SI^3

Sistema de Iluminación Inteligente Inalámbrico

Titulación: Ingeniería Informática

Autor: Jesús Nicolás Jaro

Director: Sebastià Cortes Herms

25 de Junio de 2014

DEDICATORIA

"Un gran sacrificio resulta fácil; los que resultan difíciles son los continuos pequeños sacrificios"

Goethe.

Al escribir esta dedicatoria, recuerdo a muchas personas que gracias a su ayuda, han hecho posible que llegue este momento.

A Sebastià Cortes por su consejo y apoyo en la elaboración del proyecto y a los profesores y consultores que me han guiado en este largo camino.

A los compañeros de viaje y amigos, que estaban ahí cuando los necesitaba y cuando no.

Y muy especialmente:

A mis padres, por darme la oportunidad de elegir y el ejemplo para luchar por mis sueños.

A mis hijas Andrea y Paula, por su paciencia y cariño, y por las cientos de horas de juegos y risas que las debo.

Pero sobre todo, a Chus, mi mujer, por esos miles de continuos pequeños sacrificios (y otros muchos no tan pequeños), que me ha regalado durante estos años y que tengo guardados en un rinconcito de mi corazón.

RESUMEN

El objetivo de este proyecto es el de sentar las bases para crear un sistema de iluminación compuesto por múltiples dispositivos, que de forma inalámbrica, se sincronicen entre si, para encenderse, apagarse y cambiar de color.

El sistema se ha diseñado para conectar hasta 8 dispositivos, cada uno de ellos compuesto por un LPC1769, un sensor acelerómetro, 4 leds de distinto color, y módulo Wifly RN-XV, que los interconectará de forma inalámbrica.

Para facilitar las tareas de desarrollo y depuración, se le ha dotado también de un módulo CP2102 USB/UART, que aunque no es necesario para su funcionamiento normal, permite mostrar los mensajes de debug por consola, en caso de problemas.

El conjunto opera de forma independiente y sin infraestructura adicional, es decir, no necesita de un router Wi-Fi o punto de acceso, ya que uno de los dispositivos hará las funciones de punto de acceso para que el resto se conecten. Esta interesante característica lo hace ideal para iluminar zonas de poca cobertura Wi-Fi o lugares sin infraestructura de redes inalámbricas, como exteriores y jardines.

En el diseño se ha priorizado la facilidad de uso y la simplicidad, por lo que cualquier usuario, sin ningún tipo de configuración, ni infraestructura adicional , será capaz de utilizarlo.

El modo de funcionamiento es muy sencillo. No hay más que alimentar los dispositivos y estos se conectarán entre si automáticamente. Una vez encendidos , con solo mover o inclinar uno de ellos, los leds se irán encendiendo de forma consecutiva (cada inclinación producirá un cambio de color, emulando un sistema de luz ambiente real), que se propagará de forma automática entre el resto de los dispositivos, quedando todos sincronizados. De igual manera, al hacer la misma operación sobre cualquiera de ellos, se producirá el mismo efecto.

Para el desarrollo del proyecto se han construidos dos prototipos idénticos, uno de los cuales hará de punto de acceso y el otro de cliente. La gestión de los mismos se hace mediante un desarrollo de software escrito en leguaje C, sobre el sistema operativo en tiempo real FreeRtos y la plataforma de desarrollo LPCXpresso.

El sistema escala hasta 8 unidades, incluyendo el punto de acceso. Para añadir un nuevo dispositivo a la red, solo hay que encenderlo, y el nuevo dispositivo se agregará a la misma automáticamente, sincronizándose con el siguiente movimiento de alguno uno de ellos.

Por supuesto, se trata de un sistema básico, desarrollado con fines educativos, por lo que tiene un amplio margen de evolución. Desde dotarle de mayor escalabilidad y conectividad mediante un router externo con acceso a Internet o un módulo 3G y convertirlo en un dispositivo “conectado”, a proporcionar nuevas funcionalidades al añadir más sensores (ultrasónicos, lumínicos, temperatura, etc.), gestión remota desde *smartphones* o mecanismos de ahorro de energía.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

1.1 JUSTIFICACIÓN

Los avances tecnológicos que de forma vertiginosa se están produciendo en todos los ámbitos de la sociedad, repercuten rápidamente en la forma que tenemos de interactuar con nuestro entorno, y ponen a nuestro alcance todo un mundo de posibilidades.

Con una industria en auge, nacida alrededor del *Internet de las* cosas y en un mundo donde cada vez es más habitual el poder interactuar con los elementos del entorno para gestionarlo a nuestra voluntad, he considerado interesante adentrarme, en lo que a nivel personal considero un territorio todavía por desarrollar, como es el de la iluminación inteligente.

Un sistema de iluminación inalámbrica inteligente, facilitará la gestión de la iluminación principal y ambiental de hogares, oficinas y jardines, sin la necesidad de realizar cableado especial y de forma remota e instantánea.

Con este proyecto he querido desarrollar el embrión de un sistema de iluminación compuesto por varios puntos o nodos de luz, que permitan cambiar la luz ambiente de una o varias salas o incluso de zonas exteriores, de manera inalámbrica, con solo la manipulación simple de cualquiera de la unidades que lo componen, sirviendo como un elemento más de la decoración.

El sistema diseñado sienta las bases para que en una fase posterior, y ya fuera del alcance de este proyecto, pueda ser gestionado desde un *smartphone* de forma remota, pudiendo dotarle de multitud de funcionalidades adicionales muy interesantes.

Por citar algunos ejemplos, la gestión de ahorro de energía, la iluminación selectiva, la iluminación en función de la música que reproduce el *smartphone*, e incluso la iluminación en función de datos biométricos registrados desde sensores embebidos en pulseras y otros dispositivos que interactúen con el *smartphone*, son viables con tan solo evolucionar el software.

Apagar la luz automáticamente cuando un sensor detecta que estamos dormidos, cambiar el color de la iluminación del entorno en función de nuestro estado de animo, o iluminar una área de un jardín a nuestro paso detectando nuestra presencia mediante la Wi-Fi de nuestro *smartphone* o por nuestra posición GPS, más que nunca, está al alcance de la mano.

1.2 DESCRIPCIÓN DEL PROYECTO

El proyecto consiste en el diseño y desarrollo de un sistema de iluminación multi-nodo que sincroniza el color de la iluminación que emite cada nodo, mediante tecnología Wi-Fi. El sistema funciona a modo de lámparas independientes, pero sincronizadas entre si, permitiendo iluminar una estancia o área con distintos puntos de luz, a nuestro gusto y de forma remota.

Cada uno de los nodos está compuesto principalmente por un sistema embebido basado en tecnología CORTEX-M3, concretamente el microcontrolador NPX LPC1769 y un módulo WiFly RN-XV para interconectarlos entre ellos, un acelerómetro, leds y otros componentes electrónicos, que se detallarán en capítulos posteriores.

Cada nodo dispone de cuatro leds que simulan un sistema de iluminación real, cada uno de un color (verde, rojo, azul y amarillo), que se irán iluminado secuencialmente en el momento en que el dispositivo sea girado sobre si mismo, o agitado, basándose en los cambios de posición medidos por el acelerómetro.

De forma paralela, si dentro de la red inalámbrica formada entre ellos, se encuentra otro dispositivo (el diseño actual soporta hasta un total de 8), todos ellos cambiarán automáticamente el color, activando de forma automática el led adecuado.

Una característica importante de la solución es que se ha diseñado para que su funcionamiento sea simple y autónomo, no necesitando ningún dispositivo de red adicional para funcionar, así como de configuración previa para añadir más nodos al sistema, que se conectarán automáticamente con solo encenderlos.

Dentro de esta red formada por los nodos, uno de ellos tendrá el rol de master y el resto tendrán el rol de clientes. La única diferencia fundamental entre ellos, está en la capa de comunicaciones, ya que el nodo master se comportará como un punto de acceso Wi-Fi para el resto de nodos, proveyendo mediante el protocolo DHCP de IPs dinámicas a los nuevos clientes que se vayan incorporando a la red. La solución permite que los dispositivos el rol de punto de acceso o del cliente de forma automática, en función del orden en se enciendan (el primer nodo encendido asumirá el rol de master al no detectar la presencia de ningún master en la red).

Por cuestiones de tiempo y por la dificultad añadida que tener que construir dos dispositivos idénticos para realizar las pruebas de comunicaciones, se ha decidido simplificar al máximo el diseño hardware eliminando, todos los componentes que no fueran estrictamente necesarios para implementar la esencia del proyecto, como son interruptores, leds adicionales, baterías, etc., que si serian interesantes incorporar en una solución comercial.

En cuanto al software, en su diseño se ha buscado la máxima simplicidad para el usuario, evitando tener que conectar cada nodo a un PC para configurarlo, ejecutándose en cada de ellos la misma versión de código, independientemente de

tratarse del nodo master o nodo cliente, así cómo como de dotarle de cierta escalabilidad, pudiendo crecer hasta 8 dispositivos por sistema.

Por otro lado, al tratarse de dispositivos de bajo consumo energético y al tener que estar constantemente activados para conocer el estado del resto de dispositivos, no se han aplicado técnicas de ahorro de energía en esta fase, aunque podrían considerarse como una posible mejora.

También, como mejora y pensando en una posible evolución hacia un producto comercial, dotarle de una batería recargable, que le permitiera ser ubicado en cualquier lugar, incluso en el exterior añadiendo células solares, podría ser una opción interesante a considerar.

Se ha excluido del alcance de este proyecto una funcionalidad relevante en un producto comercial de iluminación inteligente, que le dotaría de gran potencia, como es la gestión remota desde un Smartphone u otro dispositivo. La limitación de tiempo disponible para el desarrollo del proyecto ha primado a la hora de priorizar el trabajo en el campo de los sistemas embebidos y dejar para una fase posterior el desarrollar, por ejemplo, una aplicación de Smartphone con la que gestionarlo.

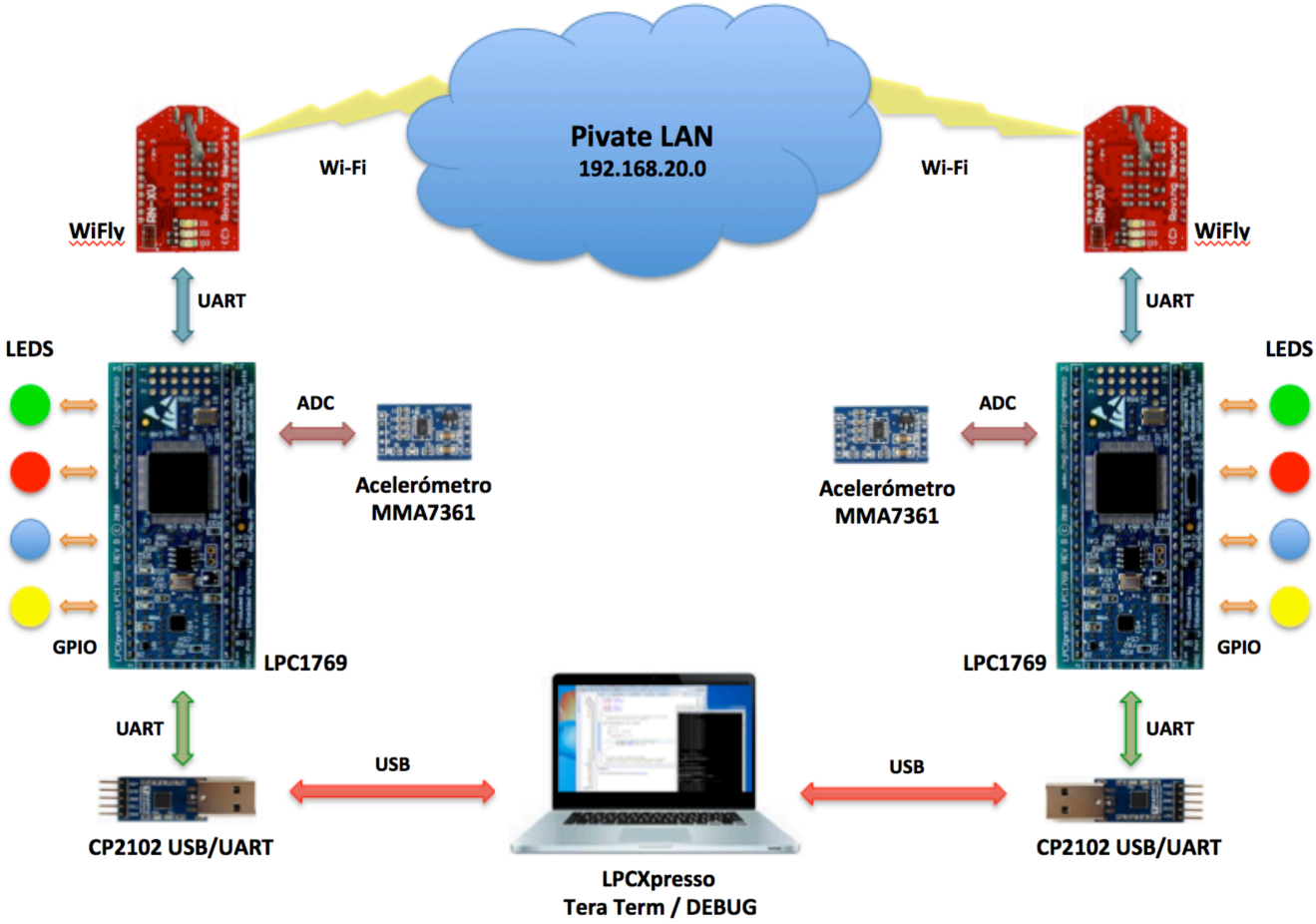


Ilustración 1: Diagrama del proyecto

Un posible producto comercial basado en tecnología, podrían ser lámparas en forma de cubo que modificaran su color al se cambiadas de cara.

Ilustración 2: Lámparas de diseño

1.3 OBJETIVOS

El objetivo de este proyecto es crear los cimientos de una solución de iluminación inteligente basada en un sistema embebido en tiempo real, que permita cambiar la iluminación ambiente de un espacio de manera inalámbrica y sincronizada, con la manipulación simple de cualquiera de la unidades que lo componen.

Los requisitos que se han definido para su desarrollo y que ha de cumplir el producto final obtenido, son los siguientes:

- Crear dos módulos de iluminación ambiental de cuatro colores que funcionen de forma autónoma o en grupo.
- Activación y cambio de color basados en la variación de posición de cualquiera de los dispositivos, registrados por el acelerómetro incorporado.
- Capacidad de interconexión inalámbrica con otros dispositivos similares para que actúen de forma síncrona, sin necesidad de infraestructura adicional.
- Soporte de interconexión escalable hasta 8 dispositivos, con la capacidad de incorporación automática a la red de los nuevos dispositivos
- Funcionamiento simple y sin necesidad de configuración inicial por parte del usuario (autoconfigurables).
- Hardware y software idéntico en todos los dispositivos.
- Diseño compatible con funcionalidades avanzadas para evoluciones futuras, como la gestión desde Smartphone (fuera del alcance de este proyecto).

Para comprobar el funcionamiento del producto final obtenido, a lo largo del proyecto se construirán dos prototipos idénticos con los cuales se realizarán las pruebas funcionales.

1.4 ENFOQUE Y MÉTODO SEGUIDO

Para la gestión del proyecto se han seguido las pautas del PMBOK y se ha dividido el ciclo de vida del mismo, en cinco etapas o grupos de procesos:

- Iniciación
- Planificación
- Ejecución
- Seguimiento y control
- Cierre

En cuanto a la metodología seguida para el desarrollo del software, por las características del mismo, se ha seguido una metodología de desarrollo clásico o en cascada, en las que el producto final ha pasado por las siguientes etapas:

- Definición de requisitos
- Análisis y diseño
- Implementación
- Pruebas

Como el proyecto se ha desarrollado bajo una filosofía didáctica y al comienzo del mismo los conocimientos en sistemas empotrados era prácticamente inexistentes, a lo largo del progreso del mismo, se han ido descubriendo características técnicas y funcionalidades concretas, que han hecho que la resolución de algunos problemas a resolver, se afrontaran desde otras perspectivas a las ideadas inicialmente.

Por este motivo, y aunque básicamente se ha seguido un proceso secuencial, se han establecido ciclos de retroalimentación en el análisis final de cada etapa, para incorporar las correcciones y modificaciones que pudieran afectar a etapas anteriores.

Siguiendo la línea de trabajo trazada por el director del proyecto, se han identificado 3 grandes apartados perfectamente diferenciados a la hora de determinar los procesos principales del proyecto:

- Proceso de aprendizaje
- Proceso de creación y desarrollo del proyecto
- Proceso de documentación

En el siguiente apartado se describirán como se ha planificado la realización de estos procesos, destellándose las tareas que los componen.

1.5 PLANIFICACIÓN DEL PROYECTO

Para el desarrollo del proyecto se ha realizado una planificación inicial que consta de tres fases, que se han llevado a cabo a lo largo de todo el semestre.

A continuación de detallan cada una de las fases tal y como se planificaron inicialmente, con sus tareas correspondientes:

FASE 1 – APRENDIZAJE

En esta fase se adquirirán los conocimientos básicos del entorno de desarrollo LPCXpresso, así como de todos los elementos hardware y software, incluyendo el sistema operativo FreeRtos. La duración de esta fase es de 51 días y consta de las siguientes tareas:

- PEC1 - Conceptos básicos. Duración 17 días.
- PEC2 - Conceptos avanzados. Duración 24 días
- PEC3 - Descripción del proyecto. Duración 10 días.

FASE 2 - DISEÑO Y DESARROLLO DEL PROYECTO

En esta fase se diseñará y desarrollará la totalidad del proyecto elegido en la tarea PEC3 de la fase 1. Su duración será de 22 días y consta de las siguientes tareas:

- Definición de Requisitos.
  Se definirán las funcionalidades y requisitos que tendrá que cumplir el sistema. Durante el desarrollo del mismo serán revisadas para ampliar o acotar las mismas en función de los resultados obtenidos. Duración 2 días.

- Diseño de Arquitectura.
  Se analizará la arquitectura más idónea para desarrollar el proyecto, en función de los requisitos definidos y de las funcionalidades de los componentes seleccionados. Duración 1 día.

- Montaje hardware del prototipo 1.
  Se montará y probará el hardware del primer prototipo. Duración 1 día.

- Desarrollo de Drivers.
  Aunque es una tarea que se realizará durante todo el proyecto, sé tendrá una versión funcional de los mismo al finalizar esta tarea. Duración 8 días.

- Desarrollo Programa principal (funcionamiento unitario).
  Se programará el programa principal que permitirá encender 4 leds en función de la orientación del sistema. Duración 2 días.

- Pruebas y corrección de errores (funcionamiento de un prototipo).
  Se realizarán las pruebas y corrección de errores de la versión inicial del sistema con un solo prototipo. Duración 3 días.

- Montaje hardware del prototipo 2.
  Construcción del segundo prototipo idéntico al prototipo 1. Duración 1 día.

- Desarrollo Función de sincronización entre dispositivos.
  Desarrollo del sistema de comunicación entre los dispositivos y su sincronización. Está es a priori, la tarea más critica del proyecto. Duración 4 días.

- Adaptación de Programa principal multi-dispositivo.
  Ampliación de programa realizado previamente para que gestionen las comunicaciones entre dispositivos. Duración 2 días.

- Pruebas y corrección de errores (funcionamiento 2 prototipos simultáneamente).
  Pruebas y corrección de errores de todo el sistemas. Duración 4 días.

- Revisión final y cierre del desarrollo.
  Revisión minuciosa del proyecto y cierre del mismo. Duración 1 día.

FASE 3: DOCUMENTACIÓN y ENTREGA

En esta fase se confeccionará de todos los entregable del PFC. Su duración total será de 15 días y consta de las siguientes tareas:

- Preparación de Memoria del Proyecto.
  Aunque durante todo el proyecto se irá documentando el mismo, en esta etapa se dará forma a la memoria. Duración 10 días.
- Preparación de Presentación del Proyecto.
  Se realizará una presentación con los aspectos más destacados del proyecto. Duración 3 días.

- Preparación paquete código fuente.
  Recopilación y empaquetado de todos los elementos del proyecto junto con el código fuente. Duración 1 día.

- Entrega del PFC.
  Revisión de todos los elementos y entrega. Duración 1 día.

A continuación se muestra el diagrama de Gantt del proyecto:

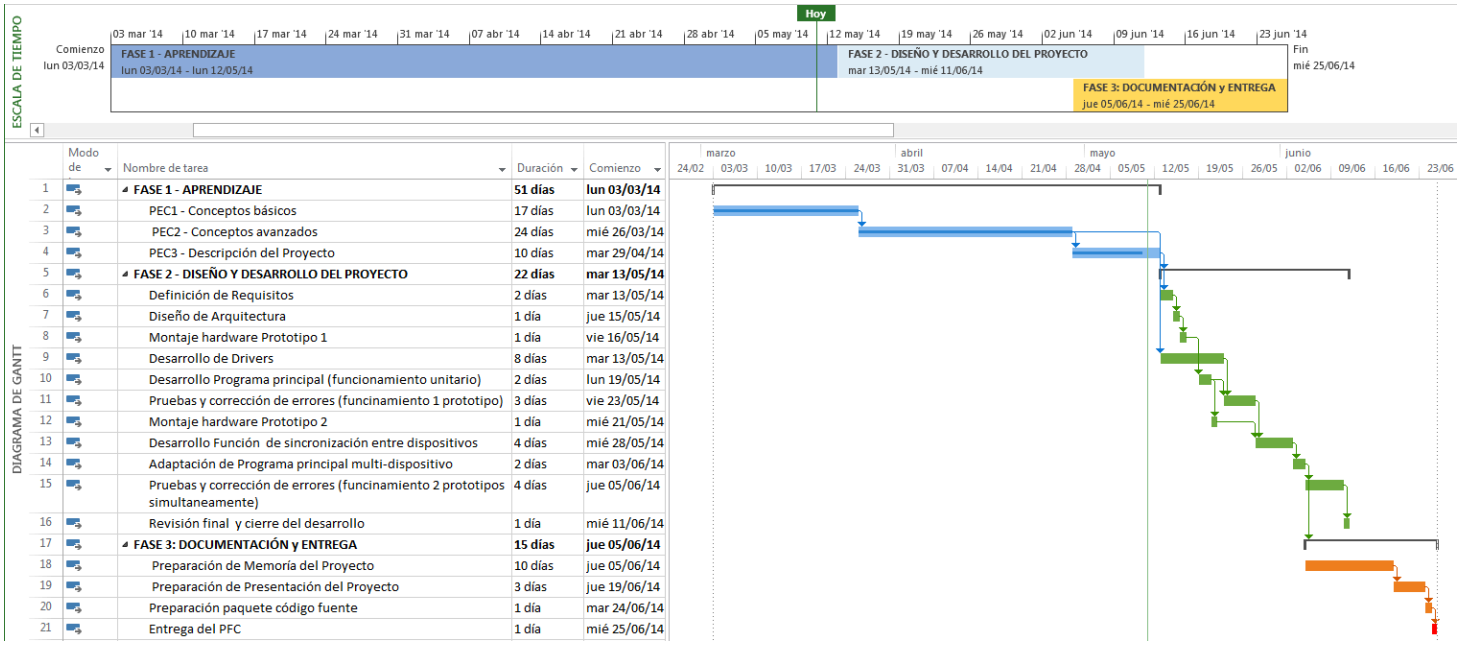


Ilustración 3: Diagrama de Gantt

1.6 RECURSOS EMPLEADOS

A lo largo del desarrollo del proyecto se han utilizado distintos elementos hardware y software para construir los prototipos, así como una serie de herramientas que han sido de gran utilidad para el progreso del mismo.

Para el desarrollo del proyecto se ha utilizado un portátil Macbook Pro i7 de 2 GHz con 8 Gb de RAM, junto con iMac i7 de 2.8Ghz con 12 Gb de RAM, ambos con VMware Fusión v6.0.3 y maquinas virtuales con Windows 7 64 bits.

El hecho de utilizar tecnologías de virtualización, ha permitido clonar los entornos de desarrollo y pruebas con mucha facilidad, así como generar *snaps* para mantener puntos de consistencia. Por otro lado, a la hora de realizar las pruebas, ha sido de gran utilidad poder suspender las máquinas virtuales en un estado concreto, para activarlas de nuevo en el momento adecuado.

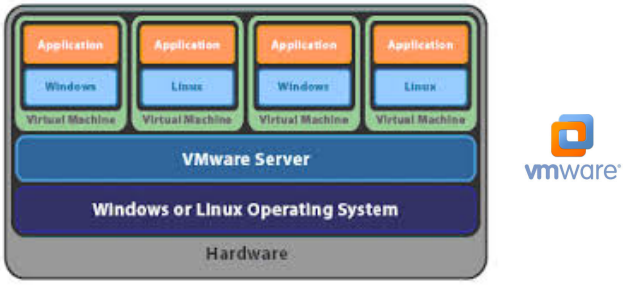


Ilustración 4: Software de virtualización VMware

En cuanto al software, se han utilizado los siguientes programas y utilidades:

- LPCXPresso v7.0.0_92 : Entorno de desarrollo de NXP Semiconductors para microcontroladores
- FreeRTOS v7.0.1: Sistema Operativo Tiempo Real para LCP y otras plataformas
- Tera Term v4.82: Terminal de comunicación serie.
- Notepad ++ v6.6.6: Editor de código fuente
- WireShark v1.10.7 para OS X: Captura de tráfico Wi-Fi en tiempo real
- Suit Microsoft Office for Mac 2011: Documentación.

Cada prototipo construido consta de los siguientes elementos:

- 1 x Placa de desarrollo LPCXpresso LPC1769
- 1 x Módulo WiFly RN-XV con adaptador Xbee
- 1 x XBEE Adapter Board Wifly-Breadboard
- 1 x Módulo CP2102 USB/UART
- 1 x Sensor acelerómetro MMA7361
- 3 x Resistencias
- 3 x condensadores
- 4 x leds de distinto color
- 1 x Breadboard
- Múltiples Cables

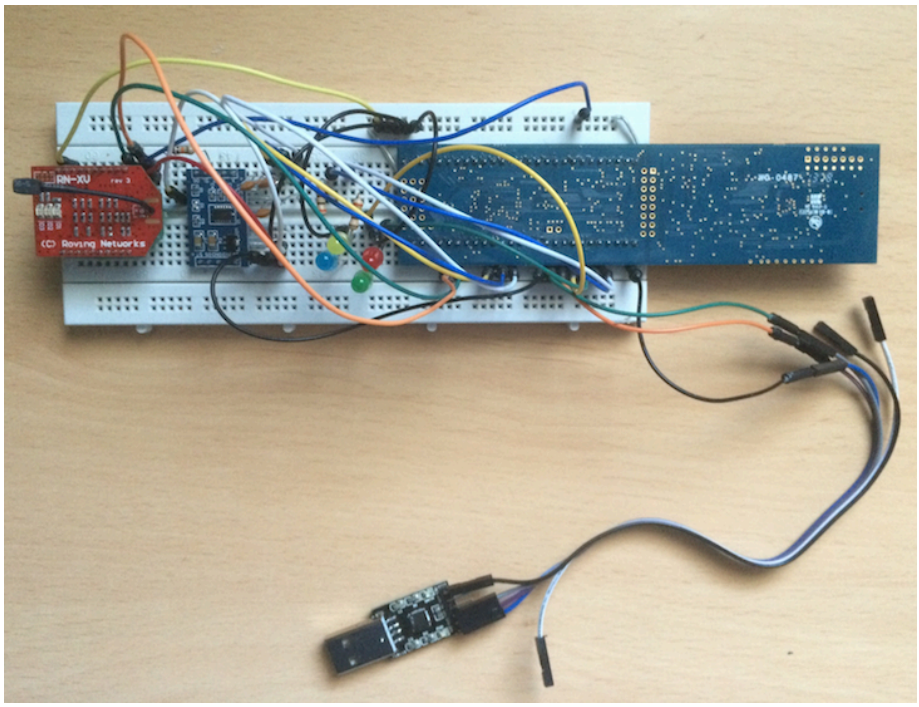


Ilustración 5: Uno de los prototipos construidos

Otras herramientas utilizadas:

- 1 x Soldador JBC 30S
- 1 x XBEE Adapter Board Wifly-Breadboard adicional
- 1 x Multímetro Digital Palm Size T33D
- Alicates, pela-cables, destornilladores, etc.
- 2 x Hub USB

Todos los recursos hardware y software relevantes serán descritos en detalle en capítulo 4.

1.7 PRODUCTOS OBTENIDOS

Tras finalizar el proyecto, el producto obtenido es una solución compuesta por 2 prototipos perfectamente funcionales, que una vez conectados a la alimentación y sin necesidad de configuración alguna, se interconectan entre si mediante una red Wi-Fi dedicada generada por ellos mismos.

El sistema se comporta como un autómata finito determinista de 5 estados.

Al efectuar un giro de cualquiera de ellos, la iluminación ambiente (que se simula encendiendo uno de los 4 leds de colores de los que dispone cada prototipo), cambia de color. Un nuevo giro, hace que ambos prototipos iluminen el siguiente led y así sucesivamente hasta quedar todos apagados y empezar el ciclo de nuevo.

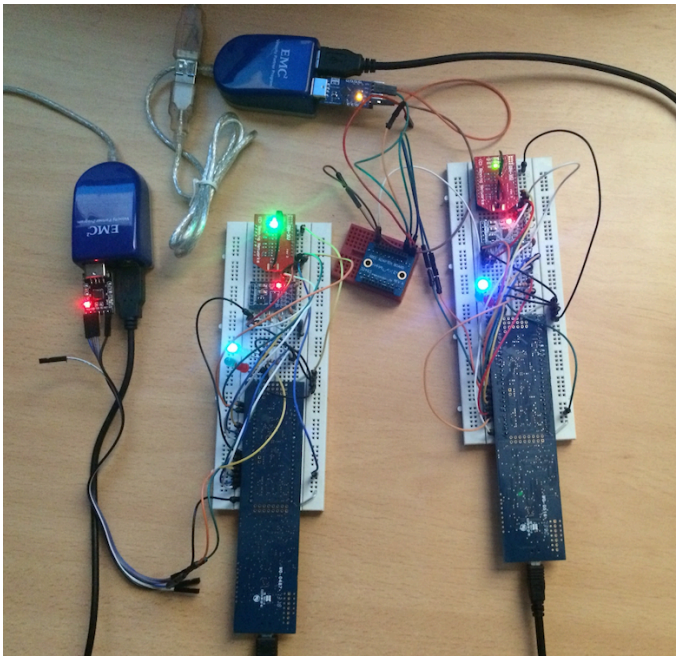


Ilustración 6: Equipos en funcionamiento

1.8 BREVE DESCRIPCIÓN DEL RESTO DE LOS CAPÍTULOS

A continuación se muestra un resumen del contenido del resto de las capítulos:

- Capítulo 2: Resumen de la situación de la tecnología utilizada en el proyecto y un análisis de mercado actual.
- Capítulo 3: Descripción funcional donde se mostrará una visión global del funcionamiento del sistema.
- Capítulo 4: Descripción detallada de los elementos hardware y software que componen el sistema y como interactúan entre si.
- Capítulo 5: Análisis de la viabilidad técnica del proyecto propuesto.
- Capítulo 6: Estudio de costes del proyecto.
- Capítulo 7: Conclusiones relativas al proyecto, revisión del nivel de cumplimiento de los objetivos propuestos y autoevaluación.
- Capítulo 8: Glosario de términos menos comunes o muy especializados.
- Capítulo 9: Bibliografía y fuentes de información utilizadas para la realización del proyecto.
- Capítulo 10: Anexos complementarios del proyecto.

2 ANTECEDENTES

2.1 ESTADO DEL ARTE

Un sistema embebido es un sistema de computación diseñado para realizar una o un reducido número de tareas específicas, generalmente en tiempo real. Suelen disponer de múltiples protocolos de comunicación, interfaces y puertos, en función del propósito para el que han sido creados, y que les permitirá interactuar con el mundo exterior.

Sus usos y aplicaciones son completamente diferentes a los que usualmente se les da a los ordenadores, diseñados pasa solucionar problemas de propósito general.

Dicho en otras palabras, un sistema embebido está basado en una electrónica programable gobernada por un microcontrolador, especialmente diseñado para soluciones específicas en tiempo real.

Consideramos un sistema en tiempo real aquel en que la respuesta a la tarea encomendada debe de ser exacta tanto a nivel lógico, como en el momento en que esta se produce. Si el sistema de control de aproximación de un avión, eligiera el rumbo correcto a seguir, pero 1 segundo antes o después del momento necesario, la catástrofe estaría servida.

Para que un sistema funcione en tiempo real, debe estar gobernado por un Sistema Operativo diseñado especialmente para trabajar en tiempo real, que garantice un tiempo de respuesta determinado (no necesariamente tiene que se rápido, sino disponer de la velocidad adecuada para cumplir puntualmente con la tarea asignada), aun en las circunstancias más desfavorables.

Aunque tradicionalmente los sistemas empotrados se han desarrollado al amparo de los sistemas de control industriales (control de motores, temperatura, presión etc., incorporados a sistemas más complejos), su evolución tecnológica (mayor potencia de computo, menor tamaño, menor consumo, etc.) y de costes, han hecho que actualmente los encontremos dentro de la gran mayoría de los dispositivos electrónicos que nos rodean en nuestra vida cotidiana.

Para hacernos una idea, citaremos unos pocos ejemplos de una lista interminable:

- Electrodomésticos (lavavajillas, hornos, lavadoras, microondas, etc.)
- Dispositivos multimedia (televisores, reproductores de música, blu-ray, consolas, etc.)
- Periféricos (routers, ratones, joysticks, tabletas gráficas etc.)
- Salud (termómetros electrónicos, medidores de tensión, marcapasos, pulsómetros, cuantificadores, etc.)

- Todo tipo de "Gadgets" y juguetes
- Teléfonos y sistemas de comunicación
- Domótica
- …

Por otro lado, la explosión del "Internet de las cosas" , "*wearables*", así como el nacimiento de plataformas de desarrollo de sistemas embebido asequibles con fines educativos para colegios, universidades y particulares (Arduino, Raspberry Pi, LPCXpresso, etc.), han propiciado el auge de los sistemas embebidos, que se han introducido en nuestros hogares para mejorar nuestra calidad de vida.

Ilustración 7: Arduino Uno

Ilustración 8: Raspberry Pi

Ilustración 9: LPC1769

2.2 ESTUDIO DE MERCADO

Cuando hablamos de iluminación inteligente, lo primero que nos viene a la cabeza son sistemas domóticos que encienden o apagan las luces de forma remota para ahorrar electricidad, por cuestiones de seguridad o para la comodidad del usuario.

Generalmente estos sistemas domóticos basados en un ordenador, controlan distintos dispositivos de nuestro hogares, limitándose a controlar la alimentación de las luces tradicionales.

El producto que se desarrolla en este proyecto va un poco más allá y a parte de las funciones arriba indicadas que se pueden implementar sin dificultad, pretende integrarse como un elemento más de la decoración de interiores o exteriores, gestionando la luz ambiente de nuestro entorno de forma sencilla y en una fase posterior, gestionarse desde un *smartphone.*

Como se comentaba en el apartado de justificación de este documento, las posibilidades son muy amplias, y los productos que existente actualmente con características similares, son relativamente escasos (sobre todo fuera de EEUU).

Tras distintas búsquedas en Internet, se han encontrado algunos productos con una vocación similar, aunque con una aproximación diferente.

El principal referente es Philips Hue.

Ilustración 10: Philips Hue

Philips, con una larga tradición en el mundo de los electrodomésticos y de la iluminación, ha traído muchas innovaciones en distintos campos y como no, en el de la iluminación inteligente. Por ejemplo, son los inventores del sistema Philips Ambilight, que ilumina el contorno del televisor con una luz ambiente similar al del contenido de la emisión, haciendo su visualización una experiencia más intensa.

Ilustración 11: Philips Ambilight

Hue es un sistema que se compone de un bridge que se conecta al router por un cable ethernet y que puede gestionar hasta 50 bombillas led especiales que soportan 16 millones de colores, aunque solo se pueden controlar 10 simultanemente. Se gestiona desde un dispositivo Android o IOs, o desde un navegador Web, y permite encender a voluntad o de forma programada caulquier bombilla del color deseado, en el momento deseado, de forma inalambrica, además de otras funciones avanzadas, como integrarse con el sistema Ambilight antes citado.

El paquete básico de Hue se compone de un bridge y 3 bombillas especiales y tiene un PVP de 199.99$. Cada bombilla independiente tiene un coste de 60$.

Ilustración 12: Paquete básico Hue

Existen otros fabricantes que han lanzado productos que intentan cubrir este segmento de mercado con soluciones con distintas caracteristicas y precios como son LIFX, Insteon, RoboSmart o GreenWave, e incluso gigantes como Samsung o LG han anunciado póximos lanzamientos de productos en este campo.

Una caracteristica comun de todos los dispositivos analizados es que la inteligencia reside en la aplicación del *smartphone* y que su arquitectura de conexión es en estrella (todos los dispositivos hablan con el Smarphone, pero no entre ellos).

A continuación se citan las caracteristicas principales y precios de los principales competidores de Philips Hue.

LIFX

- Soportan 16 millones de colores
- Gestiona hasta 100 dispositivos en una sola red
- Necesita de una red Wi-Fi existente
- Se gestiona desde el Smartphone conectado a la Wi-Fi

Ilustración 13: Bombilla LIFX

El precio de cada bombilla es de 99$.

Insteon

- Dispone de solo luz blanca
- Soporta 400 dispositivos por hub
- Necesita un *Hub* del propio fabricante
- Se gestiona desde el Smartphone conectado a la Wi-Fi

Ilustración 14: Bombilla y Hub Insteon

Su precio es de 130$ el Hub y de 30$ cada bombilla.

RoboSmart

- Dispone de solo luz blanca.
- Soporta múltiples dispositivos pero gestinados uno a uno
- No necesita infraestructura adicional
- Se gestiona desde el Smartphone por Bluetooth.

Ilustración 15: Bombilla RoboSmart

El precio de cada bombilla es de 40$.

GreenWave LED

- Dispone de solo luz blanca.
- Soporta hasta 500 dispositivos en una sola red
- Necesita un *bridge* del propio fabricante
- Se gestiona desde el Smartphone conectado a la Wi-Fi o por un mando a distancia

Ilustración 16: Paquete básico GreenWave

El paque de inicio se compone de un bridge y 4 bombillas especiales y tiene un PVP de 200$. Cada bombilla adicional cuesta 20$.

3 DESCRIPCIÓN FUNCIONAL

3.1 VISIÓN GLOBAL

Como se ha comentado en capítulos anteriores, nuestro objetivo es construir un sistema de iluminación que se gestione cambiado su orientación y que se sincronice con otros dispositivos similares de forma automática.

Desde un punto de vista conceptual, la resolución del problema se ha abordado dividiéndolo en dos partes o fases independientes, que finalmente se han integrado:

- Módulo gestión de iluminación que se activa y cambia de color en función de un cambio de orientación.
- Módulo de gestión de la comunicación y sincronización entre varios dispositivos.

Este sería diagrama de bloques básico:

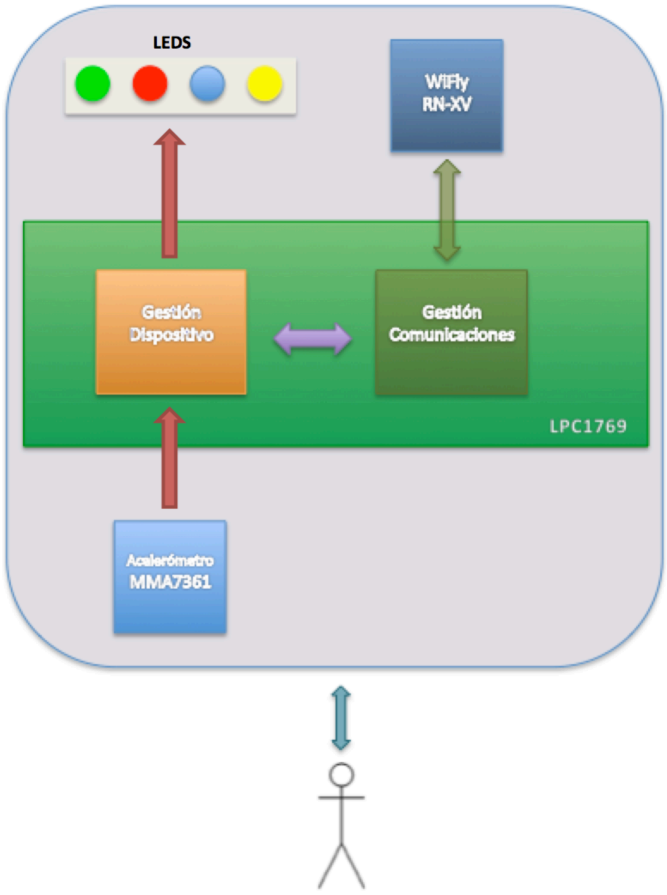


Ilustración 17: Diagrama de bloques básico

3.2 MÓDULO DE GESTIÓN DEL SISTEMA DE ILUMINACIÓN

Los requisitos que se han fijado y que debe de cumplir este bloque son los siguientes:

- Crear un módulo de iluminación ambiental de cuatro colores que funcione de forma autónoma.
- Activación y cambio de color basados en la variación de posición de cualquiera de los dispositivos, registrados por el acelerómetro incorporado.
- Funcionamiento simple y sin necesidad de configuración inicial por parte del usuario.

Para el encendido de los leds y su cambio de color se ha tomado la decisión de basarse en la acción sobre un sensor en lugar de un interruptor tradicional. La idea es dotarle de un sistema que sea fácil e intuitivo de usar, además de original. Después de barajar la opción de utilizar un sensor de ultrasonidos que activara el sistema en función de la proximidad o alejamiento, de por ejemplo la mano, siguiendo algún tipo de pauta, se ha optado por utilizar un acelerómetro que mida los cambios de posición del nodo y actué en consecuencia. El comportamiento del dispositivo es el de autómata finito determinista de 5 estados tal y como se muestra en la siguiente figura.

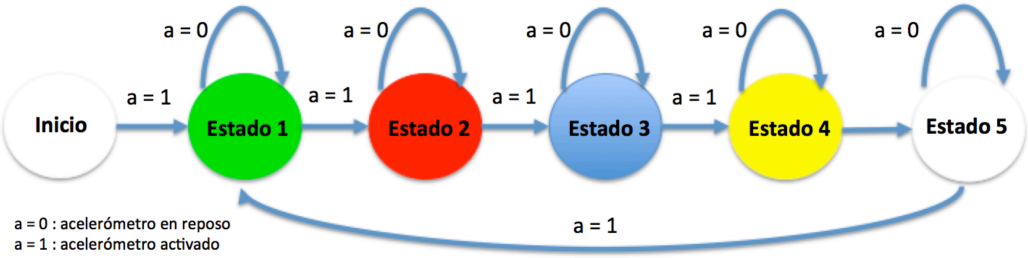


Ilustración 18: Diagramas de estados

Para su gestión intervienen dos procesos:

- El primero interactúa con el acelerómetro conectado al LPC mediante 3 canales ADC. Si detecta que las aceleraciones sufridas superan determinados umbrales configurables, calcula el siguiente estado del sistema y lo escribe en una cola.
- El segundo actúa cuando recibe una nueva orden en la cola, activando y desactivando los leds seleccionados por el proceso anterior, a través del los puertos GPIO.

A continuación se muestra el diagrama de bloques de los procesos indicados:

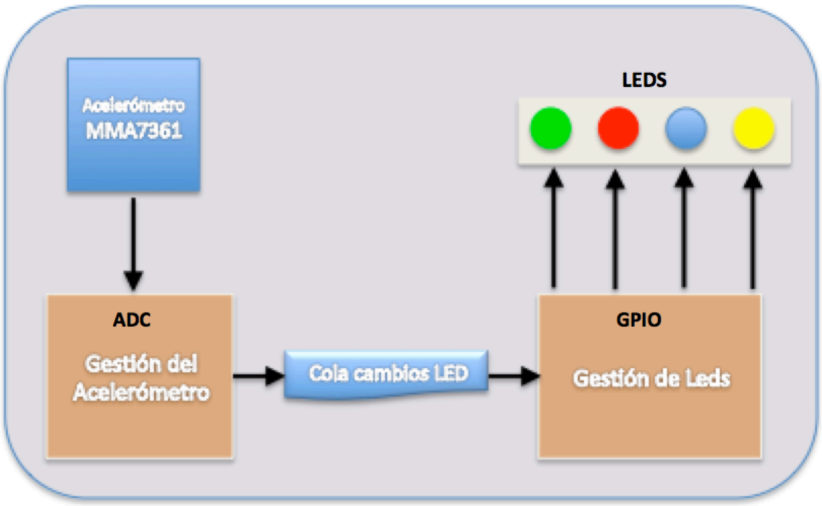


Ilustración 19: Diagramas de bloques gestión de la cola de cambios

3.3 MÓDULO DE GESTIÓN DE LAS COMUNICACIONES

Los requisitos que se han fijado y de debe de cumplir este bloque son los siguientes:

- Capacidad de interconexión inalámbrica con otros dispositivos similares para que actúen de forma síncrona, sin necesidad de infraestructura adicional.
- Soporte de interconexión escalable hasta 8 dispositivos, con la capacidad de incorporación automática a la red de los nuevos dispositivos.
- Funcionamiento simple y sin necesidad de configuración inicial por parte del usuario (autoconfigurables).
- Hardware y software idéntico en todos los dispositivos.
- Diseño compatible con funcionalidades avanzadas para evoluciones futuras, como la gestión desde Smartphone (fuera del alcance de este proyecto).

De nuevo tenemos dos procesos para la gestión del modulo:

- El primero se encarga de gestionar el WiFly que está conectado a 2 canales UART, así como de recibir y analizar los mensajes recibidos por el mismo mediante le protocolo UDP. En el caso de recibir un mensaje valido de cambio de estado, escribe en la cola un registro con dicho cambio.
- El segundo actúa cuando recibe una nueva orden de cambio en la cola, activando y desactivando, los leds indicados por el proceso anterior, mediante los puertos GPIO.

A continuación se muestra el diagrama de bloques de los procesos indicados:

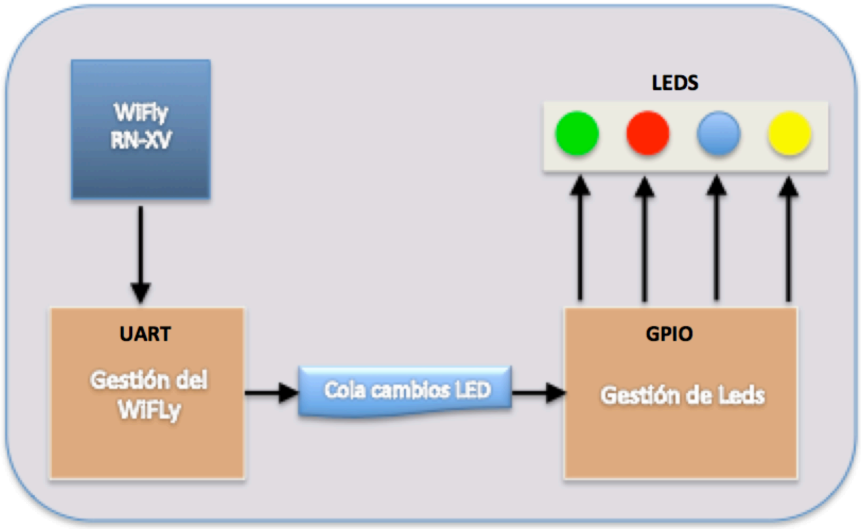


Ilustración 20: Diagrama de bloques gestión de las comunicaciones

3.4 INTEGRACIÓN DE MÓDULOS

A la hora de integrar ambos módulos y obtener el sistema con de toda la funcionalidad requerida, se han tomados las siguientes decisiones de optimización:

- Consolidar el proceso de gestión de leds para que gestione tanto peticiones locales como remotas.
- Utilizar una sola cola para gestionar todos los cambios de led, tanto los que son generados por un cambio en el acelerómetro, como los que se reciben a través del WiFly provenientes de otro dispositivo. Se añade un campo adicional a la cola para conecer cual es el origen del cambio.
- Añadir al proceso de "Gestión de Leds" la capacidad de enviar un mensaje de cambio de led a través del WiFly a los dispositivos remotos, mediante un broadcast UDP. Este proceso se realizará cuando la tarea reciba un mensaje de cambio de led local.
- Se añade en el diagrama del diseño el CP2102 para la opción de debug de las tareas.

Este es el diagrama de bloques final tras la integración de ambas fases.

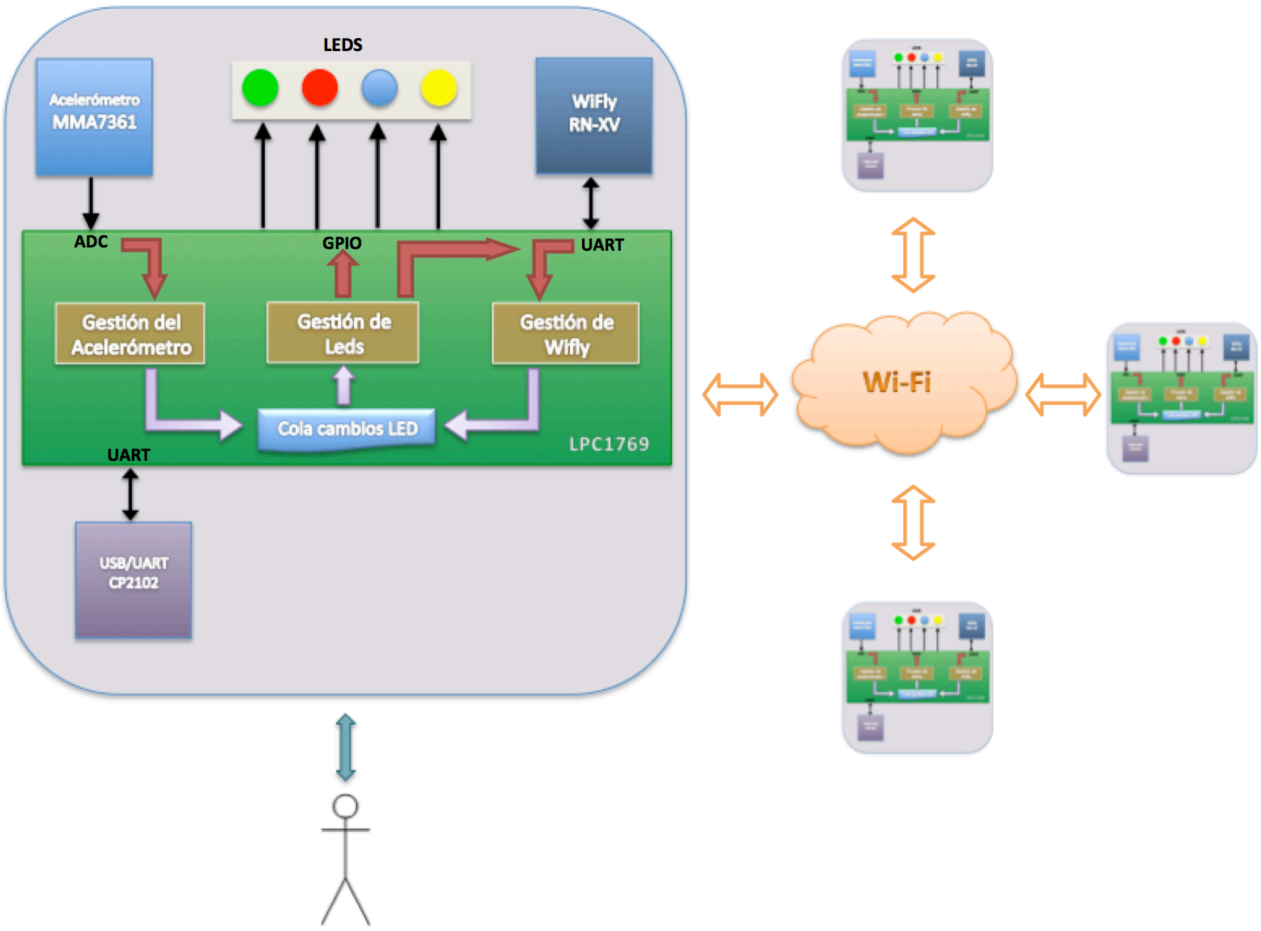


Ilustración 21: Diagrama de bloques del sistema integrado

Como se puede observar en el gráfico, los tres procesos o tareas que gestionan el dispositivo, "Gestión del acelerómetro", "Gestión de leds" y "Gestión de WiFly", interactúan con el exterior mediante distintos drivers que manejan los diferentes canales e interfaces del sistema. Estos componentes se describen en detalle en el siguiente apartado.

3.5 OPERACIONES DE ENTRADA/SALIDA

El funcionamiento del sistema se basa en las interacciones que es capaz de mantener con su entorno, tanto a la hora de tener conocimiento de lo que ocurre a su alrededor, como de responder a los estímulos recibidos con determinadas respuestas. En definitiva las operaciones de entrada/salida que pueda realizar que pueda realizar el dispositivo.

Como ya se ha explicado anteriormente, los elementos con los que se equipa cada dispositivo para relacionarse con su entorno físico, son un acelerómetro, un Wifly, un CP2102 y un grupo de leds.

A continuación se describe como se gestiona cada uno de ellos a nivel funcional y como se conectan al LPC1769.

3.5.1 Gestión del Acelerómetro

El acelerómetro se gestiona mediante el ADC (Analog-Digital Converter). El ADC convierte una señal de entrada analógica en una señal digital. El LPC1769 dispone de un ADC de 12 bits de resolución y de 8 pines de entrada. Se han utilizado 3 pines para conectar el acelerómetro (ejes X, ,Y Z, canal AD0.0, AD0.1 y AD0.2 respectivamente.

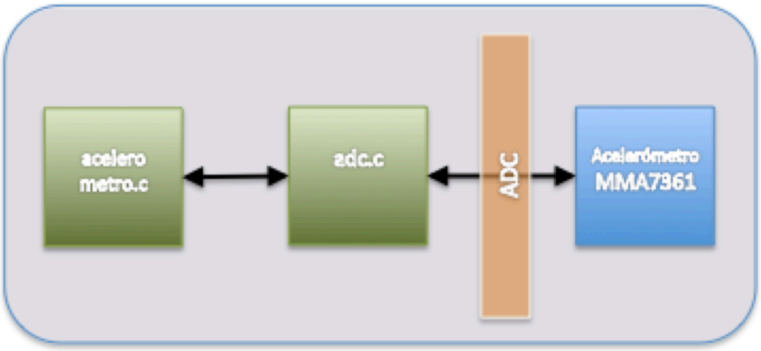


Ilustración 22: Diagrama de bloques de la gestión del acelerómetro

Como se puede observar en el diagrama de bloques, el puerto ADC se gestiona mediante el driver adc.c, mientras que el driver acelerometro.c implementa los comandos específicos para la gestión del acelerómetro e interactúa con adc.c.

3.5.2 Gestión de los leds

Los leds se gestionan mediante los puertos GPIO (General Purpose Input-Output). Los GPIO son pines de uso general de señales digitales dispuestos a modo de puertos que se pueden configurar como entrada o como salida, obteniendo una entrada lógica o generando una salida lógica. Para los leds del sistema se han utilizado los GPIO P0[9], P0[8], P0[7] y P0[6].

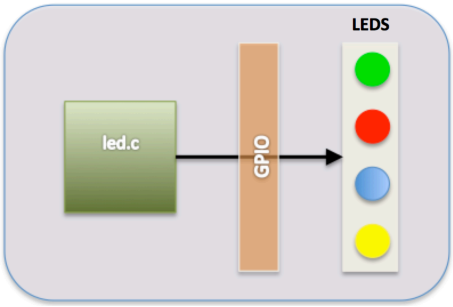


Ilustración 23: Diagrama de bloques de la gestión del leds

El driver led.c se encarga directamente de gestionar los puertos GPIO del dispositivo al que están conectados los leds.

3.5.3 Gestión de WiFly y CP2102

Tanto el Wifly como el CP2102 se gestionan mediante los puertos UART (Universal Asynchronous Receiver-Transmitter). La UART transforma la información de formato paralelo a formato serie y viceversa para facilitar su transmisión de un dispositivo a otro, añadiendo los bits necesarios para su control durante la misma. Para manejar esta comunicación se suele utilizar el estándar RS-232.

El LPC1769 tiene cuatro UART que manejan los puertos serie, dos de las cuales se han utilizado para este proyecto.

El WiFly utiliza la UART0 (RX P0[3] y TX P0[2]) y el CP2102 la UART3 (RX P0[0] y TX P0[1]).

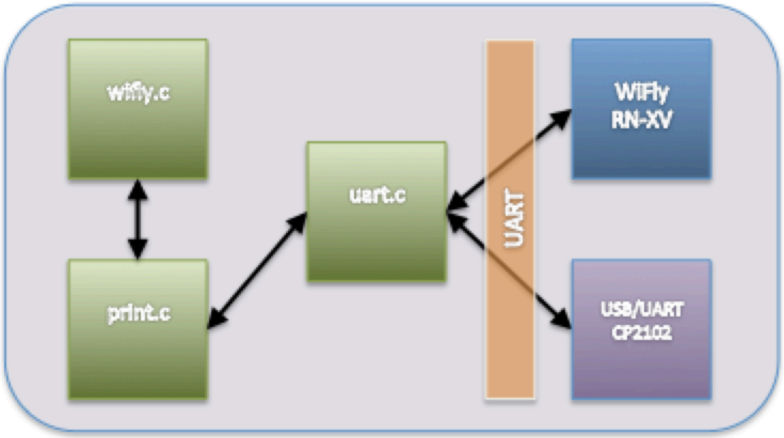


Ilustración 24: Diagrama de bloques de la gestión la UART

Como se puede observar en el diagrama de bloques, la UART se gestiona a bajo nivel mediante el driver uart.c. El driver printf.c se encarga de simplificar las operaciones de entrada/salida de la UART, así como de añadir el fin de línea y el retorno de carro cuando son necesarios.

El driver wifly.c implementa la operaciones y comandos necesarios para gestionar el WiFly a través de printf.c y uart.c.

4 DESCRIPCIÓN DETALLADA

4.1 CONSIDERACIONES DE DISEÑO

A lo largo de este capítulo se describen en detalle los elementos más relevantes de hardware y de software utilizados para la construcción de los dos prototipos, y las decisiones de diseño tomadas para conseguir los objetivos marcados.

Para la elección del hardware, se ha tomado como punto de partida los componentes proporcionados por la UOC durante la fase de aprendizaje, adaptando las funcionalidades del proyecto a los mismos. Por ejemplo, se podría haber optado por módulos *bluetooth* para las comunicaciones, en lugar de los WiFly o sensores ultrasónicos para activar los cambios de estado de los leds, en lugar de los acelerómetros, y conceptualmente, el proyecto hubiera sido similar.

También nos hemos basado en las librerías de código de CodeRed y en las desarrolladas durante la fase de aprendizaje, siguiendo las pautas de la xWiki del "Embedded Systems Lab" de la UOC, para construir la solución software.

Un punto crítico del proyecto ha sido la elección del protocolo a utilizar y la topología de las comunicaciones entre nodos, ya que no se quería renunciar a la posibilidad de que la red la formaran más de dos dispositivos y diseñar una solución de más de dos dispositivos añadía mucha complejidad al proyecto.

Para decidir el modo en que el sistema gestiona la comunicaciones, se ha realizado un análisis en profundidad de las capacidades del módulo WiFly, para poder seleccionar la más equilibra en cuanto a simplicidad y potencia.

Durante dicho estudio, se descubrió dos aspectos importantes que han pesado mucho a la hora de tomar algunas decisiones de diseño:

- El primero es que en las versiones más modernas de firmware de Wifly, existe la posibilidad de que un WiFly trabaje como punto de acceso Wi-Fi 11g, soportando por *DHCP* hasta 8 dispositivos (1 Access Point y 7 clientes), siendo además compatible con IOs y Android, siendo esto último muy útil para evoluciones futuras.
- El segundo es que los Wifly solo son capaces de mantener una conexión (dirección IP + puerto) simultanea, por lo que mantener una comunicación y un intercambio permanente de mensajes entre varios dispositivos a la vez (sobre todo si buscamos unos tiempo de respuesta agiles), sin un servidor central que, en una arquitectura en estrella, coordinara a todos los dispositivos hubiera sido muy complejo.

Una opción que se consideró fue la de montar una arquitectura en anillo, tipo Chord, en la que cada dispositivo estuviera alternativamente conectándose con el elemento anterior y el siguiente del anillo, y que fuera pasando los mensajes por los distintos nodos del anillo para hacer llegar un mensaje al otro extremo de la red, aunque de nuevo, resultaba demasiado complejo.

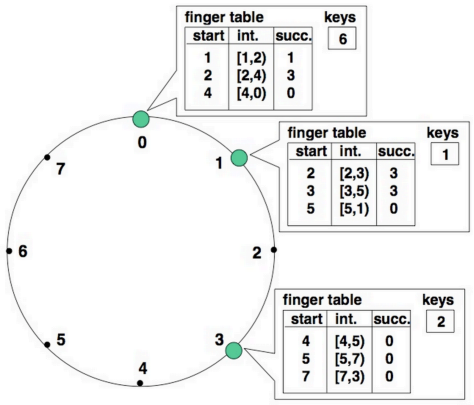


Ilustración 25: Ejemplo de topología Chord

Otra posible alternativa era la de limitar a 2 el número de dispositivos de la red, algo que hubiera condicionado demasiado la funcionalidad del sistema.

Tras un análisis en profundidad de todos los comandos y modos de funcionamiento del WiFly y tras realizar una serie de pruebas de concepto, se establece que la opción optima para las necesidades actuales, es la de utilizar el protocolo UPD (no orientado a la conexión), junto con la funcionalidad de *broadcast* disponible en los WiFly que utiliza para auto-descubrir y publicar los dispositivos dentro de la red.

La limitación a 8 dispositivos viene dada por la características del servidor *DHCP* del punto de acceso virtual de los WiFly , que en la versión actual (v4.41), solo asigna 7 direcciones IP. En el caso de asignar direcciones IP de modo manual a los dispositivos (por ejemplo de fábrica) o la de utilizar una Wi-Fi existente basada en un punto de acceso de un tercero, se elevaría notablemente el número de nodos posibles, siendo igualmente valido el sistema de comunicación basado en UPD y *broadcast* diseñado inicialmente.

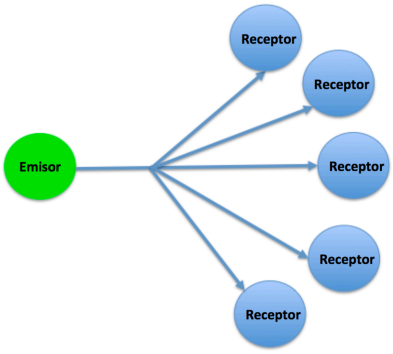


Ilustración 26: Sistema de distribución broadcast

Otra decisión de diseño relevante ha sido la forma de que el usuario interactue con el dispositivo a la hora activar los cambios de estado.

La idea original era la de anotar un cambio de estado cuando el dispositivo se girara 90º sobre si mismo gracias al acelerómetro incorporado, y que fuera rotando como un cubo en cambios sucesivos. En la práctica, la inestabilidad del sistema montado sobre la *breadboard* y los constantes fallos del hardware por falsos contactos del cableado al mover el aparato (han sido muchas horas de pruebas), han hecho que finalmente se decidiera contabilizar un cambio de estado al detectar un giro en cualquier dirección, para luego volver a la posición natural de reposo del dispositivo, totalmente estable.

Esta decisión disminuyo el numero de errores hardware producidos de forma importante, aunque no se eliminaron totalmente. El hecho de tener que mover constantemente el dispositivo para las pruebas y de no tener el cableado soldado, hacia las conexiones muy inestables.

4.2 DESCRIPCIÓN DEL HARDWARE

En este apartado se describen lo elementos hardware utilizados para la construcción de los dos prototipos.

El hardware recibido para la realización del PFC por parte de la UOC es el siguiente:

- 1 x Placa de desarrollo LPCXpresso LPC1769
- 1 x Módulo WiFly RN-XV con adaptador Xbee
- 1 x Módulo CP2102 USB/UART
- 1 x Sensor acelerómetro MMA7361
- 10 x cables

Para la realización del proyecto se han construido 2 prototipos idénticos, aprovechando el material recibido, y adquiriendo el material restante a distintos proveedores.

El material final utilizado en cada uno de ellos consta de:

- 1 x Placa de desarrollo LPCXpresso LPC1769
- 1 x Módulo WiFly RN-XV con adaptador Xbee
- 1 x XBEE Adapter Board Wifly-Breadboard
- 1 x Módulo CP2102 USB/UART
- 1 x Sensor acelerómetro MMA7361
- 3 x Resistencias
  - 1 x 10K Ohms
  - 2 x 300 Ohms
- 3 x condensadores de 3.3 nF

- 4 x leds
  - 1 x verde
  - 1 x rojo
  - 1 x azul
  - 1 x amarillo
- 1 x Breadboard
- 20 Cables

A continuación se muestra un diagrama de conexiones detallado de los prototipos que se han utilizado para realizar el proyecto.

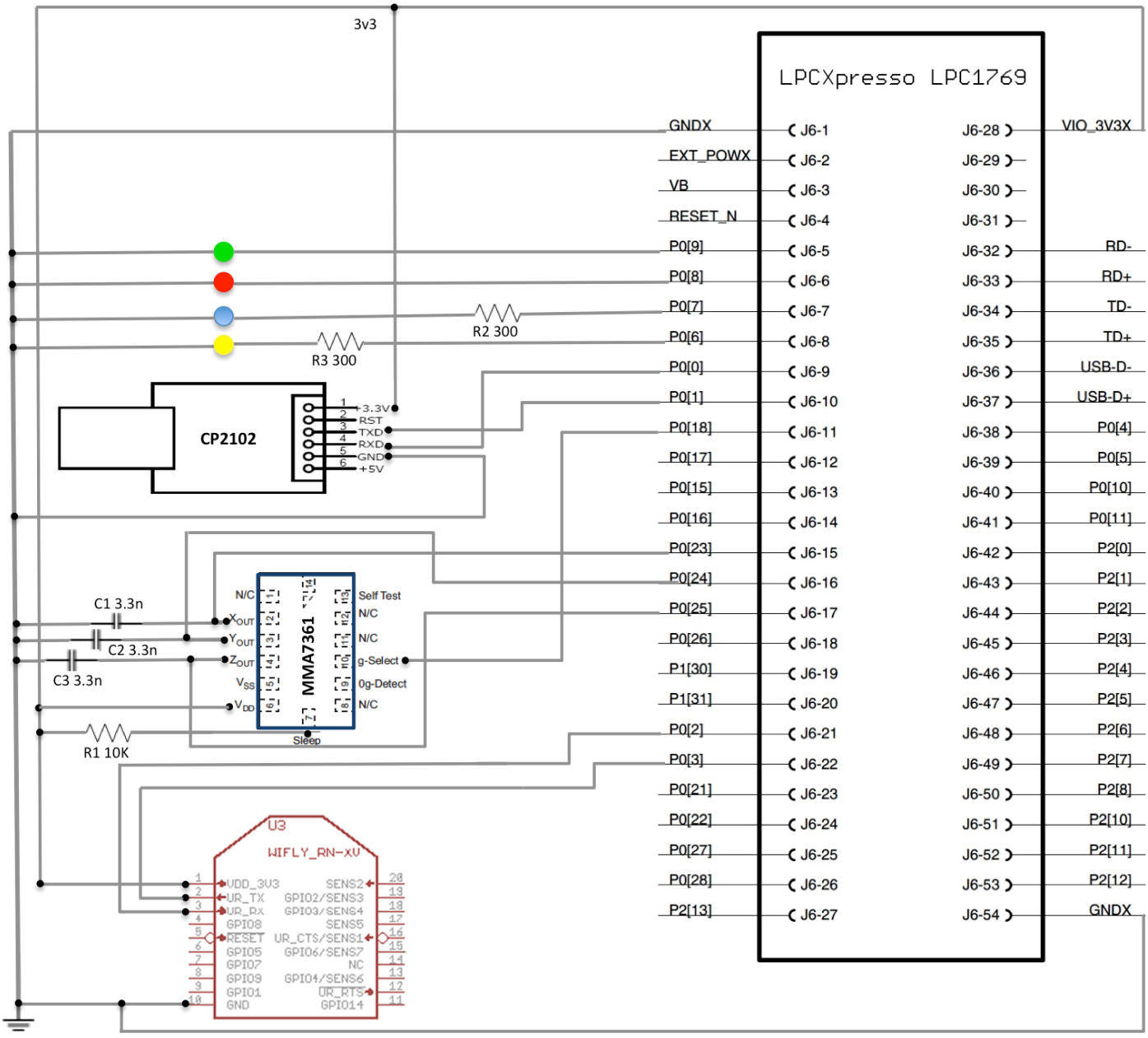


Ilustración 27: Diagrama de conexiones de los prototipos

En la siguiente fotografía se muestran los dos prototipos montados sobre unas *breadboards*:

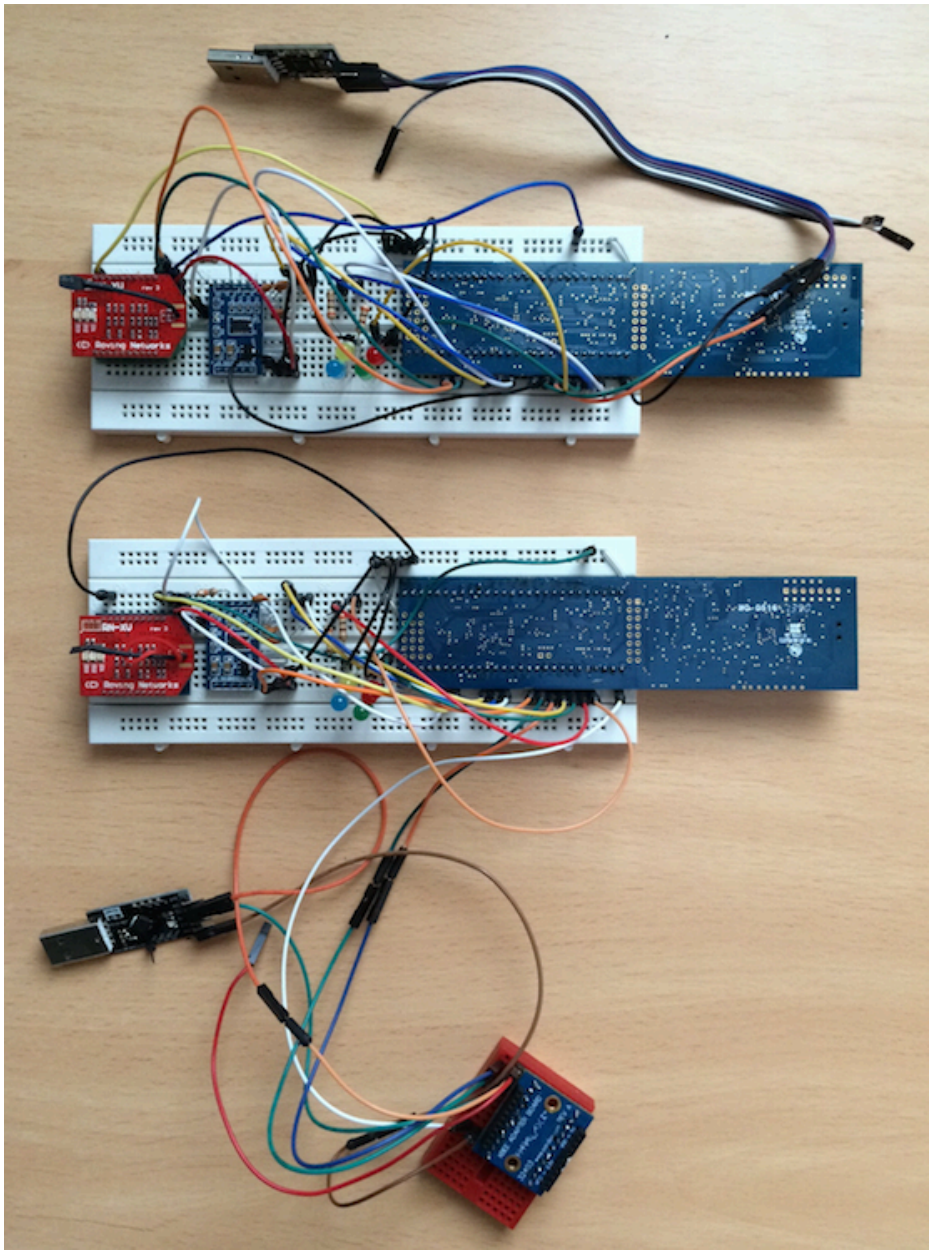


Ilustración 28: Prototipos fabricados

Dada la complejidad encontrada a la hora de realizar las pruebas con los dos prototipos, se ha creado también un modulo adicional con un zócalo para el WiFLy sobre una pequeña *breadboard* y un XBEE Adapter, conectado de forma paralela a uno de los conversores USB/UART, que permitía actuar directamente sobre los WiFly y programarlos o hacer debug de las comunicaciones, directamente desde un terminal Tera Term.

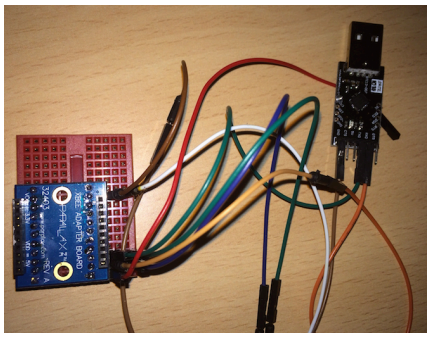


Ilustración 29: Módulo de pruebas Wifly

En los siguientes apartados, se describen en detalle los componentes hardware más relevantes utilizados para la elaboración de los prototipos.

4.2.1 LPCXpresso LPC1769

Es un placa de desarrollo fabricada por NPX, basada en un microcontrolador ARM Cortex-M3 de 32 bits con 64kB de memoria y 512 KB de memoria flash, que trabaja a 120 MHz de velocidad de reloj, además de disponer soporte USB 2.0 (*host* y *device*) y *Ethernet*.

Otras características importantes del LPC1769 son:

- 4 puertos UART
- 1 controlador DMA de 8 canales de propósito general
- 2 canales CAN
- 2 controladores SSP
- 1 interfaz SPI
- 8 canales de 12 bit ADC
- DAC de 10 bits
- 70 pines GPIO de propósito general

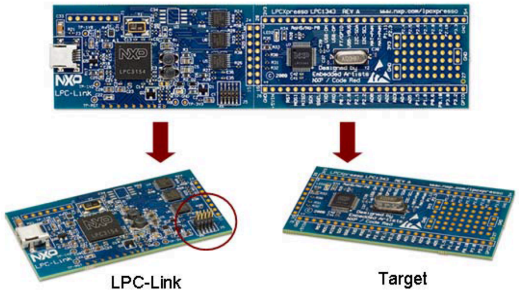


Ilustración 30: Placa de desarrollo LPCXpresso

En el diagrama mostrado a continuación podemos ver la arquitectura LCPXpresso.

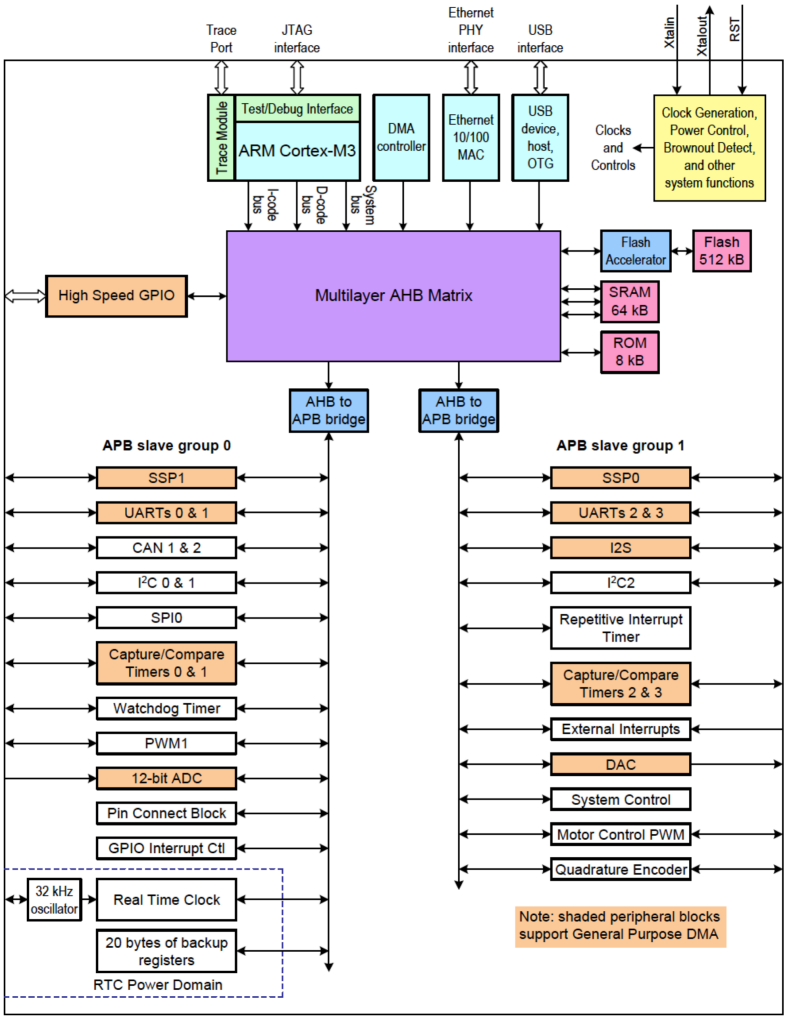


Ilustración 31: Diagrama de bloques del LPC1769

En la siguiente dirección del fabricante se puede consultar su *datasheet* :

http://www.nxp.com/documents/data_sheet/LPC1769_68_67_66_65_64_63.pdf .

4.2.2 Módulo WiFly RN-XV con adaptador Xbee

El WiFly RN-XV es un dispositivo que permite el envío y la recepción de datos mediante Wi-Fi en una red que soporte el protocolo 802.11 b/g y WEP/WPA y WPA2.

Este módulo fabricado por Roving Networks, integra un chip Wi-Fi RN-171 con un modulo de radio 802.11 b/g, un procesador de 32 bits, un stack TCP/IP, un reloj de tiempo real, un crypto acelerador, una unidad de gestión de potencia y una interface analógica. El módulo tiene preinstalado un firmware que simplifica la integración y minimiza el tiempo de desarrollo, soportando una configuración de hardware muy

simple, donde inicialmente solo son necesarias cuatro conexiones PWR, TX, RX y GND, para comenzar a utilizarlo.

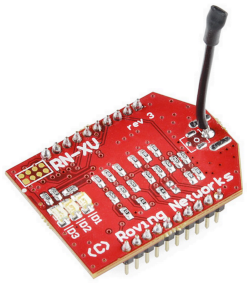


Ilustración 32: WiFLy RN-XV

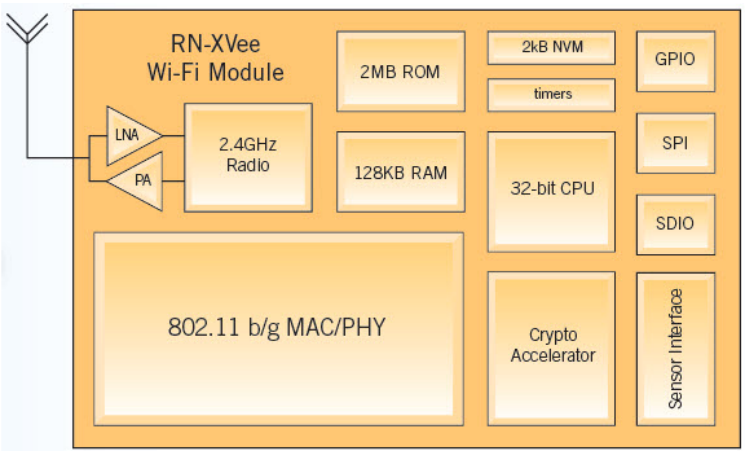


Ilustración 33: Diagrama de bloques del WiFly RN-171

El módulo WiFly tiene dos modos de funcionamiento:

- Modo de datos
- Modo de comandos

En modo de datos, el módulo puede aceptar conexiones entrantes o iniciar conexiones salientes.

Para configurar los parámetros y ver la configuración actual, se debe poner el módulo en el modo de comandos (también llamado modo de configuración) enviando tres caracteres "$".

Tras hacer un análisis en profundidad de la funcionalidades soportadas por este dispositivo, para la realización del proyecto hemos actualizado el firmware a la versión **4.41 11/25/2013, BUILD r1046**, la más reciente disponible en la actualidad, que incorpora la funcionalidad de permitir operar como un punto de acceso Wi-Fi o *apmode*. Esto va a permitir que uno de los dispositivos se comporte como punto de acceso Wi-Fi de cara al resto de los dispositivos, proporcionando una dirección IP al

resto, mediante *DHCP*. Esta versión de firmware, soporta hasta 8 dispositivos conectados en *apmode*. Este modo de funcionamiento nos permitirá en un futuro conectarnos al WiFLy mediante un Smartphone IOs o Android (ambos S.O. están soportados).

Hubiera sido trivial conectar los WiFly a la Wi-Fi local, con la ventaja evidente de escalabilidad en el número de dispositivos, así como la posibilidad de dotar de acceso a Internet a los mismos. En cambio, esta opción hubiera obligado a depender de una infraestructura (tanto hardware como de cobertura) adicional y a configurar cada dispositivo en función de red Wi-Fi utilizada, algo que hubiera añadido complejidad de manejo de cara al usuario.

Respecto a las comunicaciones entre dispositivos, por las características de la aplicación desarrollada en este proyecto, el protocolo de comunicaciones elegido es UDP.

En la siguiente dirección del fabricante se puede consultar su *datasheet* :

http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/WiFi/WiFly-RN-XV-DS.pdf

Manual de referencia de la versión 4.41 11/25/2013, BUILD r1046

http://ww1.microchip.com/downloads/en/DeviceDoc/50002230A.pdf

4.2.3 Módulo CP2102 USB/UART

Este modulo basado en el chip CP2102 fabricado por Silicon Labs, permite la comunicación bidireccional entre nuestro PC (USB) y un dispositivo que disponga de puertos UART, en nuestro caso el módulo WiFly o la placa LPC1769.

Conectado al puerto USB del ordenador, las salidas disponibles son PWR, TX, RX y GND.

Ilustración 34: Módulo CP2102

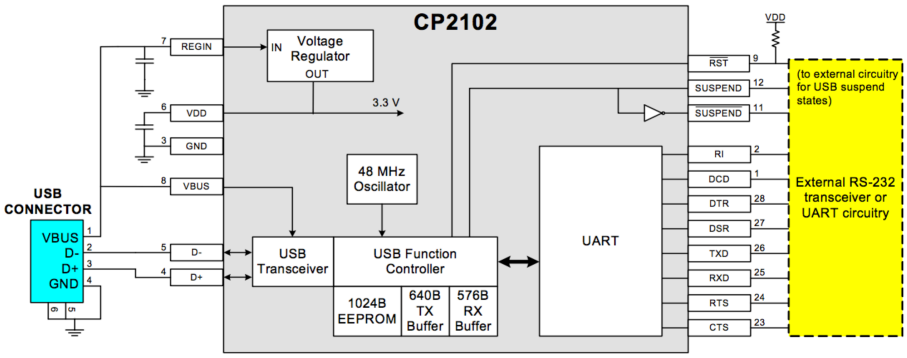


Ilustración 35: Diagrama de bloques del CP2102

En la siguiente dirección del fabricante se puede consultar su *datasheet* :

http://www.silabs.com/Support%20Documents/TechnicalDocs/CP2102-short.pdf

4.2.4 Sensor acelerómetro MMA7361

Este sensor dispone en su interior del integrado MMA7631 de la compañía Freescale, un acelerómetro de 3 ejes (x,y,z) de salida analógica, el cual detecta una fuerza máxima de hasta 6g.

Es un dispositivo capacitivo de bajo consumo (400 µA) y que soporta modo *sleep*. Dispone de una alta sensibilidad (800 mV/g a 1.5g), además de tener sensor de caída libre y la funcionalidad de autodiagnóstico.

Ilustración 36: Acelerómetro MMA7361

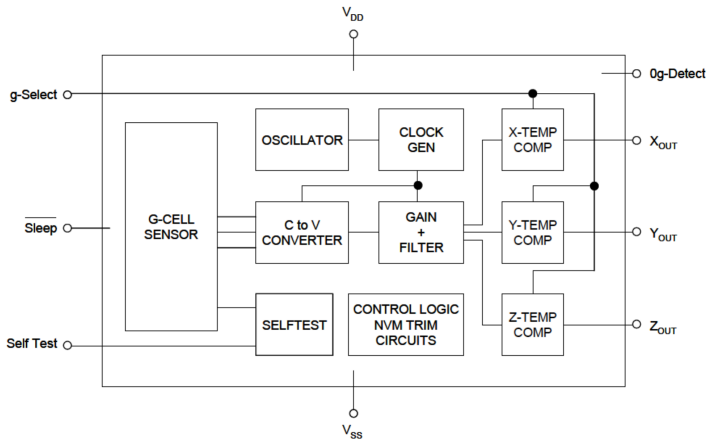


Ilustración 37: Diagrama de bloques del MMA7361

En la siguiente dirección del fabricante se puede consultar su *datasheet* :

http://cache.freescale.com/files/sensors/doc/data_sheet/MMA7361LC.pdf?pspll=1&Parent_nodeId=1238111946420745867479&Parent_pageType=product

4.2.5 Breadboard y Cableado

Para facilitar el montaje rápido de los prototipos se han utilizado varias placas *breadboard* o *protoboard*. Las *breadboard* son placas especiales compuestas por bloques de plástico con una serie de orificios conectados entre si por líneas de conducción paralelas.

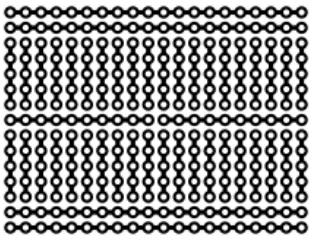
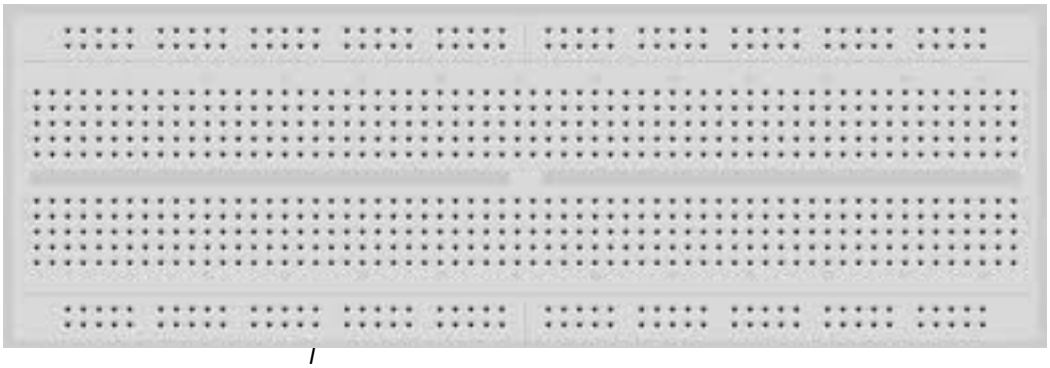


Ilustración 38: Ejemplo de las conexiones internas de una breadboard

Para el montaje de los dos prototipos se han utilizado 2 *breadboars* de 840 conexiones.



I
lustración 39: Breadboard de 840 pines

Para el montaje del módulo de pruebas WiFly una *breadboard* de 170 pines.

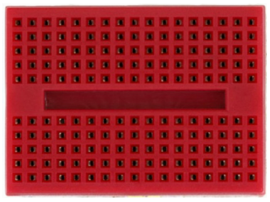


Ilustración 40: Breadboard de 170 pines

Para la interconexión de los distintos componentes se han utilizado cables especiales para *breadboard* de diferentes tamaños.

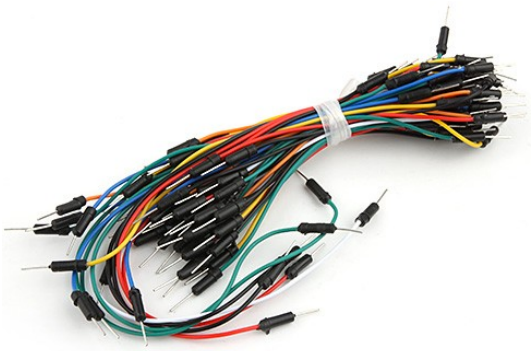


Ilustración 41: Cables breadboard

4.3 DESCRIPCIÓN DEL SOFTWARE

4.3.1 Casos de uso

Uno de los objetivos fundamentales era conseguir la máxima simplicidad del sistema. En el caso de uso adjunto, se puede apreciar que existe un solo tipo de usuario y como interactúa con el dispositivo.

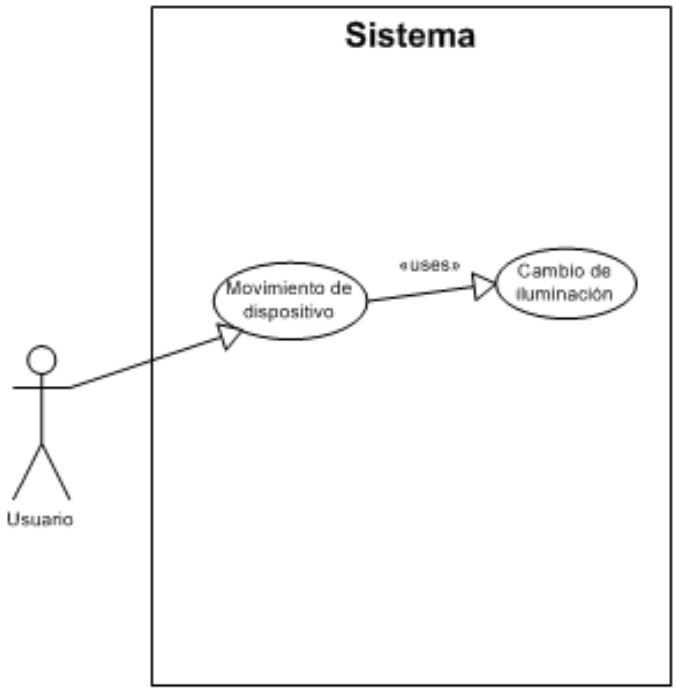


Ilustración 42: Diagrama de casos de uso

En una futura evolución, donde tuviera cabida la gestión remota desde un *smartphone*, tendríamos que ampliar los casos de uso con nuevos usuarios y nuevos usos, como la manipulación remota o la configuración de dispositivos.

4.3.2 Diagrama de bloques detallado

El diagrama de bloques detallado permite tener una visión completa de todos los elementos de la aplicación y como interactúan entre si.

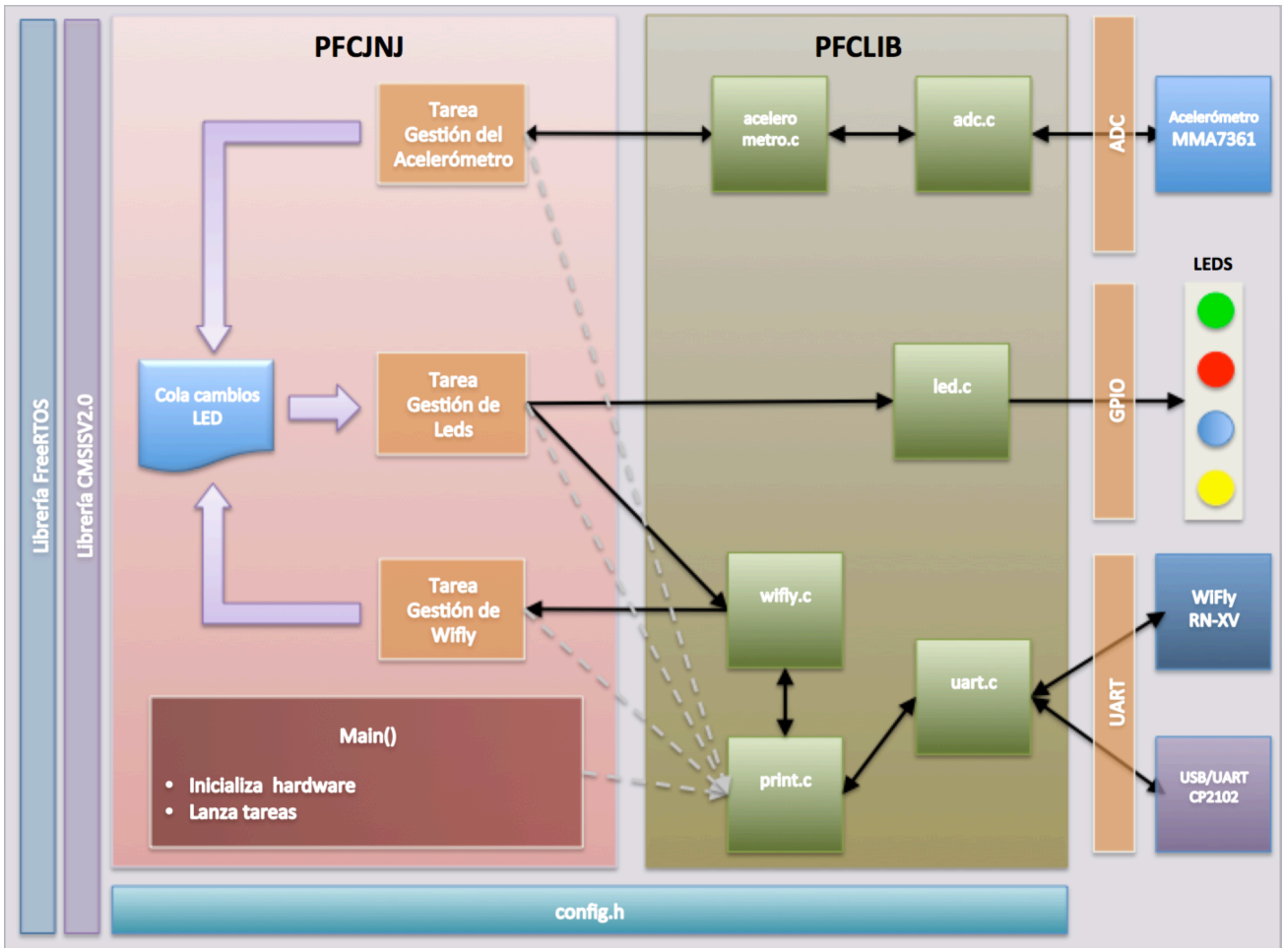


Ilustración 43: Diagrama de bloques detallado

Para el desarrollo de la aplicación se ha trabajado con un Workspace organizado en los proyectos que se describen a continuación:

- Proyectos del sistema
  - CMSISv2p00_LPC17xx
  - FreeRTOS_Library

- Proyectos del PFC
  - PFCJNJ: Proyecto de la aplicación
  - PFCLib: Librería de drivers de la aplicación

Dentro del proyecto de la aplicación PFCJNJ tenemos el fichero main.c en el que se ubica el código especifico que implementa las distintas funcionalidades del sistema y en el que se han definido 3 tareas:

- vGestionWifly
- vGestionAcelerometro
- vGestionLeds

Estas tres tareas intercambian la información para el control de leds mediante la cola xCambioLedQueue.

Tanto el código propio del main, como las tareas, hacen uso de los drivers definidos en la librería PFCLIB para gestionar todos los dispositivos y recursos del sistema.

Dentro de los distintos archivos que encontramos dentro de la librería relacionados con la aplicación para su compilación, depuración y ejecución, destacan los ".h" y ".c", que son las cabeceras y fuentes, respectivamente.

- acelerometro.c
- adc.c
- led.c
- printf.c
- uart.c
- wifly.c

Cada una de estos fuentes tiene su correspondiente cabecera ".h".

Destacamos la utilización de archivo cabecera config.h , como fichero de configuración, permitiendo de manera fácil y rápida, modificar distintos parámetros del software para configurar o modificar el comportamiento del sistema, o para activar la opción de debug y hacer seguimiento de la ejecución por consola. Las modificaciones efectuadas en este archivo tendrán efecto una vez compilada de nuevo la aplicación.

En los siguientes apartados pasaremos a explicar los diagramas de flujo del programa principal main() y de las tareas que lo integran, además de una explicación de los detalles mas relevantes de los distintos drivers.

4.3.3 Función main()

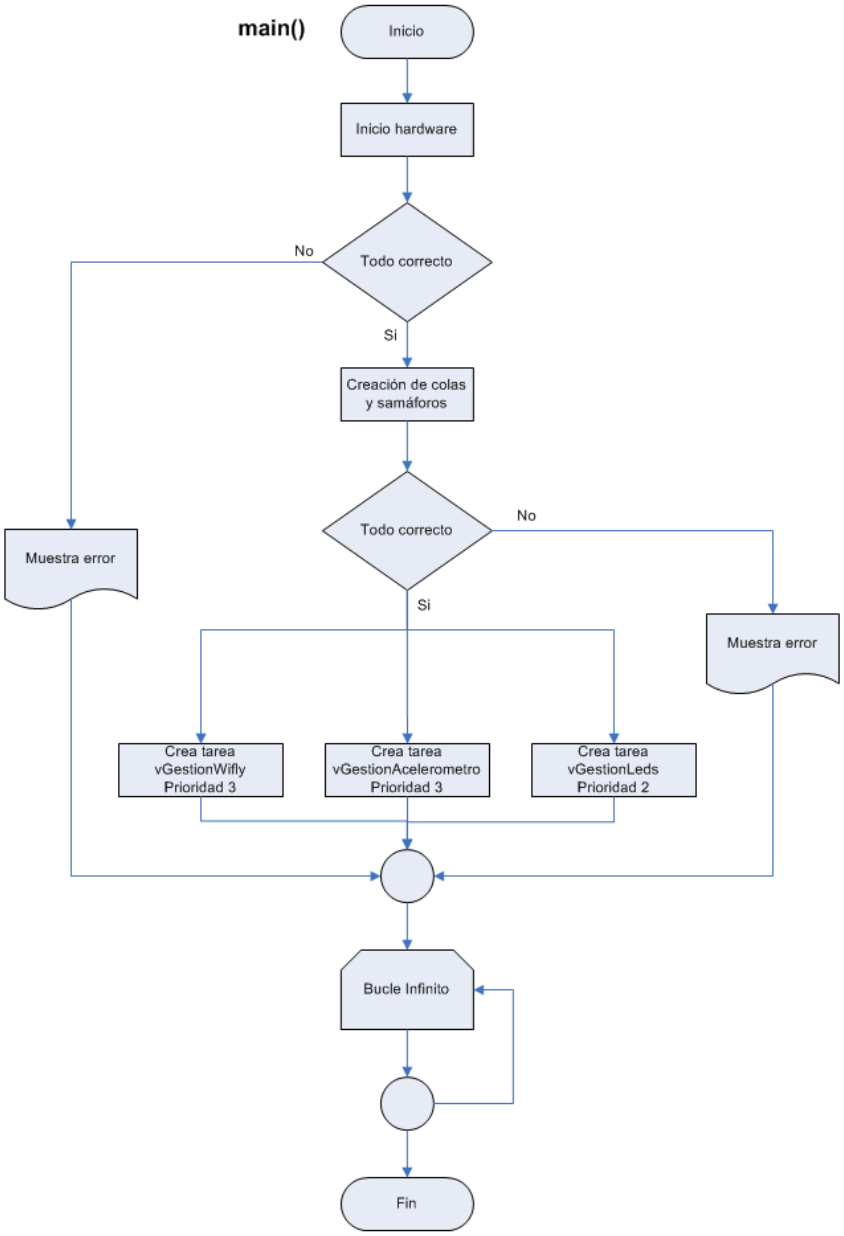


Ilustración 44: Diagrama de flujo de main()

Tras la inicialización de los distintos elementos hardware del sistema, bien directamente, bien llamando a las funciones o drivers adecuados, si no se producen errores, se crea la cola de control de leds y el semáforo que servirá para gestionar el envió y recepción de mensajes a través del Wifly.

El siguiente paso, si todo es correcto, es crear las tres tareas que son el corazón del sistema empotrado.

Desde el punto de vista de utilización de cola, dos de las tareas actuarán como productores (vGestionWifly y vGestionAcelerometro) y una como consumidora (VGestionLeds).

La cola xCambioLedQueue, se define con una capacidad de un solo elemento que contendrá la siguiente estructura de datos:

```
typedef struct {
	int origen;
	int led;
} Led_Reg;
```

Las tareas productoras vGestionWifly y vGestionAcelerometro, se crean con una prioridad mayor que la tarea consumidora vGestionLeds.

Una vez lanzado el planificador el programa entra en un bucle infinitito.

4.3.4 Función vGestionAcelerometro

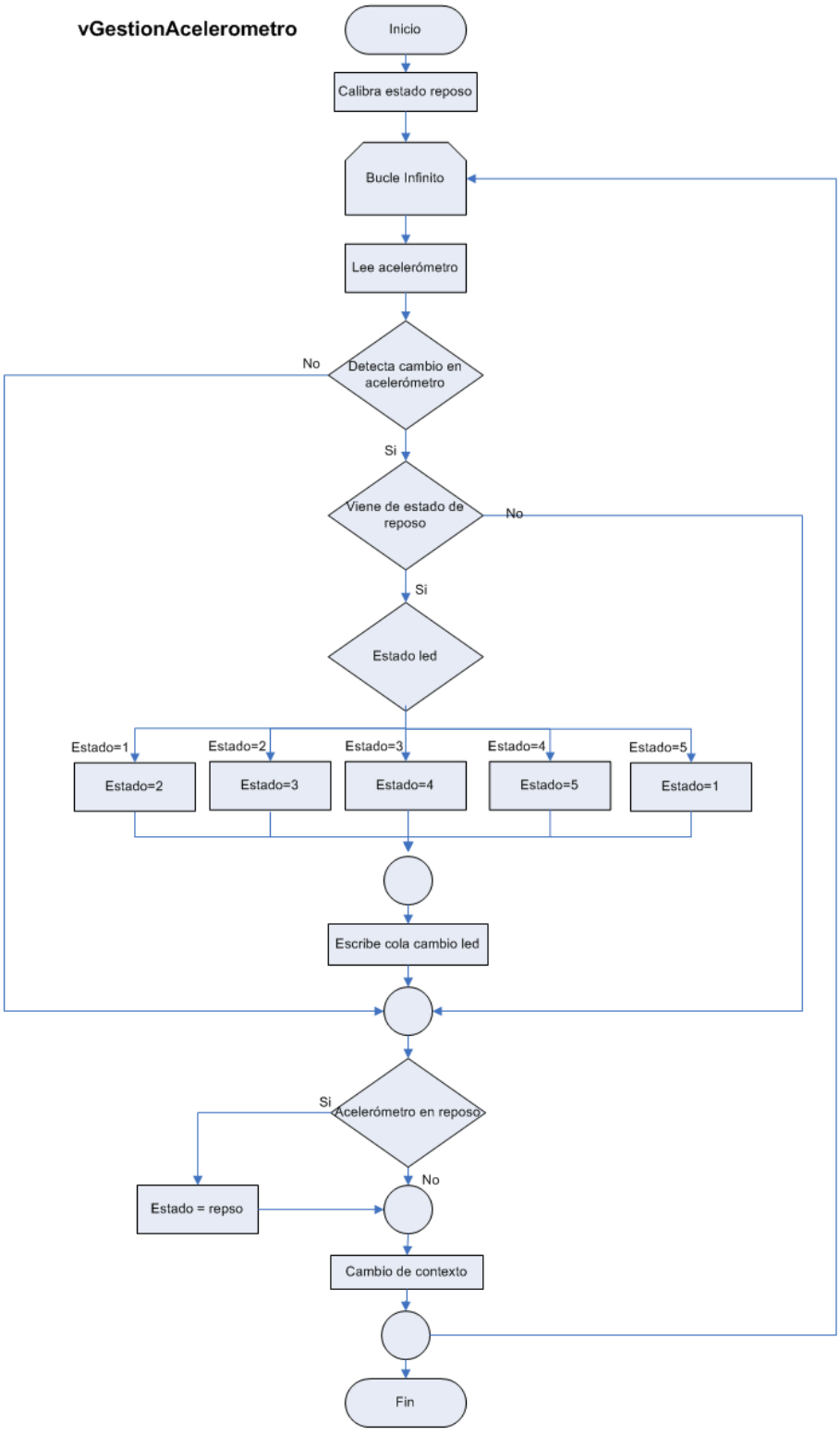


Ilustración 45: Diagrama de flujo de vGestionAcelerometro

Esta tarea se encarga de tomar lecturas del acelerómetro cada cierto tiempo y de escribir un registro en la cola cambios de leds si detecta que se ha producido un movimiento dentro de los márgenes preconfigurados en config.h.

Lo primero que hace la primera vez que se ejecuta es calibrar el acelerómetro tomando como referencia el estado de reposo del mismo.

Posteriormente entra en un bucle infinito que, en cada iteración hace una llamada a una función del driver del acelerómetro que se encarga de leer las coordenadas X, Y y Z devolviendo la media de 100 lecturas consecutivas.

El dato recibido se coteja frente al estado de reposo obtenido en el proceso de calibrado, en función de una horquilla de valores que vienen determinadas por un offset que se define en el fichero config.h y que marca la sensibilidad de acelerómetro.

Dado que el sistema trabaja a gran velocidad y para evitar que se detecte múltiples cambios de estado consecutivos en un solo movimiento produciéndose falsos positivos, se lleva un control de si el acelerómetro ha pasado de nuevo por el estado de reposo antes de dar por buena una nueva lectura, garantizando así la lectura correcta.

Si se detecta un movimiento del acelerómetro dentro de los parámetros válidos, se calcula el siguiente estado del autómata, en función del actual y se escribe en la cola xCambioLedQueue, indicando que es de origen local.

Mediante un *delay* de la tarea, se maneja el número de veces que queremos que se chequee el estado del acelerómetro.

A continuación se ejecuta taskYIELD() para provocar un cambio de contexto y liberar la CPU para el resto de tareas, y termina la iteración del bucle for.

4.3.5 Función vGestionWifly

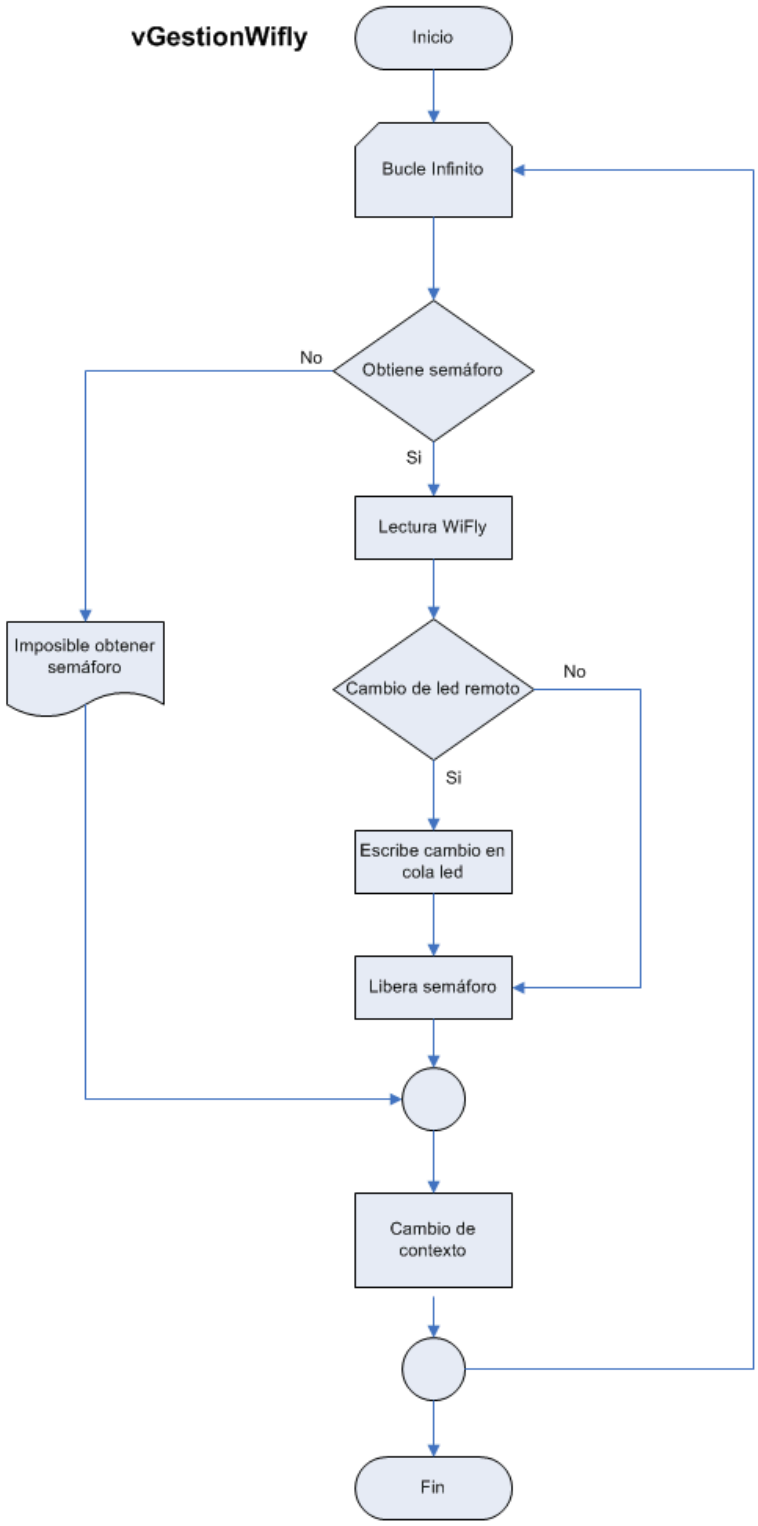


Ilustración 46: Diagrama de flujo de vGestionWifly

La función de esta tarea es la de realizar las lecturas de los datos que se reciben por el WiFly, escribiendo en la cola xCambioLedQueue un registro con el nuevo estado que indica el próximo led a iluminar, indicando que el origen del cambio es remoto.

Dado que la tarea vGestionLeds, también utiliza el WiFly para enviar al resto de los dispositivos los cambios que ella realiza en local y también la cola xCambioLedQueue, se utiliza el semáforo Mutex xControlIOSemaphore, para coordinar ambas tareas. Esto garantiza que las dos tareas no entren en conflicto por los recursos.

Una vez realizado todo el proceso, se realiza un cambio de contexto para dar paso a la siguiente tarea.

La lectura del WiFly se efectúa llamando a una función Wifly_Udp_Read () del driver WiFly**,** con una clave cuyo valor está especificado en el archivo config.h y que define al grupo de dispositivos que deben sincronizarse entre sí, permitiendo convivir distintos grupos de sincronía dentro de una misma red Wi-Fi.

4.3.6 Función vGestionLeds

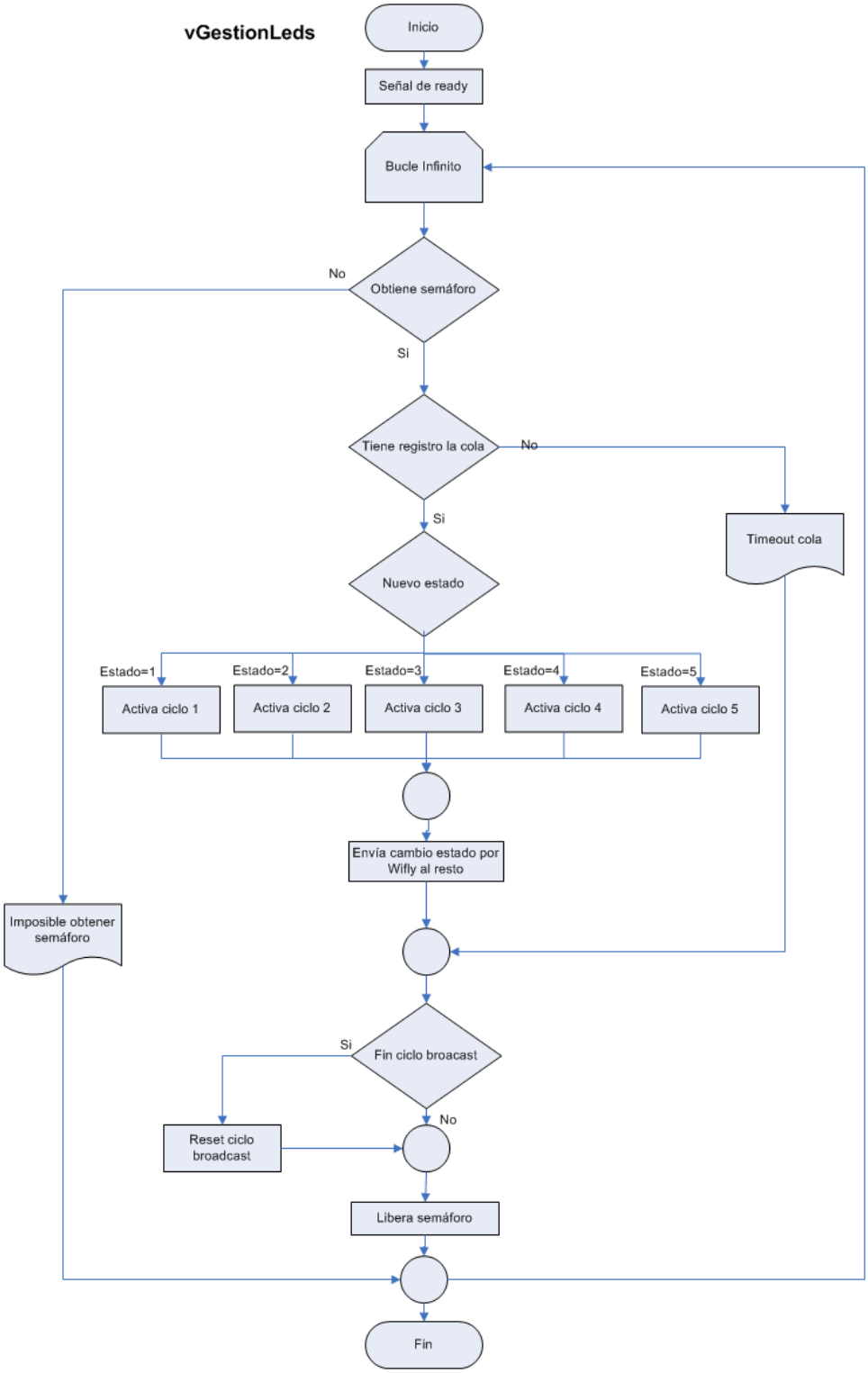


Ilustración 47: Diagrama de flujo de vGestionLeds

La misión de esta tarea es la de esperar a recibir un registro de cambio de led en la cola xCambioLedQueue, y a continuación, dar la orden por medio de una llamada a una función del driver led.c para cambiar los led al nuevo estado y enviar un registro con

dicho cambio a los dispositivos remotos. El envío al WiFly solo se realiza cuando el registro de cambio tiene origen local, es decir, lo ha producido el propio dispositivo.

Para coordinarse con la tarea vGestionWifly, utiliza el semáforo xControlIOSemaphore.

El envió del nuevo estado a los dispositivos remotos se realiza llamando a la función Wifly_Udp_Send() del driver WiFly, que además activa el intervalo de envío de mensajes broadcast a 1 segundo.

Al utilizar un protocolo de comunicaciones, como es UDP, que no está orientado a la conexión, se mantiene durante varios ciclos de la tarea activado el envío del registro que informa del cambio de led, para tener así una mayor garantía de que los mensajes llegaran a su destino.

La duración del ciclo se configura mediante una variable definida en config.h. Tras finalizar el ciclo, se restablece el intervalo de envío de broadcast a 10 segundos (valor configurable también en config.h), mediante la llamada a la función Wifly_Udp_Read().

En el apartado donde se comentan las principales características del driver wifly.c se explican todos los detalles.

4.3.7 Drivers de PFCLIB

Dentro de este apartado se comentarán las funciones más importantes de cada uno de los drivers utilizados para ayudar a tener una visión más concreta de cómo funciona el código de la aplicación.

4.3.7.1 Driver wifly.c

Como se ha comentado anteriormente, las comunicaciones son uno de los puntos más delicados del proyecto.

Tras analizar en profundidad todas las posibilidades que ofrece la última versión de firmware de WiFly, se decide utilizar el protocolo UDP, junto a la funcionalidad de *broadcast*.

El WiFly puede enviar paquetes UDP “hello/heartbeat” a un puerto determinado de forma automática para publicar información al resto de los dispositivos WiFly existentes en la red. Estás tramas se envían en intervalos parametrizables de entre 0 y 256 segundos. Este intervalo se puede modificar con el comando “set broadcast interval s”, siendo “s” el valor en segundos del intervalo. Su valor por defecto es de 7 segundos.

Cada trama UPD tiene un tamaño de 110 bytes y está compuesta por 12 campos que

sirven para compartir información que puede ser de utilidad a los distintos dispositivos de la red. En el diagrama siguiente se puede observar la estructura de los paquetes broadcast en detalle.

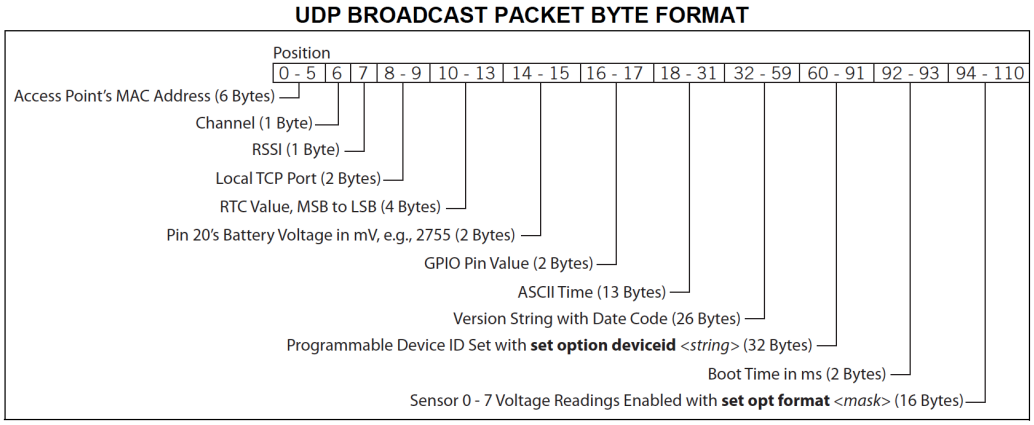


Ilustración 48: Detalle de la estructura de un paquete UDP del WiFly

Hay que prestar especial atención en el campo de 32 bytes "Device ID", que se utilizará para intercambiar la información entre los distintos dispositivos. El valor de este campo se modifica con el comando "set option deviceid <string>", siendo esta cadena el valor que tomará el campo a partir de ese momento. Su valor por defecto es "WiFly-GSX".

Basándonos es estas características, para gestionar las comunicaciones del sistema se ha diseñado un protocolo que se basa en las siguientes puntos clave:

- Modificar el campo "Device ID" de 32 bytes, para construir mensajes con una clave que identifica tanto al dispositivo, como al grupo de sincronía de la red, así cómo el valor del nuevo estado de los leds.
- Variar la frecuencia con la que se envían los paquetes automáticos de broadcast en función de si tenemos un cambio de estado o no, ajustando la misma a nuestras necesidades.

Las funciones principales del driver WiFly utilizadas para la gestión de las comunicaciones son:

- Wifly_Udp_Read()
- Wifly_Udp_Send()
- Wifly_SW_Broadcast()
- Wifly_Auto_Rol()

Con estos elementos se consigue construir un protocolo de intercambio de mensajes multi-nodo, sencillo pero muy eficiente.

Como se ha comentado anteriormente, UDP es un protocolo que no está orientado a la conexión, por lo que no se garantiza que la recepción de un paquete se ha producido

en destino, una vez enviado. WiFly dispone de la funcionalidad "UDP Retry" para aumentar el nivel de fiabilidad de estos paquetes broadcast, si añadir la sobrecarga que tiene en protocolo TCP.

Con "UDP Retry" activado, el WiFly espera una respuesta por cada paquete enviado. Si el módulo no recibe una respuesta en unos 250 ms, se vuelve a reenviar el mismo paquete. Este proceso continua hasta que:

- Se recibe una respuesta de recepción del paquete UDP
- El modulo envía un paquete UDP nuevo y se recibe la respuesta de recepción

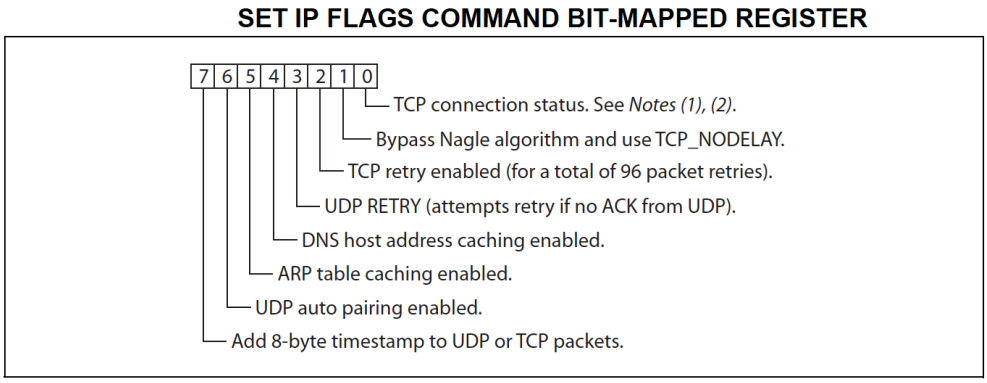


Ilustración 49: Detalle del mapa de bits de los flags IP del WiFly

Para activar la funcionalidad de "UDP Retry" hay que poner a 1 el bit 3 del los flags IP. Esto se hace mediante el comando "set ip flags <mask>" indicando la mascara de bits que queremos activar. Por defecto su valor el Ox7.

Aunque se ha considerado la posibilidad de activar esta característica como una opción muy interesante para dar robustez al protocolo, se ha descartado su utilización ante la falta de control que se tendría sobre la misma, sobre todo cuando hubiera más de dos dispositivos en la red. Se podría dar la circunstancia que un nodo reenviara constantemente un paquete UDP a todos los nodos de la red por el hecho de que uno de ellos no hubiera respondido con un "ack". Mientras tanto, la aplicación no sería consciente de ello ya que de esto se encargaría el WiFly automáticamente.

Una vez descartada la opción de usar "UDP Entry", y ante la riesgo de que un paquete UPD determinado se pierda por un fallo puntual de la red Wi-Fi o cualquier otra circunstancia y no llegue a su destino, se ha implementado dentro del código la posibilidad de manejar ciclos de *broadcast* parametrizables desde config.h.

Estos ciclos se encargan de reenviar varias veces el mismo paquete, para así mejorar su fiabilidad, con la ventaja de que pueden ser controlados e interrumpidos a voluntad desde la aplicación.

En cualquier caso, en este apartado quedaría mucho trabajo por hacer ya que se podrían implementar varias funcionalidades que harían el sistema más fiable y tolerante a fallos, como sería tener una tabla con los nodos detectados y chequear su estado mediante un ping cada cierto tiempo y reiniciar el módulo WiFly si no tuviera

conexión, o medir la calidad de la señal Wi-Fi, adaptando los reintentos o los ciclos de envío en función de la misma.

En cuanto al rol que cada Wi-Fly toma a la hora de comportarse como punto de acceso o de cliente, cada WiFly puede traer de fábrica el setup preconfigurado, o autoconfigurarse en función de si se detecta el punto de acceso del sistema, tras el escaneo de todos los canales Wi-Fi, mediante el comando scan.

Las comunicaciones entre el WiFly y el LPC se realizan con un baudrate de 115.200, por lo que el WiFly tendrá que estar configurado a esta velocidad previamente.

Ha sido fundamental para el desarrollo del proyecto disponer de una herramienta para analizar el tráfico entre los dos dispositivos como es Wireshark.

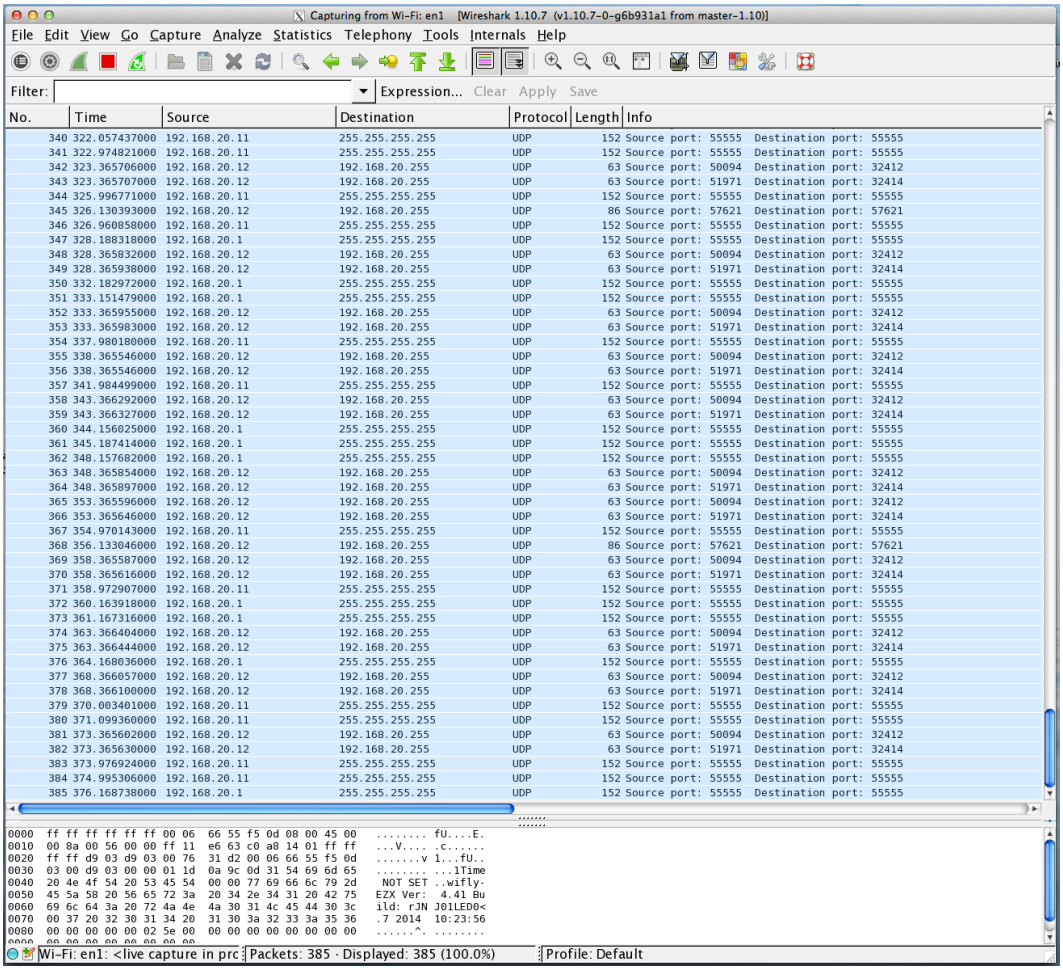


Ilustración 50: Captura de pantalla de Wireshark

Esto ha permitido conocer como se comportaban los módulos WiFly a bajo nivel, algo que desde la consola del Tera Term o haciendo debug desde LPCXpreso era absolutamente imposible.

A continuación se explica con más detalle como funcionan cada una de las funciones más importantes implicadas en el proceso. El resto de las funciones de wifly.c desarrolladas en la fase de aprendizaje, están comentadas convenientemente en el código fuente.

Wifly_Udp_Send.

```
uint32_t Wifly_Udp_Send(uint8_t PortNum, char *key, char *dispositivo,
uint8_t led)
```

Los parámetros que recibe la función son el puerto del WiFly, la clave de la red que forman este grupo de dispositivos, el dispositivo que envía el mensaje y el valor del estado del led.

Construye una cadena con la key de sincronización, el dispositivo y el valor del led correspondiente con el estado al que acaba de pasar el nodo local y se lo asigna al campo “Device ID” de la trama UDP mediante el comando “set opt deviceid”.

Posteriormente hace una llamada a la función “Wifly_SW_Broadcast” pasándole el parámetro de 1 segundo, para que envíe el mensaje inmediatamente y lo repita cada segundo mientras dura el ciclo en el que el led de “Device ID” tiene un valor asignado.

La función devuelve verdadero si todo es correcto y falso si se produce algún error.

Wifly_Udp_Read.

```
int32_t Wifly_Udp_Read(uint8_t PortNum, char *Key)
```

Los parámetros que recibe la función son el puerto del WiFly y la clave de la red que forman este grupo de dispositivos.

Esta función busca la clave Key dentro del campo “Device ID” del paquete broadcast recibido y si la identifica, devuelve el valor del nuevo estado del led que debe tomar el sistema para sincronizarse con los dispositivos remotos.

La función devuelve el valor del estado de led recibido (de 1 a 5) o 0 si no recibe nada.

Wifly_SW_Broadcast.

```
uint32_t Wifly_SET_Broadcast(uint8_t PortNum, char *Param)
```

Los parámetros que recibe la función son el puerto del WiFly y los segundos en los que se establecerá el intervalo de broadcast.

Wifly_Auto_Rol.

```
uint32_t Wifly_Auto_Rol(uint8_t PortNum, char *Ssid)
```

Los parámetros que recibe la función son el puerto del WiFly y el SSID del la red Wi-Fi.

La función sirve para autoconfigurar el rol del nodo como *apnode* (punto de acceso) o como cliente.

Mediante el comando "scan" busca las redes Wi-Fi existentes entre los canales 1 y 3. Si encuentra el SSID activo (ya existe otro nodo operando como punto de acceso), se configura como cliente y reinicia el Wi-Fly después de salvar la configuración. En el caso de que no encuentre la red operativa, se configura como punto de acceso y reinicia el WiFly tras guardar la nueva configuración.

Esta funcionalidad se puede configurar desde el fichero config.h quitando el comentario de la variable "AUTOWIFION". En el caso de estar comentada, cada nodo arranca con el rol que tenga pregrabado en su WiFly.

El arranque del los nodos es mucho más rápido cuando el auto_rol está desactivado, ya que por precaución, se aplica un *delay* de unos 15 segundos en total, para asegurar que el sistema esté listo el WiFly operativo.

Los comandos necesarios para configurar cada uno de los modos en el WiFly son los siguientes:

CLIENTE

- factory RESET
- set ip dhcp 1
- set wlan ssid **SSID**
- set option deviceid **DEVICEID**
- set broadcast interval 10
- set broadcast port 55555
- set ip remote 55555
- set ip local 55555
- set wlan phrase **PASSWORD**
- set ip proto 1
- set wlan join 1
- save
- reboot

PUNTO DE ACCESO (*apnode*)

- factory RESET
- set wlan join 7
- set wlan channel 3
- set apmode ssid **SSID**
- set apmode passphrase **PASSWORD**
- set ip dhcp 4

- set ip address 192.168.20.1
- set ip net 255.255.255.0
- set ip gateway 192.168.20.1
- set option deviceid **DEVICEID**
- set broadcast interval 10
- set broadcast port 55555
- set ip remote 55555
- set ip local 55555
- set ip proto 1
- save
- reboot

4.3.7.2 Driver led.c

Este driver es muy sencillo ya que contiene solo dos funciones:

LED_Initialize.

```
void LED_Initialize(uint16_t pinMask)
```

Inicializa el puerto GPIO correspondiente a pinMask.

LED_Switch.

```
void LED_Switch(uint16_t pinMask, bool value)
```

Enciende o apaga el led conectado al puerto indicado en pinMask en función del valor bool, que será de 1 o 0.

4.3.7.3 Driver acelerometro.c

Este driver nos permite manejar el acelerómetro del dispositivo.

Inicia_Acelerometro.

```
int Inicia_Acelerometro(int Res);
```

Inicializa el acelerómetro MMA7361. El parámetro Res le indica a que resolución debe de trabajar:

- 1 - Rango 206 mg/V
- 0 - Rango 800mg/V

Calibracion.

```
void Calibracion(int Lecturas);
```

Calcúla el offset de cada una de las coordenadas teniendo en cuenta la gravedad de la tierra, para lo cual el acelerómetro ha de estar en posición horizontal.

El parámetro "Lecturas" le indica el número de medidas que debe de tomar para sacar una media más fiable (cada medida fluctúa en un pequeño margen respecto a las demás).

Read_Acelerometro.

```
void Read_Acelerometro (int Num_muestreo, A_Reg *Datos );
```

Lee los 3 ejes del acelerómetro devolviendo la salida en tres formatos. La medida tal y como la devuelve el driver ADC, la medida calculada teniendo en cuenta los offset de la calibración y la medida en grados de inclinación de cada eje.

Los parámetros con los que trabaja son el número de muestras a recoger para calcular su media y una variable tipo registro donde devuelve todas las medidas calculadas, con el siguiente formato:

```
typedef struct {
	float Lectura_X;
	float Lectura_Y;
	float Lectura_Z;
	float Procesado_X;
	float Procesado_Y;
	float Procesado_Z;
	float Angulo_X;
	float Angulo_Y;
	float Angulo_Z;
} A_Reg;
```

El algoritmo utilizado para realizar los cálculos está basado en la información obtenida en la página http://www.starlino.com/imu_guide.html .

4.3.8 Sistema Operativo FREERTOS

FreeRTOS es un sistema operativo en tiempo real de código libre, que nos ofrece funcionalidades de abstracción de hardware y de computación orientada a tareas. Está

portado a numerosos microcontroladores y plataformas y soporta gran cantidad de herramientas de desarrollo.

FreeRTOS implementa un "scheduler" o planificador de tareas de tiempo real que gestiona cada una de las tareas que componen la aplicación y se ejecutan sobre el microcontrolador.

FreeRTOS, mediante cambios de contexto expropiativos de tareas (es el S.O. el que toma el control de la CPU, sin esperar a que la tarea se lo ceda), ofrece la sensación de ejecución concurrente de tareas en tiempo real, sin embargo el microcontrolador solo ejecuta una tarea a la vez y las va asignando CPU en función de su prioridad, bloqueo o tiempo que llevan en ejecución. Además FreeRTOS dispone de un conjunto de abstracciones para gestionar las tareas, como son mecanismos de comunicación por colas, sincronización entre tareas y gestión expropiativa basada en prioridades.

TAREAS

El programa principal contiene un bucle main() que se ejecuta al iniciarse el sistema, y es donde se definen las diferentes tareas y desde donde se lanza el gestor de tareas (vTaskStartScheduler). El buce main(), no debe tener fin, es decir, nunca debe devolver el control a S.O.

Las tareas son procesos independientes que se definen como funciones que realizan una actividad concreta. La condición que ha de cumplir una función para convertirse en una tarea es que no termine nunca, es decir, ha de contener un bucle infinito en el que ejecuta su código.

Se pueden crear tantas como sean necesarias, con la única limitación de la RAM disponible del sistema (en FreeRtos el número máximo de tareas es configurable), ya que al crearlas hay que asignar una cantidad de memoria determinada a cada tarea para su espacio de datos al que se denomina contexto.

El contexto de una tareas está compuesto por:

- La pila, que se usa para la llamada a funciones y para almacenar las variables locales.
- Los registros internos del microprocesador.
- El contador de programa.

En los sistemas con un solo procesador, solo se puede ejecutar una tarea en cada momento, por lo que el gestor de tareas de FreeRtos se encarga de cambiar el estado de cada tarea entre los distintos posibles:

- Running: Tarea en ejecución. Solo puede haber una tarea ejecutándose.
- Ready: Tarea lista y esperando para ejecutarse.
- Blocked: Tarea en espera de algún recurso o un evento.
- Suspended: Tarea suspendida.

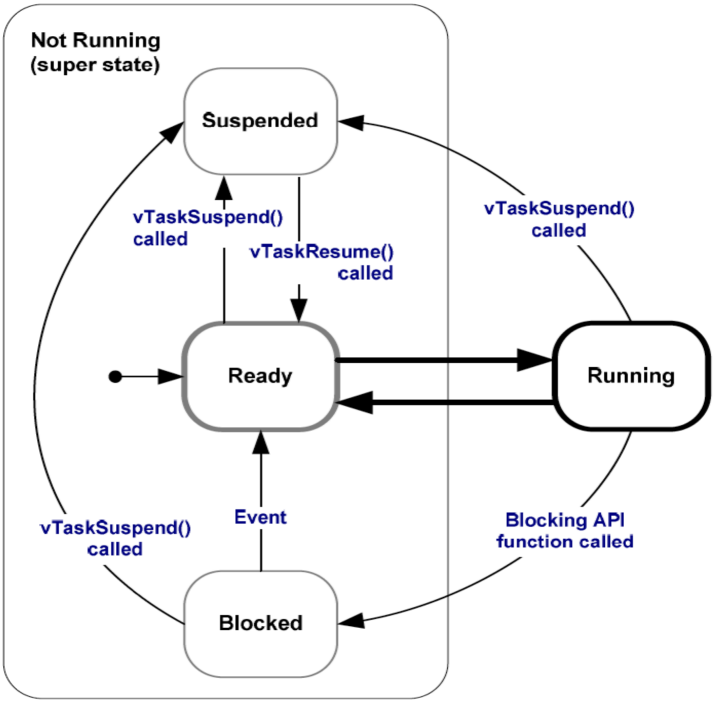


Ilustración 51: Diagramas de estados FreeRtos

El gestor de tareas decide que tarea se tiene que ejecutar en cada momento basándose en las prioridades de cada una de ellas. Las prioridades se enumeran de menor a mayor, siendo la prioridad 0, la más baja y es la utilizada por la tarea “idle” que es la que se ejecuta cuando el procesador no tiene ninguna otra que ejecutar.

La prioridad más baja de una tarea de usuario es la 1, asignando números más altos a aquellas tareas que se consideran más prioritarias (estas prioridades se pueden cambiar en tiempo de ejecución).

Solo las tareas en estado “ready” pueden ser candidatas a disponer de tiempo de CPU para ejecutarse. El planificador siempre ejecutará la tareas de mayor prioridad de todas la “ready” disponibles, hasta que esta tarea se bloquee o llegue una tarea con prioridad superior. Si varias tareas con la misma prioridad están “ready”, el gestor de tareas repartirá diferentes ”slots” de tiempo de CPU para cada una de ellas.

Las tareas en estado '”blocked” pasaran a “ready” cuando hayan obtenido el recurso que esperan o el evento que aguardaban.

COLAS

La única forma de intercambiar datos entre tareas es la utilización de variables globales o la utilización de colas.

El uso de colas es recomendable cuando se necesita un método de almacenamiento temporal para soportar ráfagas de datos, a priori desconocidas o cuando existen varios productores de datos y un sólo consumidor y no se desea bloquear a los productores a la espera de que el consumidor recoja los datos.

La gestión de las colas es realizado por el S.O. y su funcionamiento es el siguiente:

- Las colas se crean mediante una llamada al sistema. En ese momento se reserva la memoria necesaria para guardar todos los elementos de la cola, además de crear una estructura de control, que se encargará de gestionarla.
- Al crear la cola, se indica el numero máximo de elementos que contendrá. Una tarea no podrá insertar elementos en la cola, más allá del número máximo indicado, hasta que alguna tarea haya leído alguno de los existentes, liberando entonces el espacio.
- Las funciones que envían y reciben datos de la cola se pueden bloquear cuando la cola esté llena o vacía, respectivamente.
- El envío y la recepción de datos a través de la cola, se realizan únicamente mediante llamadas al sistema.
- Los elementos de la cola pueden ser de cualquier tipo, incluyendo estructuras de datos, aunque si el tamaño de los datos a pasar es muy grande, es más eficiente pasar punteros a estructuras de datos.

SEMAFOROS

Los semáforos son una herramienta que posee FreeRtos para permitir la coordinación entre tareas y evitar conflictos en el acceso a los recursos.

Sus utilidades fundamentales son:

- Sincronización para la compartición de un recurso entre varias tareas.
- Sincronización para la atención de eventos asincrónicos a modo de colas, pero sin paso de mensajes.

Se puede definir el intervalo de tiempo que una tarea ha de esperar por un semáforo. Una vez trascurrido el tiempo de espera, si no se tiene éxito en obtenerlo, la tarea continua, activando un error que la aplicación deberá gestionar.

Existen dos tipos de semáforos:

- Semáforos binarios: Cada vez que una tarea tiene que usar un recurso compartido cuyo acceso está gestionado por un semáforo, lo primero que hace es comprobar que el semáforo esté libre. En ese caso, el semáforo se tomara y la tarea podrá usar el recurso compartido, volviendo a liberar el semáforo cuando esta termine. En caso contrario, si lo encuentra ocupado, la tarea se bloquea hasta que el semáforo se libere.

- Mutex: Es un tipo de semáforo en que la tarea de baja prioridad hereda la prioridad de la de más alta prioridad cuando toma un semáforo compartido con ésta, para evitar problemas de bloqueos por “inversión de prioridad”.

Las funciones principales para el uso de semáforos son:

- De creación:
  - vSemaphoreCreateBinary: Crea un semáforo binario.
  - xSemaphoreCreateCounting: Crea un semáforo binario con capacidad de conteo, delimitado hasta una cantidad máxima posible a contar.
  - xSemaphoreCreateMutex: Crea un semáforo de características binarias pero con capacidad de exclusión mutua, siendo capaz de manejar las prioridades de las tareas involucradas. En un caso de competencia, gana el mutex la tarea que tenga mayor prioridad.
  - xSemaphoreCreateRecursiveMutex. Crea un Mutex recursivo.
- De toma de recurso o espera de evento:
  - xSemaphoreTake: Cuando se llama a esta función, la tarea esperara "xBlockTime" de tiempo para obtener el semáforo, después de ese tiempo si no lo puede obtener, se reanuda la ejecución de la tarea y se informa error. Si se libera y se logra obtener, la tarea bloqueará el semáforo para asegurar que nadie lo utilizará hasta su liberación.
  - xSemaphoreTakeRecursive: Toma de manera recursiva un semáforo para garantizar su uso tantas veces como se llamó a la función.
- De entrega de recurso o generación de evento:
  - xSemaphoreGive: En este caso se libera al semáforo para que otras tareas puedan usar el recurso que gestiona.
  - xSemaphoreGiveRecursive: Se devuelve dicho semáforo de manera recursiva. Para que otra tarea distinta lo pueda usar, la tarea que tiene bloqueado este semáforo debe llamar a esta función tantas veces como antes llamo a xSemaphoreTakeRecursive para obtenerlo.

4.3.9 LPCXpresso

El IDE LPCXpresso es un entorno de desarrollo basado en lenguaje C, creado por CodeRed y NXP, y que dispone de un entorno de Eclipse específicamente adaptado para interactuar con la target board LPC. Este IDE incluye un compilador cruzado, que permite compilar, enlazar y montar nuestra aplicación en el PC y una vez cargada en el LPC1769, hacer debug de la misma, mediante el uso del módulo CP2102.

Eclipse es un entorno de desarrollo multiplataforma de código abierto, que nos

permite trabajar con Workspaces a modo de contenedores de proyectos, donde se organizan nuestras aplicaciones y bibliotecas. Estos proyectos pueden tratarse de bibliotecas estáticas o aplicaciones ejecutables.

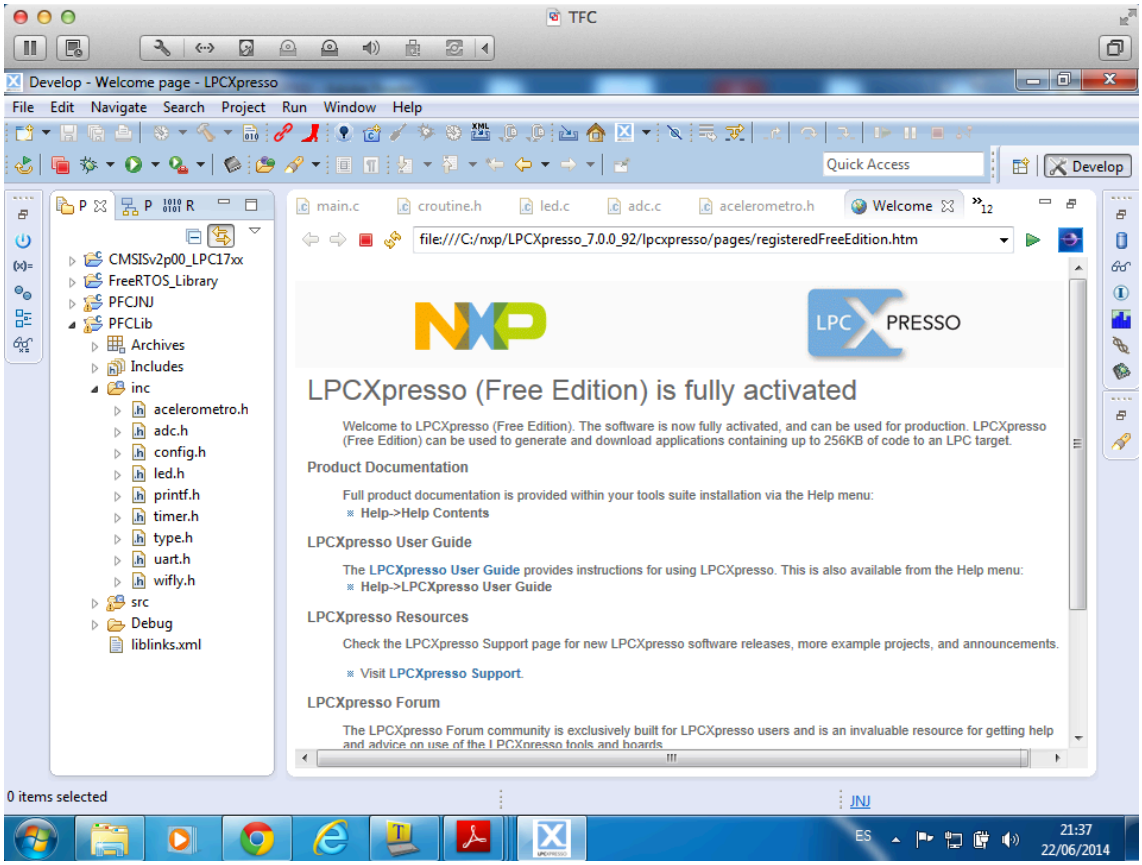


Ilustración 52: Entorno de desarrollo LPCXpresso

5 VIABILIDAD TECNICA

Partiendo de la premisa de que la idea original del proyecto era la de basarse en el hardware proporcionado para la fase de aprendizaje, evidentemente, a la hora de considerar su comercialización, habría que decantarse por un hardware mas optimizado a nivel de costes que el empleado.

Por otro lado, si quisiéramos dar un salto cualitativo a nivel de posicionamiento de mercado, habría que dotarlo de funcionalidades más avanzadas.

En cualquier caso, creo que conceptualmente se ha demostrado la viabilidad del uso de los sistemas empotrados para este tipo de aplicaciones y las posibilidades de evolución existentes.

También ha quedado de manifiesto tras analizar la situación del mercado, que el apartado de luminotecnia e incluso de diseño exterior del dispositivo es muy relevante a la hora de tener éxito comercialmente, una faceta del producto, que al no estar en los objetivos de este proyecto, no se ha abordado.

A continuación de enumeran los puntos fuertes y los débiles del sistema:

Puntos fuertes:

- Sistema autónomo, independiente de la infraestructura LAN del cliente
- Facilidad de uso e instalación. No necesita configuración (*plug&play*)
- Escala fácilmente
- Plataforma muy flexible
- Dispone de un hardware base muy potente

Puntos débiles:

- Producto inmaduro
- Solo dispone de funcionalidades básicas
- Sistema de iluminación solo para “demo”
- Hardware del microcontrolador sobredimensionado para las funcionalidades proporcionadas
- Acabado de prototipo

6 VALORACIÓN ECONÓMICA

6.1 COSTE DEL EQUIPAMIENTO HARDWARE Y SOFTWARE NECESARIO:

En este capítulo se hace un desglose del coste orientativo tanto de la construcción de un prototipo, como el del desarrollo del proyecto completo.

Evidentemente, si el objetivo fuera lanzar un producto comercial, habría que añadir equipamiento adicional (cajas, luces, interruptores, placas, embalaje, manuales, etc.), y buscar la reducción de costes de fabricación utilizando, por ejemplo, un microcontrolador más básico, y obtener cierta economía de escala a la hora de adquirir los componentes necesarios.

No se han tenido en cuenta los costes asociados al equipamiento informático (hardware y software) y otras herramientas (multímetro, soldador, alicates, pelacables, etc.), imprescindibles para el desarrollo del proyecto, que en su gran mayoría ya estaban disponibles.

PROTOTIPO			
Concepto	**Precio U.**	**Unidades**	**Total €**
Placa de desarrollo LPCXpresso LPC1769	30,00	1	30,00
Módulo WiFly RN-XV con adaptador Xbee	35,00	1	35,00
XBEE Adapter Board Wifly-Breadboard	4,00	1	4,00
Módulo CP2102 USB/UART	10,00	1	10,00
Sensor acelerómetro MMA7361	4,00	1	4,00
Resistencias	0,20	3	0,60
Condensadores	0,30	3	0,90
Leds	0,30	4	1,20
Breadboard	6,00	1	6,00
Cables	0,20	20	4,00
Software (Freeware)	0,00	1	0,00
	TOTAL PROTOTIPO		**95,70**

Tabla 1: Tabla de valoración económica de un prototipo

PROYECTO			
Concepto	**Precio U.**	**Unidades**	**Total €**
Prototipos	95,70	2	191,40
Manuales	26,00	1	26,00
Horas de I+D	12,00	250	3.000,00
	TOTAL PROYECTO		**3.217,40**

Tabla 2: Tabla de valoración económica del proyecto

7 CONCLUSIONES

7.1 OBTENCIÓN DE OBJETIVOS

Estos eran los objetivos principales que se habían marcado al inicio del proyecto y que consistía en los siguientes puntos:

- Crear un modulo de iluminación ambiental de cuatro colores que funcione de forma autónoma
- Activación y cambio de color basados en la variación de posición de cualquiera de los dispositivos, registrados por el acelerómetro incorporado
- Capacidad de interconexión inalámbrica con otros dispositivos similares para que actúen de forma síncrona, sin necesidad de infraestructura adicional
- Soporte de interconexión escalable de varios dispositivos, con la capacidad de incorporación automática a la red de los nuevos dispositivos
- Funcionamiento simple y sin necesidad de configuración inicial por parte del usuario
- Hardware y software idéntico en todos los dispositivos

Examinando el resultado final obtenido, considero que se han cumplido con éxito, a pesar de las dificultades encontradas durante el desarrollo del mismo.

Por otro lado, me hubiera gustado de disponer de más tiempo para evolucionar los prototipos y dotar de mayor funcionalidad y robustez al software, ya que solo me he centrado en que todos los elementos cumplieran con los mininos establecidos para cumplir los plazos de entrega. También hubiera sido muy interesante poder probar el sistema como más prototipos simultáneamente (al menos 3), ya que teóricamente, el sistema soporta hasta 8.

Estoy seguro que con un poco más de tiempo y algunas mejoras tanto hardware como software, junto con las habilidades ya adquiridas para trabajar con sistemas en tiempo real, el producto final obtenido hubiera “lucido” mucho más.

7.2 PROPUESTAS DE MEJORA

A lo largo del desarrollo del proyecto se han detectado muchas oportunidades de mejora que harían del producto final una solución mucho más completa.

Distinguiéremos entre mejoras hardware y mejoras software:

Mejoras hardware

- Dotar al sistema de mayor conectividad o mecanismos de iteración con el exterior, como sería la conectividad bluetooth o sensores de ultrasonidos
- Dotar al sistema de una carcasa para manipularlo adecuadamente y poder cambiarlo de posición
- Dotar al sistema de baterías recargables para que no dependiera de una toma de alimentación
- Dotar al sistema de iluminación real en lugar de utilizar leds

Mejoras software

- Dotar al sistema de una aplicación de Smartphone para su gestión remota. Con esta capacidad, sus funcionalidades crecerían exponencialmente:
  - Mando a distancia
  - Programación
  - Control individual de los dispositivos
  - Iluminación en función de pautas de uso o del entorno (hora del día, música que suena en el dispositivo, métricas de cuantificadores personales, etc.)
  - Control de consumos y de horas de uso del sistema
- Mejoras en fiabilidad y tiempos de respuesta
- Funcionalidades de ahorro de energía
- Mejoras en las comunicaciones entre dispositivos
  - Mayor numero de dispositivos simultáneos en la red
  - Posibilidad de conectar a Internet el sistema mediante el uso de la red domestica
- Mejora en la arquitectura de comunicaciones utilizando TCP y redes Chord para interconectar los nodos

7.3 AUTOEVALUACIÓN

Desde el principio, me plantee este proyecto fin de carrera como un desafío didáctico, ya que era absolutamente neófito en el mundo de los sistemas empotrados y tenia gran curiosidad por profundizar en ellos.

Siendo desde el principio consciente a las dificultades a las que me enfrentaba, tras finalizar el mismo, y echando la vista atrás, he de reconocer que me quedé corto en mi cálculo, sufriendo aun mas de lo esperado.

Entre las principales dificultades encontradas, destacan:

- La cantidad de materia a asimilar en muy corto espacio de tiempo
- Las dificultades de trabajar con dos prototipos simultáneamente que debían operar sincronizados en tiempo real y toda la logística que esto implica.
- Cada modificación en el hardware o software debía aplicarse a los 2 prototipos a la vez para hacer cualquier prueba.
- Las herramientas de debug disponibles con LPCXpresso eran poco usables para hacer debug de los 2 dispositivos simultáneamente y en tiempo real.
- La fragilidad de los prototipos montados sobre las *breadborads* y el hecho que tuviera que moverlos y girarlos para operar con ellos, hacia que tuviera muchos problemas de falsos contactos o fallos hardware, que aveces, erróneamente atribuía al software, perdiendo una gran cantidad de tiempo.
- Falta de experiencia a la hora de programar un hardware no estándar y que no siempre se comporta como se espera, dentro de un ecosistema en el que el "cuando" importa tanto o mas que el "como".

En cualquier caso, la experiencia final ha sido tremendamente positiva y satisfactoria, tanto por los conocimientos adquiridos, como por la sensación de conseguir llegar a una meta que parecía inalcanzable en muchos momentos del proyecto.

Sin duda alguna, seguiré profundizando en este apasionante mundo a nivel personal.

8 GLOSARIO

ADC. Conversor analógica digital.

Breadboard. Placa de prueba compuesta por bloques de plástico perforados y láminas delgadas interconectadas que permiten montar prototipos electrónicos muy fácilmente.

Bridge. Es un dispositivo de interconexión de redes que opera a nivel de la capa 2 OSI y conecta segmentos de red formando una sola subred.

Broadcast: Se denomina a la forma de transmisión donde existe un emisor que lanza un mensaje a multitud de receptores a la vez.

Chord: Protocolo muy escalable diseñado para crear redes descentralizadas basadas en P2P.

Expropiativos. Método en el que algunos S.O. gestionan los recursos de CPU, privando a las tareas o programas de la CPU cuando el S.O. lo considera oportuno sin a esperar a que el propio programa la libere.

GPIO. Puerto de Entrada/salida de propósito general.

Hub. También denominado concentrador, es un dispositivo de red básico que permite compartir un recurso de red a todos los elementos que se conectan a él.

Internet de las cosas. Es la red de objetos cotidianos interconectados, haciendo posible interactuar con ellos.

Mircrocontrolador. Circuito electrónico programable que dispone de CPU, memoria y unidad de E/S.

PMBOK. Conjunto de conocimientos en Dirección/Gestión/Administración de proyectos generalmente reconocidos como «buenas prácticas» y desarrolladas por el PMI (Project Management Institute).

UART. Dispositivo transmisor/receptor asíncrono universal. Se utiliza para controlar los puertos y dispositivos serie.

UDP. Protocolo de nivel de transporte basado en datagramas (capa 4 de OSI) que permite la transferencia de datos sin necesidad de establecer una conexión previa.

Wearables. Conjunto de pequeños aparatos y dispositivos electrónicos que se incorporan en alguna parte de nuestro cuerpo interactuando continuamente con el usuario. Su origen tiene un raíz inglesa que traducida al español significa “llevable”.

9 BIBLIOGRAFÍA Y FUENTES DE INFORMACIÓN

Bibliografía:

- Using the FreeRTOS Real Time Kernel – NXP LPC17XX Edition. Richard Barry. 2011.
- Sistemas Empotrados en Tiempo Real. José Daniel Muñoz Frías. 2009.
- LPC17xx User manual. NPX. Rev. 3 — 20 December 2013.
- WiFly Manual de referencia de la versión 4.41 11/25/2013, BUILD r1046. MICROCHIP. 2014. http://ww1.microchip.com/downloads/en/DeviceDoc/50002230A.pdf

Recursos Web (Consultados entre Marzo y Junio del 2014.):

- xWiki Sistemas Empotrados (UOC): http://cv.uoc.edu/app/mediawiki14/wiki/IniciCortexM3/
- LPCXpresso – NXP: Base de conocimientos: http://knowledgebase.nxp.com/
- LPCXpresso – Soporte: http://www.nxp.com/lpcxpresso-support/
- FreeRTOS: http://www.freertos.org/
- Philips: http://www.philips.es/
- Lifx: http://lifx.co/
- Insteon: http://www.insteon.com/
- Smartbotics: http://www.smartbotics.com/
- GreenWave: http://www.greenwavereality.com/

PDFs de componentes:

- LPCX1769: http://www.nxp.com/documents/data_sheet/LPC1769_68_67_66_65_64_63.pdf .
- WiFly RN-XV: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/WiFi/WiFly-RN-XV-DS.pdf
- CP2102 USB/UART: http://www.silabs.com/Support%20Documents/TechnicalDocs/CP2102-short.pdf
- MMA7321: http://cache.freescale.com/files/sensors/doc/data_sheet/MMA7361LC.pdf?pspll=1&Parent_nodeId=1238111946420745867479&Parent_pageType=product

10 ANEXOS

10.1 MANUAL DE USUARIO

El funcionamiento del sistema es muy sencillo, de hecho, ese era un de los objetivos del proyecto.

Operativa del sistema

Funcionamiento de un equipo:

- Conecte el equipo a la alimentación.
- Espere unos instantes hasta que se enciendan simultáneamente los 4 leds y se vuelvan a apagar. En ese momento el sistema quedará operativo.
- Gire el equipo en cualquier dirección y deje el equipo de nuevo en reposo, y se encenderá el led verde. Al repetir la operación, se irán apagando y encendiendo los leds en el siguiente orden, hasta que al finalizar el ciclo, permanecerán de nuevo todos apagados (Verde – Rojo – Azul – Amarillo).

Funcionamiento de varios equipos:

- Conectado el equipo a la alimentación
- A los pocos instantes se enlazará automáticamente con el resto de los equipos
- Gire cualquiera de los equipos y tanto este, como el resto de los equipos cambiarán la iluminación en la secuencia arriba indicada.
- Si el equipo no se sincroniza con el resto, desconéctelo de la alimentación y vuélvalo a conectar.

Si deseamos mostrar por una consola Tera Term los mensajes de debug activados en el programa, simplemente ha de conectarse el CP2102 del equipo a un puerto USB de un PC equipado con Tera Term, configurando el puerto serie con las siguientes características:

- Baud rate = 115.200
- Data: 8 bit
- Parity: no
- Stop: 1 bit
- Control de flujo: No

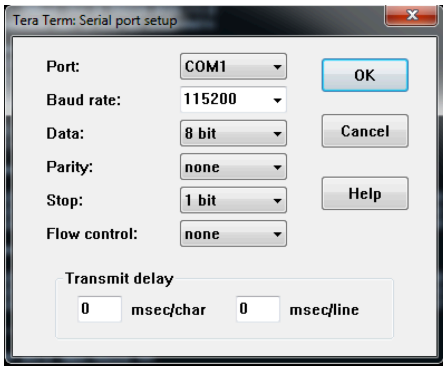


Ilustración 53: Configuración de Tera Term

10.2 PUESTA EN MARCHA DEL PROYECTO

10.2.1 Creación del entorno de Desarrollo y carga del proyecto

Para crear el entorno de desarrollo y compilar y ejecutar el proyecto se han de seguir los siguientes pasos:

- Descargar e instalar el entorno de desarrollo de LPCXpresso de la página de NXP (http://www.lpcware.com/downloads). Para tener acceso a la descarga hay que registrarse previamente en su página web.
- Crear un nuevo Workspace.

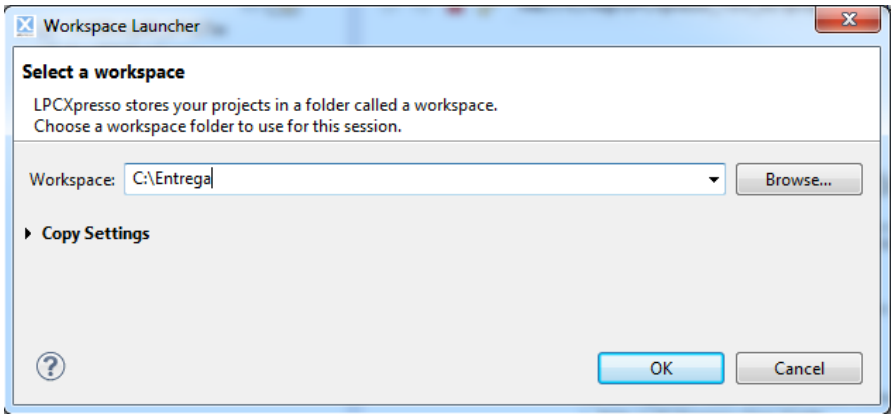


Ilustración 54: Selección de Workspace

- Importar el archivo .zip del proyecto

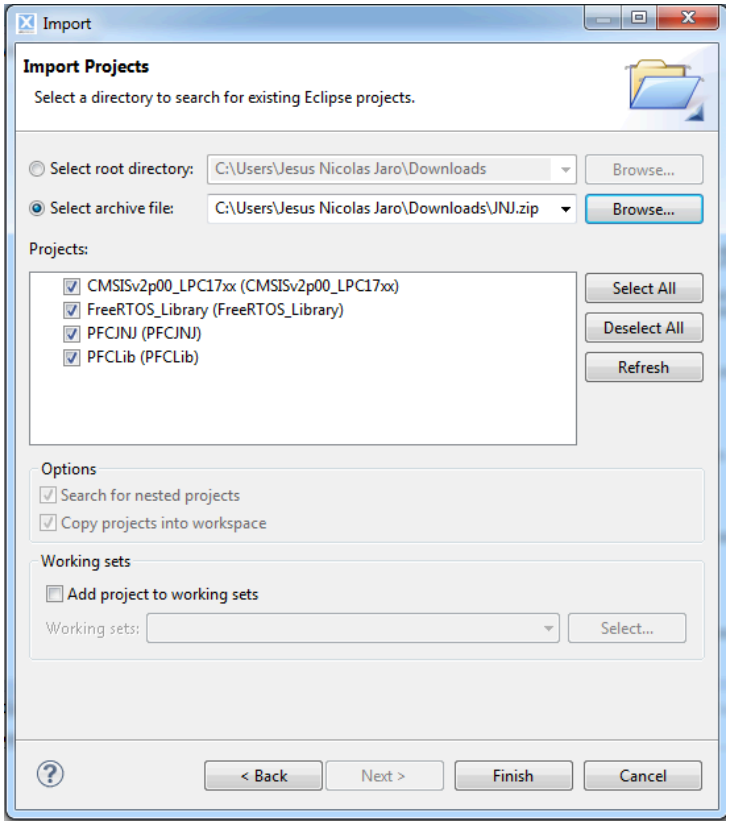


Ilustración 55: Importación de un proyecto .zip

- Seleccionar el proyecto a importar
- Una vez realizado este proceso ya tenemos acceso al código fuente del proyecto

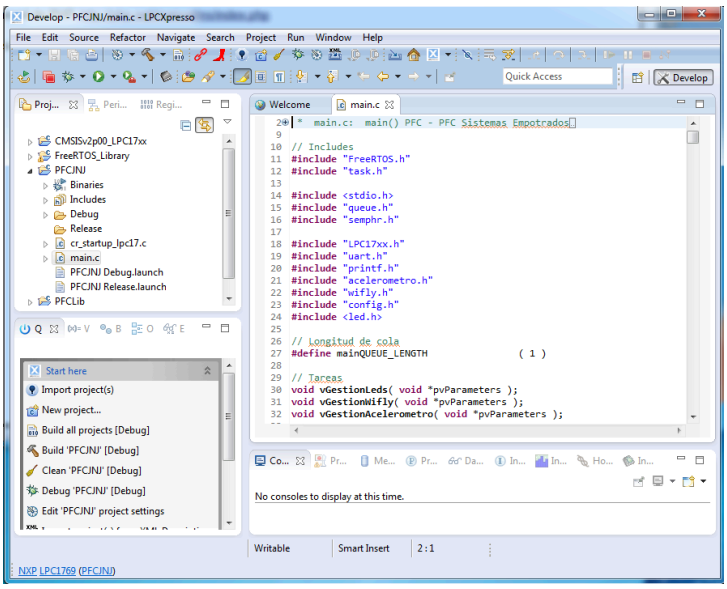


Ilustración 56: Acceso al código fuente

10.2.2 Compilación y Ejecución

Para compilar y ejecutar la aplicación hay que seguir los siguientes pasos:

- Compilar el proyecto completo seleccionando la opción “Build all” dentro del menú “Project” para compilar

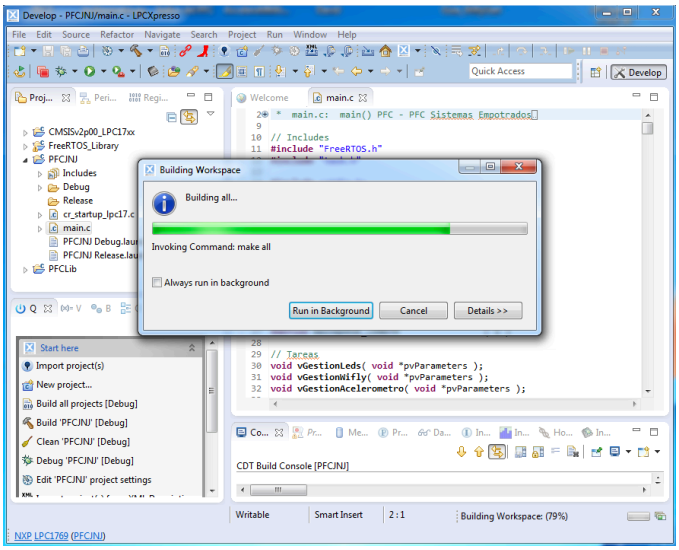


Ilustración 57: Compilación del código fuente

- Conectar el LPC1769 al PC en el puerto USB de PC
- Seleccionar el proyecto “PFCJNJ”
- Seleccionar la opción “Program Flash” de menú de opciones

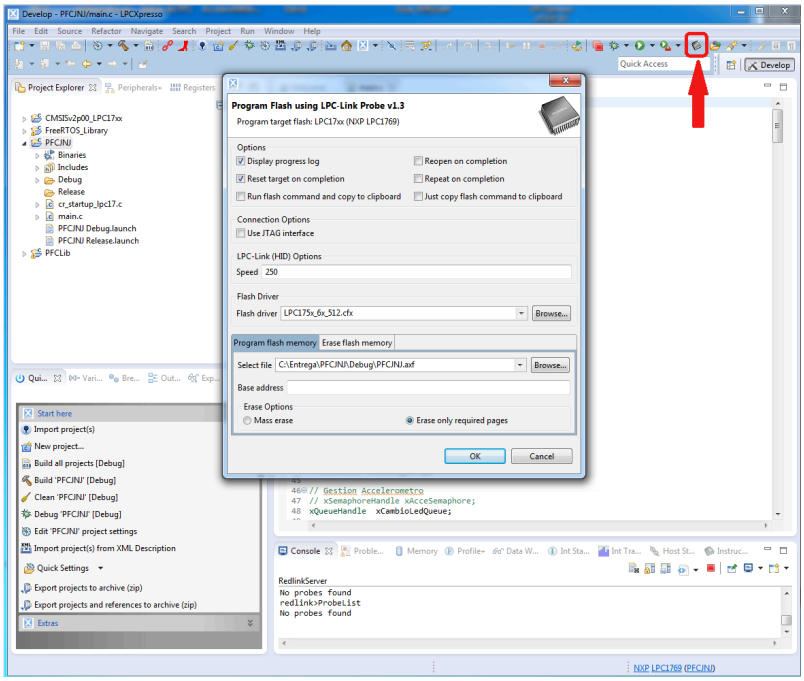


Ilustración 58: Opciones de Flash del LPC

- Una vez detectado el hardware del LPC1769 por parte del compilador, seleccionar el archivo PFCJNJ.axf a transmitir a la flash del LPC

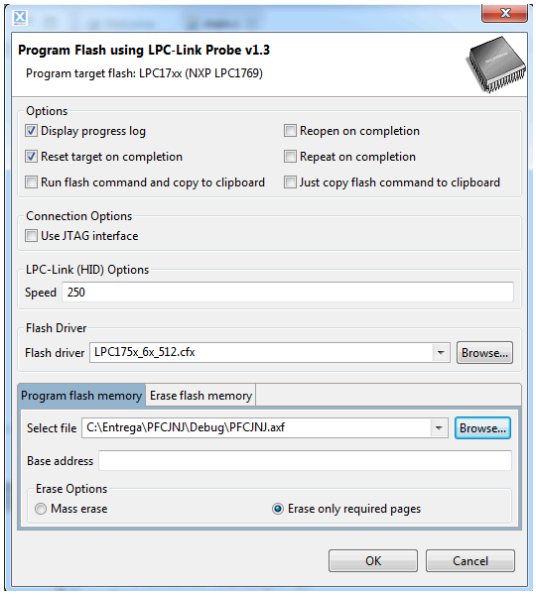


Ilustración 59: Selección del código compilado a transmitir al LPC

- Tras unos segundos y una vez que la barra de progreso se complete, el nuevo ejecutable estará instalado en el LPC y este iniciará su ejecución.

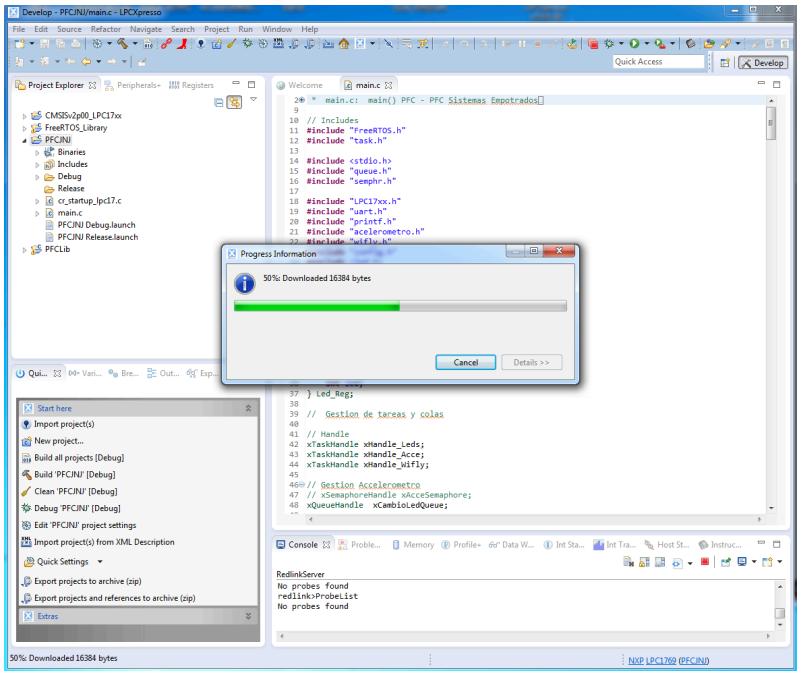


Ilustración 60: Progreso de la transmisión del ejecutable

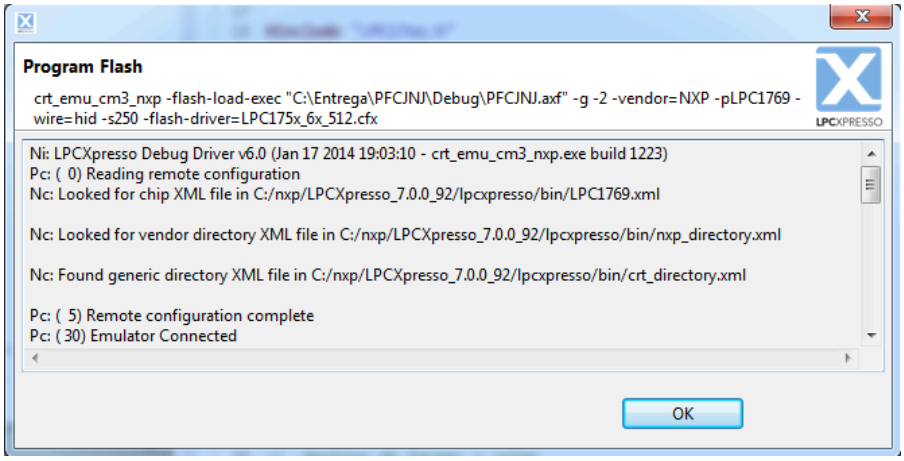


Ilustración 61: Mensaje de que la programación del LPC ha sido correcta

Conectando el CP2102 al la consola Tera Term, tendemos la salida de la aplicación en modo normal.

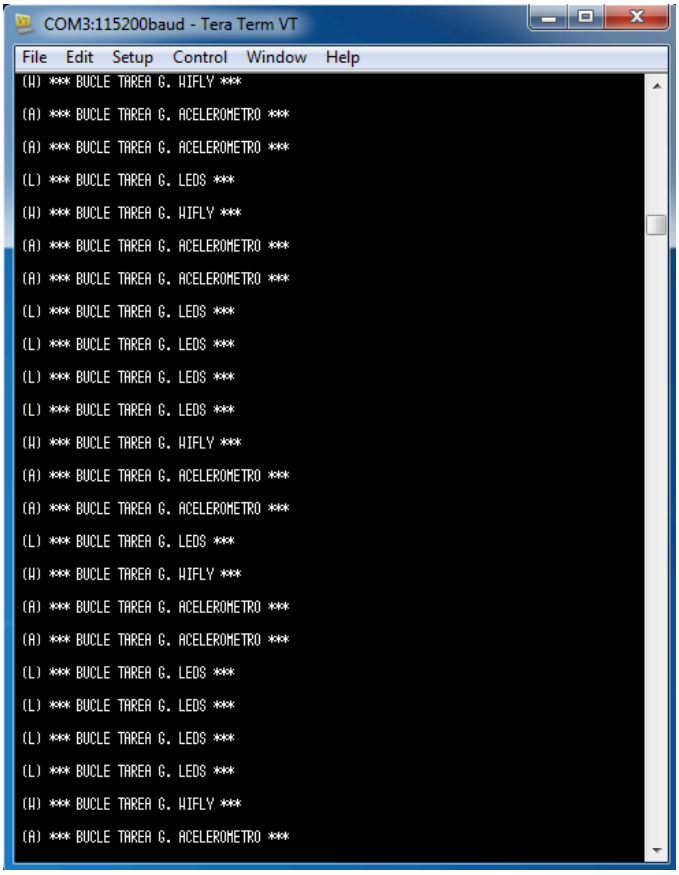


Ilustración 62: Consola en modo normal

Si en el fichero config.h d activamos la opción “MODE_DEBUG” y compilamos la aplicación, tendremos la salida con los mensajes de debug disponibles.

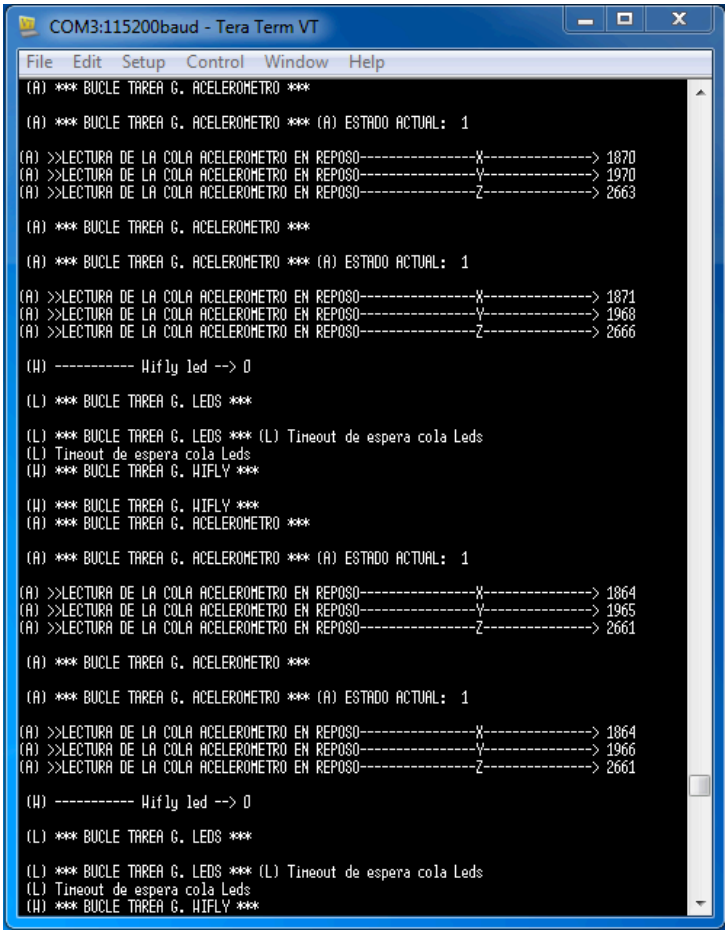


Ilustración 63: Consola en modo debug