CPDSurveyor

Sistema portable y controlable remotamente vía web para la monitorización de alarmas externas en un CPD o centro de datos asilado sobre LPC1769 y FreeRTOS.

Autor: Jaime Aivar González
Enginyeria Tècnica de Telecomunicació.

Consultor: Jordi Bécares Ferrés

15 de Enero de 2013

A mi pareja Mabel,
Gracias por tu paciencia, compresión y
por nuestro "proyecto" Lucas

Agradecimientos

Este proyecto no se podría haber llevado a cabo sin todos y cada uno de mis compañeros, consultores, tutores y toda la comunidad UOC en general. A todos ellos 1024 gracias.

Me gustaría agradecer al consultor de este proyecto Jordi Bécares Ferrés, sus buenos consejos, su experiencia para llevar a cabo proyectos de esta envergadura y sobre todo, por prepararnos para futuros proyectos en nuestras carreras profesionales.

Finalmente y no menos importante, me gustaría agradecer a toda mi familia, mis padres y en especial a mi pareja Mabel, la paciencia, apoyo y energía positiva que me han transmitido durante todos estos años, ayudándome y apoyándome en los momentos más difíciles.

Resumen

El siguiente proyecto forma parte del área TFC ***Sistemas Embebidos*** de la Universidad Oberta de Catalunya. La elección de dicha área, fue principalmente motivada, por el poder poner en práctica lo asimilado en la mayoría de las asignaturas que más interés despertaron en mi aprendizaje a lo largo de estos últimos años. Asignaturas como por ejemplo: “Fundamentos de computadores”, “Fundamentos tecnológicos II”, “Programación orientada a objetos”, etc…

En este proyecto, desarrollaremos un **sistema encastado** que una vez conectado y asociado a un punto de acceso **Wi-fi,** con conectividad a Internet, será posible controlarlo desde cualquier otro lugar y accionarle movimiento gracias a una plataforma con motores y ruedas.

Para ello, se ha creado una **GUI** (aplicación web), donde además de controlar el prototipo, se podrán visualizar datos reportados por el mismo, pudiendo obtener información física del lugar donde esté situado el prototipo (temperatura, índice de humedad relativa, etc..).

Este prototipo además será capaz de buscar autónomamente una zona con mejor cobertura, de tal forma se evitará la pérdida de control sobre si mismo.

Para llevar a cabo la implementación, se utilizará una ***mota LPC1769*** con un procesador **Cortex-M3** de ARM y como sistema operativo se usará **FreeRTOS** Real Time Kernel.

Además de todo lo anterior, se ha conectado y programado un modulo de comunicaciones inalámbricas **Wifly**, además de otro módulo independiente para el control de dos motores del fabricante “Sparkfun Electronics”.

Tendremos pues, un sistema encastado el cual podremos controlar remotamente (**telecontrol**) y del cual podremos obtener valores del entorno o estado (**supervisión**), dos acciones y características muy demandadas en la actualidad del mundo **IT**.

Índice de Contenidos

Índice de Tablas

Índice de Figuras

1 Introducción

4.1 Justificación

Cada vez son más los centros de datos que alquilan espacios dedicados (*jaulas*) a sus clientes para el alojamiento de sus racks para servidores y/o equipos de comunicaciones.

También existen muchas empresas de telecomunicaciones que tienen sus equipos en diversos emplazamientos, ya sea por necesidades del mismo despliegue de su red, por redundancia de datos o simplemente por disponer de una protección de rutas.

Como norma general, estos clientes suelen tener su base u oficinas de operaciones en otras provincias e incluso en otros países, ahorrando así en su totalidad, en costes de personal y recursos.

Este proyecto el cual llamaremos "**CPDSurveyor**" surge de la necesidad de estos clientes para controlar mecanismos y/o monitorizar alarmas de entorno en sus instalaciones externas, concretamente alarmas medioambientales como temperatura y humedad, alarmas de intrusión, incendio, etc.

Hoy en día este tipo de alarmas externas son tan importantes o más que las propias alarmas de sus equipos y/o sistemas. Por ejemplo, el control de la temperatura de un habitáculo o rack es muy importante para evitar el sobrecalentamiento de sus equipos y su consecuente avería, que normalmente éstas suelen ser muy costosas y puede demorarse mucho en su resolución.

Este proyecto será la base de partida para ofrecer un **servicio de telecontrol y monitorización** a un futuro cliente, el cual será adaptado previo estudio de necesidades con el cliente, pudiéndose añadir multitud de sensores o funciones dependiendo de las necesidades requeridas de dicho estudio.

1.2 Descripción

Como se ha introducido en el resumen y en el apartado anterior, el proyecto consiste en crear un sistema encastado con la mota LPC1769 basada en el microcontrolador Cortex-M3, el cual controlará como mínimo la siguiente serie de funciones programadas o tareas.

- *Control de motor / motores.*
- *Sistema automático de búsqueda de punto con mejor cobertura.*
- *Recepción y envío de datos desde Internet.*

Las tareas serán programadas bajo el sistema operativo de tiempo real FreeRTOS (sistema operativo basado en Lenguaje C) siendo necesaria con anterioridad la programación de nuevas librerías y drivers para poder gestionar los nuevos componentes hardware, así como la utilización de librerías ya existentes para la gestión de los puertos de E/S de nuestra mota.

Tendremos pues, un **sistema autónomo** que se conectará en una **red sin hilos**, será capaz de recibir ordenes y ejecutarlas (mover motores), estará dotado de un **sistema de protección** para impedir salirse de un rango de cobertura bajo y además será capaz de reportar estados al **sistema de supervisión y control**.

1.3 Objetivos

A continuación, se listarán los principales objetivos fijados para la ejecución del proyecto y una pequeña descripción de cada uno de ellos:

a) Conectar el sistema a redes Wireless 802.11b/g

Deberemos de conectar nuestra mota a una red inalámbrica, con lo que se deberá instalar e integrar un módulo de comunicaciones capaz de interactuar con la mota, con el fin de poderse asociar a una red con DHCP y poder obtener valores tales como la IP asignada, RSSI, etc…

b) Dotación al sistema de movimiento vía motor.

Una característica importante es el telecontrol o control remoto, con lo que deberemos de ser capaces de poder mover un motor hacía una dirección u otra, primero desde la mota, para después poderlo hacer desde la GUI diseñada.

c) Sistema bidireccional de comunicaciones.

Ha de existir un sistema bidireccional de comunicaciones entre la mota y el GUI del usuario, ya que en todo momento se ha de mantener informado al usuario final del estado de una orden de control o de la obtención de valor de un sensor.

d) Dotación al sistema de un servicio de cobertura.

Debemos de tener presente que la mota estará ubicada físicamente en un lugar remoto, por lo que ha de existir un sistema de protección para evitar la pérdida de control sobre la misma, por

lo que será muy importante la dotación de un sistema automático de búsqueda de un punto con cobertura mejorada.

e) Programación de una GUI para el control y supervisión.

Debemos de tener presente que un sistema encastado no es un PC y no está preparada para la programación por un usuario final o cliente, por lo que se ha de desarrollar una GUI apropiada y adaptarla según las necesidades del usuario final.

f) Dotación al sistema de energía autónoma.

La energía es una parte importante y se ha de dotar al sistema de una fuente autónoma y recargable que pueda alimentar a todo el sistema en global (mota, motores, etc.).

1.4 Enfoque y método seguido

Para conseguir llevar a cabo este proyecto ha sido necesario una **fase de formación** previa, en la cual se han trabajado y estudiado progresivamente los siguientes puntos:

1. Instalación del IDE de la mota y familiarización básica con FreeRTOS y con el dispositivo de comunicaciones Wifly.
2. Desarrollo de un driver para el puerto UART y conexionado del dispositivo Wifly con la mota.
3. Desarrollo de un sistema productor / consumidor sobre la aplicación Arp@Lab.

Una vez superada y asimilada la fase anterior con un **enfoque didáctico**, se ha procedido a realizar una **fase de investigación y desarrollo** con un **enfoque más productivo**.

En esta última fase, se ha trabajado igual que en la parte didáctica, es decir, primero se ha procedido a *aislar y estudiar cada componente por separado*, comprobando los comandos y sus respuestas, voltaje de salida de los pines, etc…

Una vez comprobado el correcto funcionamiento y habernos familiarizados con el hardware y su sistema de comandos, se ha *procedido al desarrollo de una librería / driver* para poder ser utilizado en el programa principal.

Este sistema tiene su origen en la filosofía **Open Source**, por lo que *se ha aprovechado otros códigos creados* de otros programadores así como los estudiados en la fase de formación para desarrollar la aplicación principal del sistema encastado.

En todo momento se ha trabajado realizando pequeños códigos de prueba para medir y testear el comportamiento, por lo que ha sido muy importante trabajar ordenadamente, aplicando una nomenclatura apropiada en las funciones y variables, separando cada librería / driver por módulo conectado o aplicación, además de comentar a nivel de código cada función, indicando los parámetros de entrada así como los parámetros de salida.

1.5 Planificación

La planificación del proyecto se ha estructurado en dos fases o etapas principales:

Fase 1 o Etapa base:

En esta fase las tareas a llevar a cabo son las necesarias y básicas para lograr alcanzar nuestros objetivos propuestos.

Fase 2 o Etapa avanzada:

Dependiendo del tiempo en ejecutar las tareas de la Fase 1, se dispone de esta segunda fase, donde se intentará ampliar o profundizar con diferentes mejoras los objetivos principales del proyecto.

En la siguiente figura representaremos el cronograma correspondiente a la **planificación inicial**, en el cual se puede observar en color verde, las fases anteriormente comentadas y en color azul, las tareas propuestas para conseguir alcanzar nuestros objetivos.

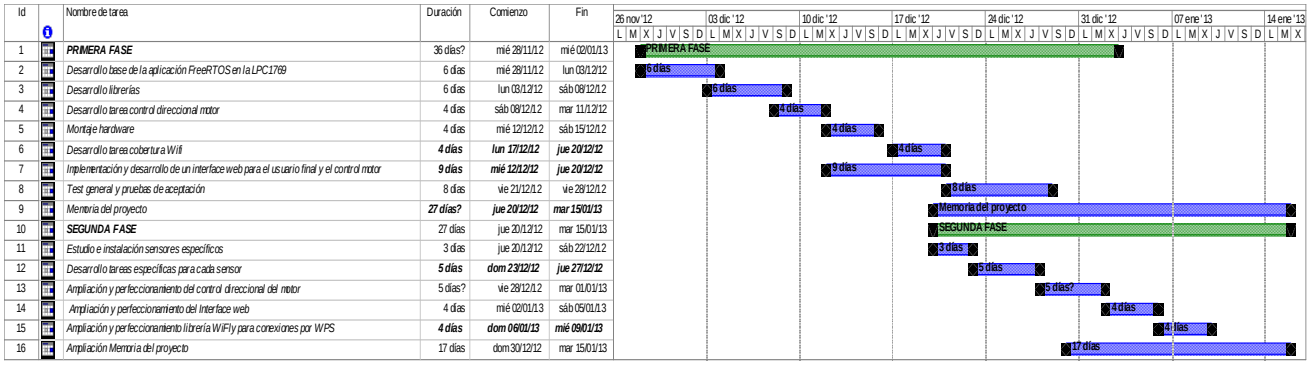
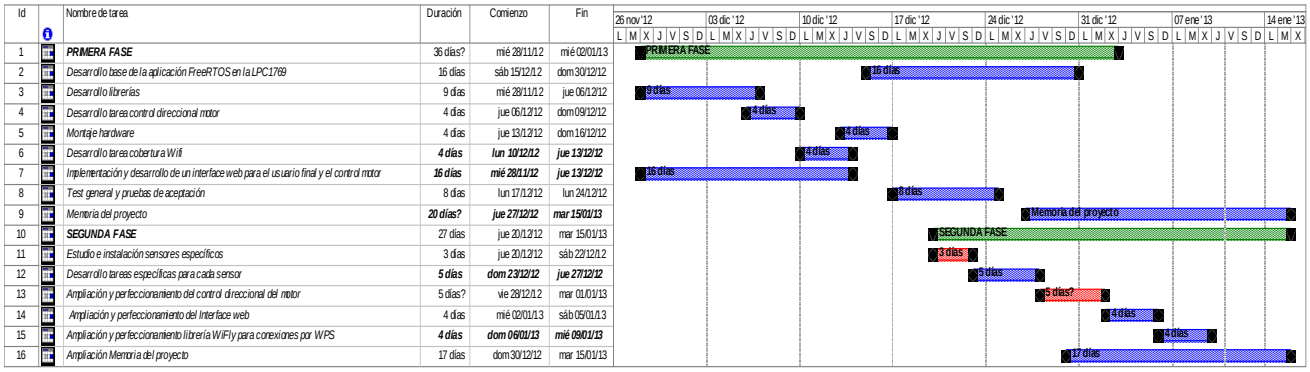


Figura 1: Diagrama de Gantt de la planificación inicial

Como en muchos proyectos, tras el comienzo con los trabajos de la planificación inicial, suelen surgir imprevistos y atrasos en las tareas, provenientes de aspectos no tomados en consideración en la fase de planificación o simplemente por consideraciones y/o recomendaciones por parte del cliente.

Es por ello que, representaremos en la figura 2, el diagrama de Gantt con la **planificación final** y real de nuestro proyecto, donde se ven plasmados los tiempos reales en llevar a cabo el sistema total:



Id	Nombre de tarea	Duración	Comienzo	Fin
1	***PRIMERA FASE***	36 días?	mié 28/11/12	mié 02/01/13
2	*Desarrollo base de la aplicación FreeRTOS en la LPC1769*	16 días	sáb 15/12/12	dom 30/12/12
3	*Desarrollo librerías*	9 días	mié 28/11/12	jue 06/12/12
4	*Desarrollo tarea control direccional motor*	4 días	jue 06/12/12	dom 09/12/12
5	*Montaje hardware*	4 días	jue 13/12/12	dom 16/12/12
6	*Desarrollo tarea cobertura Wifi*	***4 días***	***lun 10/12/12***	***jue 13/12/12***
7	*Implementación y desarrollo de un interface web para el usuario final y el control motor*	***16 días***	***mié 28/11/12***	***jue 13/12/12***
8	*Test general y pruebas de aceptación*	8 días	lun 17/12/12	lun 24/12/12
9	*Memoria del proyecto*	***20 días?***	***jue 27/12/12***	***mar 15/01/13***
10	***SEGUNDA FASE***	27 días	jue 20/12/12	mar 15/01/13
11	*Estudio e instalación sensores específicos*	3 días	jue 20/12/12	sáb 22/12/12
12	*Desarrollo tareas específicas para cada sensor*	***5 días***	***dom 23/12/12***	***jue 27/12/12***
13	*Ampliación y perfeccionamiento del control direccional del motor*	5 días?	vie 28/12/12	mar 01/01/13
14	*Ampliación y perfeccionamiento del Interface web*	4 días	mié 02/01/13	sáb 05/01/13
15	*Ampliación y perfeccionamiento librería WiFly para conexiones por WPS*	***4 días***	***dom 06/01/13***	***mié 09/01/13***
16	*Ampliación Memoria del proyecto*	17 días	dom 30/12/12	mar 15/01/13

Figura 2: Diagrama de Gantt de la planificación final

Observamos en la imagen anterior que el orden de ejecución así como el tiempo de la mayoría de las tareas han sufrido cambios por diferentes contratiempos que se citarán en el capítulo 7. Además, en la Fase 2, tan solo se ha podido realizar parcialmente las tareas de color rojo.

1.6 Recursos empleados

A continuación se detallaran los recursos empleados en la realización de este proyecto, diferenciando entre los recursos **Hardware** y **Software**:

Hardware:

- Tarjeta de desarrollo LPCXpresso, concretamente en este proyecto se ha utilizado la mota LPC1769

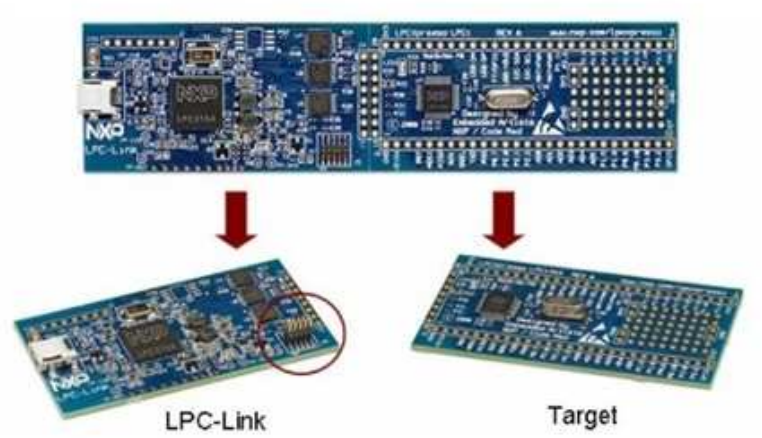


Figura 3: Detalle del el Kit educativo “FreeRTOS NXP LPCXpresso LPC1769

- Módulo de comunicaciones inalámbricas WiFly RN-XV 802.11b/g

Figura 4: Detalle del módulo WiFly RN-XV 802.11b/g

- Serial Controlled Motor Driver ROB-09571 RoHS de “Sparkfun Electronics”

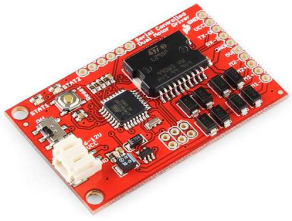


Figura 5: Detalle del Serial Controlled Motor Driver ROB-09571

- 2 Motores de 4V y una plataforma con ruedas delanteras y traseras.

Figura 6: Detalle de un motor eléctrico de 4V

- CP2102 convertidor UART to USB.

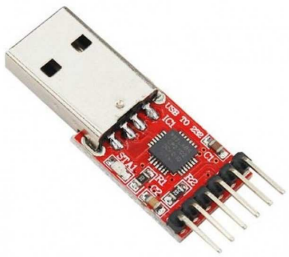


Figura 7: Detalle del CP2102

- Sensores (opcionales):

Figura 8: Detalle del sensor de temperatura digital TMP102

- Material y herramientas electrónica varias:
  - ✓ Soltador y estaño
  - ✓ Cables varios
  - ✓ 2 resistencias de 3,3K (o una resistencia de 6,6K)
  - ✓ 1 resistencia de 10K

- ✓ 2 leds
- ✓ Placa perforada pistas paso

- Batería de Litio con dos salidas USB

Figura 9: Detalle la batería de Litio de 5V

- Servidor o espacio web para alojar la aplicación web (opcional).

- Base de un coche por control remoto de juguete.

- PC Con sistema operativo Windows 7 Professional 32-bits con 2 Gb de Ram (1 Gb mínimo). También es posible con un PC Linux o Mac. Es necesario para programar con el IDE de LPCXpresso así como trabajar con el resto de software detallado en el siguiente apartado.

Software:

- Entorno de programación basado en eclipse y desarrollado por "Code Red Technologies" LPCXpresso IDE para programar en lenguaje C.

- Cliente para puertos serie como por ejemplo Putty, Cutecom o Teraterm pro

- Editor php como por ejemplo codeanywhere que es online y por lo tanto permite portabilidad para programar el interface desde cualquier PC.

1.7 Productos obtenidos

Como veremos en los próximos capítulos, obtendremos principalmente **dos productos**:

Por un lado, tendremos un **sistema encastado** en forma de prototipo o plataforma móvil con la mota LPC1769 y los módulos hardwares necesarios instalados, ejecutando en su totalidad una serie de tareas programadas para cubrir los objetivos fijados en el apartado 1.3.

Por otro lado, tendremos una **interface de usuario** (GUI) para poder controlar nuestro sistema encastado en remoto y también poder supervisar unos valores y parámetros determinados.

1.8 Descripción capítulos

En el siguiente capítulo haremos una pequeña introducción al mundo de los sistemas encastados, la actualidad y el presente de los mismos, así como pequeño estudio de mercado sobre nuestro sistema. Posteriormente en el capítulo 3 describiremos a grandes rasgos el funcionamiento y diseño global de nuestro sistema.

Entrados en el capítulo 4, nos adentraremos de una forma más técnica y detallada en el funcionamiento y diseño del proyecto. Ya en el capítulo 5, expondremos un estudio de la viabilidad técnica del sistema, para posteriormente en el capítulo 6, exponer una valoración económica orientativa del coste del sistema.

Por último, en el capítulo 7 expondremos las conclusiones extraídas de nuestro trabajo, exponiendo propuestas de mejoras y realizando una autoevaluación de los objetivos propuestos en el capitulo 1.

2 Antecedentes

2.1 Estado del arte

Un sistema encastado, también comúnmente llamado sistema empotrado o embebido, es un sistema de computación diseñado para realizar un conjunto de funciones o tareas específicas en tiempo real. Estas tareas pueden ser de control, procesamiento y/o monitorización.

El componente central es la **CPU**, la cual dependiendo de su arquitectura, puede incluir una **memoria** interna o externa.

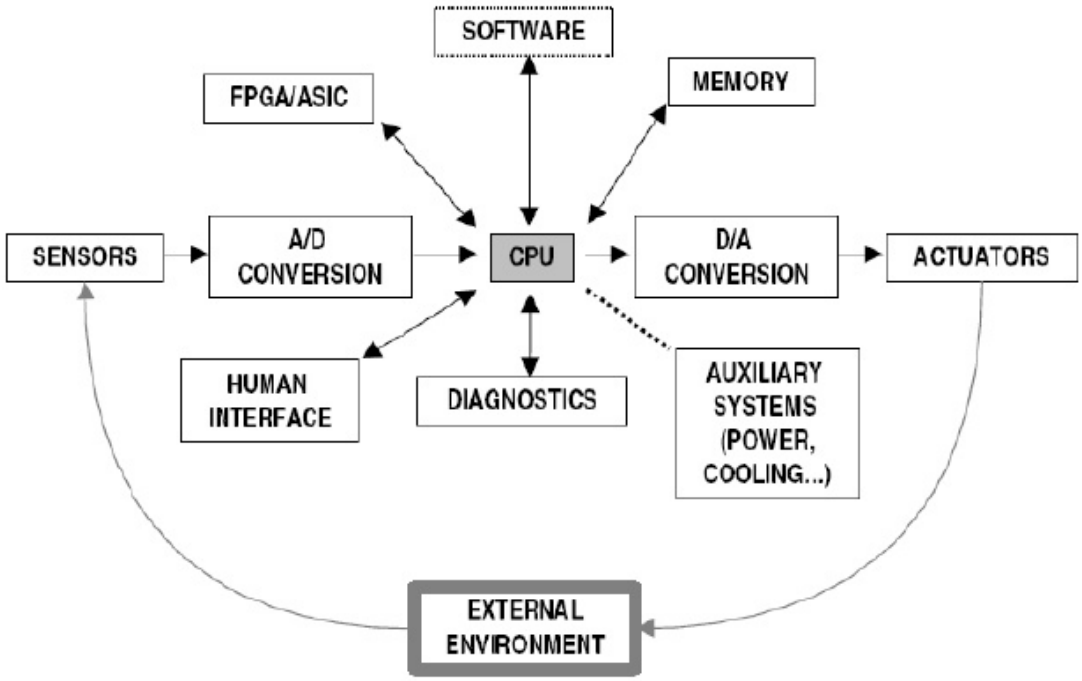


Figura 10: Diagrama conceptual de un sistema encastado

Tal y como podemos observar en la figura anterior, existen dos tipos de comunicaciones. El primer tipo es a **nivel interno** y las comunicaciones se realizan a nivel de bus de sistema. El segundo tipo es a **nivel externo**, pudiéndose comunicar mediante interfaces estándar de cable o inalámbricas.

En la actualidad un sistema encastado suele incorporar puertos y/o soportar protocolos de comunicaciones externas del tipo RS-232, RS-485, SPI, CAN, USB, IP, Wi-FI, GSM, GPRS, etc…

2.2 Estudio de mercado

Un poco de historia

El primer sistema embebido reconocido de la historia fue el sistema de guía del Apolo, desarrollado por el laboratorio de desarrollo del MIT para las misiones Apolo hacia la luna. Este sistema de cómputo fue el primero en utilizar circuitos integrados y utilizaba una memoria RAM magnética, con un tamaño de palabra de 16 bits. El software fue escrito en el lenguaje

ensamblador propio y constituía un sistema operativo básico, pero capaz de soportar hasta ocho tareas simultáneas.

La actualidad

En la actualidad existen multitud de sistemas embebidos, y cada vez más comunidades de usuarios aficionados crean sus propios sistemas, todo ello gracias a los fabricantes de microcontroladores y la aportación de documentación que aportan de cada uno de ellos. En la siguiente tabla se puede observar los **fabricantes** y las familias de **microcontroladores** más comunes:

Empresa	**8 bits**	**16 bits**	**32 bits**
Atmel	AVR (mega y tiny), 89Sxxxx familia similar 8051		SAM7 (ARM7TDMI), SAM3 (ARM Cortex-M3), SAM9 (ARM926)
Freescale (antes Motorola)	68HC05, 68HC08, 68HC11, HCS08	68HC12, 68HCS12, 68HCSX12, 68HC16	683xx, PowerPC, ColdFire
Holtek	HT8		
Intel	MCS-48 (familia 8048) MCS51 (familia 8051) 8xC251	MCS96, MXS296	x
National Semiconductor	COP8	x	x
Microchip	Familia 10f2xx Familia 12Cxx Familia 12Fxx, 16Cxx y 16Fxx 18Cxx y 18Fxx	PIC24F, PIC24H y dsPIC30FXX,dsPIC33F con motor dsp integrado	PIC32
NXP Semiconductors (antes Philips)	80C51	XA	Cortex-M3, Cortex-M0, ARM7, ARM9
Renesas (antes Hitachi, Mitsubishi y NEC)	78K, H8	H8S, 78K0R, R8C, R32C/M32C/M16C	RX, V850, SuperH, SH-Mobile, H8SX

STMicroelectronics	ST 62, ST 7		
Texas Instruments	TMS370	MSP430	C2000, Cortex-M3 (ARM), TMS570 (ARM)
Zilog	Z8, Z86E02		

Tabla. 1: Fabricantes y microcontroladores sistemas encastados

Los **sistemas operativos** pueden ser los mismos que los sistemas operativos para PC, pero eso sí, optimizados para realizar las funciones básicas y así conseguir un correcto aprovechamiento de los recursos.

Hoy en día, Microsoft tiene un solo sistema operativo (Windows CE) para sistemas embebidos, a diferencia del sistema operativo Linux, del cual existen multitud de "distros" especialmente orientadas a estos sistemas, como por ejemplo nuestro sistema operativo FreeRTOS, o el conocido Android de Google.

Estos sistemas están **evolucionando de forma exponencial** y además ya no solo van dirigidos al sector empresarial o industrial (fabricas de montaje y producción, sistemas radar de aviones, cajeros automáticos, equipos de medicina, etc…) , si no al resto de usuarios entrando en nuestros hogares en forma de dispositivos tipo smartphones o en sistemas de seguridad, electrodomésticos, etc.

En la actualidad se ha encontrado un producto muy similar al nuestro:

Figura 11: Detalle del producto i-Racer

Se trata del producto "i-Racer" . Aunque no dispone de una CPU tan potente como la que utilizaremos en nuestro proyecto, es un sistema encastado que puede ser controlado desde cualquier dispositivo móvil que disponga de sistema operativo Android a través de una red Wif-fi.

Además disponer de comunicación bluetooth , está provisto de un microcontrolador ATMega48V siendo posible poder subir otro firmware trabajado.

3 Descripción funcional

Con el objetivo de tener una visión global, comenzaremos exponiendo la funcionalidad del sistema al completo, de una forma gradual y progresiva.

3.1 Sistema total

Tal y como podemos observar en la figura 12 de la siguiente página, la **movilidad** ha sido un aspecto a considerar en el proyecto. Se pueden diferenciar los **dos productos** citados en la sección 1.7 y como ambos están **localizados en dos ubicaciones físicas** distintas y/o distantes.

Por un lado tendremos nuestro prototipo en la ubicación física o zona donde ejerceremos un control y obtendremos una supervisión. De la misma, en otra ubicación distinta, tendremos al usuario final.

Este usuario dispondrá de una GUI mediante la cual podrá mandar órdenes de control para desplazar el prototipo, recibir acuse de la última orden enviada y además, obtener valores de entorno de la zona física donde se encuentra en ese momento nuestro sistema encastado.

Destacar, que la aplicación web puede estar alojada en cualquier ubicación física, es decir, se podría instalar tanto en un servidor situado en la ubicación física del cliente como en cualquier host o servidor web situado en cualquier lugar del mundo que disponga de interconectividad con Internet.

Por último podemos observar a groso modo las tres funciones principales de nuestro proyecto, pudiendo observarse por un lado, la *capacidad de enviar y recibir datos desde internet* (flechas bidireccionales), así como *controlar el sistema con motores* desde la GUI, sin olvidarnos del sistema de protección autónomo de *control de cobertura*.

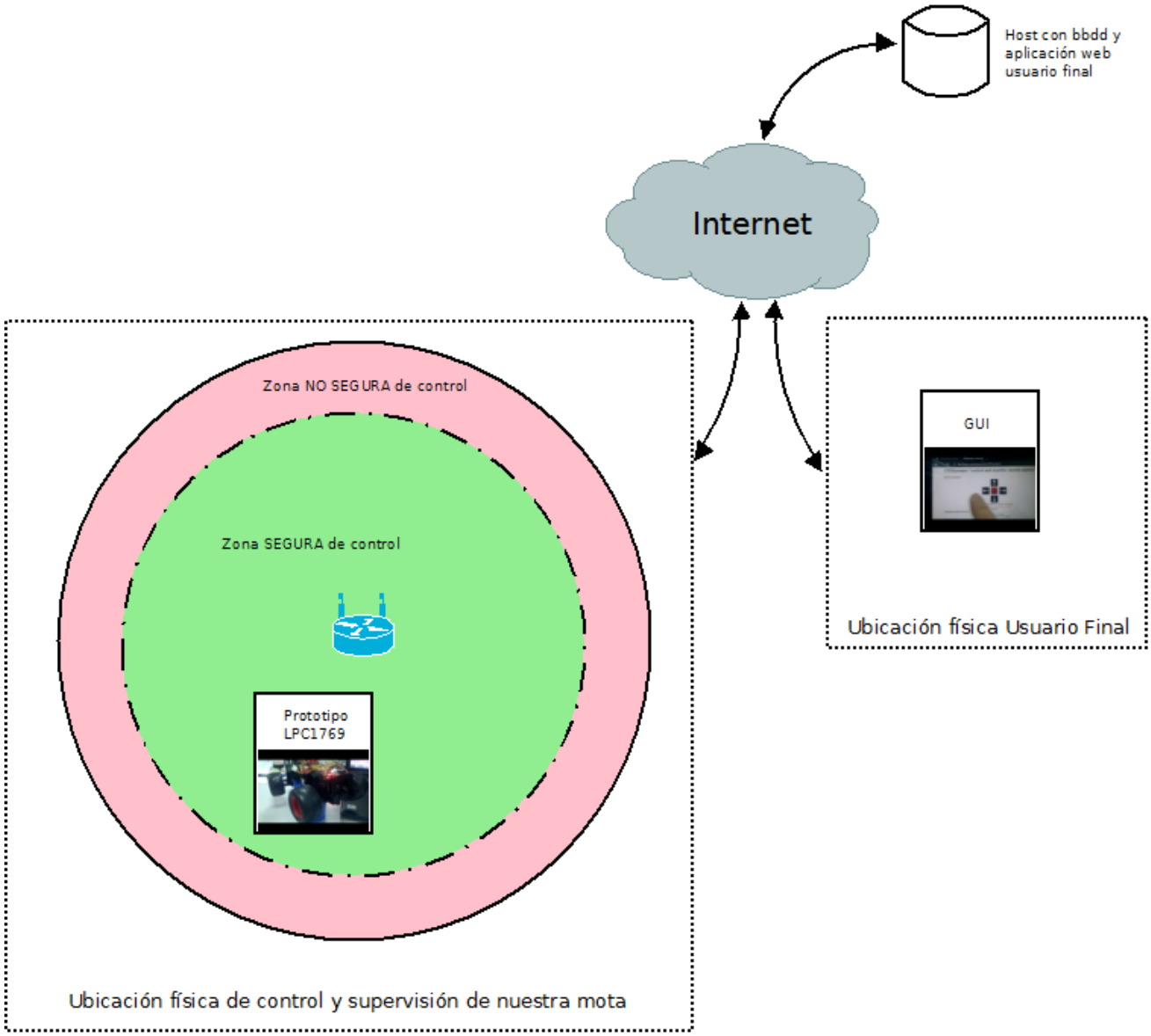


Figura 12: Diagrama explicativo sistema total

En el siguiente diagrama, podremos ver con más detalle, como interactúan los diferentes **objetos del sistema** (ver capítulo 1.6), además de las **comunicaciones internas** y **externas** empleadas:

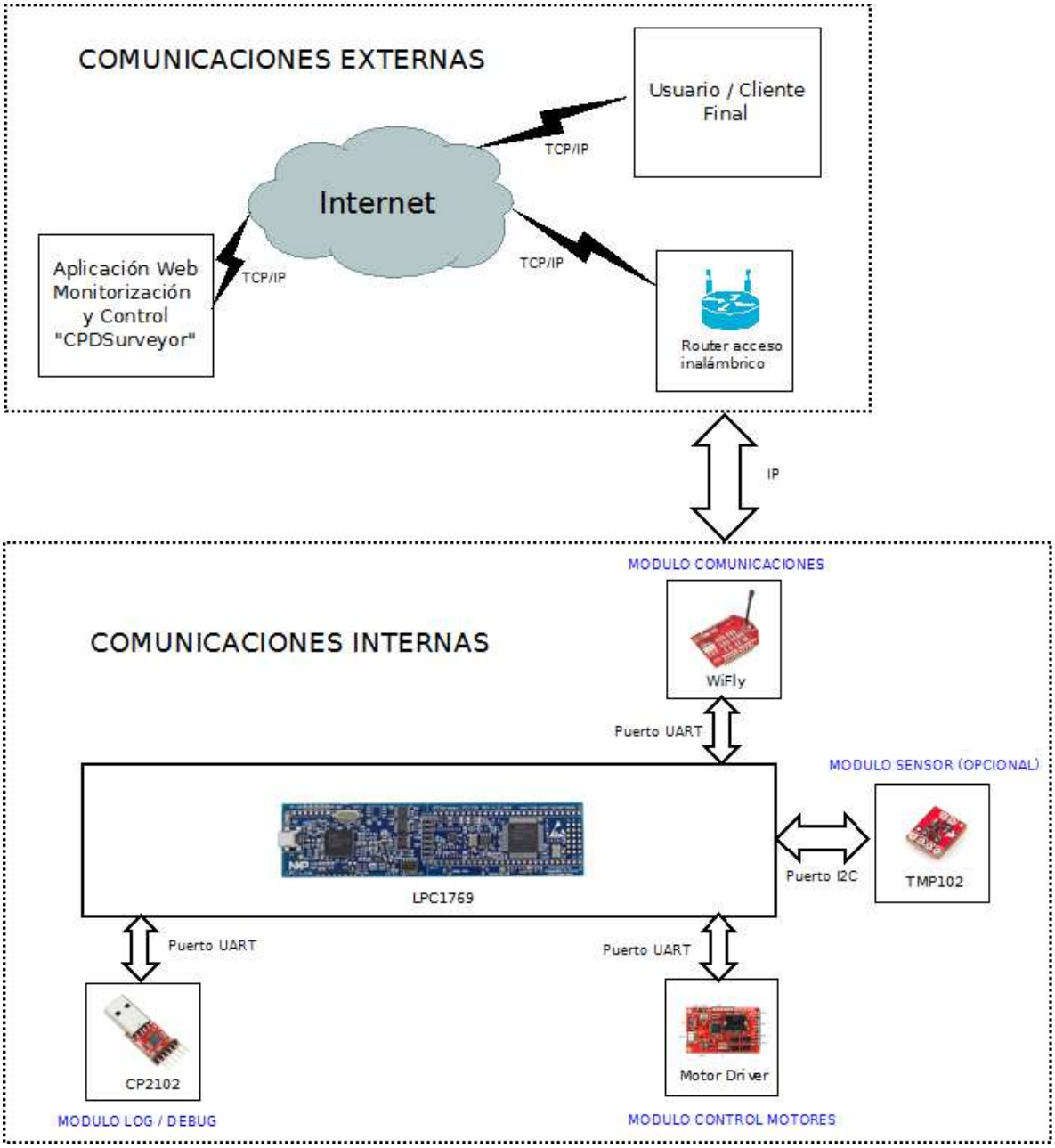


Figura 13: Diagrama de comunicaciones y módulos funcionales del sistema.

Se puede apreciar en nuestra mota los objetos hardware (módulos) implicados y que serán necesarios para poder ejecutar correctamente las tareas en nuestro sistema encastado. A continuación se listarán estos módulos y su función:

MODULO LOG / DEBUG:

En este módulo va conectado el “CP2102”, el cual es un simple convertidor de USB 2.0 a TTL UART de 6 pines.

Con este módulo podremos imprimir por la pantalla de un PC como se va ejecutando la aplicación del sistema, así como detectar errores en ejecución de tareas, en la ejecución de las mismas, etc.

MODULO COMUNICACIONES:

En este módulo irá el dispositivo “Wifly”. El principal objetivo será el poder conectar el sistema a una red inalámbrica y así poder disponer de interconectividad con internet sin hilos además de poder obtener otros parámetros informativos.

También se utilizará para el envío y recepción de datos desde y hacia nuestra aplicación web.

MODULO CONTROL MOTORES:

El objeto conectado en este caso será el “motor serial driver”.

El objetivo de este módulo será mover los motores de la mota y poder dotar de movilidad al prototipo.

MODULO SENSOR (OPCIONAL):

Este módulo es un ejemplo de otro objeto hardware que se podría conectar a nuestro sistema y por otro puerto interno de telecomunicaciones distintos a los utilizado (por un puerto I2C).

En este caso se ha optado por el sensor digital de temperatura.

3.2 Interface usuario

Se ha decidido crear una **interface básica** para el control y supervisión de la mota, ya que el diseño y programación de una aplicación web, conlleva muchas horas de trabajo y va más allá del alcance de este proyecto.

El principio básico de funcionamiento tiene su origen en la aplicación “Arp@Lab”, adaptando la interface a nuestras necesidades y diseñando una GUI portable e instalable en

básicamente cualquier servidor web que cumpla ciertos requisitos (en el siguiente capítulo se detallará estos requisitos y otros detalles técnicos).

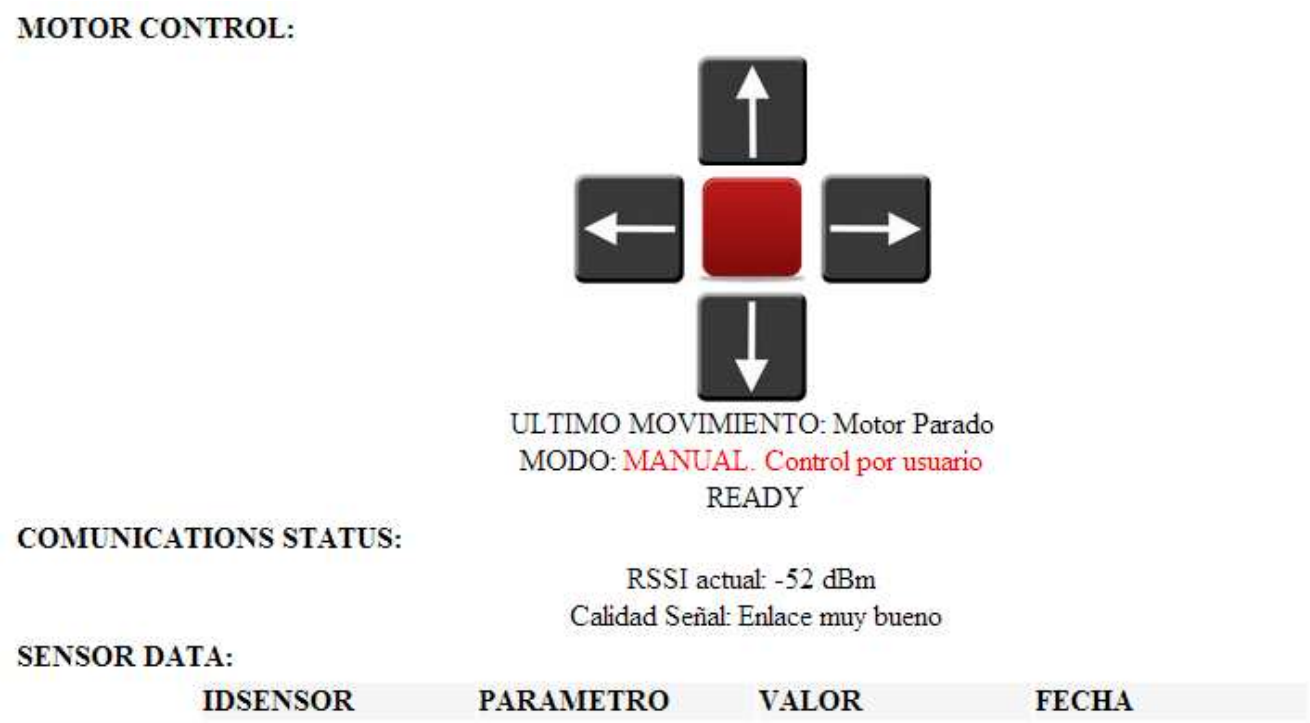


Figura 14: Captura de pantalla de la GUI

En la figura anterior podremos observar una **zona de control** (Motor Control) y una **zona de supervisión** (Comunications Status & Sensor Data).

En la siguiente imagen, podremos ver el diagrama de bloques de la GUI donde se aprecian las dos zonas anteriormente mencionadas, las principales funciones y como interactuará con el sistema encastado:

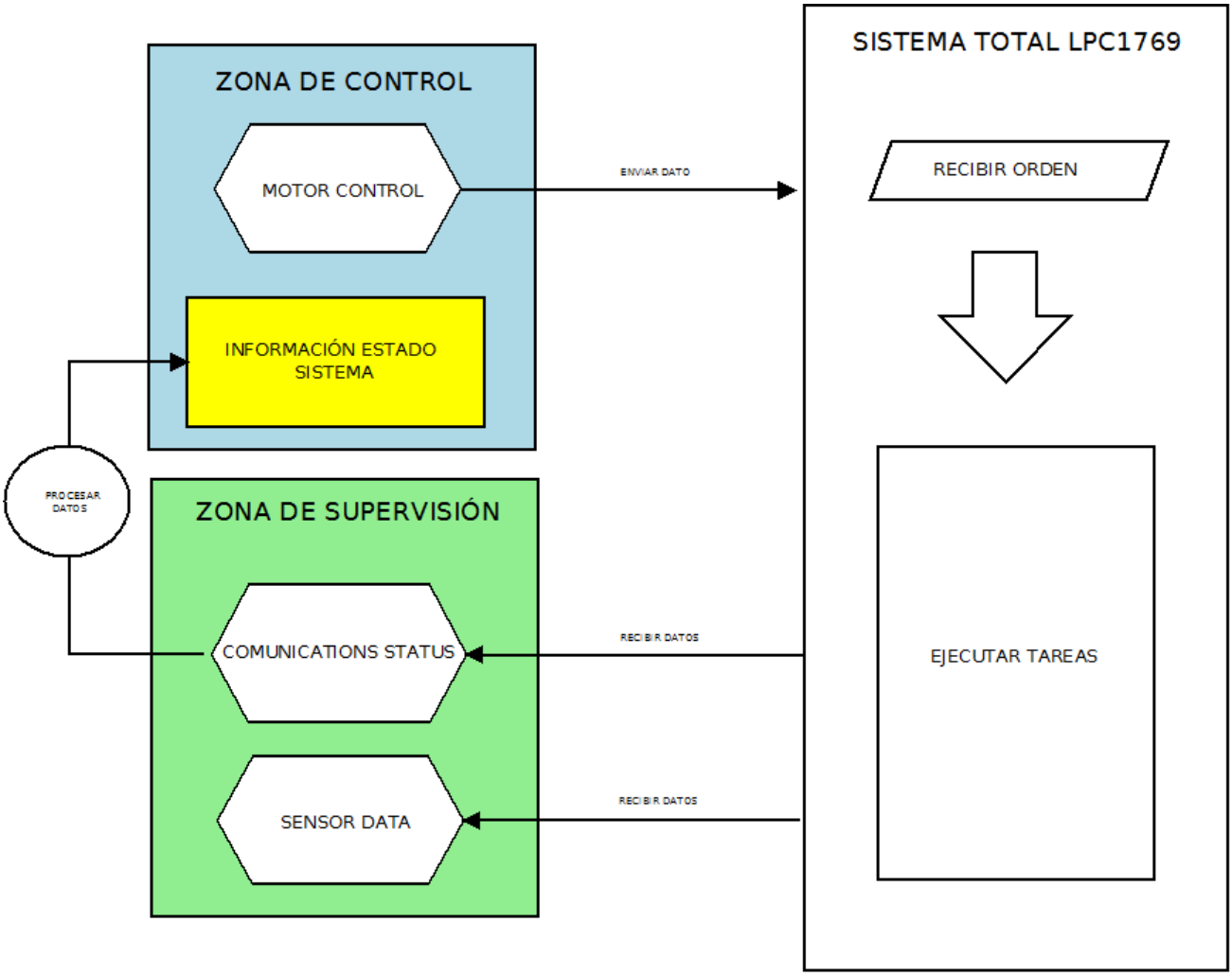


Figura 15: Diagrama de bloques de la GUI

3.3 Aplicación del sistema encastado

Como se ha comentado en capítulos anteriores, nuestro sistema encastado tendrá alojada una aplicación desarrollada bajo el sistema operativo en tiempo real FreeRTOS.

Se ha optado por desarrollar una aplicación que tiene su base en aplicaciones desarrolladas y estudiadas en la fase de formación, es decir, hemos desarrollado una aplicación con el objetivo de tener un **sistema productor / consumidor** sobre la aplicación CPDSurveyor (Aplicación Web).

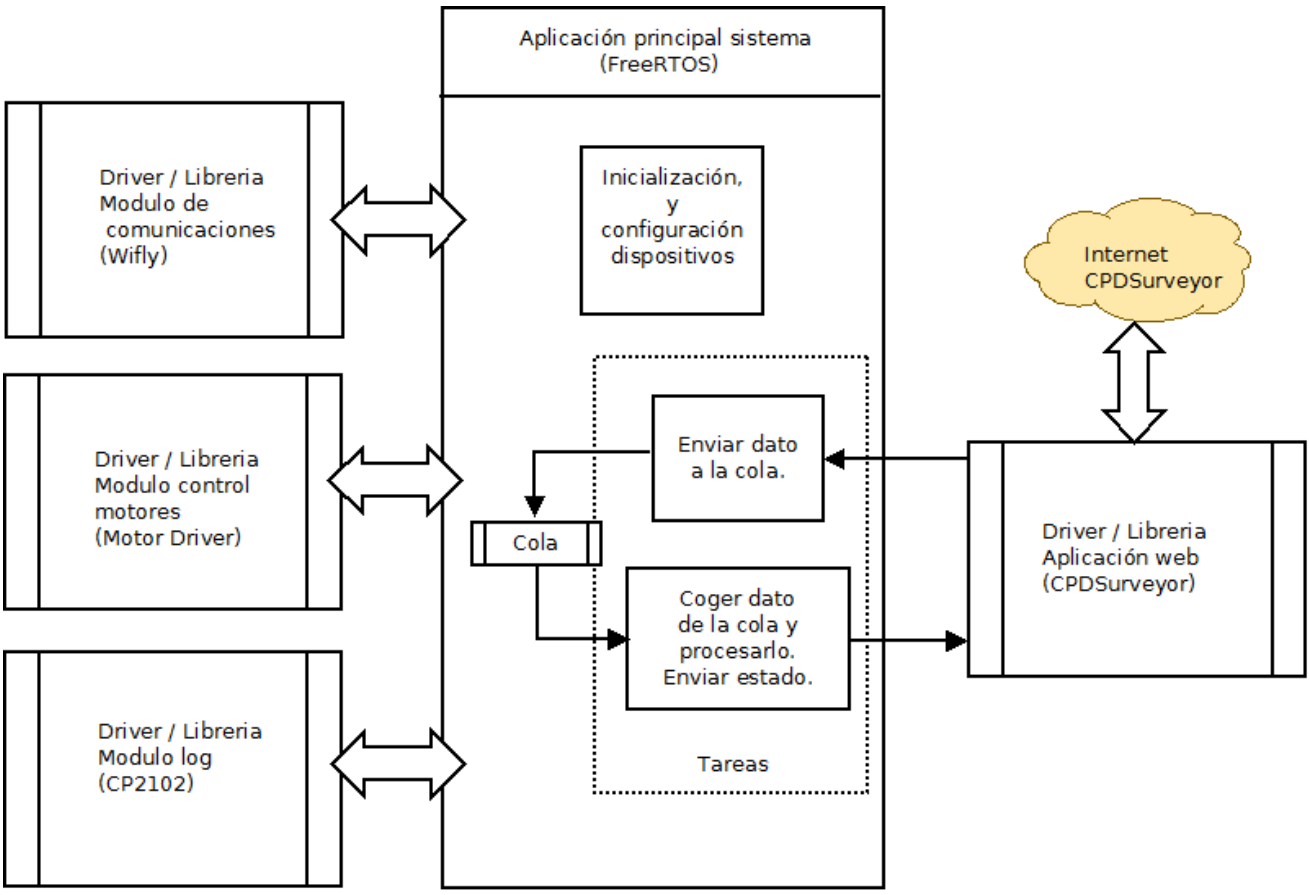


Figura 16: Diagrama de bloques de la aplicación del sistema encastado

Como se aprecia en la imagen anterior, por un lado vemos en el bloque central la **aplicación principal**, sustentándose por unas **librerías o drivers** diseñados para cada dispositivo hardware así como otra librería para la aplicación web CPDSurveyor.

Esta solución de diseño nos ha permitido desarrollar una aplicación de forma limpia, siendo más fácil la localización de errores y permitiéndonos en el futuro poder utilizar estas mismas librerías en otras aplicaciones para otros sistemas (**portabilidad e independencia**)

Como se ha comentado en este capítulo, en el siguiente capítulo nos adentraremos en detalles más técnicos de la aplicación del sistema.

4 Descripción detallada

4.1 Explicación módulos funcionales y hardware

Una vez que conocemos el funcionamiento básico de cada objeto hardware en el proyecto y a que módulo funcional pertenece, nos adentraremos en las siguientes secciones de una forma más técnica con cada uno de ellos, observando su integración progresiva en el sistema total.

4.1.2 Módulo principal sistema LPC1769

La mota LPC1769 como se ha comentado con anterioridad, esta basada en un microcontrolador Cortex-M3 de ARM y principalmente ha sido concebida para el desarrollo de aplicaciones embebidas con un **alto nivel de integración** y con un **bajo nivel de consumo**.

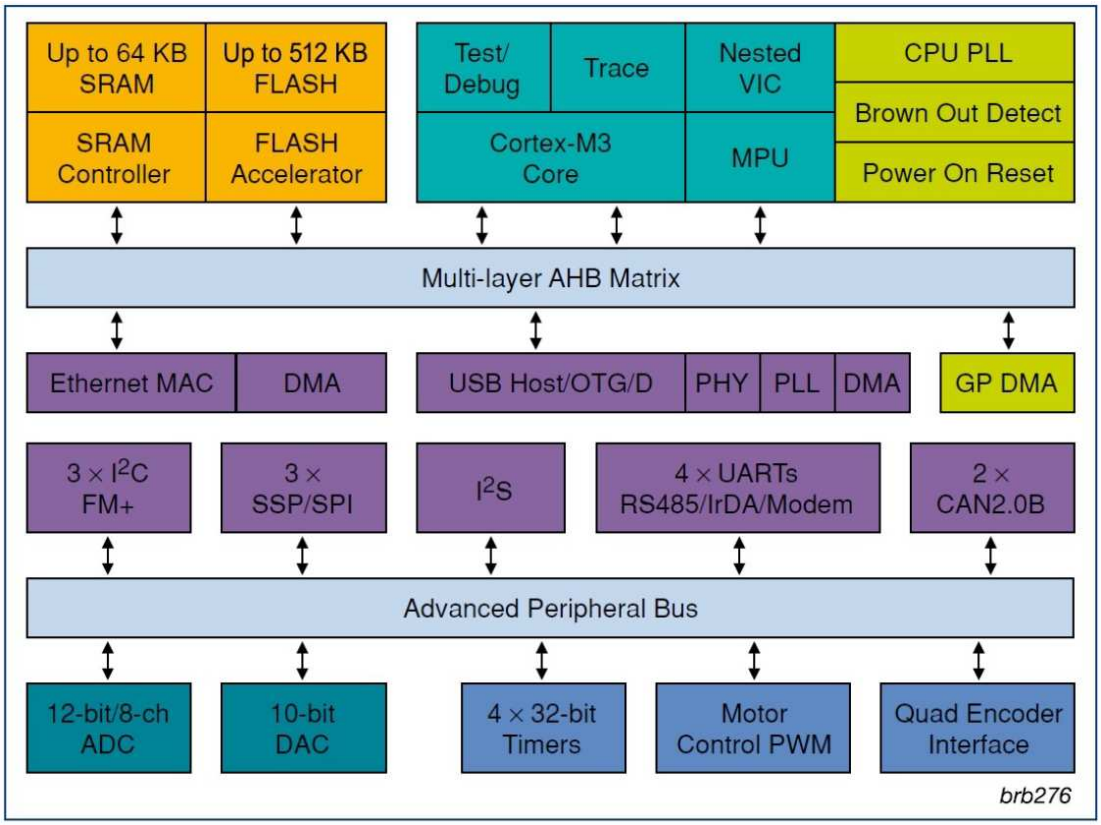


Figura 17: Diagrama de bloques de la mota LPC1769

Algunas de sus **principales características** son:

- Nuestra CPU puede trabajar en frecuencias de hasta 120MHz.
- Dispone de 512 kB de memoria flash.
- Hasta 64 kB de memoria de datos.
- Puerto Ethernet.
- Interface USB Device/Host/OTG
- 8 canales DMA.
- 4 puertos UART.
- 2 canales CAN.
- 2 controladores SSP.
- 1 SPI interface.
- 3 I2C interfaces.
- Motor control PWM

Y muchas otras características más que se pueden consultar en la página oficial del fabricante (consultar capítulo 10 Biografía).

Antes de la conexión de cualquier hardware en la mota, se ha procedido a al estudio de la misma en la fase de formación.

El primer paso ha sido familiarizarse con el IDE suministrado (basado en el entorno "Eclipse"), para posteriormente ir testeando proyectos demostrativos facilitados por el mismo fabricante. Con estos ejemplos se obtienen las primeras nociones de sistema operativo FreeRTOS así como del hardware.

Además, el fabricante pone a disposición una carpeta con varios proyectos de ejemplo demostrativos y comentados del IDE con la LPC1769 (archivo ubicado en la carpeta "Examples" del IDE NXP_LPCXpresso1769_MCB1700_2011-02-11.zip) , utilizando para ello librerías ya realizadas para cada tipo de comunicación (uart.h para los puertos UART, etc.)

Una vez formados y familiarizados con el sistema de desarrollo y la mota, se procede a crear un proyecto con una "aplicación principal de test" para realizar pruebas los módulos funcionales que instalaremos. En el apartado 4.3 se entrará en más detalle sobre la "aplicación principal final".

4.1.2 Módulo log / debug

Tal y como podemos ver en la figura 7, el **CP2102** es un módulo que nos convierte de USB 2.0 a serie (niveles TTL UART). Dispone de los siguientes 6 pines de conexionado:

PIN 1	GND
PIN 2	RXD
PIN 3	TXD
PIN 4	5V
PIN 5	3V3
PIN 6	RST

Tabla. 2: Pines conexión CP2102

A diferencia del resto de hardware (mota, WiFly, motor serial), hay que tener en cuenta que en la recepción RXD (pin 2), es donde se conecta la recepción del hardware a conectar, es decir, indica que es un pin de recepción. Lo mismo ocurre con la trasmisión TXD (pin 3), que corresponde con la transmisión del equipo a conectar).

Este hardware, aparte de utilizarse en el sistema como una salida para visualizar el proceso de ejecución del programa principal (log de la aplicación), ha sido de gran utilidad para aislar individualmente el resto de hardware implicado y poder hacer pruebas antes de conectarlo a la mota (como se puede comprobar en la figura 13 , el resto de hardware también va conectado por otros puertos UART).

Por tanto, este módulo ha tenido dos funciones en el proyecto:

1. **Abrir** sesiones serie directamente **contra** el WiFly y el Motor Serial para familiarizarnos y realizar pruebas con estos **dispositivos**.
2. **Implementar** el CP2102 en la aplicación principal como sistema de log para poder visualizar las acciones de la aplicación en forma de **log / debug**.

Continuando la segunda función anteriormente citada, en la siguiente figura podemos observar el esquema eléctrico del CP2102 conectado en el sistema por el puerto **UART3**:

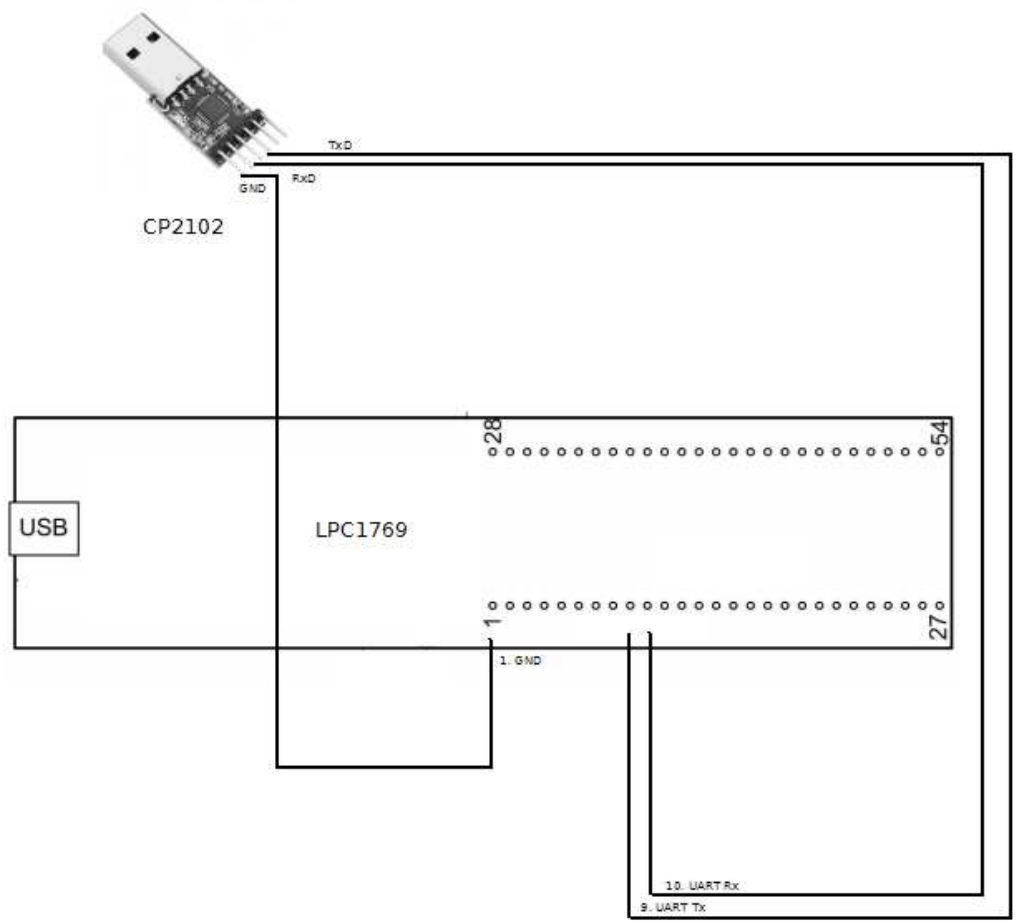


Figura 18: Esquema eléctrico de conexión del CP2102 con la LPC1769

Con esta conexión, una vez alimentada la mota a través de USB y con el CP2102 conectado a otro puerto USB del PC, se comienza con el desarrollo de una librería llamada "**printf.h**", la cual al ser llamada contendrá las siguientes funciones:

1. PrintInit :

Esta función tiene como objetivo **inicializar el UART** donde tenemos conectado el CP2102 y establecer el baudrate adecuado (en este caso utilizamos el UART 3 con un baudrate de 115200).

Esta función de inicialización también **creará un semáforo**, el cual será utilizado en las siguientes dos funciones como método de protección de exclusión mutua, de tal forma que evitaremos que dos tareas del programa principal intenten utilizar alguna de las dos siguientes funciones al mismo tiempo, es decir, intenten escribir por el log a la vez.

2. PrintText:

Con esta función enviaremos un buffer de texto al puerto Tx de la UART3 y podremos visualizar el texto por pantalla. Como se ha comentado anteriormente, esta función esta **controlada por el semáforo** creado anteriormente, por lo que antes de enviar el buffer, se ha de coger el semáforo, es decir, hay que utilizar la función **xSemaphoreTake**, para después una vez enviado el buffer de texto, volverlo a soltar con **xSemaphoreGive**.

3. PrintTextAndNum:

El funcionamiento de esta función es el mismo que la anterior, pero con la diferencia de que se le puede pasar como parámetro un número además del buffer de texto.

Para poder ver por pantalla los mensajes de log, es necesario levantar una sesión contra el CP2102 desde un PC, pudiendo utilizar el software “putty” o similar indicando el puerto COM donde esta conectado nuestro CP2102 así como el baudrate inicializado (115200).

4.1.3 Módulo comunicaciones

El módulo WyFly RN-XV es una solución ideal para dotar a nuestro sistema LPC1769 de interconectividad 802.11 b/g. Este dispositivo en sí, ya es una mota o sistema encastado, ya que esta dotado por un microcontrolador de 32 bits de SPARC , 8 Mb de memoria flash, 128 Kb de memoria RAM, ultra bajo consumo, etc..

Nuestro modelo en cuestión es el RN-171, el cual dispone de antena incorporada y el siguiente consumo en lo que ha transmisión se refiere (Tx puede ser configurable por el usuario):

4uA sleep, 35mA Rx, 185 mA Tx at 12dBm

Como veremos en la siguiente tabla, no solamente dispone de un puerto UART, si no de funciones y puertos adicionales como E/S GPIOs, ADCs, etc..

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

Tabla. 3: Descripción de pines de conexión dispositivo WiFly

Como se realizó con el CP2012, antes de conectarlo a la mota, se estudió individualmente, ya que además de servirnos para conectar la LPC1769, este módulo tiene la opción de conectarse a servidores web para poder enviar y recibir datos desde clientes HTML (peticiones GET).

Para conectarnos con el dispositivo tan solo hace falta conectar los pines del Wifly indicados en el esquema de la siguiente página, con los pines UART del CP2102 (teniendo en cuenta el detalle de la TXd y RXd comentado anteriormente en el CP2102) y comprobar en que COM tenemos conectado nuestro CP2102 abriendo sesión con un baudrate de 9600.

Posteriormente podremos comprobar que se nos abre una consola indicándonos la versión del firmare del dispositivo Wifly (la 2.32). Para entrar en modo consola es necesario introducir $$$ sin utilizar el salto de línea.

Una vez familiarizados con el entorno de configuración del dispositivo de forma aislada, se procedió a la instalación tal y como se puede apreciar en el siguiente esquema eléctrico:

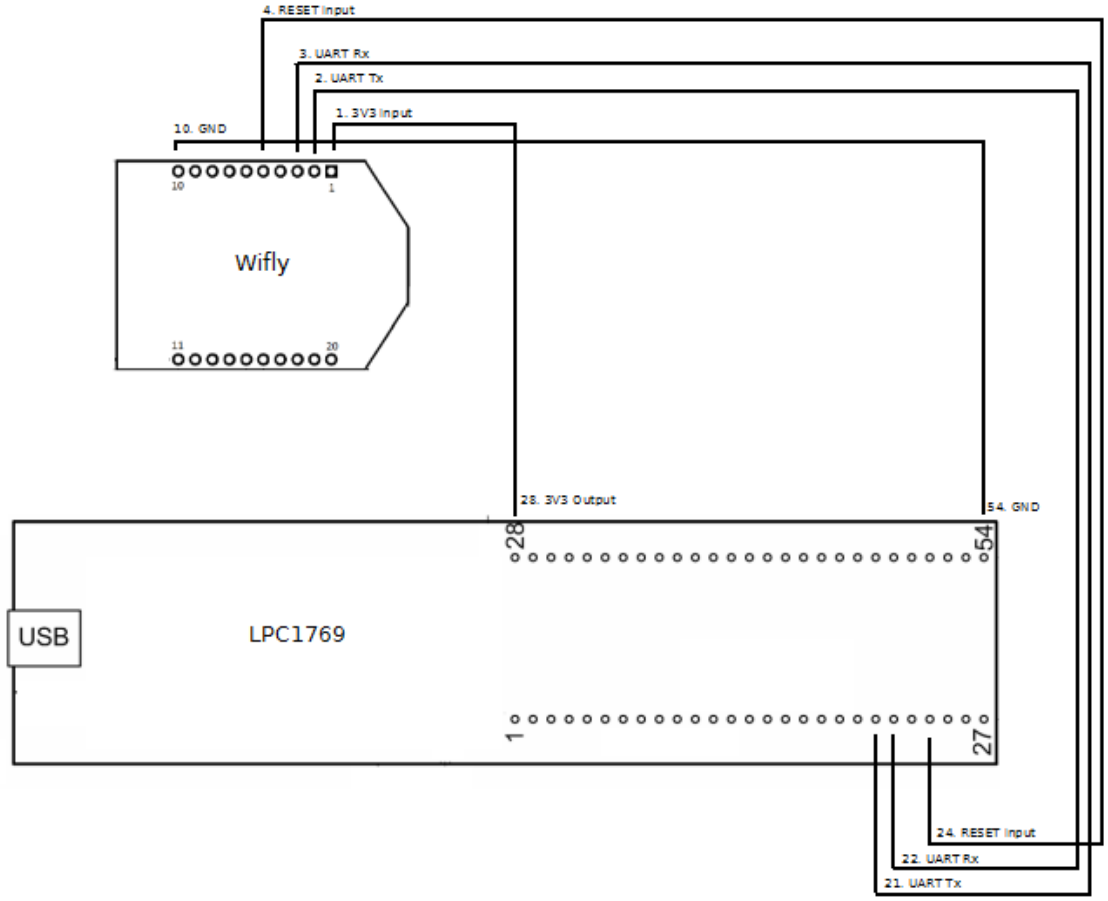


Figura 19: Esquema eléctrico de conexión del WiFly con la LPC1769

Con esta conexión se puede comenzar el desarrollo de una librería llamada "**wifly.h**", la cual al ser llamada tendrá entre otras las siguientes funciones utilizadas en nuestro proyecto:

1. WiflyInit :

Al igual que en el CP2102 **se inicializará** el dispositivo Wifly en el **UART** correspondiente (en este caso utilizamos el UART 0 con un baudrate de 9600).

También **creará un semáforo** como método de protección de exclusión mutua, que será utilizado en una función encargada de escribir enviar comandos, para así evitar que se escriba por el UART 0 más de un comando a la vez.

Por último se realizará un reset para limpiar de toda configuración previa al dispositivo.

NOTA: como se ve en el esquema eléctrico, se ha utilizado el pin 24 para mandar la **señal de reset** al dispositivo, es decir, se envía un pulso de una duración mínima de 160us tal y como se nos indica en la tabla 3.

2. WiflyComand:

Está función será la encargada de enviar un buffer de texto y añadir un salto de línea (comando + intro) al puerto Tx de la UART0 y así recibir el dispositivo por su puerto Rx el comando. Como se ha comentado anteriormente, esta función esta **controlada por el semáforo** creado anteriormente, por lo que antes de enviar el buffer, se ha de coger el semáforo, es decir, hay que utilizar la función **xSemaphoreTake**, para después una vez enviado el buffer de texto, volverlo a soltar con **xSemaphoreGive**.

3. WiflyConsola:

Función igual que la anterior, pero difiere en que no envía un salto de línea, ya que como se ha comentado en la página 31 para entrar en modo consola no se debe de pulsar el intro.

4. WiflyUART0scanf:

Esta función será la encargada de leer lo que recibimos del dispositivo Wifly una vez enviado un comando. Como parámetros se pasan el buffer donde deseamos que se guarde el texto recibido así como el tiempo que deseamos que escuche la recepción.

Es decir, recibimos del puerto Tx del Wifly al puerto Rx de la LPC1769 y leemos el contenido del buffer del UART0.

5. WiflyConect:

Con esta función nos conectaremos a la red Wi-fi pasando como parámetros de entrada la password y la BSSID de la red.

6. WiflyLogResponse:

Se ha creado esta función que tiene como principal objetivo, encapsular la función WiflyUART0scanf y hacer uso de la librería creada en el apartado 4.1.3 (print.h) para imprimir la

respuesta a los comandos, pasando como variables un buffer y por el puerto UART donde esta conectado el dispositivo de log (CP2102)..

7. WiflyRssi:

Con esta función se realiza un “parser” a la respuesta del comando para conocer la RSSI (“show rssi”) y se devuelve. Muy importante obtener este dato el cual será el que nos medirá la calidad de señal en todo momento.

De la misma forma que se ha creado WiflyRssi, se han creado otras funciones más, como por ejemplo **WiflyIp**, para conocer la ip asignada por el router de acceso a la red, **WiflyTimePower** para conocer el tiempo que lleva encendido el dispositivo WifFy, etc. (más información en el código del proyecto).

Por último, en el siguiente diagrama de bloques podemos ver todas las funciones creadas en esta librería, su interacción con la librería printf.h y los dos dispositivos conectados al sistema encastado. En verde destacamos las funciones que comunican con el puerto UART0 (Wifly).

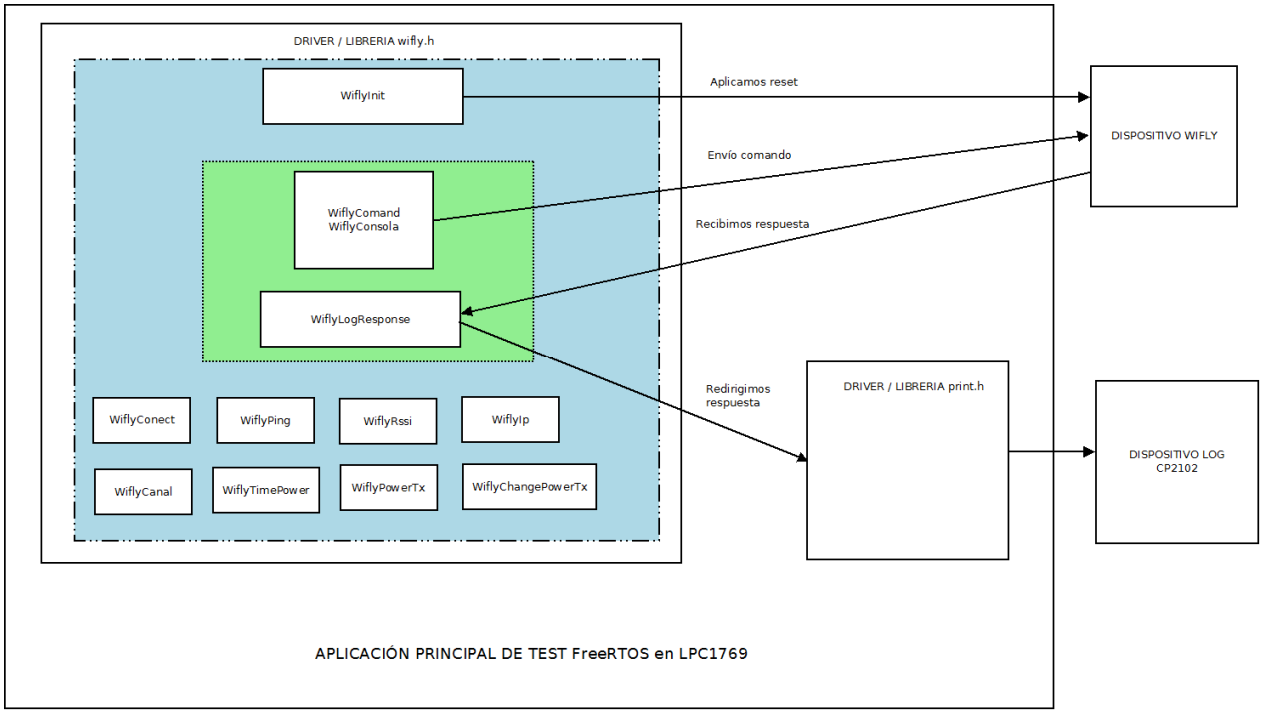


Figura 20: Diagrama de bloques funciones e interacción librería WiFly

4.1.4 Módulo control motores

El Serial Controlled Motor Driver ROB-09571 RoHS (ver detalle fig. 5) es una solución ideal para **controlar dos motores**.

Esta dispositivo esta provisto por un microcontrolador de 8 bits de bajo consumo, concretamente el microcontrolador **ATMEGA168**.

Se puede alimentar con una tensión de entrada de entre 5 y 16V, pudiendo proveer de 2 Amperios a cada motor (4 Amperios en total).

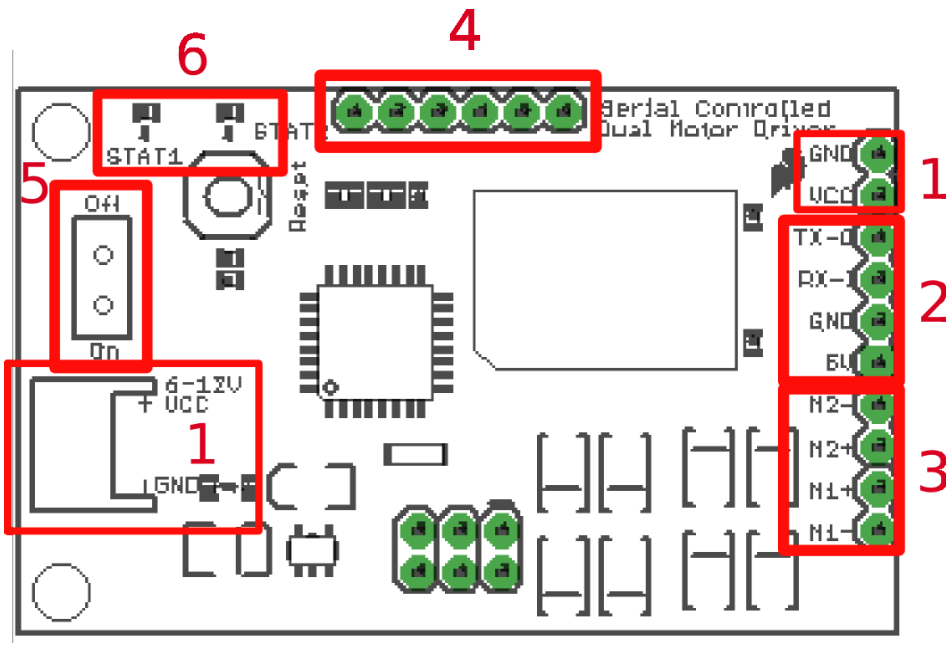


Figura 21: Detalle de zonas funcionales y pines conexión motor serial driver

Se pueden observar en la figura anterior 6 zonas funcionales de conexionado:

ZONA 1	2 posibles zonas de alimentación.
ZONA 2	Pines interface UART.
ZONA 3	Pines + y – conexionado motores.
ZONA 4	Pines extra.

ZONA 5	Switch encendido/apagado.
ZONA 6	Leds de estado.

Tabla. 4: Correspondencia zonas / funciones motor driver serial

En este caso, al igual que con el módulo Wifly, se conectó el CP2102 directamente y se alimentó el dispositivo con el pin de 5V del CP2102.

Posteriormente, se abrió una sesión (con un baudrate de 115200) para comprobar el correcto funcionamiento de la tarjeta, conectando antes dos leds, uno en M1, y el otro en M2.

Con esto conseguimos aislar la mota y la plataforma con los motores para comprobar el correcto funcionamiento de los comandos, así como para desarrollar la librería específica.

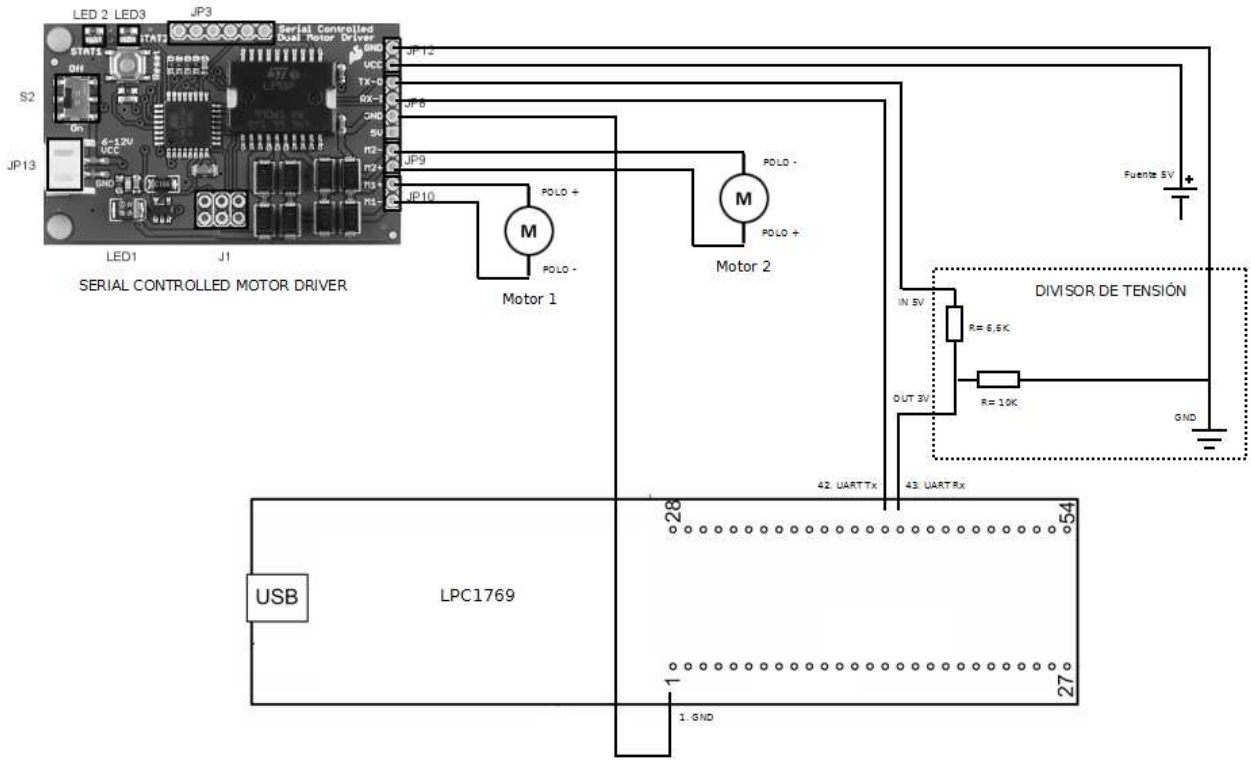


Figura 22: Esquema eléctrico de conexión del Serial Motor Driver con la LPC1769

Destacar, que gracias al procedimiento de aislar los componentes y a recomendaciones de terceros, se detector que la salida Tx del componente sacaba 5 V, en vez de 3.3 V como el resto.

Tras mirar las especificaciones de la LPC1769, y aunque se indicaba que los puertos UART disponían de una tolerancia de 5V (otros pines de la mota son solo tolerantes a 3.3V) se decidió por si fuera una errata del manual técnico, realizar un **divisor de tensión** como el indicado en la figura 22, con el objetivo de acoplar esta salida con la entrada de la mota.

Tras solventar este problema, se procedió al desarrollo de una librería llamada "**motor.h**", la cual al ser llamada, tendrá entre otras, las siguientes funciones utilizadas en nuestro proyecto:

1. MotorInit :

Al igual que las anteriores funciones de inicialización, se inicializará el dispositivo en el UART correspondiente (en este caso utilizamos el UART 1 con un baudrate de 115200).

También creará un semáforo como método de protección de exclusión mutua, que será utilizado en una función encargada de enviar comandos, evitando así que se escriba por el UART 1 más de un comando simultáneamente.

2. MotorMakeComand :

Esta función formará el comando y lo almacenará en un buffer para poder ser enviado al dispositivo. El comando dependerá de tres variables (Motor, Dirección, Velocidad) añadiendo al final un salto de línea.

En la siguientes tablas se representan la sintaxis de los comandos, teniendo en cuenta que, nosotros tendremos conectado el **motor de tracción (adelante y atrás) en M1** y el **motor de dirección (derecha e izquierda) en M2** tal y como se puede observar en el esquema eléctrico.

Motor Ind.	Direction	Speed	End of Command
'1' or '2'	'f' or 'r' (case insensitive)	'0'-'9'	'/r'

Tabla. 5: Sintaxis comandos Motor Serial Driver

Speed Value	PWM Duty Cycle
'0'	Off
'1'	11%
'2'	22%
'3'	33%
'4'	44%
'5'	55%
'6'	66%
'7'	77%
'8'	88%
'9'	100% (Full Speed)

Tabla. 6: Valores de velocidad en el motor serial driver

3. MotorSendComand :

Función similar a WiflyComand (página 33), en la cual se enviará un buffer de texto al puerto Tx de la UART1 (donde tenemos conectado el dispositivo) y así poder recibir el dispositivo por su puerto Rx el comando.

Esta función también esta controlada por el semáforo creado en la función MotorInit.

4. MotorUART1scanf :

Igual que la función WifliUART0scanf (página 33) pero para poder recibir del puerto Tx del dispositivo del motor al puerto Rx de la LPC1769 y leer el contenido del buffer del UART1.

5. MotorLogResponse:

Función cuyo objetivo es exactamente igual que la función WifliLogResponse (página 33).

6. MotorControl:

En esta función será pasado un número entero (ver tabla 7 página 45) desde la aplicación web para posteriormente efectuar el control sobre el motor.

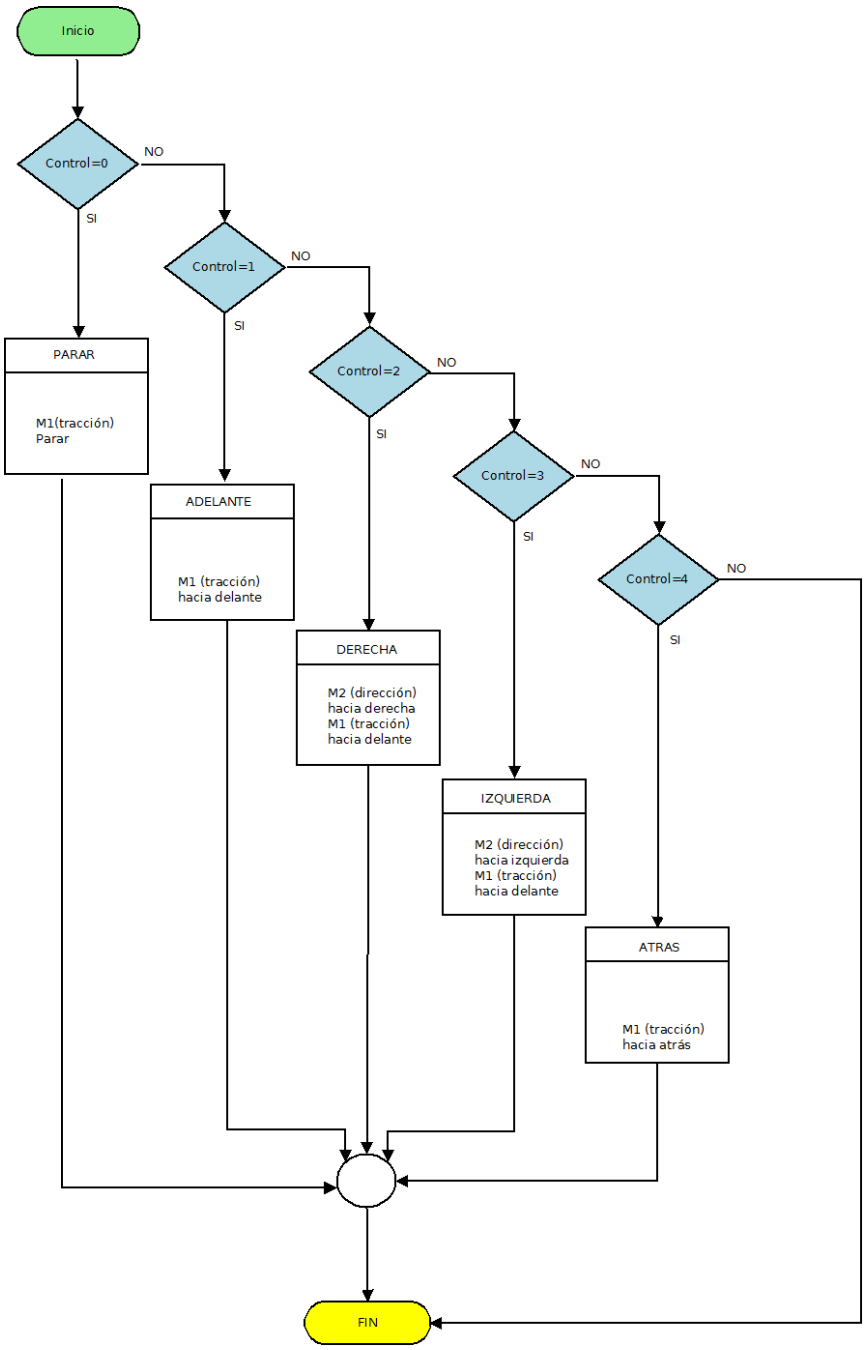


Figura 23: Diagrama de flujo de la función “MotorControl”

Como se puede apreciar en la figura anterior, se utilizarán las funciones "**MotorMakeComand**", "**MotorSendComand**" con los parámetros ajustados al movimiento que realizamos (al ser motores de pequeño voltaje, la velocidad siempre será de "9").

7. MotorCobertura:

Esta función se ha diseñado para poder tener un sistema automático de búsqueda de mejor cobertura. Para ello, comparará el **rssi actual** con un **rssi límite** que será establecido en una variable dentro del programa principal.

Además y **para que se cumpla esta protección de cobertura, deberá ser llamada justo después de realizar cualquier movimiento.**

Su funcionamiento es el siguiente:

1) Se realiza un movimiento y se obtiene el rssi.
2) Si el rssi está fuera del límite, será llamada esta función y se ejecutará.
3) Lo primero que hará será moverse hacia adelante y volver a medir el rssi.
4) Si éste mejora, seguirá hacia delante hasta conseguir un rssi inferior al límite.
5) Si empeora es que vamos en la dirección incorrecta e irá hacia detrás hasta conseguir un rssi inferior a límite.

De tal forma que hacia una dirección u otra conseguirá una zona con mejor cobertura.

Es imprescindible mover el prototipo con anterioridad hacia delante para coger una dirección como referencia.

IMPORTANTE!

Debemos tener en cosideración, que el rssi límite debe de estar también programado en la aplicación web, de tal forma, que si por ejemplo ponemos en el sistema principal rssi límite = -80, en la aplicación web al ser inferior -80 también nos avise y pase a modo automático.

En el apartado 4.2 (en la sección TRATAMIENTO DE DATOS) aparece la configuración de los modos en la aplicación web.

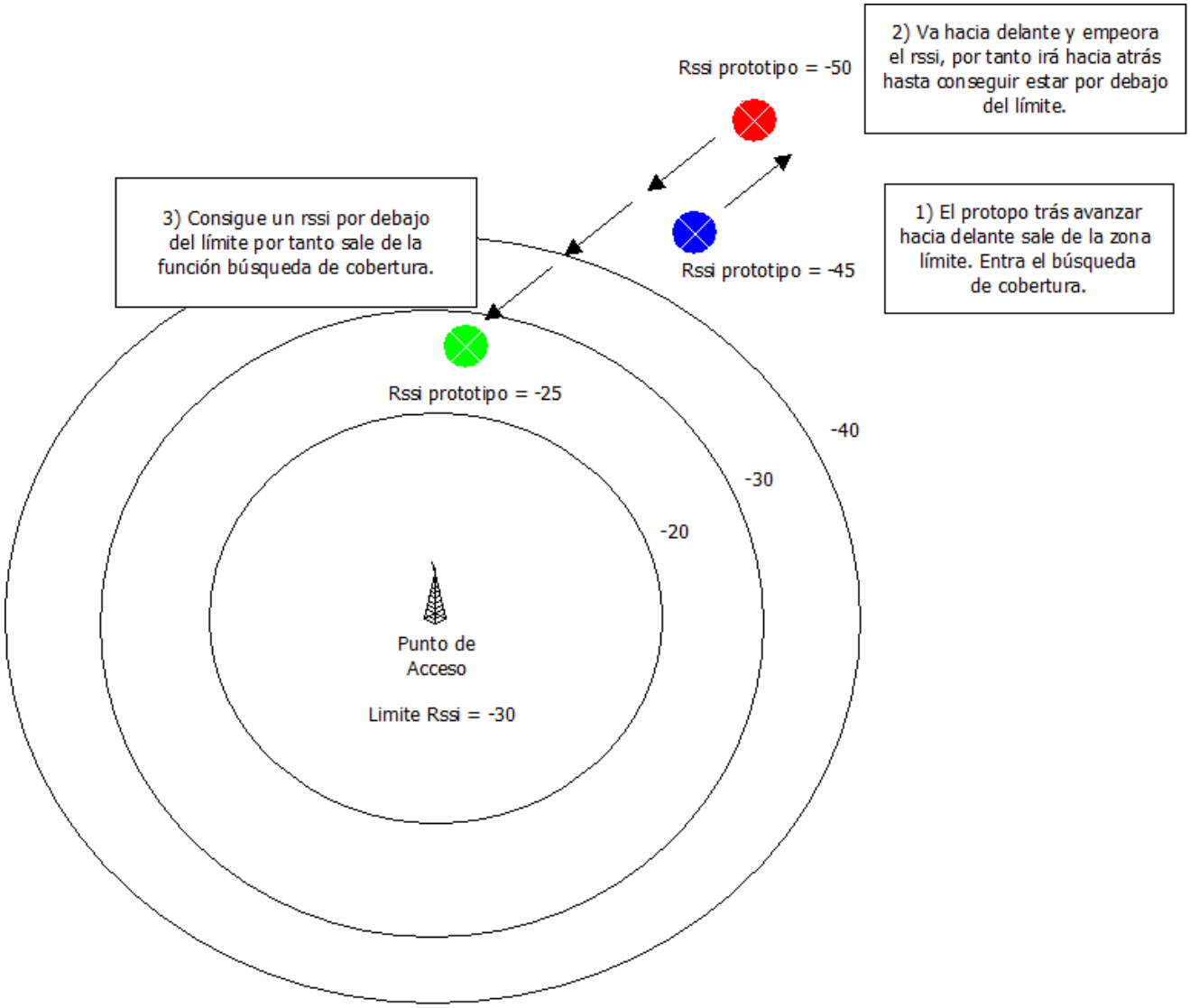


Figura 24: Diagrama conceptual del sistema de protección de la cobertura

En el anterior diagrama podemos apreciar el funcionamiento de la protección contra la pérdida de cobertura.

Para finalizar, en el siguiente diagrama flujo se puede apreciar el funcionamiento de esta función:

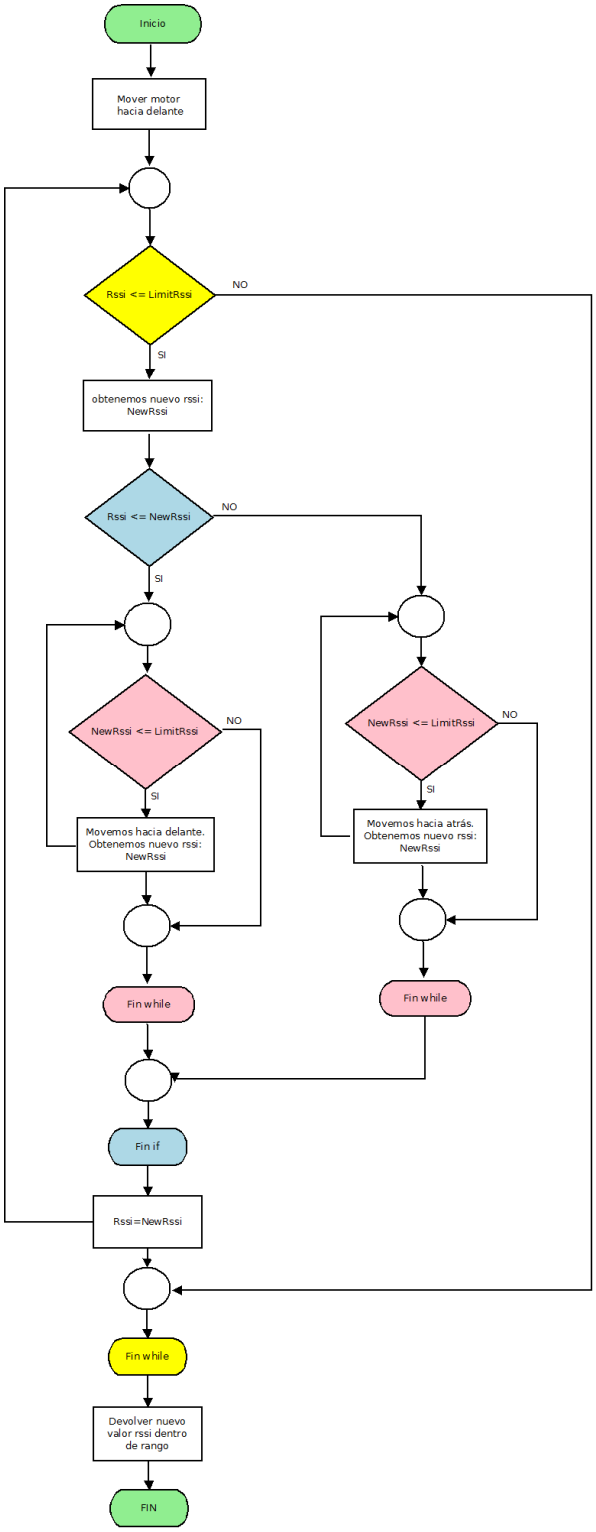


Figura 25: Diagrama de flujo de la función “MotorCobertura”

4.2 Explicación interface de usuario

La interface de usuario ha sido desarrollada con el lenguaje de **programación PHP** y apoyada con una **base de datos Mysql** donde se irán albergando los datos recibidos desde nuestro sistema. Para la realización del cursor de control del motor, se procedió a diseñar un scrpit utilizando la tecnología asíncrona **AJAX**.

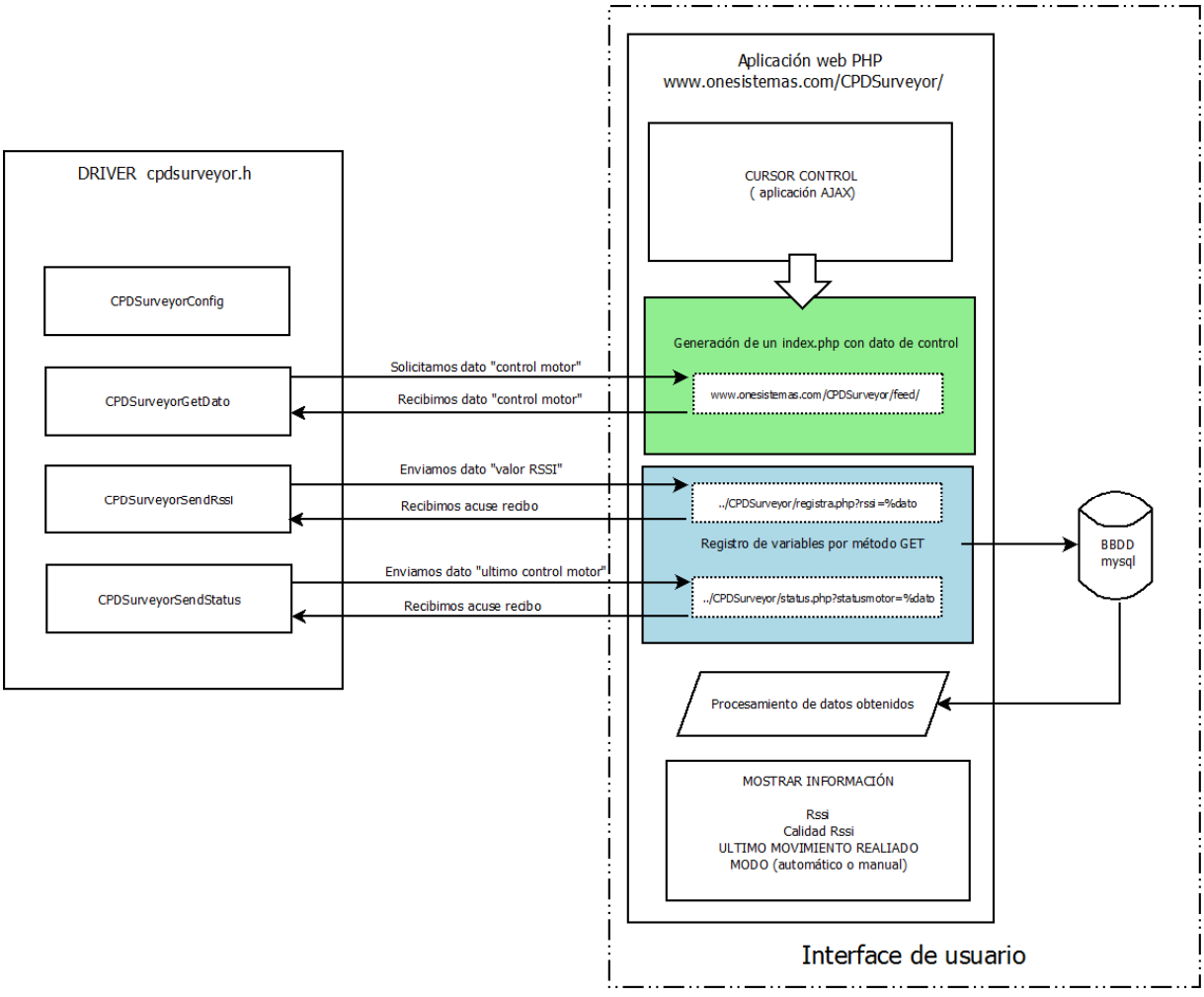


Figura 26: Diagrama funcional de la aplicación web con la librería “cpdsurveyor.h”

Como se puede observar en la figura 26, se ha diseñado un driver llamado “**cpdsurveyor.h**”, el cual actuará de interface entre nuestro sistema encastado y la aplicación web (módulo verde y módulos azules), conteniendo las siguientes funciones:

1. CPDSurveyorConfig :

Esta función será llamada una sola vez y será la encargada de configurar el dispositivo Wifly con la dirección (o IP) donde se encuentra ubicada la aplicación web, además del puerto donde escucha (normalmente suele ser el puerto 80).

Además se enviará otros comandos para configurar el dispositivo con la habilitación de los protocolos HTTP y TCP, convertir los datos de binario a formato ASCII, etc…

2. CPDSurveyorGetDato:

Esta función configurará el dispositivo Wifly con la dirección de la aplicación donde se encuentra el dato a descargar (http://www.onesistemas.com/CPDSurveyor/feed/index.php), para posteriormente enviar el comando “open”, de esta forma se descargará la página con el dato.

En esta misma función se realiza un “parser” para obtener el dato que será procesado en la aplicación principal del sistema encastado LPC1769.

3. CPDSurveyorSendRssi:

Similar a la anterior, esta función configurará el dispositivo Wiflly para enviar un dato con el valor rssi a la aplicación web. Para ello se deberá configurar la dirección con el método GET$/CPDSurveyor/registra.php?rssi=%dato

4. CPDSurveyorSendStatus:

Con estructura igual que la número 3, pero en este caso se configurará con GET$/CPDSurveyor/status.php?statusmotor=%d y se pasará como dato el último control ejecutado.

Una vez que tenemos definido el interface, detallaremos como se **generan los controles**, como se **guardan los datos** en la base de datos y como se **procesan los datos** para informar al usuario final en la GUI.

GENERACIÓN CONTROLES MOTOR

Con el script en AJAX desarrollado, al pulsar sobre cualquier botón del cursor (ver figura 14 página 23) , se generará un archivo index.php con un dato de tipo entero en su interior.

Dicho archivo index.php, será creado en la ruta de la aplicación: .. /CPDSurveyor/feed/ con el siguiente contenido:

<html> <title>CPDSurveyor Feed</title><body><h1>**3**</h1></body></html>

Podemos observar el dato de control número 3 entre las etiquetas html marcadas en rojo. Como veremos a continuación, este número corresponde a realizar el movimiento hacia la izquierda, ya que cada botón del cursor, así como el botón de paro, tienen asociados un único número que corresponde a una dirección:

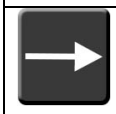
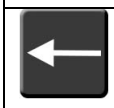
	0 (Parar)
	1 (Adelante)
	2 (Derecha)
	3 (Izquierda)
	4 (Atrás)

Tabla. 7: Equivalencia entre botones cursor y valores enteros

BBDD DE LA APLICACIÓN

Como se ha comentado con anterioridad, se ha creado una base de datos MySQL con tres tablas con la siguiente estructura y declaración de variables:

TABLA "**wifly**":

Nombre Columna:	Id	rssi	calidad
Tipo dato	Bigint(20)	float	Varchar(25)
Extra	AUTO_INCREMENT		
Ejemplo dato:	1	-41	Enlace muy bueno

TABLA "**status**":

Nombre Columna:	Id	statusmotor
Tipo dato	Bigint(20)	Varchar(25)
Extra	AUTO_INCREMENT	
Ejemplo dato:	1	Motor Izquierda

TABLA "**modo**":

Nombre Columna:	Id	modomotor
Tipo dato	Bigint(20)	Varchar(25)
Extra	AUTO_INCREMENT	
Ejemplo dato:	1	MANUAL. Control por usuario

Tabla. 8 Tablas de la base de datos

Para almacenar los datos desde nuestra aplicación a la base de datos, se ha creado un archivo llamado "**conexionmysql.php**", en el cual se indica los datos necesarios para poderse conectar a la aplicación (dirección o ip de servidor donde se encuentra la bbdd, nombre de la bbdd y por último, usuario y password de la bbdd).

En este mismo archivo, se indicarán las tablas de la bbdd que se utilizarán y los comandos php necesarios para conectar con la base de datos (mysql_conect, etc.).

También se ha creado un archivo llamado "**finconexionmysql.php**" el cual la única función que tiene es la de cerrar la conexión con la bbdd (mysql_close).

TRATAMIENTO DE DATOS

Como se puede observar en el bloque azul, de la figura 26 de la página 43, se reciben datos utilizando el método GET y posteriormente son tratados y almacenados en las tablas anteriores, teniendo como finalidad mostrar información al usuario en la GUI.

DATO RSSI:

En primer lugar esperamos recibir el valor rssi en la página registra.php desde la aplicación principal del sistema encastado para posteriormente almacenarla:

1) RECIBIMOS EN: ../CPDSurveyor/registra.php?rssi=%dato
2) ALMACENAMOS EN VARIABLE: $RSSI = $_GET['rssi'];

Al recibir el rssi y procesamos los siguientes mensajes de estado:

```
if ($RSSI>-40){
    $calidad = 'Enlace ideal';
    $modomotor = 'MANUAL. Control por usuario';
}elseif($RSSI>-60){
    $calidad = 'Enlace muy bueno';
    $modomotor = 'MANUAL. Control por usuario';
}elseif($RSSI>-70){
    $calidad = 'Enlace bueno';
    $modomotor = 'MANUAL. Control por usuario';
}elseif($RSSI>-80){
    $calidad = 'Enlace normal-bajo';
    $modomotor = 'AUTOMATICO. Atención! no controlar';
}elseif($RSSI>=-200){
    $calidad = 'Enlace malo';
    $modomotor = 'AUTOMATICO. Atención! no controlar';
}
```

Como se puede observar, en cado de que el RSSI sea menos de -80 dBm, nuestro prototipo entraría en modo automático ejecutando la función de búsqueda de cobertura.

Por último, todos estos datos se almacenarán en la bbdd, para posteriormente mostrarlos en la GUI:

TABLA wifly ($RSSI,$calidad)
TABLA modo($modomotor)

DATO STATUSMOTOR.

Procediendo de la misma forma que con el dato RSSI, esperamos recibir el valor con el último movimiento realizado, en la página status.php. Dato enviado desde la aplicación principal del sistema encastado para posteriormente almacenarla:

3) RECIBIMOS EN: ../CPDSurveyor/status.php?statusmotor=%dato
4) ALMACENAMOS EN VARIABLE: $statusmotor = $_GET['statusmotor'];

Procesamos la variable:

```
if ($statusmotor==0){
	$direccion = 'Motor Parado';
}elseif($statusmotor==1){
	$direccion = 'Motor Adelante';
}elseif($statusmotor==2){
	$direccion = 'Motor Derecha';
}elseif($statusmotor==3){
	$direccion = 'Motor Izquierda';
}elseif($statusmotor==4){
	$direccion = 'Motor Atras';
}
```

Por último lo almacenamos en la bbdd, para posteriormente mostrarlos en la GUI:

TABLA status ($statusmotor)

4.3 Explicación de la aplicación del sistema encastado

Como ya se introdujo en la figura 16 de la página 25, la aplicación principal del sistema será la encargada de ejecutar todas las tareas con las funciones necesarias para conseguir los objetivos mínimos propuestos.

Este programa principal (llamado también "main.c"), estará creado como un proyecto FreeRTOS,el cual a su vez, llamará a otros "proyectos librerías" .

Un proyecto librería será "**TFC_Library**" el cual será donde tendremos las librerías y/o drivers que hemos desarrollado a lo largo del proyecto y se han explicado anteriormente.

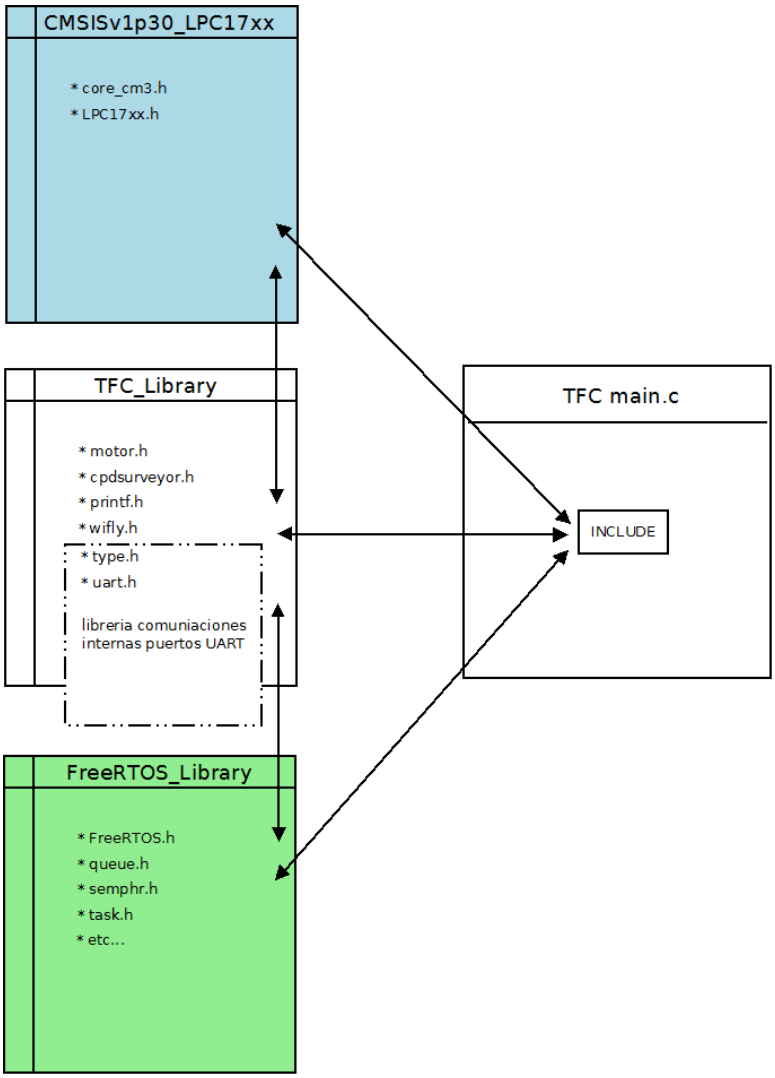


Figura 27: Estructura de la aplicación del sistema encastado

Otro proyecto será "**FreeRTOS_Library**", donde se encuentran las librerías de funciones propias del sistema operativo FreeRTOS. Dentro de esta carpeta y como se puede ver en el anterior diagrama, encontraremos librerías necesarias para el desarrollo del main.c, como por ejemplo:

"**task.h**": librería de obligado uso para la creación y uso de tareas en nuestro sistema.

"**semphr.h**" **:** librería de obligado uso para la creación de semáforos y funciones para trabajar con los mismos.

"**queue.h":** librería de obligado uso para crear y trabajar con "colas".

Una **cola** es una estructura de datos, caracterizada por una secuencia de elementos en la que la operación "push" se realiza por un extremo y la operación "pop" por el otro.

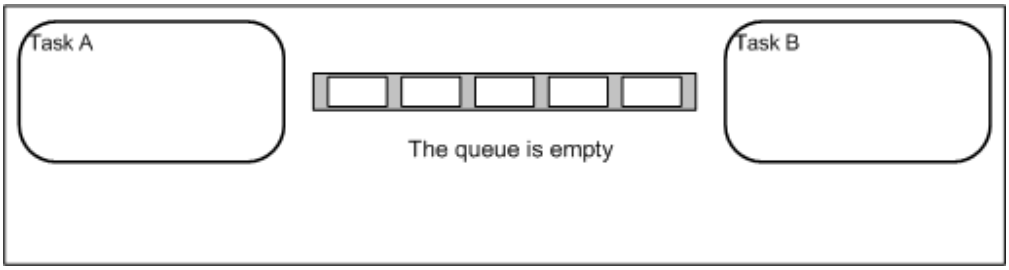


Figura 28: Dibujo de una cola vacía

Existen varias formas de crear tareas, semáforos y colas bajo FreeRTOS, al igual que muchas formas de programar una aplicación principal.

Nuestra aplicación como se ha comentado en otros capítulos parte de la fase de formación, concretamente de un sistema productor / consumidor sobre Arp@Lab, adaptada a nuestro sistema, es decir, se han incluido nuestros dispositivos con sus librerías y utilizado las funciones correspondientes.

Para finalizar, en las siguientes figuras podremos observar el diagrama de flujo de la **aplicación principal**, así como los diagramas de cada tarea (tarea "**Control**" y "**Status**"):

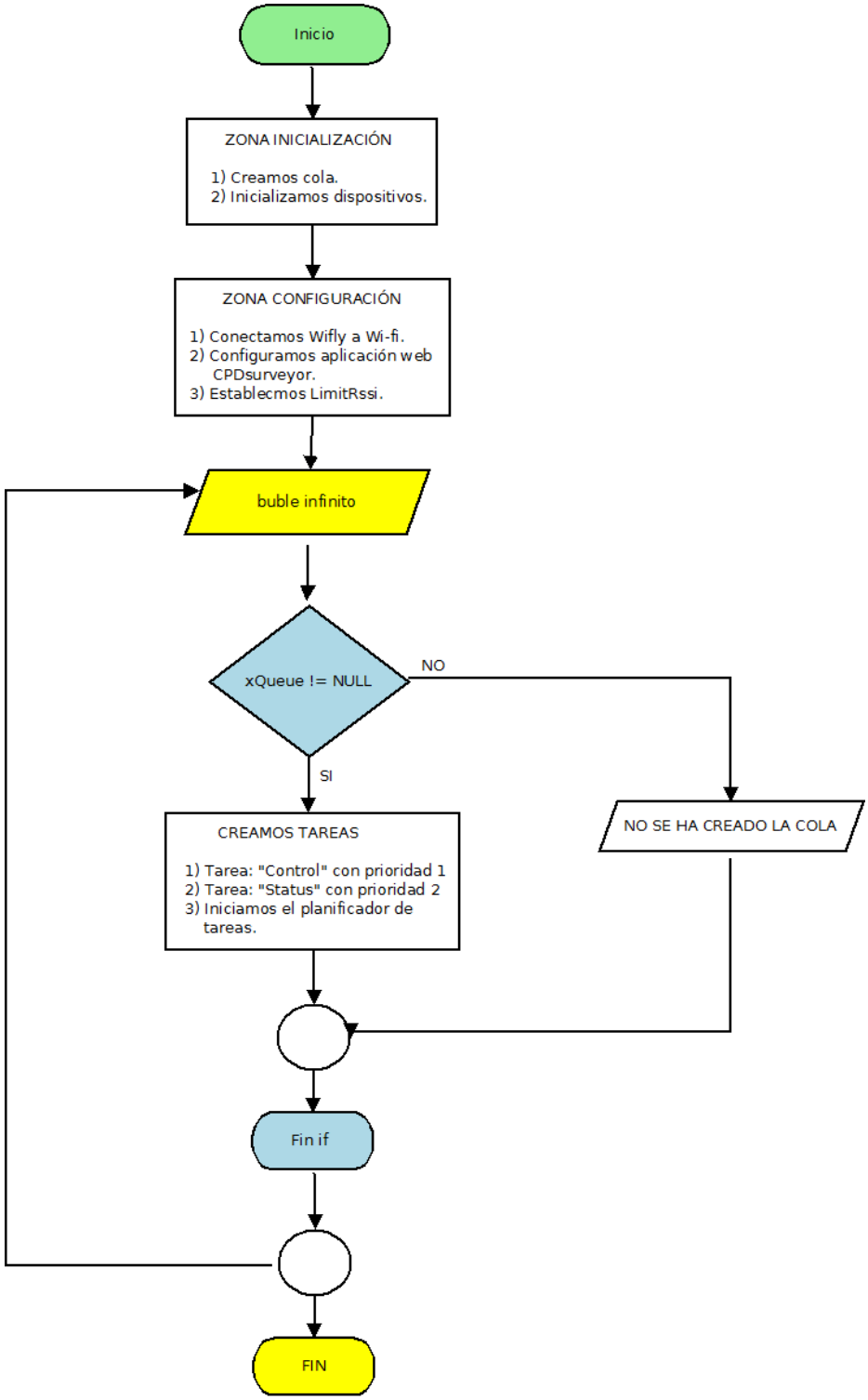


Figura 29: Diagrama de flujo de la aplicación principal “main.c”

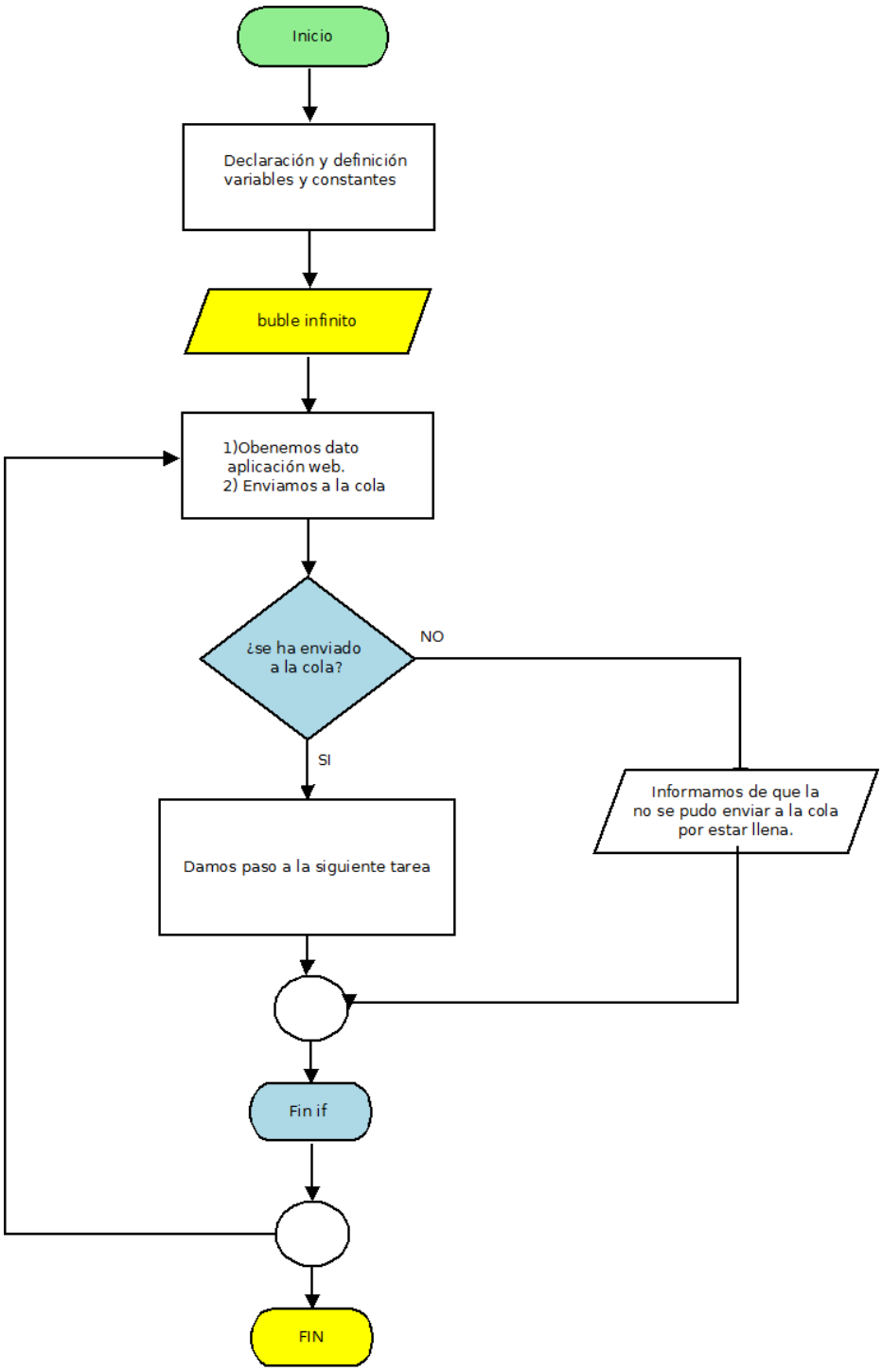


Figura 30: Diagrama de flujo de la tarea “Control”

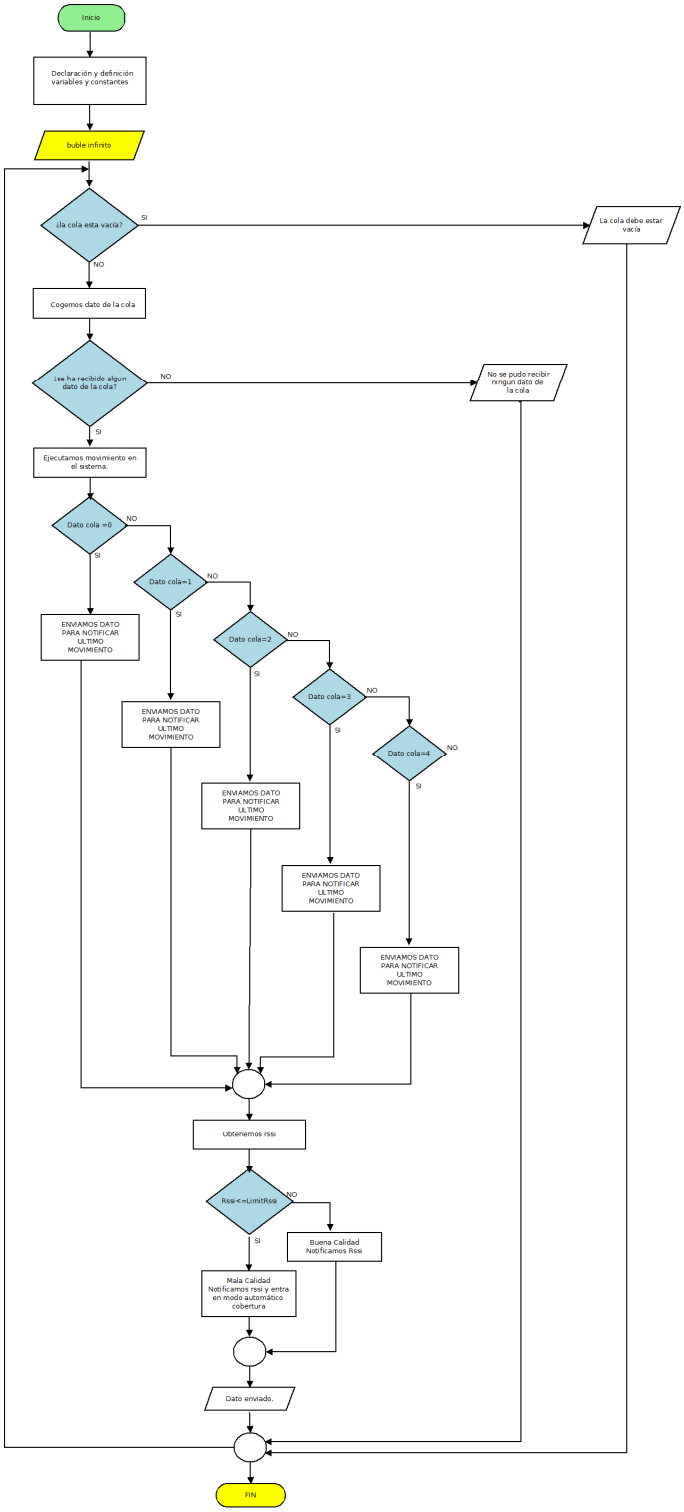


Figura 31: Diagrama de flujo de la tarea “Status”

5 Viabilidad técnica

Si se tuviera que evaluar en una escala entre el 1 y el 10 la viabilidad técnica de este proyecto, éste obtendría una nota de 6 o 7, es decir, estaría a las puertas de un notable.

Respecto a lo que en tecnología se refiere, hemos visto en los anteriores capítulos que ésta es vanguardista, siguiendo una clara línea Open Source, además de regirse de normas y protocolos muy estandarizados, lo que conlleva a numerosas fuentes de información.

Se ha demostrado como adaptar un nuevo objeto a la mota (motor serial driver), tanto a nivel hardware, como a nivel de aplicación, lo que nos puede dar una idea clara de cómo hacerlo con otros dispositivos y/o objetos, ya sean para ejercer una función de control a un objeto como para la monitorización de un parámetro u objeto.

Por último listaremos los puntos fuertes y los puntos débiles más destacados:

PUNTOS FUERTES

- Flexibilidad funcional.
- Posibilidad de conexión de diversos sensores, mecanismos de control, etc…
- Adaptación a aplicativos webs existentes.
- Ahorro de costes (por ejemplo licencias de software).
- Hardware bastante económico.
- Multitud de documentación y foros al ser Open Source.

PUNTOS DÉBILES A MEJORAR

- La autonomía energética y sistema de recarga autónoma de energía.
- Seguridad intrínseca del prototipo.
- Errores y falta de depuración del código. Mejora de código en su conjunto.
- Peso del prototipo y optimización de la parte mecánica.
- Varios puntos posible de fallo (servidor web, router wifi de acceso, etc..)

6 Valoración económica

A continuación se expondrá una **valoración económica orientativa** del coste del proyecto, incluyendo en el mismo, además del material empleado, los costes de desarrollo, instalación y mantenimiento del producto.

Con este presupuesto, se dispondrá de una **base de partida** en caso de comercialización del producto, ya que lo que se intenta reflejar son los gastos principales y las horas empleadas, incluyendo la fase de formación en el mismo.

DESCRIPCIÓN ITEM	CANTIDAD	PRECIO UNITARIO	PRECIO TOTAL
Mota LPC1769	1	102€	102€
RN-XV WiFly Module	1	27,80€	27,80€
Serial Controlled Motor Driver ROB-09571	1	15,87€	15,87€
CP2102 Serial Converter USB	1	10,82€	10,82€
DC Motor	2	1,55€	3,10€
Estructura y ruedas	1	20€	20€
Batería recargable de Lítio 5V	1	30€	30€
Horas desarrollo y programación	224	12€	2688€
Instalación y puesta en marcha	20	25€	500€
Soporte y mantenimiento anual	1	480€	480€
		COSTE TOTAL	**3.877,59€**

Tabla. 9: Presupuesto orientativo

7 Conclusiones

7.1 Conclusiones

Los requisitos mínimos del proyecto se han visto cumplidos, incluso algunos se han mejorado como veremos a continuación:

a) Conectar el sistema a redes Wireless 802.11b/g

Se ha conseguido gracias al dispositivo Wifly. Incluso se han creado otras funciones como por ejemplo cambiar la potencia de transmisión, etc.

b) Dotación al sistema de movimiento vía motor.

Gracias al dispositivo "motor serial driver" se ha conseguido mover en cuatro direcciones, en vez de dos (adelante y atrás). Se seguirá trabajando y perfeccionando el módulo.

c) Sistema bidireccional de comunicaciones.

Se ha dotado de un aplicación principal FreeRTOS que genera unas tareas obteniendo un sistema productor / consumidor con la aplicación web.

d) Dotación al sistema de un servicio de cobertura.

Se ha creado una función que cumple básicamente este objetivo. Se podría mejorar bastante dotando con un algoritmo más complejo.

e) Programación de una GUI para el control y supervisión.

Se ha creado una aplicación web básica que se utiliza no solamente para controlar el prototipo, si no para monitorizar estados.

f) Dotación al sistema de energía autónoma.

Se ha dotado de una batería recargable, aunque se debería de mejorar para ganar en autonomía y peso.

7.2 Propuesta de mejoras

Las **propuestas de mejoras** son bastante numerables, muchas de ellas se han citado en el capítulo 5. Cabe destacar, que la mayoría tienen su origen por la falta de tiempo, recurso que bien es cierto, se debe de emplear y estructurar adecuadamente.

Unas **mejoras añadidas** podrían ser sustituir los motores por unos de mayor voltaje o intentar disminuir el peso del prototipo para mayor agilidad, realizar una carcasa de seguridad, realizar una GUI más intuitiva y perfeccionada, etc.

7.3 Autoevaluación

Tal y como se cita en el resumen, este TFC en su globalidad ha cumplido con las expectativas de trabajar con muchas asignaturas estudiadas a lo largo de estos años.

En cuanto a sistemas embebidos se refiere, cumple con creces una primera toma de contacto, además de motivar al proyectista a seguir trabajando en estos sistemas.

Se ha aprendido como integrar dispositivos nuevos, analizar protocolos de comunicación, capacidad de adaptación a las necesidades del cliente, crear funciones específicas, y sobro todo, llevar lo planificado a la realidad.

8 Glosario

A

Arp@Lab: Aplicación web desarrollada para poder enviar datos y poderlos representar gráficamente. También es posible capturar datos aleatorios.

ADC: Conversión analógica digital.

AJAX: Técnica de desarrollo web para crear aplicaciones interactivas.

B

Bluetooth: Es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos.

Baudrate: Número de unidades de señal por segundo.

Buffer: Espacio de memoria.

BSSID: Nombre de identificación único de todos los paquetes de una red inalámbrica (Wi-Fi).

C

Cortex-M3: procesador de 32 bits de ARM.

CPU: Unidad central de procesamiento.

D

DHCP: protocolo de configuración dinámica de host.

Diagramas de Gantt: herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado.

Distro: Distribuciones de GNU/Linux.

E

E/S: Entrada/Salida.

F

FreeRTOS: Sistema operativo en tiempo real para dispositivo embebidos basado en Lenguaje C.

G

GUI: Interface gráfica de usuario.

GPIO: Puerta programable de entrada y salida de datos.

H

HTML: Siglas de HyperText Markup Language.

I

IT: Tecnologías de la información y la comunicación.

IDE: Entorno de desarrollo integrado.

IP: Una Dirección IP es una etiqueta numérica que identifica, de manera lógica y jerárquica, a un interfaz de un dispositivo dentro de una red que utilice el protocolo IP.

I2C: Bus de comunicaciones en serie.

J

Jaula: Espacio de data center en forma de jaula.

L

Lan: Red de área local.

M

Mota: o mote es una placa con procesador y transmisión/recepción vía radio.

Microcontrolador: circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria.

Mysql: Sistema de gestión de bases de datos relacional, y multiusuario.

O

Open Source: Es el término con el que se conoce al software distribuido y desarrollado libremente.

P

Parser: Procesador de cualquier lenguaje.

PHP: Lenguaje de programación de uso general de script del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico.

R

RSSI: Indicador de fuerza de señal de recepción.

T

Telecontrol: envío de indicaciones a distancia mediante un enlace de transmisión.

TCP/IP: conjunto de protocolos de red en los que se basa Internet y que permiten la transmisión de datos entre computadoras.

U

UART: Transmisor-Receptor Asíncrono Universal

W

Wifi: Mecanismo de conexión de dispositivos electrónicos de forma inalámbrica.

Wireless: Comunicación inalámbrica o sin cables.

9 Bibliografía

Bibliografía:

- "Using the FreeRTOS Real Time Kernel - A Practical Guide NXP LPC17xx Edition" y "FreeRTOS Reference Manual" de Richard Barry.
- "Enciclopedia del lenguaje C++ estandar" de Fco. Javier Ceballos. Ediciones Ra-Ma.
- "Sistemes Encastats" material web UOC.

Recursos Web:

- Wiki de la asignatura Sistemas Encastados de la UOC: http://cv.uoc.edu/app/mediawiki14/wiki/IniciCortexM3
- Arp@Lab: http://arpalab.appspot.com/
- Wikipedia: http://es.wikipedia.org/wiki/Sistema_embebido
- Web oficial FreeRTOS:
- http://www.freertos.org
- Ejemplos FreeRTOS: http://www.freertos.org/Documentation/code/source-co de-for-LPC17xx-editionexamples.zip
- Foro mbed.org: http://mbed.org/forum/
- Datasheet LPC1769:
- http://www.nxp.com/documents/data_sheet/LPC1769_68_67_66_65_64_63.pdf
- Sistemas embebidos UPV: http://server-die.alc.upv.es/asignaturas/PAEEES/2005-06/A07%20-%20Sistemas%20Embebidos.pdf
- PHP web oficial: http://php.net/
- Mysql web oficial: http://www.mysql.com/
- Documentación WiFly RN-XV: http://rovingnetworks.com/products/RN_XV
- Documentación TMP102 (Digital Temperature Sensor Breakout): https://www.sparkfun.com/products/9418
- I-Racer: https://www.sparkfun.com/products/11162

10 Anexos

10.1 Manual IDE LPCExpresso

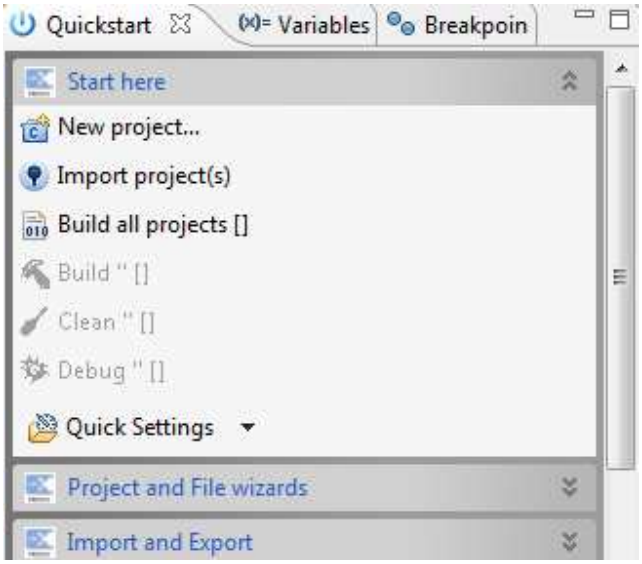
Para la instalación y familiarización con el IDE, existe un manual aportado por el fabricante con imágenes ilustrativas e intuitivas, por lo que no se duplicará información dejando el siguiente link de referencia:

http://www.nxp.com/documents/other/LPCXpresso_Getting_Started_Guide.pdf

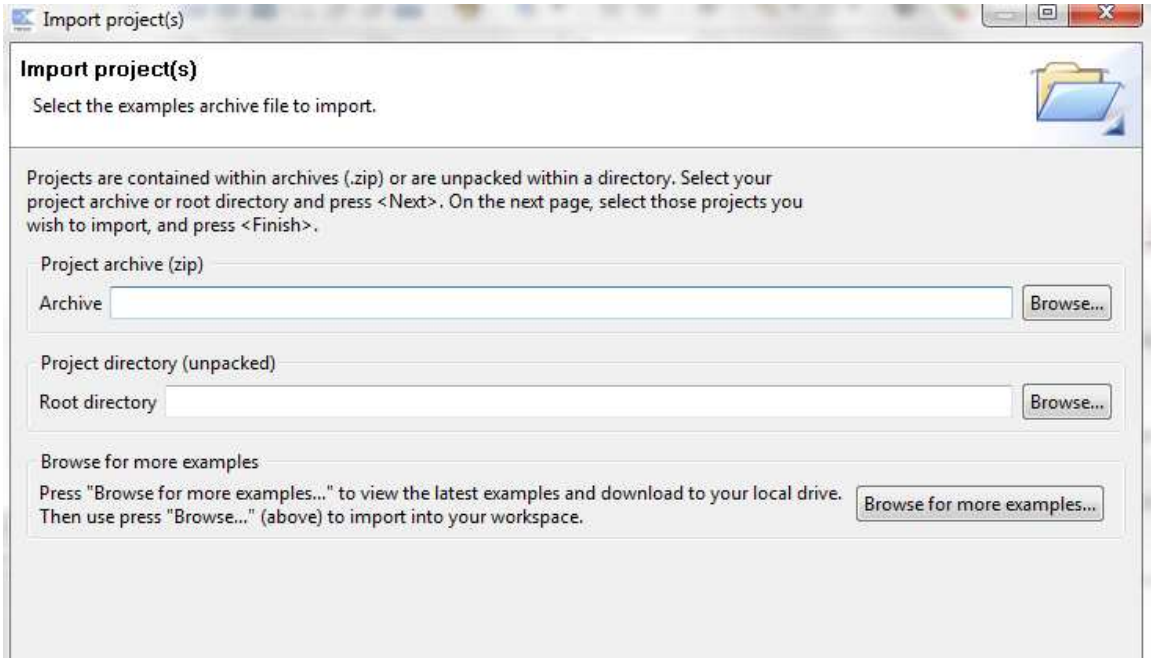
10.2 Ejecución y compilación código

Los pasos a seguir para la compilación y ejecución del código son los siguientes:

1) Importamos el “workspace” proyecto pulsando sobre “import project”



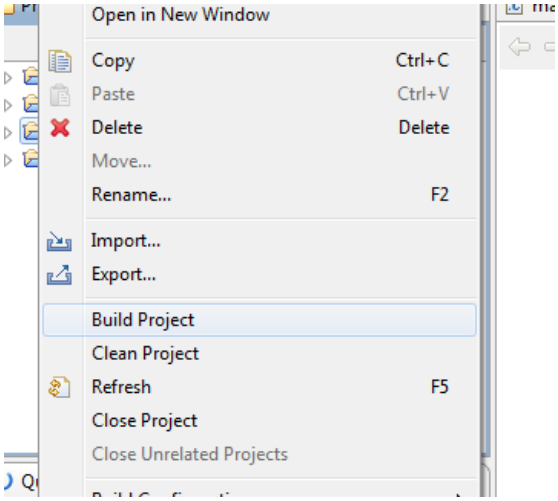
2) Seleccionamos el .zip con el código:



3) Y por último pulsamos Finish.

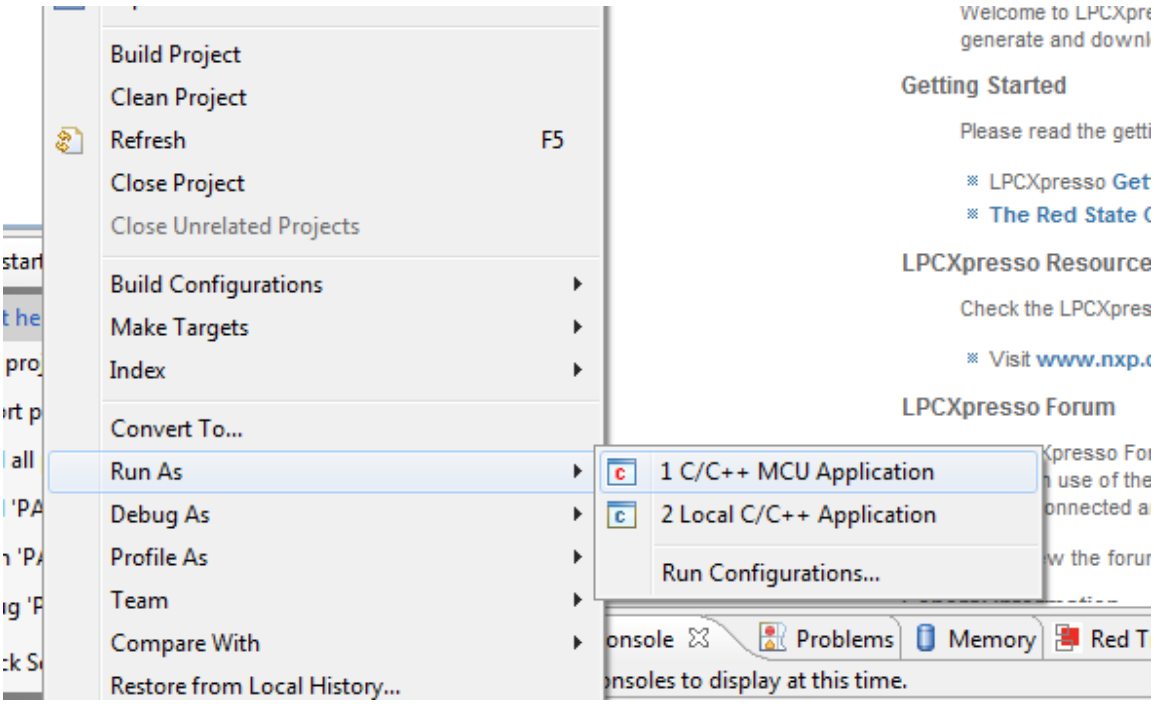
Compilación:

Una vez tenemos el código en nuestro workspace, procedemos a la compilación pulsando con el botón derecho sobre el proyecto y seleccionando "**Build Project**"



Ejecución:

Para ejecutar el código, procediendo de la misma forma que en la compilación, pero esta vez seleccionaremos "**Run As -> C/C++ MCU Application**":



10.3 Instalación del GUI en un servidor

Para la instalación y configuración de la aplicación web, será necesario de disponer un hosting con soporte PHP y MySQL.

Se podría instalar también en local, con un sistema tipo XAMPP (distribución Apache que contiene MySQL, PHP y Perl, disponible para PC, MAC y Linux).

1) Crear una base de datos e importar la base de datos de ejemplo (cpdsurveyor.sql).
2) Editar el archivo conexionmysql.php con los datos correspondientes a la base de datos creada.

10.4 Vídeo demostrativo del prototipo y capturas de pantalla sistema log

www.youtube.com/watch?v=-JgNotJkK5Y