Red de sensores sin hilos para control domótico

Ingeniería Técnica de Telecomunicaciones
TFC Sistemas empotrados. Memoria
Enero de 2013
Autor: Félix Herrera Piña
Consultor: Sebastiá Cortes Herms

A program is no more than half a conjecture. The other half of the conjecture is the functional specification the program is supposed to satisfy
(Dijkstra, 1998)

A mi familia

ÍNDICE

Félix Herrera Piña fherrerap@uoc.edu

TABLA DE FIGURAS

1 Descripción del proyecto

El presente, es trabajo de fin de carrera de la Ingeniería Técnica de Telecomunicaciones, especialidad telemática, encuadrado en el área de Sistemas empotrados.

El proyecto cubre el desarrollo de un aplicativo que permite capturar información física del entorno, proporcionada por los sensores y el sistema empotrado, y reacionar a ella según las especificaciones introducidas en remoto por el usuario en la interface web.

En concreto, nuestro sistema empotrado captura los valores de luminosidad y temperatura de su ubicación, y los transmite, mediante wifi, a una aplicación web que proporciona información gráfica en tiempo real. El usuario puede consultar en tiempo real los valores medidos, empleando cualquier dispositivo con navegador web y compatibilidad con Java, y, en el mismo punto, especificar los valores de los umbrales de control para transmitir al sistema empotrado.

El sistema empotrado captura los valores de los umbrales proporcionados por el usuario y actúa en consecuencia, simulando el encendido de luces o calefacción cuando el valor de luminosidad o temperatura medidos son inferiores al de los umbrales proporcionados por el usuario.

1.1 Sistemas empotrados y redes de sensores sin hilos

Una red de sensores sin hilos o WSN (wireless sensor network) está compuesta, de manera genérica, por un conjunto de sensores distribuidos. Cada sensor consta de un sistema empotrado o embebido, y de la sensorización específica necesaria para cada proyecto (LOS SANTOS, 2004).

El sistema embebido es un sistema microcontrolador con CPU, memoria RAM, y memoria EEPROM o Flash. Además, incluye periféricos y algún interface de comunicación, como ethernet, wifi o ZigBee. El sistema debe incluir una fuente de energía, y, en conjunto, tiene limitaciones hardware muy estrictas por su orientación, y necesidades de muy bajo consumo (para ser autónomos) y de muy bajo coste (para permitir despliegues masivos).

La sensorización puede ser parte del sistema embebido, o puede acoplarse a través de las oportuna interfaces analógicas o digitales, siendo en este caso tratadas por el sistema embebido como periféricos.

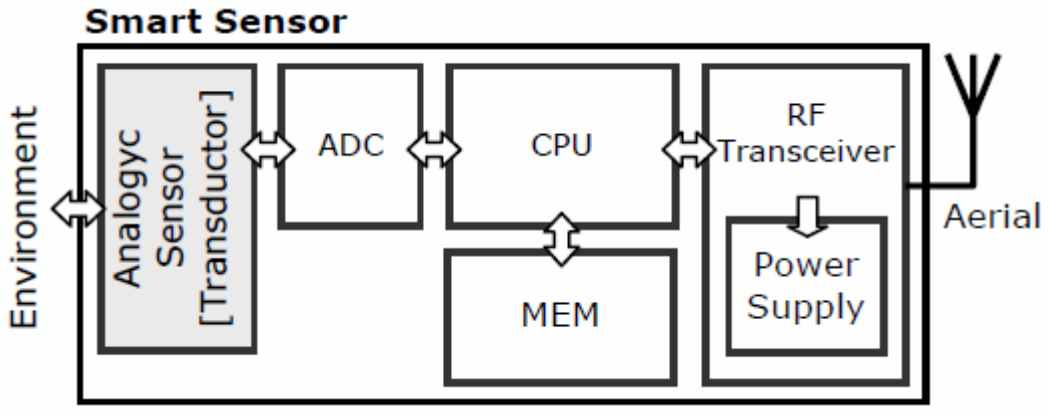

Figura 1. Estructura básica genérica de sensor inteligente

En nuestro caso, por las características del kit de sistemas empotrados que se nos proporciona para el desarrollo de la asignatura, adoptamos esta última manera de incluir los sensores al sistema. Trabajaremos con un microcontrolador ARM Cortex M3, con un módulo wifi conectado mediante puerto serie UART y dos sensores analógicos conectados mediante puerto ADC. Para la simulación de los actuadores, emplearemos algunas salidas digitales mediante puertos GPIO.

Detallaremos todo más adelante en un apartado específico.

1.2 Justificación y contexto

Los equipos sensores, que permiten medir las variables físicas del entorno que nos rodea, están evolucionando hacia dispositivos embebidos, cada vez más pequeños, de menor consumo y con mayor capacidad de proceso y facilidad para las comunicaciones (LOS SANTOS, 2004). También podemos apreciar que las redes de sensores están avanzando en los últimos años hacia los agentes inteligentes. Como tales, permiten percibir su entorno, procesar tales percepciones y actuar en el medio mediante actuadores.

Uno de los campos con más crecimiento y más potencial en el desarrollo del "entorno inteligente" es el de la domótica y el ahorro energético. La monitorización y el ajuste automático de los consumos permiten una gestión personalizada según las necesidades del usuario, corrigiendo pautas de comportamiento y optimizando el ahorro energético, a la vez que se logra un mayor confort.

En otro ámbito, sabemos que los dispositivos móviles comienzan a ser el medio más habitual para acceder a internet. La tendencia de crecimiento parece consolidada y está en plena expansión, llegando incluso a sustituir a los tradicionales ordenadores personales.

Los proyectos que nacen integrando estos tres vectores con potencial explosivo, están, por tanto, de plena actualidad. Este proyecto nos permite introducirnos, explorar y aprender en aspectos de estos tres campos, y trabajar en su integración.

Aunque de manera muy modesta, esta integración nos acerca a uno de los conceptos más novedosos y revolucionarios del futuro cercano: Internet de las Cosas (Internet of Things, IoT).

1.3 Objetivo del proyecto

El proyecto es la simulación de un sencillo sistema de control domótico. Partiendo del kit de sistemas empotrados que se nos proporciona para la asignatura, y teniendo en cuenta el criterio específico de simplificar y abaratar el despliegue y mantenimiento del sistema, usando las herramientas y sistemas libres de coste que tengamos disponibles, las aplicación a desarrollar debe ser capaz de cumplir con los siguientes requisitos:

- Leer la temperatura de la estancia mediante un sensor analógico.
- Leer la luminosidad de la estancia mediante una foto-resistencia.
- Conectar el sistema a la red wifi determinada para dar salida a internet al sistema empotrado.
- Enviar periódicamente a Arp@Lab Network Stats los valores leidos por los sensores.

- Recibir periódicamente desde un servidor de internet los valores de los umbrales de temperatura y luminosidad definidos por el usuario.

- Si el valor leído de temperatura es menor que el umbral establecido por el usuario, encender la calefacción.

- Si el valor leído de luminosidad es menor que el umbral establecido por el usuario, proceder a encender la iluminación.

- Por simplificación del hardware, simularemos los dispositivos a encender mediante un led para el procesado de temperatura y de otro led para el de luminosidad.

- Para la visualización y manejo de los valores de los umbrales, se desarrollará una aplicación alojada en Google App Engine, GAE. Se pretende que la interface de usuario web no añada requisitos técnicos superfluos, para hacerla accesible desde el mayor número de dispositivos posible.

1.4 Tecnologías empleadas

En el proyecto se usan las siguientes tecnologías en la parte del sistema embebido:

- lenguaje C

- sistema operativo en tiempo real FreeRTOS v7.3

- IDE LPCXpresso v4.3, basado en Eclipse Helios 3.6

- HTML 1.1

Y en la aplicación web:

- Java 1.6

- GWT 2.4

- Eclipse Java EE IDE for Web Developers, versión JUNO 4.2

- App Engine Java SDK 1.7.4

- DataNucleus AccessPlatform 3.1, JDO
- HTML 1.1
- XML
- CSS

Como podemos ver, trabajamos con un muy amplio espectro de tecnologías. Trabajamos con funciones de "bajo nivel" como el lenguaje C, sincronización de procesos mediante semáforos y colas, y multitarea. A la vez, en el desarrollo empleamos lenguajes de "alto nivel", como el lenguaje Java, con orientación a objetos, HTML, XML ó CSS.

El código programado se ejecutará en un sistema embebido, donde habremos tenido que diseñar un sencillo circuito electrónico eligiendo los componentes, y, también, desarrollamos una aplicación web que desplegamos en la nube.

2 Kit de sistemas empotrados

Para la realización del tfc en el área de sistemas empotrados se nos hace llegar un kit. El kit está compuesto de 3 dispositivos electrónicos y se apoya en un IDE descargable desde internet.

2.1 Descripción del hardware. Kit de sistemas empotrados.

La parte fundamental del kit es la placa LPCXpresso Board "LPC1769 Target", comercializada por Embedded Artists. La placa consta de dos partes, unidas pero independientes, la placa del microcontrolador y la electrónica necesaria para su funcionamiento o target board, y la placa del programador y depurador o JTAG.

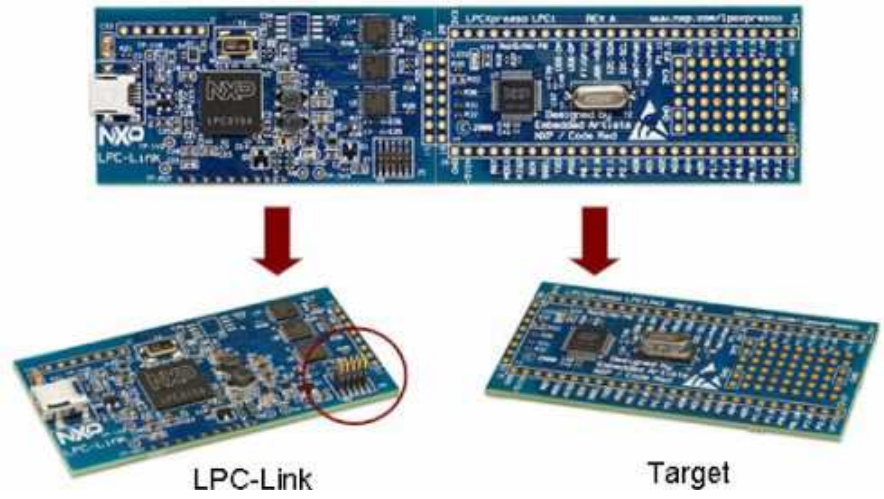

Figura 2. LPC1769 Target

LPC1769 se presenta con un procesador ARM Cortex-M3 de NXP (antigua Philips Semiconductors), memoria de datos de 64 kB SRAM y memoria flash de 512 kB. El procesador opera con una frecuencia de 120 Mhz.

Podemos ver el esquema de la placa aquí: www.embeddedartists.com/sites/default/files/docs/schematics/LPCXpressoLPC1769revB.pdf

Entre los periféricos integrados, destacamos 4xUART, 3xl2C serie, SPI serie, 2xSSP serie, 2xCAN bus interface, PWM modulación por ancho de pulsos, USB 2.0 Device/Host/OTG, RTC, Ethernet, diversos puertos ADC y GPIO, etc.

La placa puede alimentarse mediante fuente externa de entre 3.15V y 3.3V o desde el conector usb del JTAG.

Para nuestra fase de entrenamiento podemos usar la target board y el JTAG unidos, tal y como vienen. También podríamos cortar la placa en sus dos partes y manejar ambas, de manera independiente pero en conjunto, rehaciendo las conexiones. Podemos, incluso, utilizar otro JTAG que satisfaga de mejor manera nuestras necesidades de desarrollo. En nuestro caso, mantendremos las dos partes unidas, empleando el JTAG en las primeras fases como programador y depurador, y, al final del proyecto, como la manera más sencilla de alimentar el circuito mediante el conector usb.

El segundo elemento del kit es un adaptador USB-UART. Hace de puente, bridge, entre estos dos formatos. Gracias a los drivers disponibles, permite emplear dispositivos RS-232 a través de un puerto USB. El dispositivo dispone de una memoria EEPROM para programar su identificación, facilitando su uso para aplicaciones en desarrollos OEM. El adaptador tiene los pines necesarios para alimentar otros dispositivos a 3.3V y 5V.

El tercer y último elemento de nuestro kit es un módulo WiFly RN-171 de Roving Networks. Permite interactuar con redes wifi TCP/IP estándar mediante una interface serie RS-232. El dispositivo es de consumo ultra bajo, soporta redes Adhoc y de infraestructura, y su stack TCP/IP incluye soprte para TCP, DHCP, UDP, DNS, ARP, ICMP y clientes HTTP y FTP.

El módulo WiFly incluye antena, y la potencia de emisión puede configurarse mediante el firmware entre 0 dBm y 12 dBm. Además, incluye pines para 8 entradas/ salidas digitales y 3 entradas para sensores analógicos.

Podemos ver el esquema del módulo aquí: http://www.rovingnetworks.com/resources/download/16/Frn_xv

2.2 Entorno de desarrollo

El entorno de desarrollo recibe el nombre de LPCXpresso IDE, y ha sido desarrollado en conjunto por CodeRed y NXP. El IDE está basado en Eclipse, manteniendo básicamente su aspecto y manera de trabajar. Como principal aporte, indicar que está adaptado específicamente para manejar e interactuar con el JTAG y el target.

Nuestro IDE se instala sin problemas ejecutando un único fichero, descargable desde la web de CodeRed. La versión básica y gratuita incluye la funcionalidad de la versión superior de pago, con la única limitación de tamaño del programa ejecutable del target. Se exige registro, proporcionando identificación y una dirección de correo electrónico. El proceso entero de localizar, descargar, instalar y registrar el entorno lleva unos 10 minutos, en nuestro caso en una máquina con Windows 7.

El IDE se basa en los conceptos de workspace y proyecto como manera de ordenar nuestra labor de programación.

Los proyectos pueden ser bibliotecas estáticas o aplicaciones ejecutables, y contienen archivos de código fuente, encabezados y cualquier otro archivo necesario. Podemos compilar cualquier proyecto, pero serán las aplicaciones ejecutables, con main(), las que descargaremos y ejecutaremos en nuestro microcontrolador.

El espacio de trabajo o workspace funciona como contenedor de proyectos, sean aplicaciones o bibliotecas, y, también, incluye las configuraciones del entorno. Esta disposición agrupada de toda la información involucrada en la gestión de proyectos de desarrollo, permite copiar o mover fácilmente el contenido de una máquina a otra.

2.3 Sistema operativo. FreeRTOS.

FreeRTOS (FREERTOS, 2012) es un sistema operativo en tiempo real para sistemas embebidos escrito en lenguaje C. Los sistemas embebidos disponen de una memoria reducida y de baja potencia de cómputo, a pesar de ello, el sistema operativo consigue lograr tiempos reducidos de reacción y tiempos de respuesta predecibles. Proporciona acceso rápido al hardware y capacidad de priorización.

Para lograr todo ello, el sistema consta de una sola imagen binaria, que incluye el kernel y las aplicaciones, y dominio del algoritmo de scheduling (LORENZATI, 2010).

Parte de la configuración del funcionamiento del OS se produce en tiempo de compilación, como la definición del timer del sistema o la pila de memoria (FreeRTOSConfig.h).

En tiempo de compilación, mediante las constantes correspondientes que funcionan como parámetros, podemos configurar el tipo de multitarea (preemptive or cooperative), tick propio o externo, velocidad en Hz de la cpu, velocidad en Hz del tick si es interno, prioridad máxima de las tareas, tamaño de pila y cola, aspecto de mutexes, semáforos, etc.

Otra parte se gestiona en tiempo de ejecución, mediante el manejador de tareas. La tarea es la unidad básica de ejecución en un RTOS, y lo es también en FreeRTOS. Cada tarea puede instanciarse, con un contexto propio e independiente para cada una de las instancias. Con limitaciones y solo en modo cooperativo, entre distintas instancias de tareas podemos utilizar corutinas. Las corutinas comparten el mismo contexto entre diferentes instancias.

Las aplicaciones en tiempo real, que usan un RTOS, se estructuran como un conjunto de tareas independientes, TASKS. Las tareas se ejecutan en contextos independientes, sin relación entre los contextos de distintas tareas, excepto en lo que se refiere al planificador del sistema operativo, o scheduler, que las gestiona.

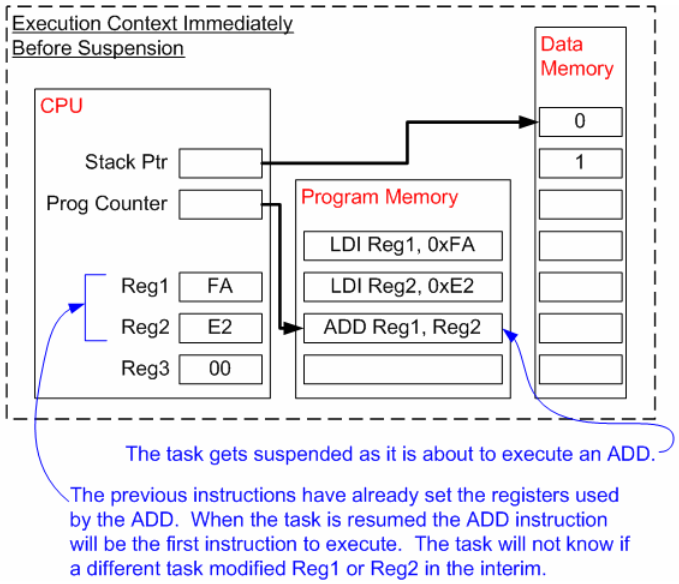

Figura 3. Contextos en FreeRTOS

En el caso de sistemas con muy limitados recursos de procesador y memoria, podríamos emplear otra aproximación al concepto de multitarea, con gestión cooperativa mediante lo que FreeRTOS denomina Co-Routines. Como no es nuestro caso, no le dedicamos más espacio a este concepto.

Las tareas utilizan el procesador, sus registros, la memoria RAM y la memoria ROM, la pila, los eventos o las señales como cualquier otro programa. Estos recursos en conjunto forman lo que se denomina contextos de ejecución.

Las tareas no saben cuando van a ser suspendidas, pueden serlo en cualquier punto, es el kernel quien las suspende o reactiva. Y es el kernel el responsable de guardar y restaurar el contexto de las tareas que suspende o reactiva de manera transparente para las mismas.

Las tareas no instanciadas no consumen tiempo de procesador, ni ocupan memoria. Las tareas en espera no consumen tiempo de procesador, pero sí memoria. Es importante tener esto en cuenta y usar los mecanismos que proporciona FreeRTOS para sincronizar tareas.

Las tareas son interrumpibles por IRQ de hardware, o por cambios de prioridades o contexto de FreeRTOS. También pueden interrumpirse por cambios de estado forzados desde las tareas.

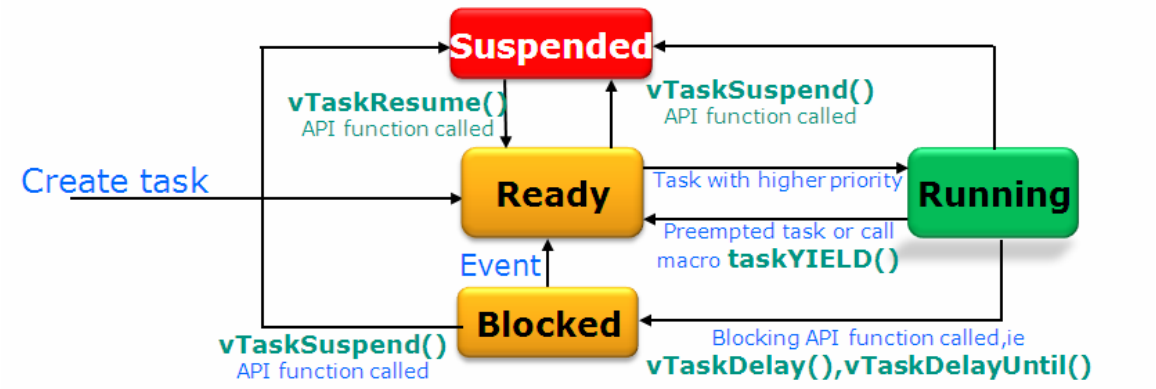

Figura 4. Estado de tareas en FreeRTOS

Las tareas pueden interactuar con otras tareas. Para sincronizar, proteger y compartir los procesos y los datos, disponemos de

semáforos binarios y semáforos contadores. FreeRTOS proporciona un tipo especializado de semáforos binarios denominados MUTEX, que están indicados para la protección del acceso a un recurso común serializado.

El intercambio de información entre tareas en FreeRTOS, en general, puede hacerse mediante el uso de variables globales y la adecuada sincronización de sus procesos de lectura y escritura que garanticen la integridad de la información procesada.

No usaremos este método general, porque la API de FreeRTOS proprciona también un método específico de intercambio de mensajes entre tareas, y entre tareas e interrupciones, denominado QUEUES. Las colas contienen elementos, o items, de tamaño fijo. Pueden contener un máximo de elementos que se definen en el proceso de creación de la cola.

Las llamadas a la API de FreeRTOS manejan y garantizan las cuestiones de de exclusión mutua entre las tareas que hagan uso de las colas, por lo que simplifican enormemente el intercambio de mensajes entre tareas.

Otros elementos de la API de FreeRTOS son gestión de sincronización, de captura y liberación de recursos compartidos, colas de mensajes, pipes, gestión de memoria compartida entre tareas, etc.

Además del núcleo del sistema operativo, FreeRTOS pone a nuestra disposición una capa adicional que abstrae los detalles de las llamadas a periféricos, incluyendo posibles interrupciones que requieran para su funcionamiento. Esta capa se denomina FreeRTOS+IO, y se descarga y gestiona de manera independiente, aunque similar, al núcleo del sistema.

La librería permite llamadas a los periféricos al estilo POSIX, y permite cuatro modos de lectura: sondeo (polled), cola circular gestionada mediante interrupciones, cola circular sin copia (zero copy) y gestionada mediante interrupciones con cola de caracteres. Con los modos primero y último podemos, también, escribir (PITCHER, 2012).

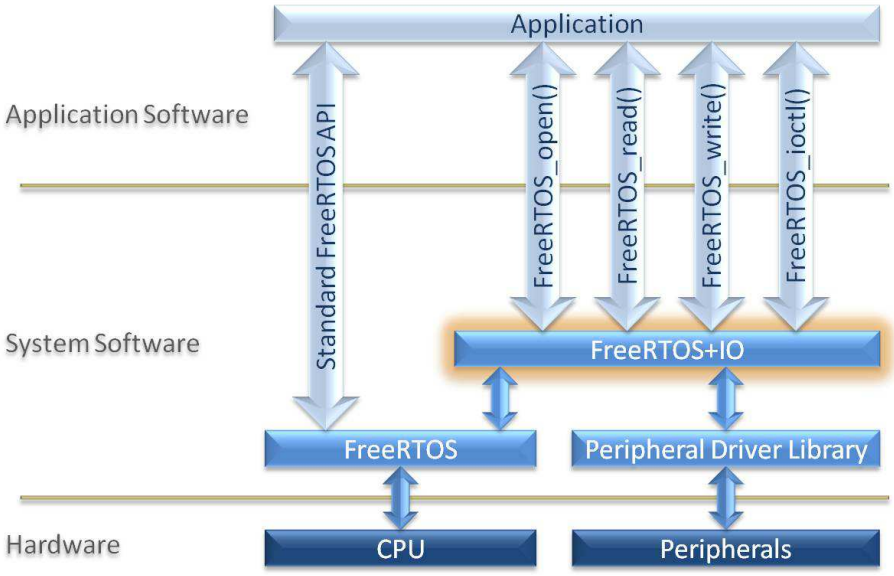

Figura 5. FreeRTOS+IO

2.4 Programas de ejemplo

El punto de partida imprescindible en el proceso de aprendizaje, son los programas de ejemplo. Los obtenemos descargando los workspaces correspondientes, aunque también podemos hacerlo empleando proyectos. Los fuentes están en lenguaje C, puesto que nuestro IDE no admite C++.

Por si no estuvieran desarrollados para nuestro dispositivo específico, debemos asegurarnos de incluir la librería CMSIS adecuada para el LPC1769, la última versión actualmente es la CMSISv2p00. Debemos incluir también el kernel de nuestro sistema operativo, con la libería FreeRTOS.

Antes de llegar a ejecutar cualquiera de los ejemplos, hay que asegurarse de configurar el target adecuado (LPC1769), y configurar las dependencias y el orden de las distintas librerías. La configuración puede hacerse informando los valores adecuados en los distintos menús del IDE. En mi caso, al desconocer el manejo de Eclipse, ha sido la parte menos intuitiva y la que más tiempo me ha llevado para entender el manejo del proceso de software.

Una vez configuradas las dependencias de las librerías y nuestro target, podemos compilar el código fuente sin errores. Se ejecutará

automáticamente el linker como parte de nuestro tool-chain para ya obtener el fichero descargable con nuestro ejecutable.

Generado el ejecutable, lo descargamos en la memoria flash de nuestro LPC, indicando al IDE que lo haremos mediante el JTAG. El JTAG reseteará el target, y, automáticamente, el microcontrolador ejecutará nuestro programa.

Gracias al JTAG, y siempre mediante nuestro IDE, podemos depurar nuestro programa. Disponemos de las opciones habituales en cualquier depurador, con ejecución paso a paso, inspección de valores de variables y registros, modificación de la instrucción a ejecutar, o gestión de puntos de ruptura, trabajando con distintas perspectivas según estemos depurando o en proceso de desarrollo (codificación). Es en este punto donde más he podido apreciar la potencia de un IDE basado en Eclipse.

2.5 Módulo Wifly

En este apartado describiremos la configuración inicial del módulo WiFly para verificar su funcionamiento, y establecer una configuración inicial para preparar el uso del módulo como parte de nuestro sistema en el desarrollo de nuestro producto software final.

Figura 6. Módulo Wifly

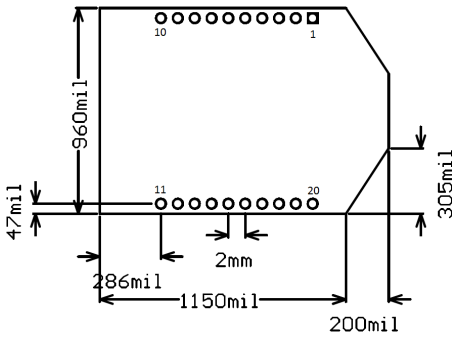

Figura 7. Dimensiones módulo Wifly

Podemos ver a simple vista, con la tabla de pines, que el módulo Wifly no es un simple adaptador que nos de salida a la red wifi. De hecho, podríamos desarrollar una solución similar a la del proyecto, aunque limitada, conectando los sensores directamente al módulo Wifly y prescindiendo del LPC.

Pad Number	Signal Name	Description	Optional Function	Direction
1	VDD_3V3	3.3V regulated power input to the module		POWER
2	UART_TX	UART TX, 8mA drive, 3.3V tolerant		OUT →
3	UART_RX	UART RX, 3.3V tolerant		IN ←
4	GPIO 8	GPIO, 24mA drive, 3.3V tolerant		IN / OUT
5	RESET	**Optional** Module Reset Signal (active low), 100k Pull up, apply pulse of at least 160us, 3.3V Tolerant		INPUT
6	GPIO 5	GPIO, 24mA drive, 3.3V tolerant	Data TX/RX	OUT
7	GPIO 7	GPIO, 24mA drive, 3.3V tolerant		IN / OUT
8	GPIO 9	Enable Adhoc mode, Restore factory defaults, 8mA drive, 3.3V tolerant		IN / OUT
9	GPIO 1	GPIO, 8mA drive, 3.3V tolerant		IN / OUT
10	GND	Ground		GND
11	GPIO 14	GPIO, 8mA drive, 3.3V tolerant		IN / OUT
12	UART_RTS	UART RTS flow control, 8mA drive, 3.3V tolerant		OUT →
13	GPIO 4/SEN 6	GPIO, 24mA drive, 3.3V tolerant/ADC input , (**3.3V tolerant).** Defaults to GPIO 4	Association Status	IN / OUT
14	Not Used			No Connect
15	GPIO 6/SEN 7	GPIO, 24mA drive, 3.3V tolerant/ADC input , (**3.3V tolerant).** Defaults to GPIO 6	Connection Status	POWER
16	UART_CTS	UART CTS flow control, 3.3V tolerant		IN ←
17	SENSOR 5	Sensor Interface, Analog input to module, (**3.3V tolerant)**		INPUT
18	GPIO 3/SEN 4	GPIO, 8mA drive, 3.3V tolerant/ADC input (**3.3V tolerant**). Defaults to GPIO 3		IN / OUT
19	GPIO 2/SEN 3	GPIO, 8mA drive, 3.3V tolerant/ADC input (**3.3V tolerant**). Defaults to SEN 3		IN / OUT
20	SEN 2	Sensor Interface, Analog input to module, **3.3V tolerant**		INPUT

Figura 8. Tabla pin out módulo Wifly

2.5.1 Conexiones hardware

El módulo WiFly requiere de la conexión de los pines de tierra y alimentación, 3.3V. En nuestro caso se preveía la alimentación inicial del módulo desde el pin correspondiente del adaptador USB-UART. Parece ser que hay un problema con el módulo adaptador de mi kit de sistemas empotrados, por lo que a pesar de repetidos intentos y mucho tiempo dedicado a buscar algún error en las conexiones, no logré hacer funcionar el módulo de esta manera. Una vez conectado a las patillas correspondientes del LPC1769, el módulo WiFly funcionó sin mayores inconvenientes.

Con la alimentación eléctrica podemos manejar el módulo desde la red, pudiendo entrar en modo adhoc conectando la patilla correspondiente a 3.3V.

En nuestro caso, procedimos a conectar las patillas tx y rx al adaptador. A partir de ese momento, podíamos comunicarnos con el módulo a través del puerto USB del ordenador y de un programa de comunicaciones.

2.5.2 Configuración

Para configurar nuestro WiFly, y darle acceso a la red wifi, puede emplearse cualquier programa de comunicaciones estándar que soporte conexiones serie. En mi caso empleé el programa CoolTerm, por potencia y facilidad de uso. Se activa el modo comando introduciendo $$$ (sin retorno de carro). A partir de ahí, los comandos se introducen añadiendo <enter> al final de cada uno. Adjunto copia del resultado de los comandos básicos, que son autoexplicativos.

Debemos establecer la SSID de nuestra red, su nivel de seguridad, WPA2 en nuestro caso, y la password asociada si es necesario.

Para finalizar, grabamos la configuración para no tener que repetir las órdenes cada vez que vayamos a utilizar el módulo.

A partir de ese momento, salvo que especificáramos lo contrario, el módulo Wifly intentará asociarse a la red que tiene configurada cada vez que se encienda. Podemos ver el resultado de todo el proceso en

la copia de los comandos introducidos y las respuestas del sistema en la figura de más abajo.

Durante todo el proceso, el módulo informa de su estado de manera muy visual mediante leds de colores, que parpadean o no según proceda.

No he encontrado ninguna dificultad reseñable en este proceso. Creo que el módulo WiFly es muy sencillo de manejar a nivel de comandos para las tareas básicas de configuración inicial.

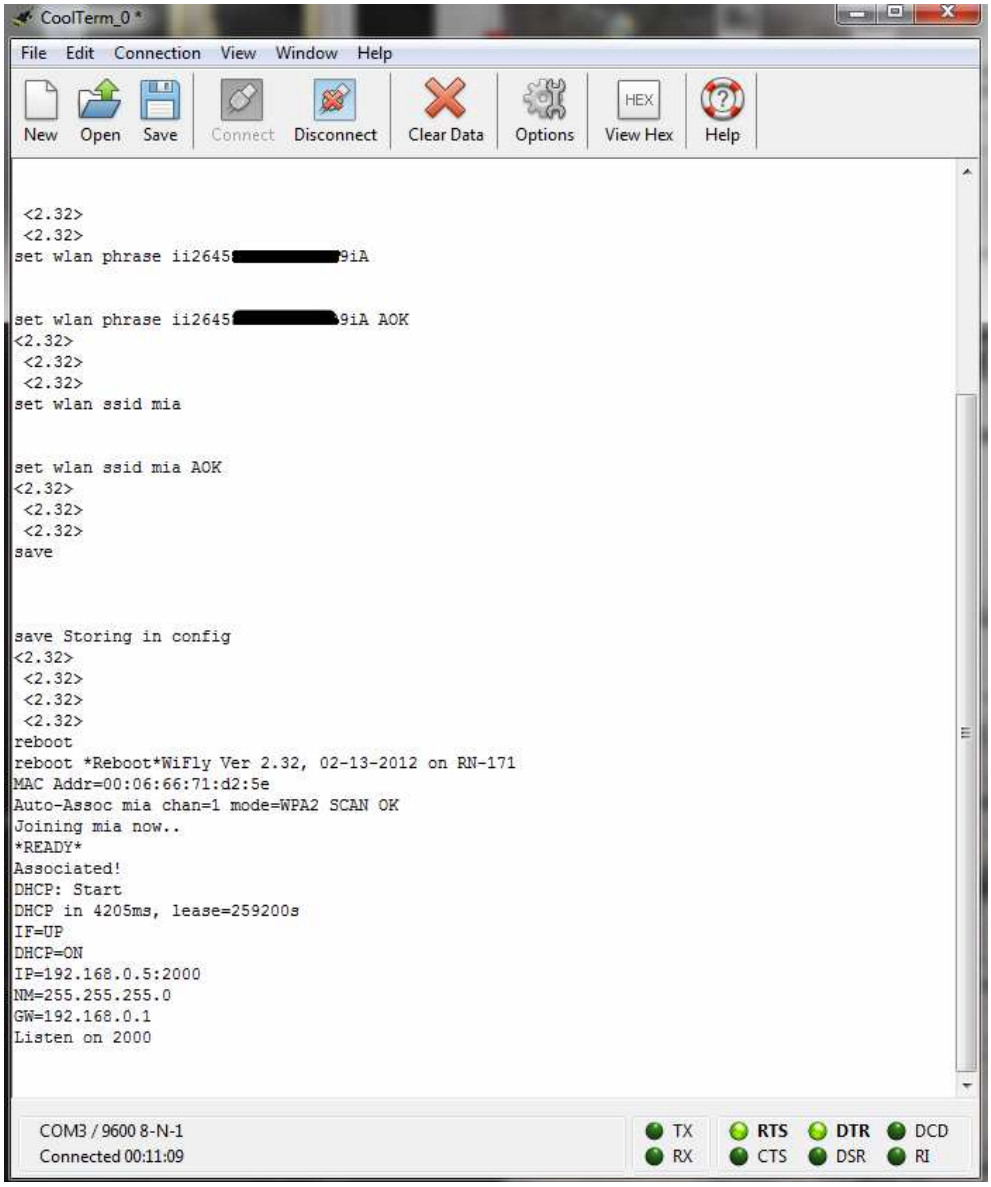

Figura 9. Resultado configuración Wifly

3 Manejo de periféricos en LPCXpresso y LPC1769

Un aspecto fundamental en el desarrollo de nuestro proyecto es el manejo de los periféricos de nuestro sistema empotrado.

El punto de partida inicial en la fase de aprendizaje, lógicamente, ha sido el código de ejemplo proporcionado en el aula de la asignatura y en la wiki.

La referencia al ir avanzando en complejidad, o cuando se presentaban problemas a los que no era fácil encontrar solución, han sido los foros de soporte de NXP. La documentación del fabricante es muy amplia, aunque desde mi punto de vista tiene una organización mejorable. Trata los aspectos relevantes en muy alto grado de profundidad, pero creo que no está bien orientada para principiantes. Una vez que se dominan los aspectos básicos de funcionamiento del hardware y del propio IDE con ayuda de otros medios externos, volveremos a los foros oficiales para resolver los problemas más específicos de la plataforma (NXP, 2012).

Como contrapunto a la complejidad de la organización de la información en NXP, hay multitud de sitios web dedicados al desarrollo para el ARM Cortex M3 que se caracterizan por su sencillez y claridad en la presentación y comprensión de la información, y del código, que presentan. En concreto, para el manejo de los periféricos ADC y GPIO hemos optado por el uso de las librerías SimpleCortex, de BRC-Electronics (BRC-Electronics, 2012).

En el manejo del módulo Wifly se ha partido de librerías de Mbed, principalmente la denominada Wifly (http://mbed.org/cookbook/wifly). Ha sido necesario reescribir la librería a lenguaje C desde C++, sustituyendo las llamadas a la API de Mbed por llamadas a código de NXP o propio. También ha sido necesario adaptarla al target LPC1769 desde el LPC1768, ya que las librerías de Mbed sólo soportan las versiones de procesador de las que tienen hardware en el mercado. (MBED, 2012). Para explorar y probar el código de Mbed y proyectos de jemplo del fabricante o de usuarios, ha sido útil trabajar con su compilador online (http://mbed.org/handbook/Compiler-Tour).

En este proceso de conversión entre plataformas, ha sido de gran ayuda trabajar con la demo del equivalente al LPCXpresso en versión de pago, denominada Red Suite. Respecto a la versión gratuita, este

IDE no tiene limitación en el tamaño de los ejecutables a manejar, tiene funcionalidades específicas de apoyo al proceso de depuración y, principalmente, admite código con fuentes en lenguaje C++.

3.1 Conexiones

Para trabajar con los periféricos, debemos realizar las conexiones adecuadas entre nuestro sistema embebido y los dispositivos externos que queramos emplear. El primer paso es identificar los distintos pines de nuestro LPC1769.

La mayoría de los pines admiten distintos usos, cuestión que debemos tener en cuenta al escoger el número de puerto de cada tipo que necesitemos en nuestro diseño para cada uno de los dispositivos a conectar. Lo vemos más claramente en la tabla pin-out de referencia del fabricante:

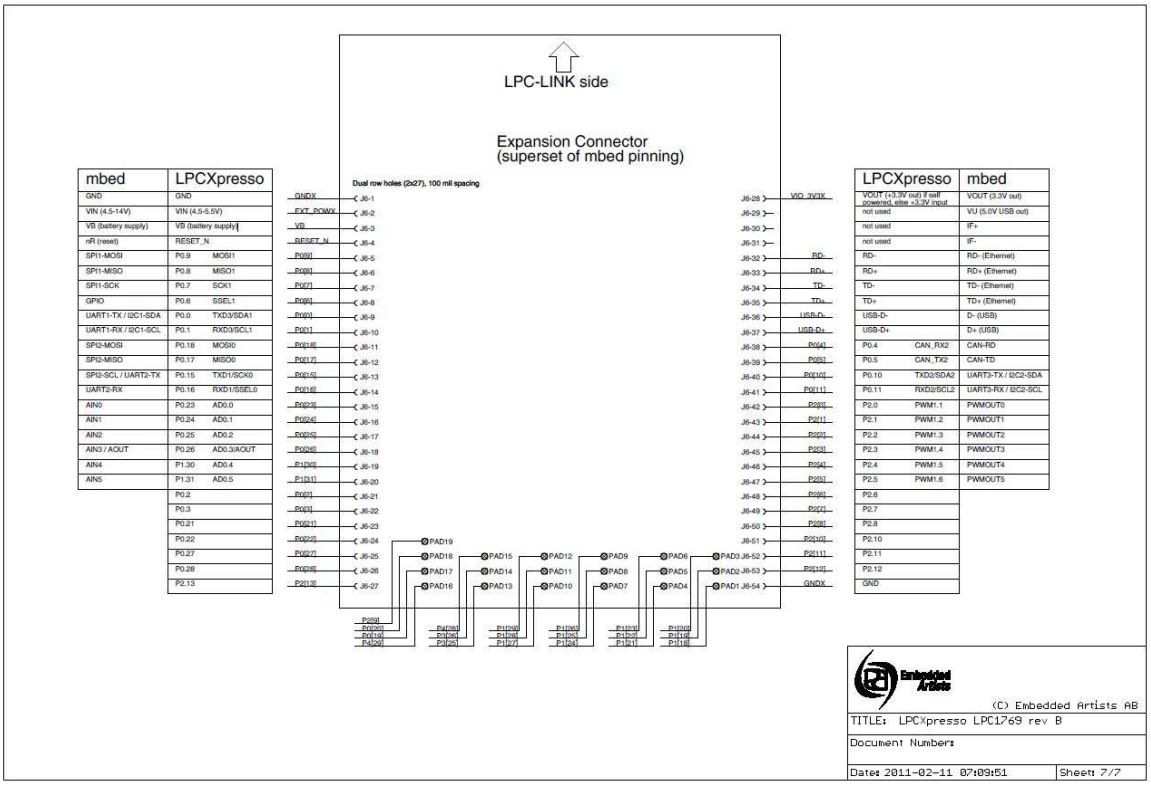

Figura 10. Pin out LPC1769

Reseñar que el paso, la distancia entre pines, en la placa del LPC1769 y del módulo Wifly son distintas, por lo que no podremos emplear la misma placa para ambos circuitos. Es por eso que hay adaptadores dip para el formato Xbee del módulo Wifly.

Figura 11. Adaptador DIP para Xbee

Dado que requieren soldadura "de precisión", y ante la duda de disponer de uno a tiempo, optamos por realizar las conexiones con la ayuda de una placa de inserción. A pesar de que, a simple vista, aparenten ser conexiones débiles, en el transcurso de los meses demostraron ser suficientemente fiables.

Estas fotografías muestran las conexiones iniciales realizadas para la fase de entrenamiento en el manejo de periféricos con la que era posible ejecutar el código presentado para la pec1.

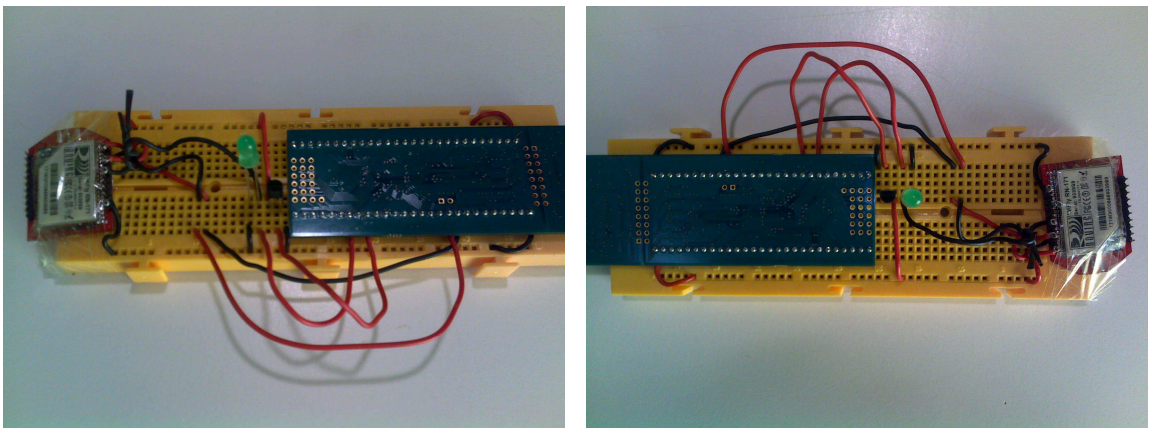

Figura 12. Conexiones programa de entrenamiento

3.2 UARTs

El uso de UARTs es básico para la comunicación con el módulo Wifly, y muy práctico como medio de extraer información del sistema en la fase de desarrollo y depuración del código.

El LPC1769 dispone de varios puertos UART. En nuestro código, gestionamos el puerto UART3, mediante los pines J6-9 para TX y J6-10 para RX.

La opción de emplear el puerto UART0, mediante los pines J6-21 para TX y J6-22 para RX, se descartó, para evitar limitaciones, por coincidir los pines con los puertos digitales P0.2 y P0.3, que también pueden ser puertos digitales.

3.3 GPIO

El manejo de puertos digitales mediante GPIO es el más sencillo de los posibles. En nuestro código, leemos en el puerto P0.3, pin J6-22 y escribimos en el puerto P0.2, pin J6-21, puerto P2.6, pin J6-48 y puerto P2.7, pin J6-49

- GPIO output: manejamos un led, encendiéndolo y apagándolo. Al activarlo, se pone en nivel alto, 3.3V, y se enciende el led. Al desactivarlo, se pone en nivel de GND y se apaga. Cuando sea necesario, podremos ajustar a lo necesario el tiempo de encendido o apagado, o regular la velocidad de parpadeo mediante timers del sistema en la fase de entrenamiento y mediante llamadas a delay de la API de FreeRTOS en la fase final de codificación.

- GPIO input: la gestión de los puertos de entrada, requiere que en su inicialización se informe al sistema del tipo de activación que presenta. Los más habituales son pulldown y pullup. En el primer caso, el pin se detecta como activo hasta que haya una muy baja resistencia a tierra. En el segundo caso es justo al contrario, se lee como activo si el voltaje leido es 3.3V. Hay que tener esto en cuenta en las conexiones a otros dispositivos, como el módulo Wifly, o botones, por ejemplo.

3.4 ADC

Los puertos ADC, que es la abreviatura de analog to digital converter, permiten leer el voltaje de uno de los pines que permita esta función, y convertirlo a un número digital.

Para la lectura de la temperatura ambiente, decidimos optar por un sensor TMP36, uno de los más populares y sencillos de utilizar. Proporciona una respuesta lineal, puede operar entre -40ºC y 100ºC, y no necesita calibración para un margen de error del 1%.

El precio es de alrededor de 2,30 EUR. Hoja de características: www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf

Figura 13. Sensor de temperatura TMP36

En nuestro código utilizamos el puerto AD0.4, P1.30, pin J6-19, conectado a la patilla central del TMP36. El sensor debe conectarse de manera adecuada, puesto que equivocar las patillas de GND y 3.3V inutiliza el circuito en segundos.

Para la lectura de la luminosidad elegimos una sencilla fotorresistencia LDR CE-C2795. Como cualquier resistencia variable, para poder realizar las lecturas necesitaremos acoplarla a una resistencia de valor conocido (4K7 ohmnios) y montar un divisor de voltaje.

El precio es de alrededor de 0,55 EUR, aunque varía mucho de un distribuidor a otro. Podemos ver la hoja de características aquí: http://www.cebek.com/docs/C-2795.pdf

Figura 14. Fotoresistencia LDR

En nuestro código utilizamos el puerto AD0.5, P1.31, pin J6-20, conectado a las patillas centrales del divisor de voltaje.

Para evitar errores en las mediciones, es necesario evitar interferencias en las lecturas provocadas por otros puertos declarados como ADC, o por ruidos en la línea de GND. También ayuda declarar como puertos digitales todos los puertos ADC que no necesitemos. Con unas cuidadosas conexiones a GND y 3.3V, descarte de medidas muy fuera de rango para nuestro entorno (>37º ó < 8º) y mediando repetidas muestras, el resultado resulta muy estable y fiable.

3.5 Cliente Wifly HTTP

El uso del módulo Wifly proporcionado en el kit de sistemas empotrados, es fundamental para abrir nuestro microcontrolador al mundo exterior a través de la red y de internet.

En nuestro caso, ha sido un proceso largo y complejo, porque el módulo original ha quedado totalmente inutilizado por un fallo de hardware. El módulo presentaba un comportamiento no predecible, incluso errático a ratos, dificultando y retrasando nuestro desarrollo en un mínimo de 3 semanas. El primer síntoma fue no encenderse proporcionándole alimentación de 3.3V desde el adaptador UART-USB. En días posteriores, no respondía adecuadamente a los comandos introducidos desde el LPC1769, e, incluso, desde el propio adaptador. No puedo asegurar si el fallo de hardware ha sido provocado por alguna mala conexión que yo haya realizado, que provocara algún cortocircuito por ejemplo, o que el módulo fuera defectuoso desde el principio. La solución fue sustituirlo por uno nuevo del mismo modelo.

Para el manejo del módulo Wifly, empleamos el puerto UART3 para enviar comandos y recibir información. También, mediante GPIO con puerto P0.2, pin J6-21, manejamos el pin de reset del módulo Wifly, pin 5. El reset del Wifly se provoca conectando a tierra, y lo usamos en su inicialización, y como manera de resolver los posibles problemas imprevistos en el transcurso de las conexiones con servidores o clientes de internet.

Está previsto, aunque no implementado, el control de los estados de las conexiones a internet del Wifly mediante la patilla 15 del módulo, denominada connection status. Requiere establecer por comando lo que Roving Networks denomina "alternate GPIO functions". Dado que todavía no se usa, por simplicidad no aparece en las fotografías de las conexiones.

A modo de entrenamiento, realizamos un programa que lee la temperatura desde el sensor analógico antes descrito, y lo envía al servidor Arp@ Network Stats de la wiki del TFC mediante las llamadas correspondientes. También, lee y extrae el valor recibido en una llamada al Arp@Lab Feed, y lo vuelve areenviar al servidor Arp@ anterior.

Usamos el cliente HTTP 1.1 implementado en el módulo Wifly, intentando mantener activa la conexión con el servidor para reducir el coste de las conexiones.

Resultado del proceso en la web de Arp@Lab, con la lectura del sensor de temperatura y con la lectura y reenvío del dato del feed.

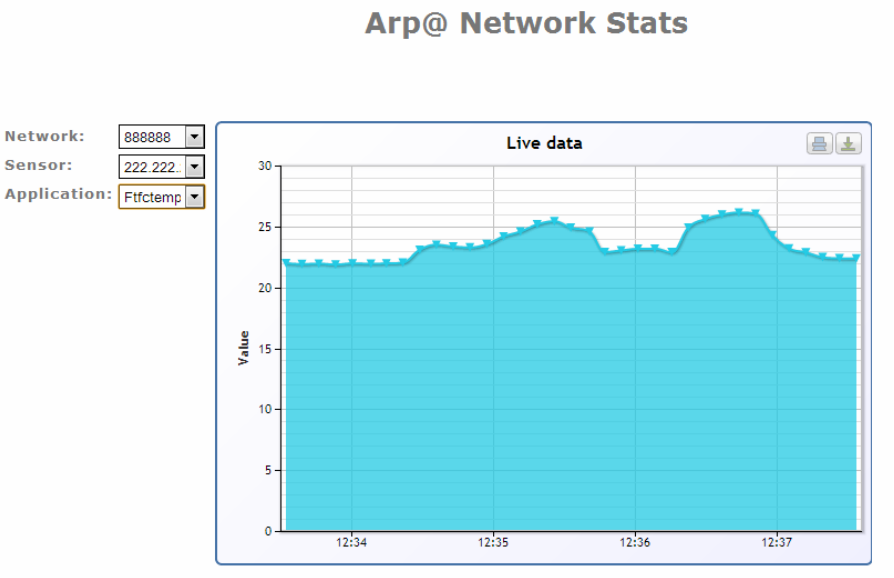

Figura 15. Lectura del sensor de temperatura en Arp@Lab

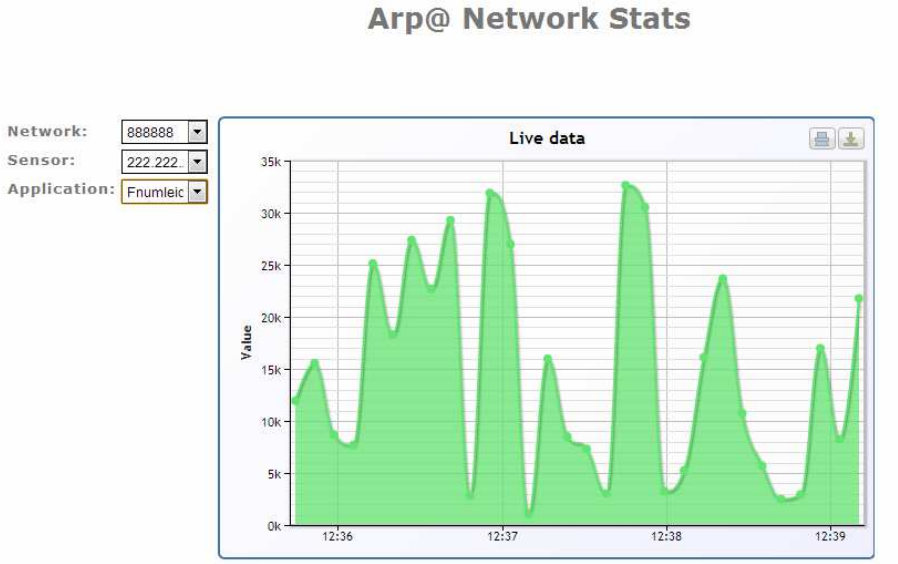

Figura 16. Datos recibidos del feed reenviados a Arp@Lab

4 Diseño

4.1 Diseño general del sistema

La aplicación consta de una parte que se ejecuta en el sistema empotrado y de otra parte que se ejecuta en la web.

Como requisitos generales, hemos buscado diseñar una aplicación que sea muy fácil de usar por parte del usuario, con un punto único de acceso, que abarque consulta y gestión.

También, hemos optado por un servidor en la nube, accesible desde internet.

La aplicación está desarrollada con tecnologías que permiten utilizar prácticamente cualquier dispositivo, bien sea un ordenador personal estándar, tableta o móvil.

Por último, y no menos importante, en la elección del servidor se ha tenido en cuenta escoger el de menor coste disponible, monetario y en tiempo de instalación y mantenimiento.

4.2 Aplicación web

La aplicación web se encarga de la interacción con el usuario y de la gestión de la persistencia de los datos por él introducidos.

Para el despliegue hemos optado por un servidor de Google App Engine, en su versión gratuita.

Se ha utilizado un lenguaje estándar con orientación a objetos, Java en versión 1.6.

La aplicación está desarrollada empleando el IDE Eclipse Java EE for Web Developers, versión Juno 4.2. Se trata de un entorno de desarrollo de los más usados actualmente, gratuito, y adaptable mediante plugins a multitud de lenguajes y sistemas. Dispone de una

ingente cantidad de tutoriales e información de soporte disponible en la web.

Dado que hemos optado por desplegar la solución en un servidor de Google App Engine, usamos App Engine Java 1.7.4 y Google Web Toolkit GWT 2.4.

Para la persistencia la opción elegida ha sido DataNucleus AccessPlatform 3.1, JDO. Potencia y sencillez. Nuestras necesidades en este punto eran muy básicas, pero el manejo demuestra que ha sido buena elección.

Para la presentación, determinado por las elecciones anteriores, el estándar HTML 1.1, XML y CSS como no podía ser de otra manera.

Como podemos ver, todas las tecnología empleadas en la aplicación web son gratuitas, y, a la vez, algunas de las que están más de actualidad en su campo.

Procedemos a continuación a detallar la funcionalidad de la parte servidor y de la parte cliente de la aplicación web.

4.2.1 Aplicación web: cliente.

La parte cliente de aplicación web permite al usuario introducir en el sistema los valores de los umbrales de temperatura y luminosidad que determinan el encendido de la calefacción y la iluminación.

El acceso a la información y gestión de la misma por parte del usuario se realiza a través de una única página web, que se encarga de presentar la información y realizar las llamadas correspondientes al servidor de manera transparente para el usuario.

Las llamadas al servidor, para recuperar o actualizar los valores de los umbrales, son asíncronas, tal y como se acostumbra, de manera que respuestas lentas desde el servidor no hagan parecer la aplicación como poco ágil o bloqueada.

En cuanto a la presentación de la información de las lecturas de los sensores, hemos utilizado la aplicación Arp@ Network Stats, tal y como se nos ha indicado en la asignatura. No procedemos a comentar más detalles al respecto, puesto que no hemos introducido ninguna

modificación al código. Nos limitamos a incrustar un iframe en la web de control para facilitar el acceso a la misma.

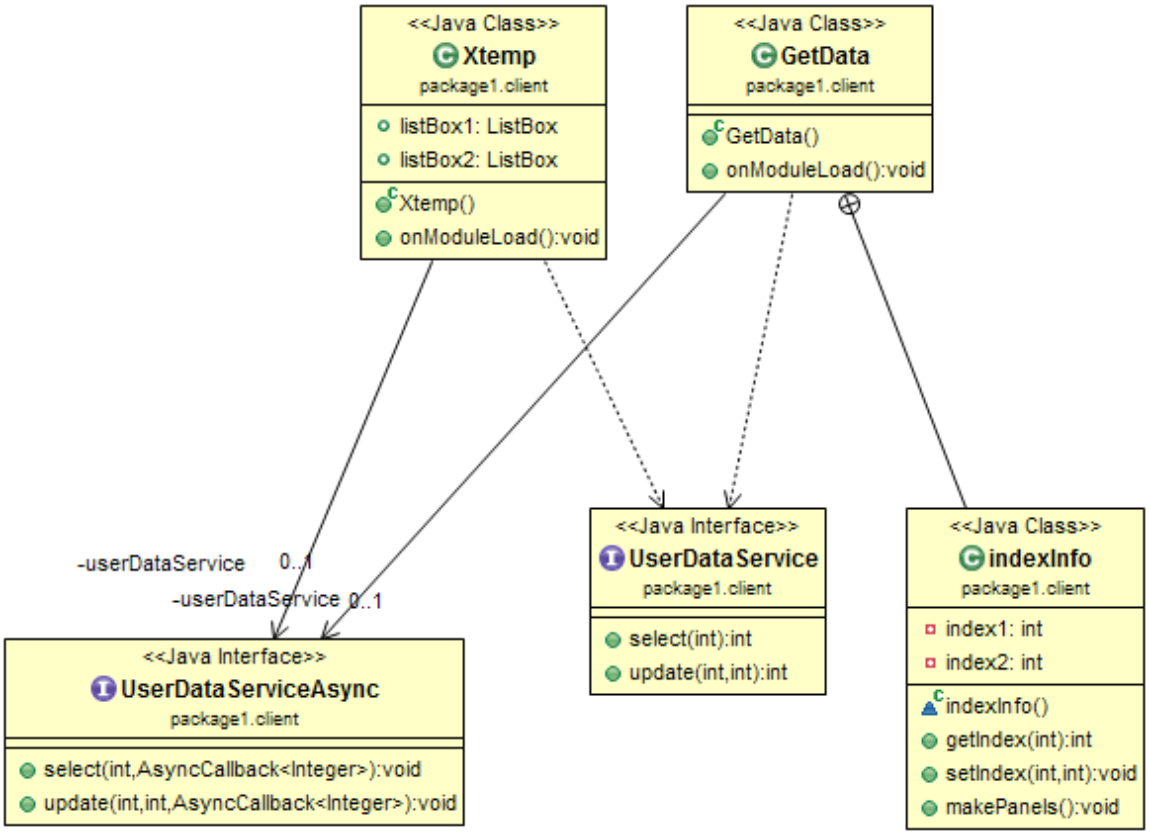

Figura 17. Diagrama UML: cliente

4.2.2 Aplicación web: servidor.

La parte del servidor de la aplicación web se encarga principalmente de la gestión de la persistencia. Permite recuperar los valores de los umbrales, bien para presentarlos al usuario a modo de consulta, bien para trasladarlos al sistema empotrado cuando éste los requiera.

La persistencia se resuelve con DataNucleus, JDO. Mantenemos exclusivamente el último valor informado de cada uno de los dos sensores, por lo que las necesidades de espacio son ínfimas (unos pocos bytes) y los tiempos de respuesta muy buenos. Hay que pensar

que tratamos con un servidor gratuito, por lo que restringir estas variables se hace imprescindible.

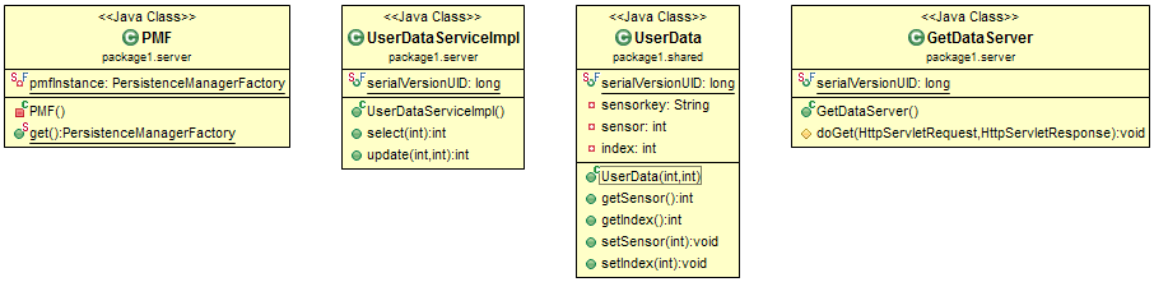

Figura 18. Diagrama UML: servidor

La parte del servidor, también se encarga de responder a las consultas del valor de los umbrales que le lanza el sistema empotrado. Esta función se resuelve en el servlet GetDataServer, accesible desde http://tfcsistemasempotrados.appspot.com/GetDataServer. La llamada devuelve el valor de los últimos umbrales de temperatura y luminosidad introducidos por el usuario, y válidos en ese momento.

Como es lógico, el cliente HTTP del módulo Wifly no soporta Java, por lo que la respuesta está en texto plano. Por simplicidad del tratamiento en el sistema empotrado, la longitud de los valores enviados es fija. La posición de los campos se determina por marcadores (flechas), y se añade un campo de control al final de la cadena para verificar que se ha recibido la respuesta completa (<-END).

Podemos ver a continuación un ejemplo de consulta con la información que devuelve este servlet y recibe el sistema empotrado a través del módulo Wifly:

```
Temp---->22<---- Lum---->0800<-END
```

Figura 19. Ejemplo de datos enviados desde servidor web al sistema empotrado

4.3 Aplicación del sistema empotrado

Nuestro programa tiene 5 tareas o TASKS:

- HttpInTask: se encarga de leer desde el servidor de internet los valores de los umbrales definidos por el usuario y almacenar los datos extraídos en su cola de salida.

- HttpOutTask: encargada de enviar a Arp@ Network Stats la información que le envían las 2 siguientes tareas a su cola de entrada. Los valores enviados pueden ser vistos gráficamente en la web.

- TempTask: encargada de la lectura del sensor de temperatura. Envía la información a HttpOutTask y a ProcInTempTask.

- LumTask: obtiene la lectura de la luminosidad ambiente y la envía a HttpOutTask para que le de salida a internet y a ProcInTempTask.

- ProcInTempTask: procesa la información proporcionada por HttpInTask con los valores de los umbrales de temperatura y luminosidad definidos por el usuario en la web, y compara con la información recibida desde TempTask y LumTask con las últimas lecturas de temperatura y luminosidad disponibles. En función del resultado enciende o apaga los leds que simulan el encendido de la calefacción o la iluminación.

Las tareas HttpIn y HttpOut manejan el módulo Wifly mediante el puerto UART3. Un uso simultáneo de este recurso provocaría mezclas entre los mensajes de entrada y salida al módulo de ambas tareas, y el correspondiente malfuncionameinto del sistema. Por tanto, debemos garantizar el uso exclusivo del recurso en disputa por una sola de las tareas. Una de las opciones presentes en FreeRTOS para la sincronización de procesos son los semáforos.

En nuestro caso usamos un semáforo binario, que en FreeRTOS se denomina MUTEX:

- HttpMutex: las tareas HttpIn y HttpOut comparten el uso de este mutex para garantizar el acceso exclusivo al puerto UART3 y a través de éste al módulo Wifly.

Las tareas LumTask y TempTask manejan puertos ADC para acceder a la lectura de sus sensores. Un uso simultáneo de este recurso provocaría lecturas erróneas. Por tanto, debemos garantizar el uso exclusivo del recurso en disputa por una sola de las tareas. Como en el caso anterior, usamos un MUTEX:

- AdcMutex: las tareas LumTask y TempTask comparten el uso de este mutex para garantizar el acceso exclusivo a los puertos ADC, y, a través de ellos, acceso a los sensores.

El intercambio de información entre tareas en FreeRTOS, en general, puede hacerse mediante el uso de variables globales y la adecuada sincronización de sus procesos de lectura y escritura que garanticen la integridad de la información procesada. No usaremos este método general, porque la API de FreeRTOS proprciona también un método específico de intercambio de mensajes entre tareas, y entre tareas e interrupciones, denominado QUEUES. Las colas contienen elementos, o items, de tamaño fijo. Pueden contener un máximo de elementos que se definen en el proceso de creación de la cola. Las llamadas a la API de FreeRTOS manejan y garantizan las cuestiones de exclusión mutua entre las tareas que hagan uso de las colas, por lo que simplifican enormemente el intercambio de mensajes entre tareas.

Hacemos uso de las siguientes colas, o QUEUES:

- HttpinQueue: la tarea HTTPInTask utiliza esta cola como salida, y la tarea ProcInTempTask la utiliza como entrada. La cola, de un elemento de longitud, almacena la lectura de la última información recibida desde la aplicación web con la información de los umbrales definidos por el usuario, leídos en la tarea HTTPIn.

- HttpOutQueue: la tarea HTTPOutTask utiliza esta cola como entrada, con la información que tiene que enviar a Arp@ Network Stats. Las tareas TempTask y LumTask utilizan esta cola como salida, almacenando en la misma el resultado de las lecturas de sus sensores.

- TempQueue: la tarea TempTask utiliza esta cola como salida, almacenando en ella el valor de la última lectura de temperatura procesada. Al necesitar sólo el último valor leído, la cola tiene sólo un elemento con dicha lectura. La tarea ProcIntTemp utiliza esta cola como entrada.

- LumQueue: la tarea LumTask utiliza esta cola como salida, almacenando en ella el valor de la última lectura de luminosidad procesada. Como en el caso anterior, al necesitar sólo el último valor leído, la cola tiene sólo un elemento con dicha lectura. La tarea ProcIntTemp utiliza esta cola como entrada.

Podemos ver el esquema de las tareas, las colas y los dos mutex en el siguiente esquema, donde las flechas indican el sentido del flujo de la información:

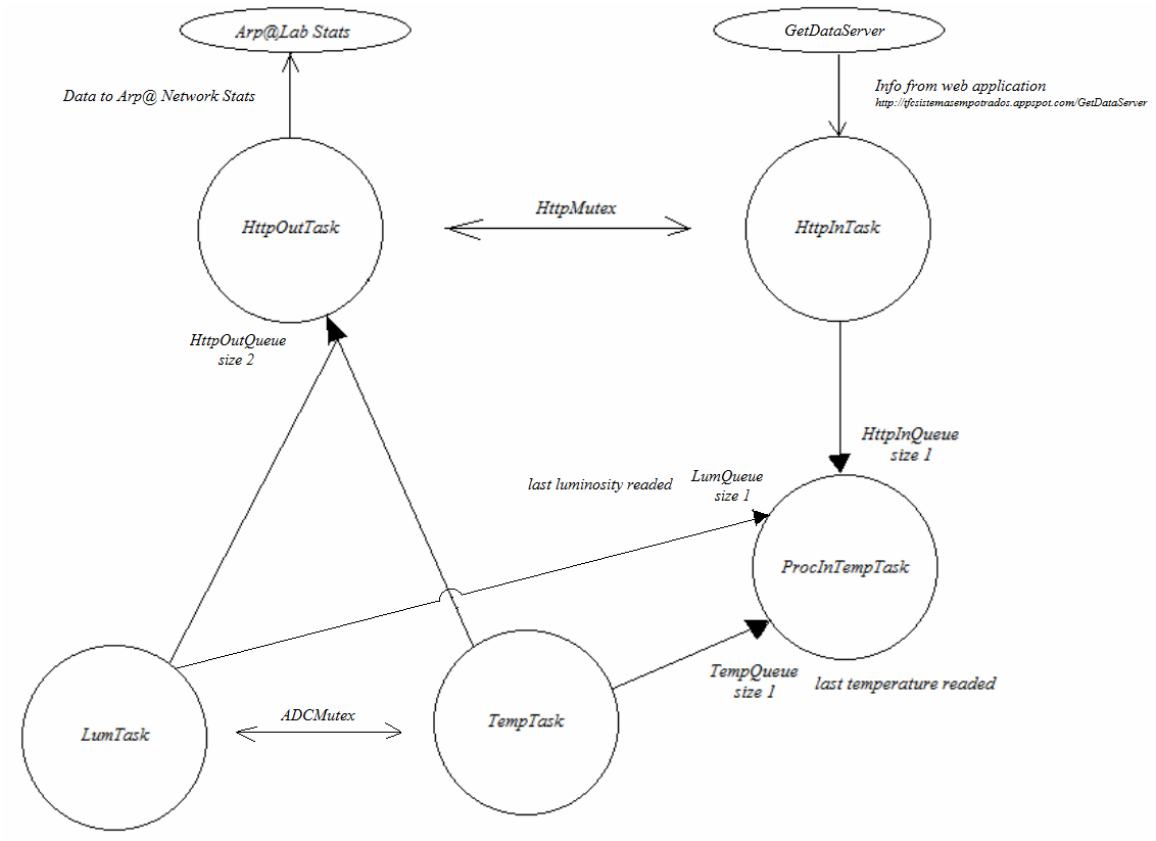

Figura 20. Esquema de la aplicación en el sistema empotrado

5 Implementación

La implementación lleva a la práctica lo visto en el diseño. Presentamos la parte de la aplicación web por un lado, y la parte del sistema empotrado por otro.

5.1 Implementación aplicación web

Detalle del código de la aplicación web.

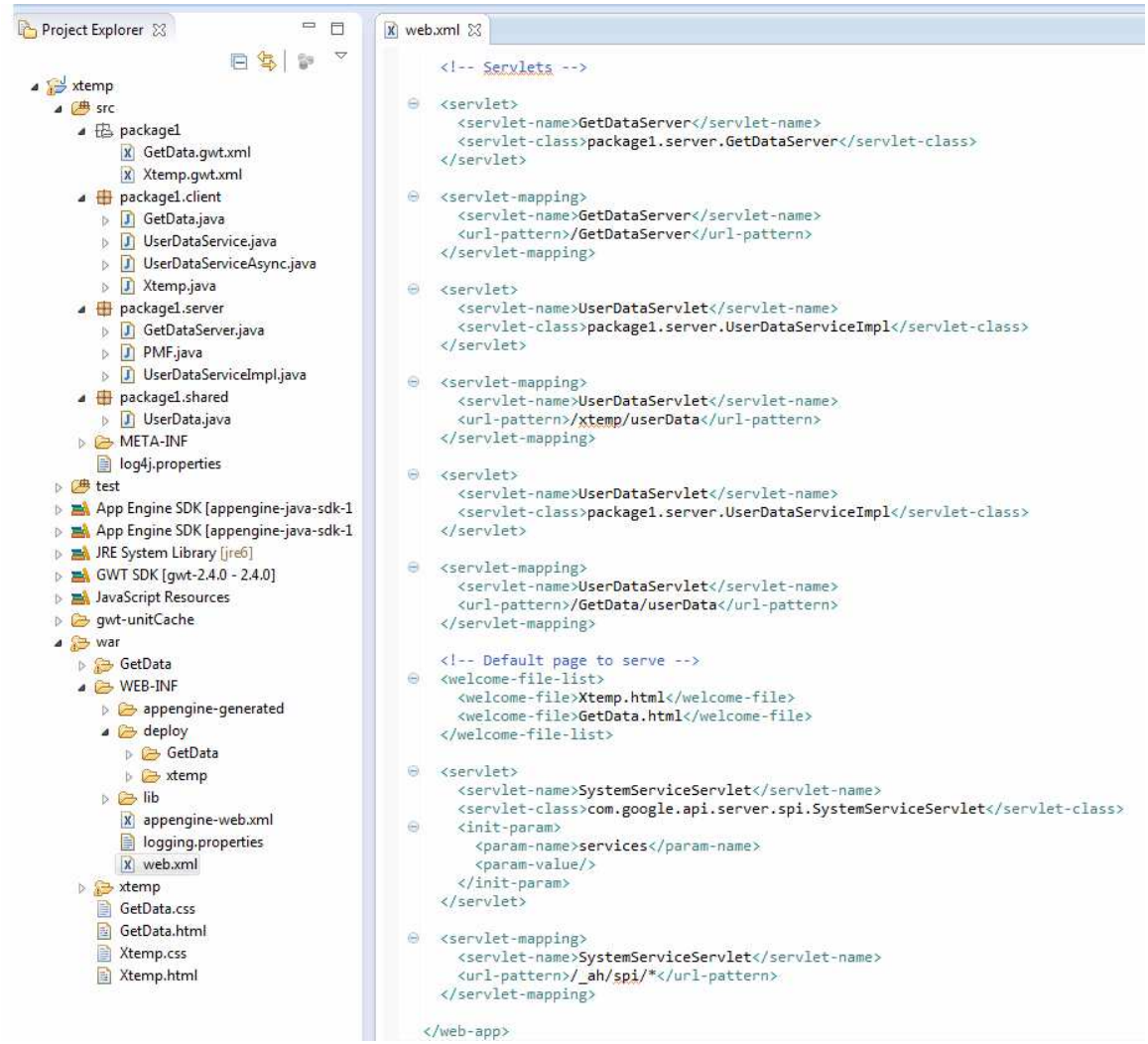

Figura 21. Detalle del código de la aplicación web

5.2 Implementación aplicación sistema empotrado

El código del sistema empleado, cumpliendo con lo especificado en el apartado de diseño. Podemos ver el punto donde es necesario informar los datos de la red wifi que queramos emplear.

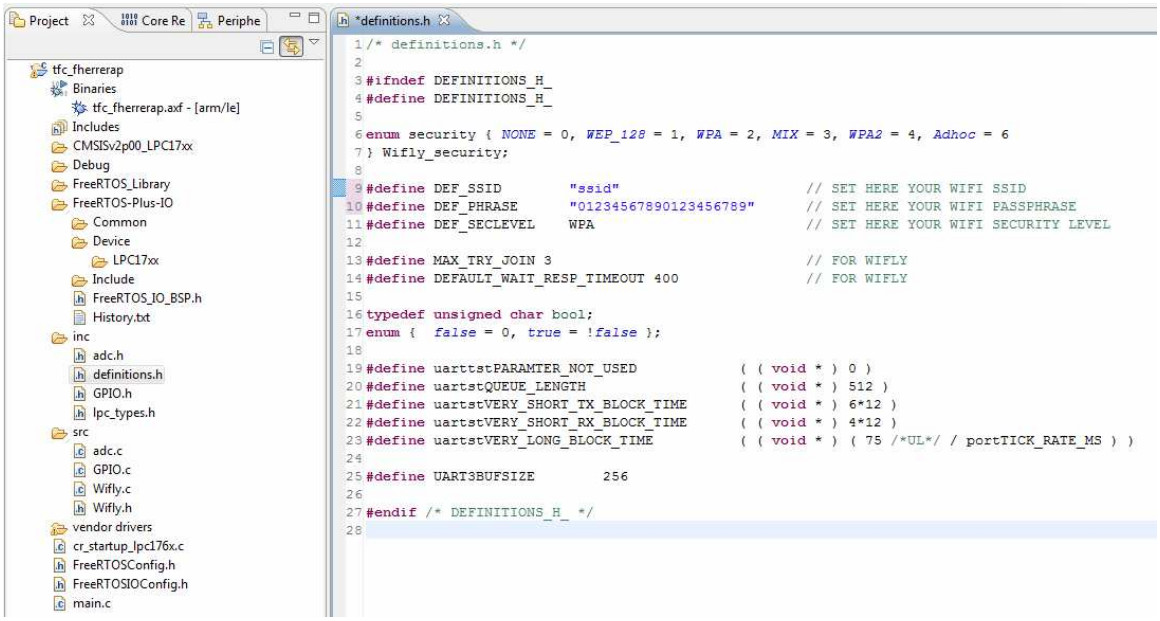

Figura 22. Detalle del código del sistema empotrado

En la estructura de archivos del proyecto, podemos ver:

- El archivo main.c, con el código del programa principal, definición y creación de tareas, colas y mutexes; el código de las tareas y lectura de sensores

- Carpetas inc y src, con nuestro código para gestión de periféricos GPIO y ADC, y gestión del módulo Wifly

- Aunque parte de la carpeta inc, destacamos el archivo definitions.h, que incluye la configuración de la red wifi a emplear, así como otros parámetros que afectan al funcionamiento global del sistema

- Archivos FreeRTOSConfig.h y FreeRTOSIOConfig.h, con la configuración del FreeRTOS

- Archivo cr_startup_lpc176x.c, proporcionado por el fabricante con el código de inicio necesario para la ejecución en el LPC1769

- Carpeta vendor_drivers, con los drivers de los periféricos del fabricante

- Carpeta FreeRTOS_library, con el código de FreeRTOS v7.3

- Carpeta FreeRTOS-Plus_IO, con la ampliación de FreeRTOS+IO

- Carpeta CMSISv2p00_LPC17xx, con código específico para la plataforma proporcionado por el fabricante

Merece un comentario específico la elección de la implementación de la pila. Como parte de la librería del sistema operativo FreeRTOS, en la parte "portable", contiene código específico para el hardware concreto de la plataforma en la que se ejecutará el código. Se proporcionan 4 implementaciones de pila distinta, con distintas funcionalidades y complejidades. Por ejemplo, delegando en el compilador, o permitiendo o no liberar la memoria ya reservada o juntar áreas separadas en un bloque único cuando sea necesario.

Hemos optado por utilizar la versión más sofisticada de las disponibles, la 4, que se ha demostrado absolutamente estable cuando el sistema se deja funcionando durante varios días seguidos.

5.3 Construcción de la solución. Circuitería.

Como puede verse en las fotografías, hemos empleado una placa de inserción corriente para dar soporte físico a los componentes. Una solución más limpia y sólida sería diseñar una placa impresa con los componentes soldados y encargar su fabricación, pero esta alternativa estaba fuera del ámbito del proyecto.

Las conexiones que requiere nuestro circuito son las siguientes:

- VOUT: +3.3V pin J6-28 a línea alimentación placa de inserción

- GND: pin J6-1 a lína GND placa inserción

- TXD3: UART3 pin J6-9 a RX Wifly pin 3

- RXD3: UART3 pin J6-10 a TX Wifly pin 2

- P0.2: GPIO output pin J6-21 a RESET Wifly pin 5 (led verde)
- AD0.4: ADC P1.30 pin J6-19 a patilla central sensor TMP36
- AD0.5: ADC P1.31 pin J6-20 a patilla central divisor de voltaje (fotoresistencia LDR + resistencia 4K7 ohmnios)
- P2.6: GPIO output pin J6-48 a sistema calefacción (simulado con led rojo)
- P2.7: GPIO output pin J6-49 a sistema iluminación (simulado con led amarillo)

Las conexiones no tienen requisitos especiales, salvo verificar que sean estables. Hay que ser muy cuidadoso en la alimentación del sensor TMP36, puesto que confundir los pines de la alimentación destruye el circuito en pocos segundos.

Indicar que los sensores podríamos ubicarlos separados de la placa principal si la instalación así lo aconsejara. Teóricamente, con cable eléctrico de dos hilos estándar podemos alcanzar alguna docena de metros sin problemas.

Vemos a continuación un par de fotografías en las que puede verse en detalle las conexiones antes enumeradas.

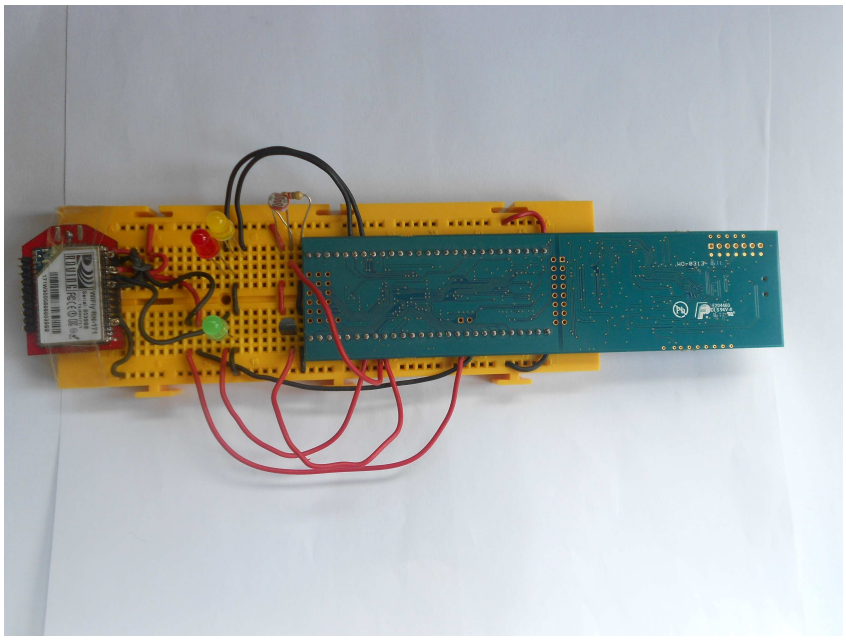

Figura 23. Detalle de las conexiones del sistema empotrado 1

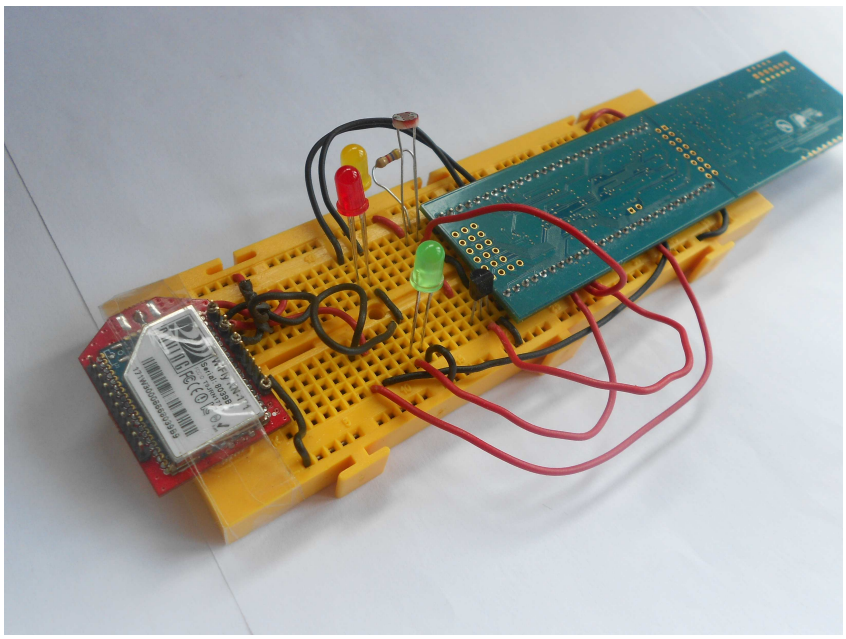

Figura 24. Conexiones del sistema empotrado 2

5.4 Instalación

Para la instalación, una vez realizadas las conexiones del apartado anterior, tenemos que cumplir con los siguientes pasos:

- Desplegar en Google App Engine la aplicación web
- Informar en el fichero "definitions.h" de los parámetros de la red wifi a utilizar. Compilar y descargar el programa ejecutable del sistema empotrado en la memoria flash del microcontrolador, el LPC1769.
- Alimentar el circuito.

6 Manual del usuario

Una aplicación bien diseñada debe tener un uso simple e intuitivo. Podemos ver que nuestra aplicación cumple con este requisito.

- Carga del software del sistema empotrado: Abrir el workspace con el código fuente de la aplicación en LPCXpresso y seleccionar la opción "Project/Buil Project". El progama se compilará y se ejecutará automáticamente el linker como parte del tool-chain.

  Conectar el microcontrolador a un puerto USB disponible y descargar el fichero ejecutable generado en la memoria flash del microcontrolador. Para ello, clicar en el icono "Program Flash", en el menú emergente y seleccionar "Use JTAG interface", para indicar al IDE que lo descargue mediante el JTAG, seleccionar también "Reset target on completion" y "Mass erase"; dejar el resto de opciones como no seleccionadas. Especificar la ruta y el nombre del fichero a cargar en "Select file" y clicar el botón "OK". En la nueva ventana emergente, seleccionar el JTAG con doble clic (sólo aparecerá uno). En unos segundos, el JTAG cargará el ejecutable en el microcontrolador, reseteará el target, y, automáticamente, el microcontrolador ejecutará nuestro programa.

  El programa así cargado se ejecutará cada vez que se proporcione alimentación al circuito.

- Configuración inicial: La configuración inicial debe ser realizada por personal técnico cualificado y sólo es necesaria la primera vez que usamos el dispositivo o si cambiamos los parámetros de la red wifi a utilizar. Debemos programar nuestro microcontrolador con los datos específicos de la red wifi a la que queramos conectar nuestro dispositivo. Necesitamos SSID, nivel de seguridad y clave, si procede.

  Opcionalmente, también podemos configurar la identificación de red, sensor y aplicación que presentará la aplicación web (Arp@ Network Stats), de la manera que sea más comprensible para el usuario.

  El procedimiento a seguir en este punto es el mismo que en el del apartado anterior, "carga del software del sistema

empotrado". Se tarda aproximadamente dos minutos en el total del proceso, incluyendo el tiempo de arrancar el IDE y elegir el workspace.

- Instalación del dispositivo físico: Para usar la aplicación, el usuario debe ubicar el circuito del sistema empotrado en la estancia que quiera gestionar. Como hemos mantenido la parte del JTAG de nuestro kit de sistemas empotrados, es posible alimentar el circuito a través del conector USB, conectándolo a un ordenador disponible o a un cargador con conector estándar mini-USB.

  Debemos elegir una ubicación alejada de las fuentes de calor o frío, como radiadores o puertas, y en un punto en el que no se vea afectado directamente por la iluminación a manejar.

  Los sensores, mediante un sencillo cableado de dos hilos, podrían ubicarse separados del microcontrolador, en estancias distintas o, incluso en interior - exterior. En este caso, recomendamos también que la instalación sea realizada por personal cualificado.

- Gestión y control del sistema: El usuario tiene a su disposición la pantalla de visualización de información y gestión accediendo a la web http://tfcsistemasempotrados.appspot.com. En esta única página están accesibles todas las opciones disponibles, y los puntos siguientes se refieren a ella.

- Control de temperatura de la estancia: El usuario puede consultar la temperatura ambiente, de manera gráfica, seleccionando en la sección titulada "Arp@ Network Stats" los valores de red, sensor y aplicación configurados en la instalación para la temperatura. Por defecto son "888888", "192.168.1.30" y "vTempTask".

  La temperatura se presenta en grados centígrados.

- Control del sistema de calefacción: En la parte superior de la página web de control, a la izquierda, tenemos disponible una lista de selección (listbox) con la etiqueta "Temperature thershold" donde podremos elegir el umbral de temperatura ante el que reaccionará el dispositivo.

  El sistema funciona como un termostato corriente. Si la temperatura medida es inferior a la definida por el usuario en

este punto, se encenderá la calefacción. Si la temperatura medida es superior a la especificada como umbral, la calefacción permanecerá apagada.

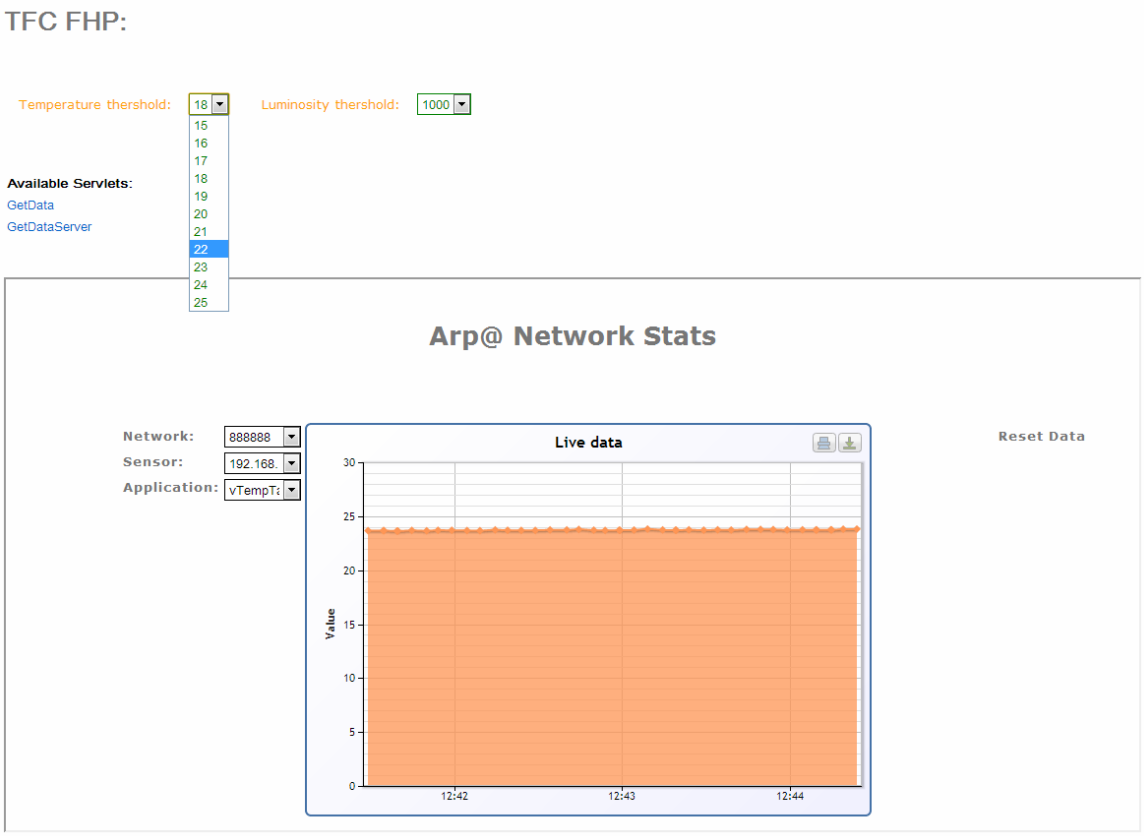

Figura 25. Consulta de temperatura y manejo de su umbral

- Control de luminosidad de la estancia: El usuario puede consultar la luminosidad ambiente, de manera gráfica, seleccionando en la sección titulada "Arp@ Network Stats" los valores de red, sensor y aplicación configurados en la instalación para la luminosidad. Por defecto son "888888", "192.168.1.30" y "vLumTask". La luminosidad se presenta aproximadamente en candelas.

  Atención: tenga en cuenta que la respuesta del sistema no es lineal en esta variable, por lo que en caso de requerir un valor específico deberá tomar las mediciones necesarias con el equipo adecuado para verificar la respuesta del sistema.

- Control del sistema de iluminación: En la parte superior de la página web de control, a la derecha, tenemos disponible una lista de selección (listbox) con la etiqueta "Luminosity thershold"

donde podremos elegir el umbral de iluminación ante el que reaccionará el dispositivo.

Si la intensidad de luz medida es inferior a la definida por el usuario en este punto, se encenderá la iluminación. Si la iluminación medida es superior a la especificada como umbral, la iluminación permanecerá apagada.

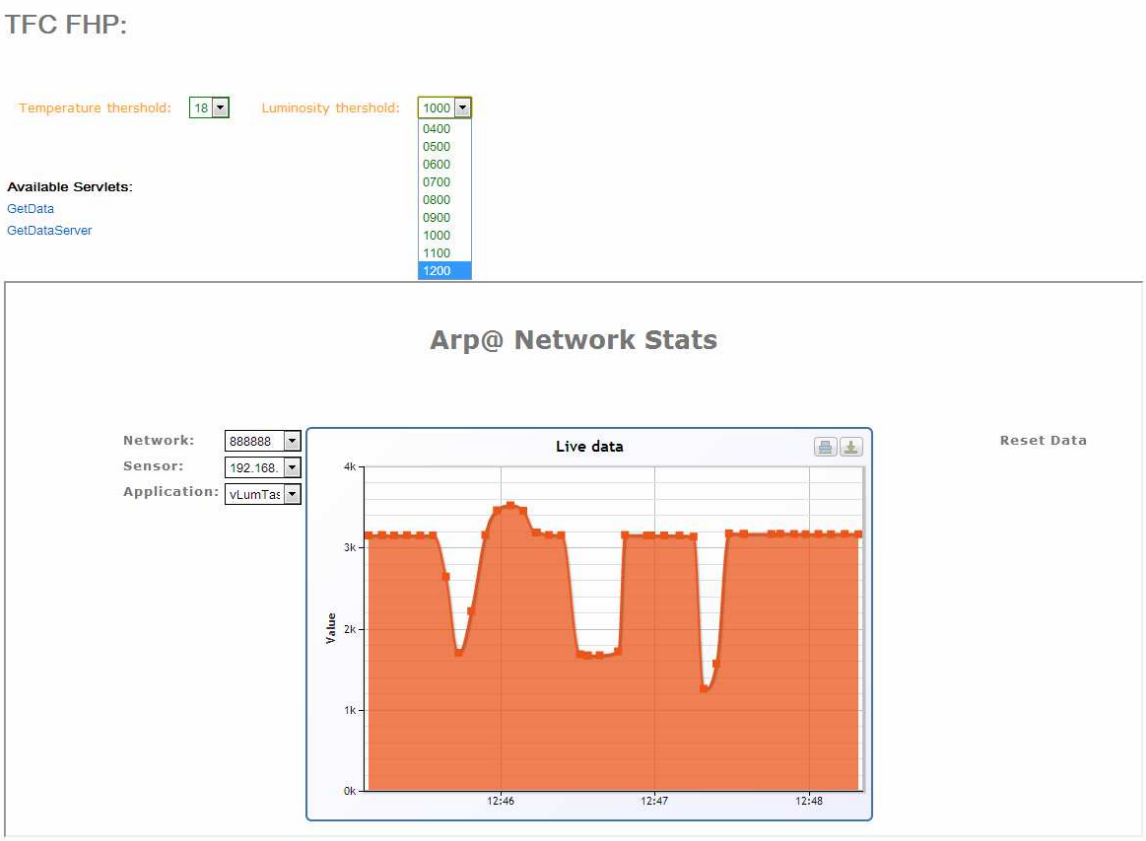

Figura 26. Consulta de luminosidad y manejo de su umbral

7 Evaluación de resultados

7.1 Retos y problemas

El principal reto ha sido lograr el tiempo suficiente de dedicación que me permitiera aprender todas las técnicas y conceptos que se requieren para el proyecto. Al escoger el área de sistemas empotrados para desarrollar el tfc, era plenamente consciente del esfuerzo que suponía, por lo que había dejado el TFC como única asignatura del semestre. Estoy convencido de que no habría sido capaz de compaginar este proyecto con ninguna otra asignatura de la carrera.

Los problemas que he tenido en el transcurso del proyecto tienen que ver con el punto anterior. Inicialmente hubo un error con el envío de materiales de la signatura, que se retrasó hasta entrado septiembre, dentro del plazo, pero casi dos meses más tarde de lo que podría haber sido matriculando en julio.

Los siguientes problemas han sido fallos de hardware. Primero, no fue posible alimentar el módulo Wifly desde el adaptador UART-USB. Después, el módulo Wifly quedó totalmente inutilizado y tuve que reemplazarlo por uno nuevo del mismo modelo. Parece razonable pensar que el problema estaba originalmente en elmódulo Wifly, pero, con certeza, no puedo valorar si el fallo fue de un módulo defectuoso o yo mismo provoqué la avería por alguna mala conexión o cualquier otro uso incorrecto. Estimo que la avería del módulo supuso unas tres semanas de retraso, que hubo que recuperar con muchas horas diarias de dedicación.

7.2 Aprendizaje

Desde mi punto de vista es el resultado más relevante del proceso:

- Sólo había trabajado en lenguaje C en la asignatura "ampliación de estructura y tecnología de computadores" de ITIS.
- No había trabajado nunca con Eclipse.
- No había trabajado nunca con Java.

Entiendo que el resultado del aprendizaje de una carrera como la Ingeniería Técnica de Telecomunicaciones no es el entrenamiento en el manejo de algunos lenguajes o herramientas. Lo fundamental, es la base teórica global, la capacidad y el conocimiento necesarios para enfrentarse con éxito a cualquier nuevo lenguaje o tecnología, que resulten necesarios para llevar adelante los proyectos a los que tengamos que enfrentarnos.

7.3 Puntos de mejora

Para llegar a convertirse en un sistema realmente funcional, el primer aspecto a mejorar es el físico. Sería relativamente rápido y barato encargar un circuito impreso con los componentes integrados. Con un encapsulado adecuado podría ser más resistente, incluso podría tolerar una instalación en exterior o en ambientes "sucios".

El segundo aspecto a mejorar sería la autonomía de alimentación. Con el diseño actual es muy sencillo acoplarle unas baterías de 3.3V, liberándonos de la necesidad del conector USB. El pin J6-28, permite un uso como V-OUT y como V-IN. En este caso podríamos prescindir del JTAG, reduciendo el tamaño del circuito prácticamente una tercera parte.

En cuanto a software, podemos plantear varias opciones de mejora manteniendo la estructura básica del sistema. En un primer nivel, plantería optimizar el número de veces que se envían o reciben datos desde la red. Es relativamente sencillo enviar los datos de los sensores en una sola operación y recibir los datos de los umbrales como parte de la respuesta. Habría que modificar las tramas de los datos a enviar y a recibir, tanto en la parte del sistema empotrado, como en la parte de la aplicación web, pero no resulta una modificación demasiado compleja.

Una vez implementada la mejora del punto anterior, deberíamos centrarnos en reducir el consumo de energía del conjunto del sistema. Para ello, podemos reducir la frecuencia de ejecución de las tareas del sistema empotrado y programar tanto el microcontrolador como el módulo Wifly para entrar en estado de hibernación el tiempo que estén inactivos. Un objetivo ambicioso, pero entiendo que alcanzable, a lograr en este aspecto sería alargar el funcionamiento autónomo del sistema con unas pilas estándar a unos 2 años.

La interface de usuario puede mejorarse, proporcionándole un mejor aspecto visual, adaptado específicamente para los dispositivos más frecuentes en el mercado.

Para aportar valor a la información capturada, podríamos almacenar información histórica. Disponiendo de la misma, es sencillo tratarla de manera adecuada y presentar información estadística, alarmas, etc.

En otro ámbito, habría que trabajar en la seguridad del sistema. En el caso de la aplicación web, incluyendo un sistema de seguridad en el acceso a la aplicación. En el caso de la información transmitida a través de la red, con la encriptación adecuada.

Para finalizar, y dependiendo del uso al que se dedique el sistema, deberíamos incorporar mecanismos de seguridad en el sistema empotrado, que permitan proteger los datos recogidos por los sensores en caso de fallo en la red wifi para enviarlos cuando volviera a ser posible.

8 Conclusiones

Sobre el proyecto y su desarrollo:

Un proyecto en el que prácticamente todo a lo que te enfrentas es nuevo conlleva mucho esfuerzo y paciencia. El aprendizaje de nuevas tecnologías o lenguajes implica siempre momentos en los que parece que no se avanza, a veces parece incluso que se retroceda.

La falta de precedentes y el uso de hardware, introducen un mayor riesgo en la estimación de los tiempos que se consideran razonablemente necesarios para cada tarea. Este riesgo adicional nos obliga a trabajar con mayor flexibilidad en las distintas fases del proyecto.

Para contrarrestar lo anterior contamos con nuestros estudios de Telecomunicaciones, que nos aportan la seguridad, la convicción personal, de que seremos capaces de llevar adelante el proyecto de manera eficaz y en el tiempo adecuado.

Entiendo que el resultado final del proyecto, el software y documentación entregados, cumplen con los requisitos definidos en el planteamiento del mismo. El software es fácil de manejar, robusto y fiable, y la documentación es clara y concisa, y cumple con los requisitos formales de un trabajo fin de carrera.

En lo personal, por las dificultades encontradas en estos meses, quedo satisfecho con el resultado. He disfrutado mucho del proceso, he aprendido enormemente, empezando a dominar algunas tecnologías que quería aprender desde hace tiempo y he terminado con éxito el proyecto.

Respecto a los sistemas empotrados:

Los sistemas empotrados están evolucionando hacia los sistemas inteligentes, con capacidades avanzadas de procesamiento y conectividad para acceso a redes, acople con otros dispositivos y con la nube. Se considera a los sistemas inteligentes como "la siguiente gran oportunidad" para los proveedores de tecnología, vendedores de sistemas y proveedores de servicios (IDC, 2011).

Monitorización y diagnóstico, gestión y control de energía, gateways de servicios personales… Se prevé un mercado por encima de los 2 billones de dólares en ingresos, y más de 11.000 millones de unidades vendidas para el 2015. Sensores de todo tipo medirán y actuarán sobre nuestro entorno, con conectividad total y aplicaciones y servicios de análisis de datos basados en la nube. Mejorará la interacción con el usuario y, con el middleware apropiado, mejorará la interconexión y los servicios recíprocos entre los distintos dispositivos. Todo ello nos lleva a que "Internet de las Cosas" (Internet of Things, IoT) esté, cada vez más, cerca de la realidad.

Para que todo esto sea posible, el desarrollo de soluciones viables comercialmente requiere del dominio de una ingente variedad de tecnologías. La complejidad de este ámbito de desarrollo radica en esa variedad de tecnologías a emplear y en la necesidad de hacerlas funcionar eficazmente de manera conjunta. En mi opinión, la masificación del despliegue de soluciones de este tipo obligará al establecimiento de estándares y a la aparición de soluciones de middleware que faciliten la interacción entre dispositivos y con los usuarios, ocultando la complejidad de cada sistema físico mediante la oportuna capa de abstracción.

Este proyecto ha supuesto una manera, muy humilde, de entrar en contacto y aprender sobre uno de los sectores con futuro más prometedor de la actualidad, que, por sí mismo, definirá un nuevo paradigma de interacción con el entorno.

Anexo I. Bibliografía

- LOS SANTOS, 2004. Alberto Los Santos Aransay, Aplicación de las redes de sensores en el entorno vehicular. Mayo 2009. www.albertolsa.com/wp-content/uploads/2010/04/rsi-aplicacion-de-las-redes-de-sensores-en-el-entorno-vehicular-alberto-los-santos.pdf

- LORENZATI, 2010. Ing. Marcelo Lorenzati, Desarrollo de drivers y aplicaciones para FreeRTOS. Marzo 2010. http://laboratorios.fi.uba.ar/lse/sase/2010/slides/SASE-2010_-FreeRTOS_Drivers_-Lorenzati.pdf

- PITCHER, 2012. Graham Pitcher, Extensions to FreeRTOS bring productivity gains to embedded design engineers. Mayo 2012. http://www.newelectronics.co.uk/electronics-technology/extensions-to-freertos-bring-productivity-gains-to-embedded-design-engineers/42128/

- FREERTOS, 2012. Richard Barry y Real Time Engineers Ltd. FreeRTOS – Market leading RTOS for embedded systems. 2004-2012. http://www.freertos.org/

- NXP, 2012. NXP Semiconductors. LPCXpresso. 2012. http://knowledgebase.nxp.com

- BRC-Electronics, 2012. Libraries & Examples. 2011. http://www.brc-electronics.nl/librarys

- MBED, 2012. Rapid Prototyping for Microcontrollers. 2012. www.mbed.org

- IDC, 2011. Mario Morales, Shane Rau y otros. Intelligent Systems: The Next Big Opportunity. 2011.

Anexo II. División del trabajo

El proyecto está condicionado por el esfuerzo que exige y permite la carga en créditos de la asignatura, y por la planificación temporal de la misma, que se concreta en tres pruebas de evaluación continua, PECS, y la memoria y la presentación virtual. Tal y como está predefinido en la asignatura, debemos emplear un kit de sistemas empotrados, que se nos hace llegar. Se sugiere, también, que la interface de usuario se realice mediante una modificación de la aplicación GWT Arp@Lab. Optamos por desarrollar una aplicación propia desde cero e integrar la funcionalidad de Arp@Lab en la página web de nuestra aplicación.

Con estas premisas, la división del trabajo a realizar queda como sigue:

Preparación del proyecto:

- Instalación del IDE.
- Ejecución de ejemplos.
- Familiarización con el software y el hardware.
- Configuración inicial del módulo Wifly.

Aprendizaje de majejo de periféricos, primeros programas en sistema empotrado:

- Programación de periféricos.
- Programación del driver para manejo del módulo Wifly mediante UART.

Aprendizaje de FreeRTOS:

- Programación de primeros programas en FreeRTOS, con creación de tareas y sincronización e intercambio de información entre ellas.
- Reescritura de programas con manejo de periféricos desde versiones iniciales a FreeRTOS.

Proyecto final:

- Definición de objetivos de propuesta final del proyecto.
- Desarrollo para sistema empotrado.
- Desarrollo app Google Web Tollkit.
- Redacción de documentación.

Actualización bibliográfica:

Puesta al día bibliográfica según necesidades.

Hitos y calendario

Hito 1: PEC1

Marca el fin de la preparación del proyecto. Una vez llegados a este punto, tendremos instalado el IDE, sabremos ejecutar ejemplos y estaremos familiarizados con el kit de sistemas empotrados. Tendremos el módulo Wifly configurado para conectarse a la red wifi.
FECHA: 18-10-2012

Hito 2: PEC2

Marca el fin del aprendizaje del manejo de periféricos. Tendremos programado un driver para el manejo del módulo Wifly mediante un puerto serie, controlando el mismo desde nuestros programas para configurarlo y lanzar peticiones mediante HTTP capturando la información contenida en las respuestas.
FECHA: 16-11-2012

Hito 3: PEC3

Al alcanzar este hito tendríamos control de FreeRTOS suficiente para programar aplicaciones sencillas, con creación y manejo de tareas, colas y mutexes. También seremos capaces de emplear dispositivos mediante el uso de los periféricos adecuados.
FECHA: 26-11-2012

Hito 4: Entrega código final

Entregamos la versión final de la aplicación del sistema empotrado y de la aplicación web. Las aplicaciones deben ser totalmente funcionales y cumplir con los requisitos determinados en la definición del proyecto.
FECHA: 24-12-2012

Hito 5: Entrega Memoria y Presentación virtual

Se presentan la memoria del TFC y la presentación virtual. Esperamos la evaluación.
FECHA: 21-1-2013

Podemos ver que los 3 primeros hitos abarcan el periodo de familiarización y aprendizaje iniciales, definididos por las fechas y contenido de las 3 pecs de la asignatura.

El cuarto y quinto hitos abarcan la definición, programación y pruebas del proyecto final, y, lo que no es menos importante, la cumplimentación de la documentación formal del proyecto, que se materializan en la presente memoria y en la presentación.

Anexo III. Estimación de tiempos y planificación

El proyecto comienza el 18-9-2012 con la apertura del aula de la asignatura, y termina con la presentación de la memoria el 21-1-2013 como fecha tope. La estimación de tiempos está basada en la previsible disponibilidad de dedicación al TFC, que se estima en unas 30 horas a la semana. Se ha planteado el fin del proyecto unos días antes de la fecha tope del plan docente, en concreto, la fecha de fin del mismo es el 3-1-2013.

La tarea "Actualización bibliográfica" se considera que tiene que abarcar la duración completa del proyecto, se le dedicará tiempo según surjan necesidades en las distintas fases.

En el transcurso real del proyecto, el grado de cumplimiento de las estimaciones de tiempos ha sido bastante bajo, en particular en las tareas determinadas por los tres primeros hitos.

Una vez solucionados los problemas con el hardware ya detallados en la sección correspondiente, en la fase final del proyecto se ha logrado recuperar el retraso acumulado. Con el necesario esfuerzo durante estos meses, se ha reconducido la situación, llegando al hito de la entrega final del código fuente y de la memoria y presentación en las fechas previstas, incluso con un pequeño margen de casi una semana.

III.1 Planificación

La estimación del esfuerzo necesario para realizar un proyecto, se determina utilizando modelos aplicables o, según la experiencia de quien planifica, en función de precedentes. La planificación del desarrollo de una aplicación para un sistema empotrado, que emplea tecnologías de funcionamiento desconocido para el autor, introduce un mayor riesgo en la estimación de los tiempos que se consideran razonablemente necesarios para cada tarea.

El uso de hardware en el proyecto, añade a lo anterior una mayor dificultad inherente, puesto que en el transcurso del proyecto puede resultar muy complicado determinar cuándo el sistema no funciona por un defecto de construcción o por un fallo del propio hardware. Los

fallos de hardware menos costosos en tiempo son los que están provocados por un fallo total de algún componente, mientras que los más costosos son los que se manifiestan de manera intermitente o aleatoria, puesto que se confunden fácilmente con otras causas ajenas al hardware.

La falta de precedentes y el uso de hardware, nos obligan a trabajar con mayor flexibilidad en las distintas fases del proyecto. En nuestro caso, hemos podido contar con flexibilidad en las fechas de los hitos de las tres pecs, cuestión fundamental sin la que el resultado final del proyecto no habría sido el mismo.

En la ejecución del proyecto, se ha llegado a los hitos iniciales previstos con bastantes días de retraso. Retraso provocado por los problemas de hardware ya comentados en el apartado específico.

En detalle:

- La PEC1 prevista para el 18/10/2012 se entregó el 31/10/2012.
- La PEC2 prevista para el 16/11/2012 se entregó el 2/12/2012.
- La PEC3 prevista para el 26/11/2012 se entregó en 14/12/2012.

Una vez solucionados los problemas del hardware del módulo Wifly, sustituyéndolo por uno nuevo del mismo modelo, el proyecto avanzó al ritmo adecuado.

Definida y aprobada la propuesta de proyecto, esta es la planificación prevista desde ese momento:

	Nombre	Duración	Inicio	Terminado	Pred...
1	**Entrega final TFC**	**20 days**	**10/12/12 8:00**	**3/01/13 17:00**	
2	Definición de objetivos	1 day	10/12/12 8:00	10/12/12 17:00	
3	**Desarrollo sistema empotrado**	**4,5 days**	**11/12/12 8:00**	**15/12/12 13:00**	**2**
4	Codificación pgm FreeRTOS	4,5 days	11/12/12 8:00	15/12/12 13:00	
5	Pruebas de programa	1,5 days	11/12/12 8:00	12/12/12 13:00	
6	**Desarrollo app Google Apps**	**4,5 days**	**15/12/12 13:00**	**20/12/12 17:00**	**3**
7	Codificación app	4,5 days	15/12/12 13:00	20/12/12 17:00	
8	Pruebas app	1,5 days	15/12/12 13:00	17/12/12 17:00	
9	Pruebas de sistema	2 days	21/12/12 8:00	24/12/12 17:00	6
10	Documentación	10 days	21/12/12 8:00	3/01/13 17:00	

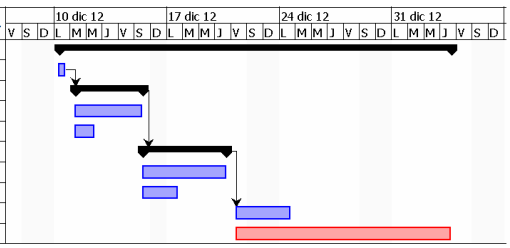

Figura 27. Diagrama de Gantt

La previsión era tener código y documentación entregados el día 3 de enero, para, a partir de ahí, incorporar las correcciones indicadas por el consultor de la asignatura.

Los principales riesgos, a priori, eran:

- Uso de hardware de funcionamiento desconocido para el desarrollador.

- Falta de experiencia en el desarrollo de aplicaciones web con tecnología GWT, como la empleada para la gestión de los umbrales por parte del usuario.

Finalmente, el código completo se entregó el día 27/12/2012, y se aprobó como bueno el 30/12/2012.

La memoria, pendiente de correcciones se entregó el 7-1-2013, con un margen razonable para incluir los cambios que nos indicara el profesor. La memoria final se entregó el 15-1-2013, siendo la fecha tope de entrega el día 21-1-2013.

La presentación se entregó el día 15-1-2013, siendo también la fecha tope de entrega de la misma el 21-1-2013.

Como podemos ver, por las causas ya indicadas, se acumuló un retraso de hasta 18 días con las fechas previstas. Con esfuerzo y mucho trabajo, se ha logrado reconducir la situación hasta llegar con un pequeño margen a las fechas de finalización del proyecto.