dispositivo SLAVE

9.3 Decisiones tomadas

Antes de empezar a implementar el protocolo se han tomado varias decisiones con respecto a la implementación. En este apartado se detallaran las decisiones tomadas y la razón por la que se ha elegido una u otra opción.

A la hora de empezar a configurar la aplicación la primera decisión que hay que tomar es que protocolo utilizar. Esta aplicación funciona solamente con dos dispositivos y en consecuencia, no necesita enrutar mensajes. Como la cantidad de dispositivos es mínima y no necesita enrutar mensajes se puede optar por utilizar cualquiera de los tres protocolos. En este caso se ha optado por el protocolo MiWi Mesh.

Una vez elegido el protocolo a utilizar, es necesario definir el rol de cada dispositivo. Cada red al menos tiene que tener un dispositivo que será el coordinador PAN. Este dispositivo será el encargado de crear la red. Para la realización de este proyecto se decidió decantarse por el dispositivo MASTER y tendrá la dirección EUI 0000000000000000. El otro dispositivo será un nodo final FFD y tendrá la dirección EUI 1111111111111111.

Tras esta configuración ya se podría crear la red e iniciar su funcionamiento pero no obstante falta decidir qué características especiales necesita la red.

Como cada vez que se encienden los dispositivos tienen que hacer el *handshaking* no habrá que activar la característica de guardar la red anterior en una memoria no volátil (*NetworkFreezer*).

Al dormir el microcontrolador, se tendrá que apagar el *transceiver* pero no como define el protocolo sino bajando la corriente a 0. Además en este caso el dispositivo se tendrá que desconectar de la red por lo tanto no se tendrá que activar la característica de dormir.

Los mensajes que intercambian no llevan nada confidencial por lo que no es necesario activar la seguridad del protocolo.

9.4 Implementación de la aplicación

Antes de empezar a implementar la aplicación se han realizado las configuraciones necesarias.

Para ello no solo se ha tenido que configurar las características del protocolo mencionadas en el *punto 9.3*, también se ha tenido que configurar el microcontrolador, ya que las placas utilizadas para esta prueba tienen el microcontrolador PIC24FJ32GA102. Al cambiar de microcontrolador también se han tenido que modificar los PINs.

Al implementar la aplicación, por lo que respecta a las funciones del protocolo MiWi (configurar, obtener conexión, recibir mensaje, etc.) se ha seguido el mismo procedimiento documentado en las pruebas anteriores.

Respecto a dormirse el microcontrolador, el compilador CCS tiene la función *sleep(SLEEP_FULL).* Al pulsar el pulsador *power* la variable *apagado* se activará y esto lo conducirá a la ejecución a la dicha función. Una vez que se ha dormido el microcontrolador, solamente se puede despertar mediante el reseteo del microcontrolador o mediante la interrupción externa INT0. En este caso será la interrupción externa quien despierte el microcontrolador. Esta interrupción saltará al pulsar el botón *power* por lo tanto la variable *apagado* se actualizara en la rutina de dicha interrupción. Al dormir el microcontrolador tendrá que apagar el *transceiver* para ello está la placa dispone de un pin(*POWERON_MRF*).

Para gestionar el botón test, no hemos usado ninguna interrupción pero sí que hemos usado una para la gestión del rebrote de la misma. Para ello hemos configurado un nuevo *timer*. Este *timer* cada X tiempo saltara la interrupción. Por otro lado tendremos una variable de 2 bytes llamada *debounce*. Esta variable estará inicializada a 0x0000. Al saltar la interrupción del temporizador si está el botón test pulsado desplazara 1 uno hacia la izquierda (<<1) y saldrá de la interrupción. Tras transcurrir un tiempo volverá a saltar la interrupción del temporizador y volverá a desplazar otro 1. Si durante un tiempo considerable ha estado el pulsador pulsado la variable *debounce* tendrá el valor 0xFFFF y será entonces cuando se tendrá en cuenta que se ha pulsado la interrupción.

10

CONCLUSIONES

Tras finalizar el proyecto "Reconversión de Protocolo de Comunicaciones MIWI™ XC16 en CCS" se puede afirmar que el resultado ha sido satisfactorio, puesto que se ha conseguido cumplir tanto con el alcance como con los objetivos propuestos al principio de dicho proyecto.

Fundamentamos la anterior afirmación en la justificación desarrollada a continuación sobre los objetivos y competencias alcanzadas:

- Indudablemente para desarrollar este proyecto ha sido necesario colaborar con la empresa Atelei Engineering. A pesar de que gran parte de trabajo se ha realizado desde casa esto no ha impedido conocer el funcionamiento de la empresa y trabajar en ella. Por lo que se puede afirmar que este primer objetivo ha sido alcanzado.
- El siguiente objetivo a conseguir era comprender a fondo el protocolo de comunicación inalámbrica MiWi™. Como se ha explicado a lo largo de este trabajo previo a la migración se elaboró una exhaustiva documentación sobre dicho protocolo. Además, conseguir la correcta migración y la solución de los problemas surgidos durante el desarrollo de esta habría sido imposible sin un conocimiento profundo del protocolo de comunicación inalámbrica citado. Basándonos en esta información afirmamos que también el segundo objetivo ha sido alcanzado.
- En cuanto al objetivo de dominar el lenguaje de programación C orientado al compilador CCS y XC16 consideramos que también ha sido alcanzado. Si este objetivo no hubiera sido alcanzado habría resultado imposible realizar la migración correctamente; ya que para ello es necesario comprender que hace cada parte de código y como migrar el mismo; para lo que resulta imprescindible dominar el lenguaje C orientado al compilador XC16 y al CCS respectivamente.
- Al inicio del desarrollo del proyecto el conocimiento sobre los conceptos SPI, UART, interrupciones, timers, etc. era limitada pero al tener que trabajar con ellas se ha tenido que comprender el funcionamiento y la programación de las mismas. Por ello se puede decir que se ha conseguido el objetivo de profundizar en la programación de los microcontroladores PIC.
- El último objetivo era dominar el software MPLAB. Como tanto la migración como la aplicación se ha realizado desde el principio a fin utilizando el software en cuestión se ha cumplido con el objetivo.

Respecto al alcance de la migración la justificación desarrollada es la siguiente:

- Se ha conseguido que la migración pueda trabajar los tres protocolos como se ha podido ver en el *punto 7* de la documentación.
- Para hacer las pruebas y la aplicación real se ha utilizado el *transceiver* MRF89XAM8A por lo tanto el producto funciona con las misma.
- La empresa quería que hubiera la opción de mandar mensajes encriptados. El producto creado ofrece esa opción tal como se puede apreciar en el *punto 7.2.5*.
- Para ahorrar energía los dispositivos finales se suelen dormir. La empresa estaba interesado que el producto tuviera esta característica pero al conseguir dicha característica la empresa no quedo convencida con la gestión de los mensajes indirectos. Por ello tuve que hacer algunos cambios consiguiendo así la conformidad de la empresa.

- La última exigencia de la empresa fue que el producto tuviera la opción de recuperar el estado de la red anterior. Esta característica también incorpora el producto migrado.

El objetivo de este proyecto también era crear *templates* de proyecto para cada protocolo y dispositivo. Las pruebas del *apartado 7* se han realizado utilizando estos templates por lo tanto se puede decir que este objetivo también se ha logrado.

Aparte de los objetivos del proyecto también se ha podido crear una aplicación real y el resultado de esta también ha sido satisfactorio.

Aunque se haya logrado cumplir tanto como con el alcance de la migración y el objetivo de crear los *templates* como con los objetivos transversales propuestos al principio del proyecto, en un futuro sería interesante seguir trabajando estas líneas:

- Una de las virtudes del protocolo MiWi™ es el poder cambiar tanto el protocolo de pila como el transceiver solamente cambiando unos parámetros de forma muy sencilla. En el producto creado se puede cambiar el protocolo de pila de una forma muy sencilla pero es imposible cambiar de *transceiver* ya que solamente hemos migrado la capa MAC del *transceiver* utilizado. En un futuro sería interesante migrar las capas MAC de los otros *transceivers* que soporta el protocolo MiWi™ para que se puedan cambiar de una forma muy sencilla.

- Si se cambiara de *transceiver* se podría hacer uso de más características como por ejemplo escaneo activo, escaneo de energía… En este proyecto no se han migrado dichas características del protocolo por el hecho de que el *transceiver* utilizado tiene solamente un canal.

- Respecto a la memoria no volátil en el protocolo se pueden utilizar tres tipos: memoria externa, memoria de datos y la memoria del PIC donde se almacena el programa. En nuestro caso se ha utilizado solamente la memoria externa y por ello solamente se ha migrado partes del código necesarias para escribir y leer en la misma. En un futuro sería interesante migrar para que fuera posible trabajar con los otros dos tipos de memoria.

Glosario

A

ADDI
Es un depósito de documentos digitales, cuyo objetivo es organizar, archivar, preservar y difundir en modo de acceso abierto la producción intelectual resultante de la actividad docente e investigadora ejercida en la Universidad del País Vasco.

B

Baliza
Una baliza es un mensaje que envían los coordinadores para que los otros dispositivos sepan que éste está a su alcance y su dirección.

Broadcast
Es el envío de la información desde un único emisor a todos los dispositivos de la red.

C

CCS
Es un compilador para microcontroladores PIC de Microchip basado en c y creada por la empresa CCS.

H

handshaking
Es un proceso automatizado de negociación que establece de forma dinámica los parámetros de un canal de comunicaciones establecido entre dos entidades antes de que comience la comunicación normal por el canal.

I

ICSP
Es una tecnología incluida en todos los microcontroladores PIC de Microchip más recientes y posibilita la reprogramación de los mismos sin que sea necesaria la remoción de éstos de su circuito de aplicación.

IEE 802.15.4
Es un estándar que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos.

M

MiWi™
Es un estándar que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos.

Multicast
Es el envío de la información desde un único emisor a múltiples destinos simultáneamente.

O

OEM
Son las empresas que manufacturan productos que luego son comprados por otra y vendidos al por menor bajo la marca de la empresa compradora.

P

PAN
Es una red de computadoras para la comunicación entre distintos dispositivos cercanos al punto de acceso.

PIC
Son una familia de microcontroladores tipo RISC fabricados por Microchip Technology Inc. El nombre completo es PICmicro, aunque generalmente se utiliza como PeripheralInterface Controller.

R

RJ11
Es un conector normalmente utilizado en las redes de telefonía., 41

S

Sniffer
Es una aplicación especial para redes informáticas, que permite como tal capturar los paquetes que viajan por una red.

T

Templates

Es una plantilla. Es un medio, aparato, sistema o programa, que permite guiar, portar, o construir, un diseño, esquema o programa predefinido.

Transceiver

Es un dispositivo que cuenta con un transmisor y un receptor que comparten parte de la circuitería o se encuentran dentro de la misma caja. Cuando el transmisor y el receptor no tienen en común partes del circuito electrónico se conoce como transmisor-receptor.

U

Unicast

Es el envío de información desde un único emisor a un único receptor.

USB

(Bus Universal en Serie) Es un bus estándar industrial que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre computadoras, periféricos y dispositivos electrónicos.

X

XC16

Es un compilador para microcontroladores PIC de Microchip creada por la empresa Microchip.

Bibliografía

[1] Anibal Alejandro Airoldi, «Comunicaciones Wireless con PIC,» Primera ed., Buenos Aires, mcelectronics, Octubre de 2014, pp. 263-327.

[2] Yang Yifeng, «Microchip Wireless (MiWi™) Media Access Controller – MiMAC,» Microchip Technology Inc., 2009.

[3] «MPLAB® X IDE,» Microchip Technology Inc., 2011-2014.

[4] Microchip Technology Inc, «MPLAB X IDE,» http://www.microchip.com/mplab/mplab-x-ide.

[5] CCS, Inc., «CCS» http://www.ccsinfo.com/.

[6] «ZENA™ Wireless Adapter,» Microchip Technology Inc., 2011-2012.

[7] Microchip Technology Inc., «Wireless Development Studio» http://www.microchip.com/SWLibraryWeb/product.aspx?product=Wireless%20Development%20Studio.

[8] «RealTerm,» http://realterm.sourceforge.net/.

[9] Saleae, Inc., «Saleae LLC» https://www.saleae.com/.

[10] «Dropbox» www.dropbox.com.

[11] Skype y/o Microsoft, «Skype,» https://www.skype.com/es/

[12] TeamViewer, «TeamViewer 11,» https://www.teamviewer.com

[13] «32K SPI Bus Serial EEPROM,» Microchip Technology Inc., 2009.

[14] David Flowers y Yang Yifeng, «Microchip MiWi™ Wireless Networking Protocol Stack,» Microchip Technology Inc., 2010.

[15] Yang Yifeng, «Microchip MiWi™ P2P Wireless Protocol,» Microchip Technology Inc., 2010.

[16] Yang Yifeng, «Microchip MiWi™ PRO Wireless Networking Protocol,» Microchip Technology Inc., 2011.

[17] Yang Yifeng, «Microchip Wireless (MiWi™) Application Programming Interface – MiApp,» Microchip Technology Inc, 1009.

[18] «MRF89XAM8A Data Sheet,» Microchip Technology Inc., 2010.

[19] «PIC24FJ128GA310 FAMILY Data Sheet,» Microchip Technology Inc., 2010-2014.

[20] «CCS C Compiler Manual,» CCS inc., October 2015.

[21] Andrés Raúl Bruno Saravia, «Aprendiendo a programar Microcontroladores PIC en Lenguaje C con CCS,» http://www.edudevices.com.ar/download/articulos/MCUsPIC/Programando_PICs_CCS_02.pdf

[22] Lucio Di Jasio, Programming 16-BIT microcontrollers in c, learning to Fly the PIC 24, Segunda ed., Oxford & Waltham: Elsevier inc., 2012.

Anexo A: Distribución de la memoria EEPROM

Este anexo es un ejemplo de la distribución de la memoria EEPROM durante el proyecto. La EEPROM tiene 128 páginas y cada página es de 32 bytes.

Este ejemplo se ha realizado con la siguiente configuración del protocolo: El protocolo de pila utilizado es MiWi Pro, en la red podrá haber 32 coordinadores y cada dispositivo podrá tener máximo 10 conexiones. El dispositivo tiene una dirección EUI de 8 bytes y la información adicional del nodo tiene un solo byte.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	PANID	PANID	Current channel	Connection mode	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table
1	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table
2	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table
3	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table
4	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table
5	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table
6	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table
7	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table
8	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table
9	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
127																

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table
1	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table
2	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table
3	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table	Connection table
4	Out frame counter	Out frame counter	Out frame counter	Out frame counter	My short address	My short address	My parent	Routing table	Routing table	Routing table	Routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table
5	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table
6	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table
7	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table
8	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Neighbour routing table	Family tree	Family tree	Family tree	Family tree	Family tree
9	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Family tree	Role					
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
127																

Anexo B: Consumos de la aplicación real

	Intensidad (mA)	Potencia (V)	Consumo (W)
Intentando asociarse y asociado	26	3,3	0,0858
Haciendo el test	34	3,3	0,1122
Dormido	0,029	3.3	0,0000957

Anexo C: Archivos de configuración

Mediante los archivos de este anexo se configura la capa de aplicación, el protocolo de pila y la capa MAC. A lo largo de este documento se han mencionado muchos de estos parámetros para configurar diferentes características del protocolo. En este anexo se definirán cada uno de los parámetros configurables del protocolo y se anotarán las restricciones de ellas.

Config_MiApp.h

El archivo *config_MiApp.h* es el archivo de configuración de la capa MiApp. En ella se definen diferentes parámetros. Es una plantilla en la que según que protocolo de pila quieras utilizar, que transceptor quieras utilizar y que funcionalidades quieres que tenga el dispositivo lo comentas o lo descomentas dicha definición. Las definiciones de dicho archivo y efecto de las cuales son las siguientes:

EJEMPLO DE LA DEFINICION	FUNCIONALIDAD	RESTRICCIONES
#define PROTOCOL_P2P **#define PROTOCOL_MIWI** **#define PROTOCOL_MIWI_PRO**	Selecciona el protocolo inalámbrico que vaya a ser utilizado en la aplicación.	Solamente se puede seleccionar un protocolo. El protocolo de los dispositivos de la misma red deberá ser el mismo. Si el protocolo de red definida es P2P no habrá diferentes roles. En cambio sí es MiWi o MiWi PRO se deberá definir el rol del dispositivo.
#define NWK_ROLE_COORDINATOR **#define NWK_ROLE_END_DEVICE**	Define el rol del nodo actual. Un nodo puede ser coordinador o end device. El coordinador PAN es un coordinador pero además de eso es el que crea la red.	El protocolo P2P no utiliza estas definiciones.
#define MRF24J40 #define MRF49XA **#define MRF89XA**	Selecciona el transceptor que se vaya a utilizar en la aplicación.	Solamente un transceptor puede ser seleccionado.
#define MY_ADDRESS_LENGTH 8	Define el tamaño de las direcciones EUI.	Mínimo tiene que tener 2 bytes y máximo 8 bytes.
#define EUI_7 0x11 **#define EUI_6 0x11** **#define EUI_5 0x11** **#define EUI_4 0x11** **#define EUI_3 0x11** **#define EUI_2 0x11** **#define EUI_1 0x11** **#define EUI_0 0x11**	Define la dirección EUI.	En este caso como es de 8 bytes tenemos que definir los 8 bytes. Si fueran 4 tendríamos que definir solo (EUI_0, EUI_1, EUI_2, EUI_3).
#define TX_BUFFER_SIZE 40	Define el tamaño del buffer de escritura, las diferentes cabeceras del protocolo no están incluidos en este tamaño.	Puede haber restricciones de hardware del transceptor de RF en el tamaño del búfer que puede ser transmitida. La restricción de hardware incluye todas las cabeceras de protocolo. Tiene que ser menos que 127 y mayor que 10.

#define RX_BUFFER_SIZE 40	Define el tamaño del buffer de lectura, las diferentes cabeceras del protocolo no están incluidos en este tamaño.	Puede haber restricciones de hardware del transceptor de RF en el tamaño del búfer que puede ser transmitida. La restricción de hardware incluye todas las cabeceras de protocolo. Tiene que ser menos que 127 y mayor que 10.
#define MY_PAN_ID 0x0000	Define el identificador de la red de área personal.	Todos los dispositivos de la misma red personal tienen que tener el mismo PANID.
#define ADDITIONAL_NODE_ID_SIZE 0	Define el tamaño del identificador adicional que se añade al paquete durante la asociación.	El identificador se define en main.c
#define CONNECTION_SIZE 10	Define el tamaño máximo de conexiones directas que puede tener el nodo.	Tiene que ser más pequeño que 0xFE y más grande que 0. Si es 0 no se podrá conectar con ningún dispositivo.
#define TARGET_SMALL	Elimina el soporte de la comunicación inter PAN y otras características para ahorrar espacio en la programación.	
#define ENABLE_HANDSHAKE	Permite a la pila del protocolo hacer el apretón de manos antes de comunicarse entre sí.	
#define ENABLE_NETWORK_FREEZER	Almacena la información crítica de la red en una memoria no volátil. Así, cuando ocurra una bajada de potencia el dispositivo podrá recuperar el estado de la red en el que se encontraba.	
#define ENABLE_ED_SCAN	Habilita el protocolo inalámbrico y el transceptor RF para realizar un análisis de detección de energía.	Algunos transceptores como MRF89XA no tienen capacidad de hacer la exploración de energía.
#define ENABLE_ACTIVE_SCAN	Permite al dispositivo hacer una exploración activa para detectar redes existentes.	Si el transceptor no es capaz de hacer la exploración de energía, el dispositivo no podrá tener esta funcionalidad.
#define ENABLE_SECURITY	Activa la seguridad a la hora de enviar paquetes.	El modo de seguridad y las claves se definen en un archivo de configuración del transceptor RF (config_89xa.h).
#define ENABLE_SLEEP	Permite al dispositivo irse a dormir y despertarse más tarde. Esta funcionalidad permite ahorrar energía pero solo pueden tener los nodos finales.	Solo algunos transceptores son capaces de dormirse.
#define RFD_WAKEUP_INTERVAL 5	Permite definir cada cuanto tiempo se despertara el nodo.	Solo es efectivo cuando los mensajes indirectos están activados. Esto se define en el dispositivo FFD para controlar cuando se va a despertar el dispositivo RFD.
#define ENABLE_INDIRECT_MESSAGES	Permite guardar mensajes dirigidos a los dispositivos dormidos para entregarlos una vez que hayan despertado y hayan preguntado por los mensajes.	Solo los dispositivos FFD pueden almacenar los mensajes de otros dispositivos.

#define ENABLE_BROADCAST_TO_SLEEP_DEVICE	Permite gestionar mensajes broadcast para los dispositivos dormidos.	Solo los dispositivos FFD pueden almacenar los mensajes de otros dispositivos.
#define ENABLE_FRECUENCY_AGILITY	Permite al dispositivo hacer los procedimientos para el cambio de frecuencia.	Solo algunos transceptores pueden cambiar de frecuencia.
#define HARDWARE_SPI	Permite utilizar el hardware SPI para comunicarse con el transceptor. Si no se define así la comunicación se hará a través de software SPI.	
#define ENABLE_CONSOLE	Permiten mandar información a la híper terminal.	

Config_Protocol.h

El archivo Config_Protocol.h es el archivo de configuración dl protocolo de pila. En ella se definen diferentes parámetros. Cada protocolo de pila puede contener un archivo propio o dentro de un mismo archivo se pueden diferenciar los tres protocolos de pila. En este proyecto se ha elegido utilizar un único archivo por facilidad.

- Si el protocolo de pila definido es P2P:

EJEMPLO DE LA DEFINICION	FUNCIONALIDAD	RESTRICCIONES
#define ENABLE_DUMP	Permite imprimir la tabla de conexiones utilizando la función DumpConnection(index).	
#define RFD_DATA_WAIT 0x00003FFF	Un dispositivo después de haberse despertado envía un DATA REQUEST al nodo asociado y después espera un tiempo para recibir un mensaje. El tiempo que espera es el tiempo que se define aquí. Después de este tiempo de espera, el dispositivo RFD puede seguir funcionando y luego ir a dormir para ahorrar energía de la batería.	Para que tenga efecto esto tiene que estar definido ENABLE_SLEEP.
#define CONNECTION_RETRY_TIMES 3	Define la cantidad máxima que el dispositivo va a intentar establecer conexión. Cuando llegue a 0 quiere decir que no ha conseguido establecer conexión.	
#define CONNECTION_INTERVAL	Define el tiempo transcurrido entre dos intentos de establecer conexión en segundos.	
#define FA_BROADCAST_TIME 0x03	Define el número de veces que se comunica al resto de la PAN que se va a cambiar de frecuencia antes de empezar a cambiar.	Para que tenga efecto esto tiene que estar definido ENABLE_FRECUENCY_AGILITY .
#define RESYNC_TIMES 0x03	Define el número máximo de veces que trata resincronizar en cada uno de los canales disponibles antes de entregar el control a la capa de aplicación	
#define ACTIVE_SCAN_RESULT_SIZE 4	Define el resultado de la exploración activa máxima que la pila puede contener. Si se sobrepasa este número de lecturas se descartaran las últimas.	Para que tenga efecto esto tiene que estar definido ENABLE_ACTIVE_SCAN
#define INDIRECT_MESSAGE_SIZE 2	Define el máximo número de paquetes que pueda guardar un dispositivo al dispositivo asociado, cuando este último este dormido.	Para que tenga efecto esto tiene que estar definido ENABLE_INDIRECT_MESSAGES
#define INDIRECT_MESSAGE_TIMEOUT (ONE_SECOND * RFD_WAKEUP_INTERVAL * (INDIRECT_MESSAGE_SIZE + 1))	Define el tiempo que guardara los paquetes del dispositivo dormido. Una vez sobre pase este tiempo y el dispositivo no haya pedido se procederá a eliminar.	Para que tenga efecto esto tiene que estar definido ENABLE_INDIRECT_MESSAGES
#define ENABLE_ENHANCED_DATA_REQUEST	Al definir esto, el dispositivo dormido combina el mensaje de despertar con el DATA REQUEST. Esto sirve para ahorrar más batería.	Para que tenga efecto esto tiene que estar definido ENABLE_SLEEP.
#define ENABLE_TIME_SYNC	Permite que el coordinador sepa cuando se va a despertar el dispositivo que duerme así podrá asociarse con más de un dispositivo que duerme.	Para que tenga efecto esto tiene que estar definido

		ENABLE_INDIRECT_MESSAGES y tiene que estar previsto que se asocie a más de un dispositivo que se duerme. Para definir esto es necesario definir los siguientes dos parámetros.
#define TIME_SYNC_SLOTS 10	Define el número total de intervalos de tiempo disponible dentro de un ciclo. Como regla general, el número de intervalos debe ser igual o mayor que el número total de dispositivos asociados que duermen para que a cada dispositivo de le pueda asignar un intervalo de tiempo para que se despierta. La duración del intervalo depende de la precisión del oscilador principal del dispositivo FFD y la frecuencia del cristal en los dispositivos que duermen.	Para que tenga efecto esto tiene que estar definido ENABLE_TIME_SYNC
#define COUNTER_CRYSTAL_FREQ 32768	Define la frecuencia del cristal que está conectada al contador MCU para realizar la funcionalidad de la sincronización cuando el MCU se encuentra en reposo.	Para que tenga efecto esto tiene que estar definido ENABLE_TIME_SYNC

- Si el protocolo de pila definido es MESH:

EJEMPLO DE LA DEFINICION	**FUNCIONALIDAD**	**RESTRICCIONES**
#define ENABLE_DUMP	Permite imprimir la tabla de conexiones utilizando la función DumpConnection(index).	
#define RFD_DATA_WAIT 0x00003FFF	Un dispositivo después de haberse despertado envía un DATA REQUEST al nodo asociado y después espera un tiempo para recibir un mensaje. El tiempo que espera es el tiempo que se define aquí. Después de este tiempo de espera, el dispositivo RFD puede seguir funcionando y luego ir a dormir para ahorrar energía de la batería.	Para que tenga efecto esto tiene que estar definido ENABLE_SLEEP.
#define CONNECTION_RETRY_TIMES 3	Define la cantidad máxima que el dispositivo va a intentar establecer conexión. Cuando llegue a 0 quiere decir que no ha conseguido establecer conexión.	
#define OPEN_SOCKET_TIMEOUT (ONE_SECOND * 3)	Define el período de tiempo de espera en símbolos para que el dispositivo abandone el intento de establecer una conexión indirecta.	
#define OPEN_SOCKET_POLL_INTERVAL (ONE_SECOND)	Define el periodo de sondeo (en símbolos) para adquirir datos de su padre en el proceso de establecer la conexión indirecta. Solo se define para aquellos dispositivos que se duermen.	Para que tenga efecto esto tiene que estar definido ENABLE_SLEEP.
#define ENABLE_MIWI_ACKNOWLEDGEMENT	Permite mandar el ACK de la capa de red y reconocer.	Es diferente a la de capa MAC. El de la capa de MAC se configura en el archivo de configuración MiMac.

#define MIWI_ACK_TIMEOUT (ONE_SECOND)	Define el tiempo máximo (en símbolos) que pueda transcurrir desde que se envía un paquete de datos hasta que se recibe el ACK de la capa de red.	
#define ENABLE_BROADCAST_TO_SLEEP_DEVICE	Permite transmitir mensajes broadcast a un dispositivo que se duerme.	
#define BROADCAST_RECORD_SIZE 4	Define el número máximo de mensajes broadcast que se puedan guardar. Estos se guardan para realizar un seguimiento de los mensajes broadcast para saber si el mensaje broadcast es el mensaje recibido anteriormente.	Para que tenga efecto esto tiene que ser un dispositivo que no se duerma.
#define BROADCAST_RECORD_TIMEOUT (ONE_SECOND)	Define el tiempo máximo de espera (en símbolos) para expirar el historial de broadcast.	Para que tenga efecto esto tiene que ser un dispositivo que no se duerma.
#define INDIRECT_MESSAGE_TIMEOUT_CYCLE 2	Cuando se pueden enviar mensajes broadcast a los dispositivos que están durmiendo, para el coordinador es difícil saber a cuál dispositivo a enviado y a cual no. Esto puede llevar a que envié varias veces el mismo mensaje de difusión al mismo dispositivo. MIWI soluciona este problema rastreando el mensaje broadcast el dispositivo que duerme en vez de rastrear el coordinador. Este parámetro define el total de mensajes broadcast recibidos antes de que se espire el tiempo de guardar el registro de los mensajes broadcast.	
#define MAX_ROUTING_FAILURE 3	Define la máxima cantidad de fallos que pueda haber en el enrutamiento entre coordinadores para quitar esta ruta de la tabla de enrutamiento.	Para que tenga efecto esta definición tiene que ser coordinador
#define FA_BROADCAST_TIME 0x03	Define el número de veces que se comunica al resto de la PAN que se va a cambiar de frecuencia antes de empezar a cambiar.	Para que tenga efecto esto tiene que estar definido ENABLE_FRECUENCY_AGILITY .
#define RESYNC_TIMES 0x03	Define el número máximo de veces que trata resincronizar en cada uno de los canales disponibles antes de entregar el control a la capa de aplicación	
#define ACTIVE_SCAN_RESULT_SIZE 4	Define el resultado de la exploración activa máxima que la pila puede contener. Si se sobrepasa este número de lecturas se descartaran las últimas.	Para que tenga efecto esto tiene que estar definido ENABLE_ACTIVE_SCAN
#define INDIRECT_MESSAGE_SIZE 2	Define el máximo número de paquetes que pueda guardar un dispositivo al dispositivo asociado, cuando este último este dormido.	Para que tenga efecto esto tiene que estar definido ENABLE_INDIRECT_MESSAGES
#define INDIRECT_MESSAGE_TIMEOUT (ONE_SECOND * RFD_WAKEUP_INTERVAL * (INDIRECT_MESSAGE_SIZE + 1))	Define el tiempo que guardara los paquetes del dispositivo dormido. Una vez sobre pase este tiempo y el dispositivo no haya pedido se procederá a eliminar.	Para que tenga efecto esto tiene que estar definido ENABLE_INDIRECT_MESSAGES
#define ENABLE_ENHANCED_DATA_REQUEST	Al definir esto, el dispositivo dormido combina el mensaje de despertar con el DATA	Para que tenga efecto esto tiene que estar definido ENABLE_SLEEP.

	REQUEST. Esto sirve para ahorrar más batería.	
#define ENABLE_TIME_SYNC	Permite que el coordinador sepa cuando se va a despertar el dispositivo que duerme así podrá asociarse con más de un dispositivo que duerme.	Para que tenga efecto esto tiene que estar definido ENABLE_INDIRECT_MESSAGES y tiene que estar previsto que se asocie a más de un dispositivo que se duerme. Para definir esto es necesario definir los siguientes dos parámetros.
#define TIME_SYNC_SLOTS 10	Define el número total de intervalos de tiempo disponible dentro de un ciclo. Como regla general, el número de intervalos debe ser igual o mayor que el número total de dispositivos asociados que duermen para que a cada dispositivo de le pueda asignar un intervalo de tiempo para que se despierta. La duración del intervalo depende de la precisión del oscilador principal del dispositivo FFD y la frecuencia del cristal en los dispositivos que duermen.	Para que tenga efecto esto tiene que estar definido ENABLE_TIME_SYNC
#define COUNTER_CRYSTAL_FREQ 32768	Define la frecuencia del cristal que está conectada al contador MCU para realizar la funcionalidad de la sincronización cuando el MCU se encuentra en reposo.	Para que tenga efecto esto tiene que estar definido ENABLE_TIME_SYNC

- Si el protocolo de pila definido es PRO:

EJEMPLO DE LA DEFINICION	**FUNCIONALIDAD**	**RENSTRICCIONES**
#define NUM_COORDINATOR 32	Define el número máximo de coordinadores que puede haber en la red de área personal.	Tiene impacto directo en la memoria RAM y en la MNV.
#define FAMILY_TREE_BROADCAST 3	Define el número de transmisiones broadcast de FAMILY TREE REPORT después de que el coordinador haya unido a la red.	Esta característica solo tiene efecto si el dispositivo tiene el rol de coordinador.
#define ROUTING_TABLE_BROADCAST 3	Define el número de broadcast para el ROUTING TABLE REPORT.	
#define COMM_RSSI_THRESHOLD 0x03	Define la intensidad mínima de la señal para que pueda ser enrutada. Cualquier coordinador que tenga mayor intensidad de señal que esta puede entrar en la lista de vecinos y enrutar.	
#define RANDOM_DELAY_RANGE 100	Rango de retardo aleatorio (ms). Al retransmitir un mensaje broadcast es recomendable esperar un tiempo aleatorio para que no retransmitan todos los coordinadores al mismo tiempo. El retraso aleatorio real será seleccionado al azar entre 0 y RANDOM_DELAY_RANGE.	
#define PACKET_RECORD_SIZE 5	Define el número máximo de mensajes broadcast que se puedan guardar. Estos se guardan para realizar un seguimiento de los mensajes broadcast para saber si el mensaje broadcast es el mensaje recibido anteriormente. Es lo mismo que	Para que tenga efecto esto tiene que ser un dispositivo que no se duerma.

	BROADCAST_RECORD_SIZE en el protocolo MESH.	
#define PACKET_RECORD_TIMEOUT (ONE_SECOND/2)	Define el tiempo máximo de espera (en símbolos) para expirar el historial de broadcast. Es lo mismo que BROADCAST_RECORD_TIMEOUT en el protocolo MESH.	Para que tenga efecto esto tiene que ser un dispositivo que no se duerma.
#define FA_MAX_NOISE_THRESHOLD 0x30	Define el nivel de ruido máximo que acepta el coordinador para el canal propuesto por el coordinador PAN en el proceso de cambio de canal. Si el ruido es mayor que el definido enviara FRECUENCY AGILITY AGAINST CHANNEL	
#define COMM_INTERVAL (ONE_SECOND)	define el intervalo de tiempo entre la difusión del mensaje relacionado FRECUENCY AGILITY entre FA_BROADCAST_TIME	
#define ENABLE_DUMP	Permite imprimir la tabla de conexiones utilizando la función DumpConnection(index).	
#define RFD_DATA_WAIT 0x00003FFF	Un dispositivo después de haberse despertado envía un DATA REQUEST al nodo asociado y después espera un tiempo para recibir un mensaje. El tiempo que espera es el tiempo que se define aquí. Después de este tiempo de espera, el dispositivo RFD puede seguir funcionando y luego ir a dormir para ahorrar energía de la batería.	Para que tenga efecto esto tiene que estar definido ENABLE_SLEEP.
#define CONNECTION_RETRY_TIMES 3	Define la cantidad máxima que el dispositivo va a intentar establecer conexión. Cuando llegue a 0 quiere decir que no ha conseguido establecer conexión.	
#define OPEN_SOCKET_TIMEOUT (ONE_SECOND * 3)	Define el período de tiempo de espera en símbolos para que el dispositivo abandone el intento de establecer una conexión indirecta.	
#define OPEN_SOCKET_POLL_INTERVAL (ONE_SECOND)	Define el periodo de sondeo (en símbolos) para adquirir datos de su padre en el proceso de establecer la conexión indirecta. Solo se define para aquellos dispositivos que se duermen.	Para que tenga efecto esto tiene que estar definido ENABLE_SLEEP.
#define ENABLE_MIWI_PRO_ ACKNOWLEDGEMENT	Permite mandar el ACK de la capa de red y reconocer.	Es diferente a la de capa MAC. El de la capa de MAC se configura en el archivo de configuración MiMac.
#define MIWI_PRO_ACK_TIMEOUT (ONE_SECOND)	Define el tiempo máximo (en símbolos) que pueda transcurrir desde que se envía un paquete de datos hasta que se recibe el ACK de la capa de red.	
#define ENABLE_BROADCAST_TO_ SLEEP_DEVICE	Permite transmitir mensajes broadcast a un dispositivo que se duerme.	
#define INDIRECT_MESSAGE_ TIMEOUT_CYCLE 2	Cuando se pueden enviar mensajes broadcast a los dispositivos que están durmiendo, para el coordinador es difícil saber a cuál dispositivo a enviado y a cual no. Esto puede llevar a que envié varias veces el mismo mensaje de difusión al mismo dispositivo. MiWi soluciona este	

	problema rastreando el mensaje broadcast el dispositivo que duerme en vez de rastrear el coordinador. Este parámetro define el total de mensajes broadcast recibidos antes de que se espire el tiempo de guardar el registro de los mensajes broadcast.	
#define MAX_ROUTING_FAILURE 3	Define la máxima cantidad de fallos que pueda haber en el enrutamiento entre coordinadores para quitar esta ruta de la tabla de enrutamiento.	Para que tenga efecto esta definición tiene que ser coordinador
#define ENABLE_ROUTING_UPDATE	Habilita la actualización dinámica de las tablas de enrutamiento	
#define ROUTING_UPDATE_INTERVAL (ONE_HOUR)	Define el tiempo transcurrido entre dos actualizaciones te tablas de enrutamiento en símbolos.	
#define ROUTING_UPDATE_EXPIRATION	Define cuantos ROUTING_UPDATE_INTERVAL tienen que transcurrir para deshacerse de una ruta valida sin un informe valido de actualización de la tabla de enrutamiento.	Tiene que ser mayor que 1 y normalmente menor que 10
#define FA_BROADCAST_TIME 0x03	Define el número de veces que se comunica al resto de la PAN que se va a cambiar de frecuencia antes de empezar a cambiar.	Para que tenga efecto esto tiene que estar definido ENABLE_FRECUENCY_ AGILITY .
#define RESYNC_TIMES 0x03	Define el número máximo de veces que trata resincronizar en cada uno de los canales disponibles antes de entregar el control a la capa de aplicación	
#define ACTIVE_SCAN_RESULT_SIZE 4	Define el resultado de la exploración activa máxima que la pila puede contener. Si se sobrepasa este número de lecturas se descartaran las últimas.	Para que tenga efecto esto tiene que estar definido ENABLE_ACTIVE_SCAN
#define INDIRECT_MESSAGE_SIZE 2	Define el máximo número de paquetes que pueda guardar un dispositivo al dispositivo asociado, cuando este último este dormido.	Para que tenga efecto esto tiene que estar definido ENABLE_INDIRECT_ MESSAGES
#define INDIRECT_MESSAGE_TIMEOUT (ONE_SECOND * RFD_WAKEUP_INTERVAL * (INDIRECT_MESSAGE_SIZE + 1))	Define el tiempo que guardara los paquetes del dispositivo dormido. Una vez sobre pase este tiempo y el dispositivo no haya pedido se procederá a eliminar.	Para que tenga efecto esto tiene que estar definido ENABLE_INDIRECT_ MESSAGES
#define ENABLE_ENHANCED_DATA_REQUEST	Al definir esto, el dispositivo dormido combina el mensaje de despertar con el DATA REQUEST. Esto sirve para ahorrar más batería.	Para que tenga efecto esto tiene que estar definido ENABLE_SLEEP.
#define ENABLE_TIME_SYNC	Permite que el coordinador sepa cuando se va a despertar el dispositivo que duerme así podrá asociarse con más de un dispositivo que duerme.	Para que tenga efecto esto tiene que estar definido ENABLE_INDIRECT_ MESSAGES y tiene que estar previsto que se asocie a más de un dispositivo que se duerme. Para definir esto es necesario definir los siguientes dos parámetros.

#define TIME_SYNC_SLOTS 10	Define el número total de intervalos de tiempo disponible dentro de un ciclo. Como regla general, el número de intervalos debe ser igual o mayor que el número total de dispositivos asociados que duermen para que a cada dispositivo de le pueda asignar un intervalo de tiempo para que se despierta. La duración del intervalo depende de la precisión del oscilador principal del dispositivo FFD y la frecuencia del cristal en los dispositivos que duermen.	Para que tenga efecto esto tiene que estar definido ENABLE_TIME_SYNC
#define COUNTER_CRYSTAL_FREQ 32768	Define la frecuencia del cristal que está conectada al contador MCU para realizar la funcionalidad de la sincronización cuando el MCU se encuentra en reposo.	Para que tenga efecto esto tiene que estar definido ENABLE_TIME_SYNC

Config_89xa.h

El archivo Config_89xa.h es el archivo de configuración de la capa MiMac. En ella se definen diferentes parámetros. Es una plantilla en la que según en qué banda quieras transmitir, con que tasa de datos quieras transmitir, que funcionalidades quieres que tenga el dispositivo… lo comentas o lo descomentas dicha definición. Las definiciones de dicho archivo y efecto de las cuales son las siguientes:

EJEMPLO DE LA DEFINICION	FUNCIONALIDAD	RENSTRICCIONES
#define BAND_902 #define BAND_915 #define BAND_863	Son las tres bandas de frecuencia compatibles para el transceptor MRF89XA.	Solo uno de ellos se podrá definir. Los otros dos deberán ir comentados. En Europa solo está regulado por la ETSI (European Telecommunications Standards Institute) la banda 863.
#define DATA_RATE_5 #define DATA_RATE_10 #define DATA_RATE_20 #define DATA_RATE_40 #define DATA_RATE_50 #define DATA_RATE_66 #define DATA_RATE_100 #define DATA_RATE_200	Son las 10 tasas de datos soportados por el transceptor MRF89XA.	Solo uno puede ser definido
#define LNA_GAIN LNA_GAIN_0_DB	Define el intervalo de ganancia para el transceptor.	
#define TX_POWER TX_POWER_1_DB	Define la potencia de salida para el transceptor.	
#define RSSI_THRESHOLD RSSI_THRESHOLD_79	Define el umbral para la salida digital RSSI	

#define ENABLE_CCA	Permite llevar a cabo Clear Channel Assessment (CCA) antes de transmitir los datos en la capa MiMac.	
#define ENABLE_ACK	Permite enviar automáticamente un paquete de confirmación en la capa MAC después de recibir un paquete si el paquete lo necesita.	
#define ENABLE_RETRANSMISSION	Permite retransmitir el paquete hasta las veces especificadas en RETRANSMISSION_TIMES, si ENABLE_ACK está definido y no ha recibido el ACK correcto dentro de un periodo de tiempo.	
#define SOURCE_ADDRESS_ABSENT	No envía la dirección de origen del mensaje, si al destino no le importa de dónde vienen los mensajes.	Esta función sólo está disponible para transceptores que soportan formato de trama MiMac.
#define MAX_ALLOWED_TX_FAILURE 20	Define el número máximo de intentos para transmitir un paquete antes de emitir un fallo en la transmisión a la capa de protocolo superior.	
#define RETRANSMISSION_TIMES 15	Define las retransmisiones máximas que se pueden realizar si el paquete ACK adecuado no llega en el período de tiempo predefinido.	Para que tenga efecto esta característica deberá estar definido ENABLE_RETRANSMISSION
#define CCA_TIMES 5	Define el número total de Clear Channel Assessment en el procedimiento CCA.	El procedimiento CCA es la siguiente: Realiza CCA hasta CCA_TIMES. Después mira los tiempos de fracaso. Si los tiempos de fracaso superan CCA_THRESHOLD se repite todo el procedimiento hasta CCA_RETRIES.
#define CCA_THRESHOLD 65	Define los tiempos de umbral de fallo de Clear Channel Assessment en el procedimiento CCA.	El procedimiento CCA es la siguiente: Realiza CCA hasta CCA_TIMES. Después mira los tiempos de fracaso. Si los tiempos de fracaso superan CCA_THRESHOLD se repite todo el procedimiento hasta CCA_RETRIES.
#define CCA_RETRIES 4	Define el número máximo de reintentos se pueden realizar en el caso de fallo de Clear Channel Assessment en el procedimiento CCA.	El procedimiento CCA es la siguiente: Realiza CCA hasta CCA_TIMES. Después mira los tiempos de fracaso. Si los tiempos de fracaso superan CCA_THRESHOLD se repite todo el procedimiento hasta CCA_RETRIES.
#define BANK_SIZE 2	Define el número máximo de paquetes que puede recibir y conservar la capa MiMac hasta que	

	dichos paquetes pasen a la pila del protocolo.	
#define ACK_INFO_SIZE 5	Define el número de ACKs que pueden ser almacenadas para evitar el paquete duplicado.	
#define USE_IRQ0_AS_INTERRUPT	Habilita MRF89XA para poder utilizar interrupciones IRQ0 e IRQ1.	
#define SECURITY_KEY_00 0x00 **#define SECURITY_KEY_01 0x01** **#define SECURITY_KEY_02 0x02** **#define SECURITY_KEY_03 0x03** **#define SECURITY_KEY_04 0x04** **#define SECURITY_KEY_05 0x05** **#define SECURITY_KEY_06 0x06** **#define SECURITY_KEY_07 0x07** **#define SECURITY_KEY_08 0x08** **#define SECURITY_KEY_09 0x09** **#define SECURITY_KEY_10 0x0a** **#define SECURITY_KEY_11 0x0b** **#define SECURITY_KEY_12 0x0c** **#define SECURITY_KEY_13 0x0d** **#define SECURITY_KEY_14 0x0e** **#define SECURITY_KEY_15 0x0f**	Definen cada byte de la clave usado para el cifrado de bloques. La longitud de la clave depende del tamaño de la clave del cifrado de bloque.	El 00 es el byte 0 el 01 el byte 1 y así sucesivamente.
#define KEY_SEQUENCE_NUMBER 0x00	Define el número de secuencia que se utiliza para identificar la clave. Si la aplicación utiliza múltiples claves, cada clave tendrá un número de secuencia diferente.	
#define SECURITY_LEVEL SEC_LEVEL_CTR **#define SECURITY_LEVEL SEC_LEVEL_CBC_MAC_16** **#define SECURITY_LEVEL SEC_LEVEL_CBC_MAC_32** **#define SECURITY_LEVEL SEC_LEVEL_CBC_MAC_64** **#define SECURITY_LEVEL SEC_LEVEL_CCM_16** **#define SECURITY_LEVEL SEC_LEVEL_CCM_32** **#define SECURITY_LEVEL SEC_LEVEL_CCM_64**	Define el modo de seguridad utilizado por la aplicación.	Solamente un modo de seguridad puede ser definido.
#define FRAME_COUNTER_UPDATE_INTERVAL 1024	El intervalo para actualizar el contador de tramas en la memoria no volátil	