HOW
Universidad Zaragoza

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
Y COMUNICACIONES, ÁREA DE
TECNOLOGÍA ELECTRÓNICA

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

PROYECTO FIN DE CARRERA

Z-mota Move 2.0.

INTERFAZ BASADO EN LA MONITORIZACIÓN DEL MOVIMIENTO E IDENTIFICACIÓN POR RADIOFRECUENCIA

AUTOR: EDUARDO MOLINA TOMÁS
DIRECTOR: RUBÉN BLASCO MARÍN
PONENTE: ANTONIO BONO NUEZ

INGENIERÍA TÉCNICA INDUSTRIAL, ESPECIALIDAD ELECTRÓNICA INDUSTRIAL

CURSO 2011 - 2012

RESUMEN:

Proyecto que aborda la realización de un dispositivo de avanzado de control que, mediante sensores inerciales y un lector RFID (Radio Frequency IDentification), sea capaz de aportar información sobre el movimiento de la persona y del contexto que la rodea.

CAPITULO 1. INTRODUCCIÓN

1.1 INTRODUCCIÓN

La rápida evolución y la proliferación de sistemas informáticos de todo tipo, desde la llegada de los ordenadores personales hasta la actualidad, ha traído consigo la necesidad de construir métodos de control más sofisticados de interacción con los mismos. Se precisa que sean cada vez mas intuitivos, confortables, y capaces de aprovechar la potencia que ofrecen dichos sistemas.

Este hecho, que en los últimos años ha sido más patente de cara a la población general dentro del ámbito del ocio electrónico, como es el caso de la salida al mercado de los últimos controles inerciales para videojuegos, o su inclusión en teléfonos móviles, no está ligado únicamente a dichas aplicaciones. El desarrollo de controles inerciales puede ofrecer innumerables nuevas posibilidades a la industria, la medicina, la educación, el deporte, a tecnologías asistenciales, de apoyo a personas con discapacidad, y un largo etc.

Bajo este precepto nace la idea de realizar un dispositivo avanzado de control que, mediante sensores inerciales y lectores RFID (Radio Frequency IDentification), sea capaz de aportar información sobre el movimiento de la persona y del contexto que la rodea.

1.1 MARCO DE DESARROLLO DEL PROYECTO

Z-Mota Move es un proyecto que pertenece, junto a otros subproyectos abiertos enfocados a diversas aplicaciones, al Z-Project. La base común del Z-Project es la utilización el estándar Zigbee Pro, y categoriza sus aplicaciones como Z-Motas (para pequeños dispositivos), Z-Litas (para dispositivos de enrutamiento), Z-Petas (para dispositivos que proporcionan el control de cargas tales las luces, motores, u otros sensores con conexión a la red eléctrica).

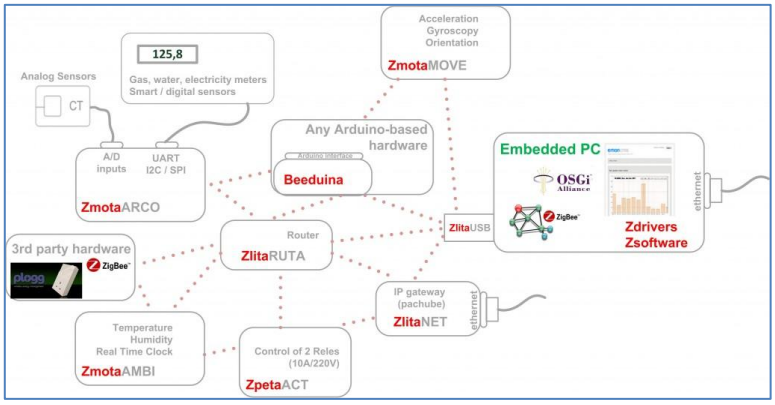


FIGURA 1.1 – Clasificación de dispositivos Z-Project (openlab.unizar.es)

El primer producto desarrollado bajo la definición Z-Mota MOVE v.1.0 (Balaitus,) es un dispositivo con forma de pulsera, capaz de capturar el movimiento de la mano, contando además con un sensor de temperatura.

Z-Mota MOVE v.2.0 continua con el planteamiento de su predecesor y lo amplia, mejorando la precisión de la captura de movimientos mediante la implementación de un giróscopo y un magnetómetro. Además, incluye un modulo RFID como nueva funcionalidad añadida, lo que permitirá el desarrollo de nuevos interfaces inerciales más completos, dada la posibilidad de detección objetos y del propio entorno.

Este proyecto ha sido realizado en colaboración con el grupo de investigación HOWLab de la Universidad de Zaragoza el cual forma parte del Instituto de Investigación en Ingeniería de Aragón (I3A). Está compuesto por un grupo interdisciplinar que abarca perfiles tan diferentes como industrial, telecomunicaciones, informática, diseño o ciencias sociales. Su objetivo principal es la mejora de la calidad de vida de las personas y su entorno mediante el desarrollo de nuevas tecnologías.

Como se ha indicado, los dispositivos Z-Project tienen una clara orientación al hardware libre, estando su información disponible al alcance de cualquiera a través del portal ***Openlab.unizar.es***. La plataforma Openlab se basa principalmente en dos propósitos. Servir como repositorio de proyectos de investigación desarrollados en la Universidad (en principio desde el grupo HOWLab) para su libre distribución bajo licencias Creative Commons, además de promover dichos proyectos fuera del ámbito de la comunidad universitaria.

1.2 OBJETIVOS

El objetivo principal de este P.F.C. es la realización de un dispositivo capaz de capturar el movimiento mediante sensores inerciales y que además sea capaz de capturar información de los distintos objetos presentes en el entorno mediante un lector RFID

Para cumplir este objetivo se plantean los siguientes objetivos parciales:

- La realización de un estudio previo del estado de la tecnología existente en este campo de aplicación.
- Determinar las especificaciones funcionales que ha de cumplir este diseño en base al estado del arte.
- La selección de todos los componentes que conformaran el hardware.
- El diseño de esquema electrónico para dichos componentes.
- El diseño de la placa de circuito impreso (**P**rinted **C**ircuit **B**oard, PCB) que sirva como soporte a los citados componentes.
- Montaje y verificación de la PCB
- La programación de las librerías de control necesarias para la integración de dichos componentes con el diseño de firmware final.
- La realización de una aplicación firmware que permita la comprobación de la integración y la funcionalidad de dichos elementos.

1.3 PLANIFICACIÓN

Para la realización de este proyecto se han empleado aproximadamente 1200h. En el siguiente diagrama de Gantt (ver Figura [1.3]) se detalla cómo se ha distribuido el tiempo en base a las tareas realizadas.

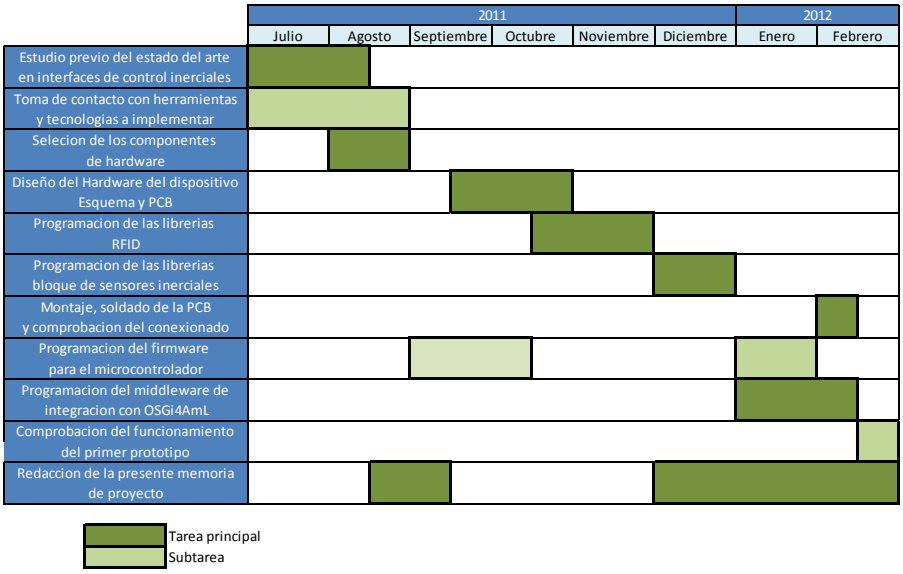


FIGURA 1.3 – Estimación de la distribución del tiempo dedicado

1.4 ESTRUCTURA DEL PRESENTE DOCUMENTO

A continuación se expone un pequeño resumen de la ordenación y el contenido de las distintas secciones que componen esta memoria:

En el capitulo uno se presenta el planteamiento inicial del proyecto así como el marco en el cual se realiza, relatando tanto su objetivo final, como los objetivos parciales que deberán ser realizados para alcanzar a este ultimo.

En el capítulo dos se describen los conceptos empleados para estudio de sistemas de interacción hombre-maquina, y como el desarrollo de los mismos a permitido realizar una mejora de los interfaces de control. También se realiza un recorrido a través de la evolución de dichos dispositivos desde la incursión de los primeros entornos gráficos hasta a la actualidad, relatando la razón de su origen, valorando su repercusión y resaltando las características principios de funcionamiento. Por último se establece un paralelismo entre todos ellos y se plantean los requisitos mínimos que debe cumplir un dispositivo de control para ser competitivo en la actualidad.

En tercer capítulo se relata la forma de abordar los requisitos citados anteriormente. Se plantea una estructura general del hardware y la selección de los componentes más adecuados para este fin, junto con la razón de su elección y sus características técnicas. Para ello se exponen las diferentes alternativas existentes tanto en tecnologías como en los componentes cuando sea necesario para una mayor comprensión de las soluciones adoptadas. Concluyendo con un

resumen de las consideraciones finales más destacables junto con un diagrama representativo de las mismas.

En el cuarto capítulo se dispone una descripción del proceso diseño del software realizado en este proyecto. Planteando de forma estructurada y a través de diferentes perspectivas los requerimientos para después relatar la forma de abordar estos.

El quinto capítulo aborda la elaboración del diseño de la PCB que da soporte a los diferentes componentes seleccionados, describiendo los aspectos más representativos de ese proceso, como son los principales criterios a seguir, así como las consideraciones particulares derivadas de las características de este proyecto en concreto.

Las conclusiones y una valoración del resultado obtenido son contempladas en el sexto capítulo. Señalando el cumplimiento de los objetivos alcanzados y las posibilidades de continuación del proyecto así como su ampliación y mejora. También se hará incidencia sobre los inconvenientes acaecidos durante la realización del proyecto y la experiencia tanto profesional como personal.

Se proporcionan al final de esta memoria, en las secciones de Anexos, información complementaria que amplía el detalle y comprensión de los distintos capítulos desarrollados, así como las referencias bibliográficas empleadas consultadas para la realización de este proyecto.

CAPITULO 2. INTRODUCCION A LOS INTERFACES INERCIALES

2.1 FUNDAMENTOS DE UNA INTERFAZ HOMBRE-MAQUINA

Una interfaz es el medio bidireccional a través del cual el usuario se comunica con la maquina. El intercambio de información desencadena como resultado, la realización de un trabajo por parte de la maquina y/o un estimulo en respuesta que pueda ser interpretado por el usuario. Este concepto puede ser aplicado a maquinas o herramientas de cualquier tipo, aunque en la actualidad y generalmente se encuentra más enfocado a entornos informáticos.

FIGURA 2.1 - Interfaz háptico aplicado a entornos virtuales (www.neoteo.com/haptics-tocando-lo-virtual)

Para que se lleve a cabo esa interacción correctamente, han de superarse ciertas barreras tales como: capacidad de memoria, medio de transferencia de datos, y lógica de la maquina. Además de otras conceptuales como son la comprensión del entorno y de las tareas por parte de ésta, debido a que tanto usuario como maquina difieren en su estructura, en el uso de un lenguaje distinto y en una diferente percepción del entorno.

La interfaz como medio está constituida por una serie de dispositivos físicos (hardware) y lógicos (software), atendiendo a criterios como simplicidad, fiabilidad, seguridad, comodidad y eficiencia.

Mayor simplicidad implica una relación menor de esfuerzo por parte del usuario, por adaptar su propio lenguaje al que es comprensible por la maquina. En este caso la interfaz es más intuitiva.

En relación a los otros criterios expuestos, también se busca que la interfaz sea fiable y segura, de modo que la transmisión de información sea recibida correctamente tanto por usuario como por la máquina de forma íntegra. Es decir, que no sufra interrupciones o se degrade.

En última instancia también debe ser cómoda y eficiente, de manera que no se demande una cantidad de recursos elevada del usuario al actuar sobre ella (esfuerzo en este caso). Ni tampoco de los requeridos por la propia interfaz para llevar a cabo la función para la cual está diseñada, es decir, su consumo de energía.

La disciplina llamada HCI (**H**uman-**C**omputer **I**nteraction [1]) o IPO (**I**nteracción **P**ersona-**O**rdenador) surge a partir de los años 60, orientada a una mejora continua de estas características, englobando áreas tales como psicología, sociología, lingüística, diseño industrial e informática. Por tanto, atiende a múltiples factores que repercuten en varios aspectos referentes a la propia persona.

Su salud, ya que un mal diseño de un interfaz puede producir desde tensión muscular a dolor de cabeza, o incluso estrés. No contemplar en su justa medida ciertas pautas en cuanto a lo que se refiere a ergonomía se traducirá finalmente en una mayor incomodidad y una mayor insatisfacción por parte del usuario.

El factor psicológico, profundizando en los procesos cognitivos del usuario durante su experiencia en el uso de interfaces y maquinas con características similares a otras ya empleadas. Siendo su principal objetivo que el usuario se sienta familiarizado desde una primera toma de contacto con la interfaz, reduciendo al máximo el proceso de adaptación por parte de éste en tiempo y esfuerzo.

En beneficio de un mejor diseño de interfaz, más sencilla, intuitiva, cómoda, se hace necesario un fuerte hincapié en la mejora de los dispositivos de entrada y salida, prestando un gran interés a las estructuras de dialogo empleadas para la comunicación.

En cuanto a la salida de datos, el principal objetivo es potenciar el factor multimedia. Medios para tal fin son el empleo de colores, imágenes sonidos, o aplicando técnicas que incluso alcancen a otros sentidos como el tacto, en el desarrollo de los interfaces hápticos.

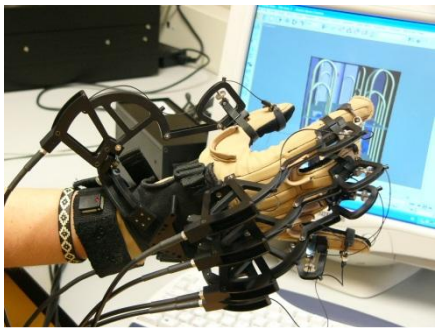


FIGURA 2.2 – Cybergrasp, el único guante háptico disponible comercialmente (dac.escet.urjc.es)

Respecto a la entrada de información, la posibilidad de que el usuario interactué de la manera más natural posible. Con sus propios movimientos corporales empleando tecnologías inerciales y de captura de movimiento. Empleando su voz sin tener que adaptar su propio lenguaje, con el uso de técnicas de reconocimiento del habla.

En un futuro, la consecución de los últimos pasos de la HCI nos permitiría la completa materialización de conceptos tales como la realidad virtual y la realidad aumentada, de los cuales existen ya algunas muestras actualmente.

2.2 ESTADO DEL ARTE EN INTERFACES INERCIALES

2.2.1 LOS PRIMEROS PASOS

El camino que dista hasta la consecución de los objetivos de la HCI anteriormente citados es bastante largo. El estudio de la interacción hombre-maquina ha avanzado notablemente en este campo desde la definición del concepto de informática tal y como es conocido actualmente.

Las vías tomadas para la superación de las barreras existentes en cuanto a una comunicación más natural con los computadores han experimentado fuertes avances, alcanzando mayor sofisticación y viendo ampliada la variedad de opciones disponibles.

Si realizamos una retrospectiva, el primer dispositivo enfocado a la interacción del usuario en entornos computacionales mediante el uso de movimientos, fue el “X-Y Position Indicator for a Display System”, posteriormente bautizado como “Mouse” o ratón de ordenador.

FIGURA 2.3 - Detalle interno de un "mouse" mecánico (http://es.wikipedia.org)

Desarrollado en el Stanford Research Institute por Douglas Engelbart y William **“**Bill” English durante los años 60, no apareció comercialmente hasta el año 1981 con la salida al mercado de ordenador personal de la compañía Xerox PARC, Xerox Star 8010. Sin embargo, el uso de este periférico esta enormemente extendido y su estructura ha permanecido casi inalterada hasta la actualidad, variando únicamente su composición interna y elementos empleados para su fabricación.

El mouse opera detectando los movimientos producidos al desplazarse este sobre una superficie plana en dos ejes y reproduciéndolos en la pantalla. La captación de estos desplazamientos ha variado en cuanto a las técnicas empleadas, basándose en un principio en medios mecánicos y ópticos para más tarde tan solo necesitar de la implementación de estos últimos.

A nivel de hardware, los primeros ratones mecánicos basan su funcionamiento en la transmisión de los movimientos ejercidos sobre la superficie a unos rodillos circulares a través del giro de una bola solidaria a estos. Dichos rodillos poseen un disco con aperturas que en su giro impiden o permiten el paso de un haz de luz de un LED a un fotodiodo según una codificación (binaria o código Gray). Con dicha información es posible conocer el ángulo de giro de los discos rotatorios codificados, y por ende tener una estimación de la dirección de movimiento del ratón y de su velocidad, para ser posteriormente transmitida a la aplicación.

En la actualidad, los llamados ratones ópticos analizan la superficie en la cual son desplazados mediante la radiación reflejada en esta, proveniente de un emisor LED, que posteriormente es captada por un dispositivo de adquisición de imagen y evaluada por un procesador digital de señal.

2.2.2 LOS ENTORNOS GRAFICOS COMO ESTANDAR

La tendencia basada en interfaces graficas o GUIs, WIMP (**W**indows, **I**cons, **M**enus, **P**ointer [2]), causó una verdadera revolución en cuanto a la interacción hombre-maquina se refiere, permitiendo que los ordenadores llegasen a ser mucho más accesibles y manejables para el público general, lo que a su vez permitió su rápida proliferación y desarrollo posterior.

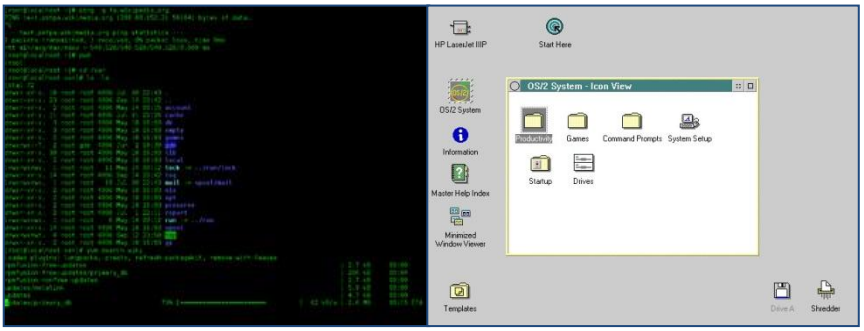


FIGURA 2.4 - Interprete de comandos frente a un entorno grafico

Los entornos gráficos sustituyeron casi por completo a los intérpretes de comandos, surgiendo nuevas aplicaciones orientadas específicamente a ellos, que permitiría a su vez la aparición de nuevas y más intuitivas interfaces electrónicas táctiles.

Desde su invención, por el Dr. Samuel C. Hurst en 1971, esta tecnología ha seguido manteniéndose hasta la actualidad, empleándose en multitud de dispositivos de uso cotidiano. Pantallas táctiles de dispositivos móviles de última generación, touchpad incluidos como método de control en ordenadores portátiles, tabletas digitalizadoras que emplean diseñadores gráficos y artistas para reproducir los trazos de sus obras a mano alzada en el computador, por citar sólo algunos ejemplos.

Existen principalmente dos tipos de interfaces táctiles atendiendo a su tecnología de fabricación: capacitivas y resistivas.

Un panel táctil resistivo está compuesto por dos láminas rígidas solapadas una sobre otra, con una capa resistiva en sus caras internas. Los lados opuestos de las láminas disponen de contactos para acceder a una conexión externa, que por lo general estará comunicada a una tensión de alimentación por un extremo y a masa por la terminación opuesta.

El procedimiento para determinar las coordenadas de la posición del panel que ha sido presionada consta de dos pasos, que son la detección de las dos coordenadas X e Y por separado. Para cada una de ellas, presionando sobre las láminas resistivas se realiza un contacto de ese punto a masa, lo que es similar a obtener un divisor de tensión variable dependiente del dicho punto de presión en el panel táctil. El valor de la tensión obtenida en el divisor puede ser traducido a la distancia en el eje correspondiente, respecto del origen de coordenadas.

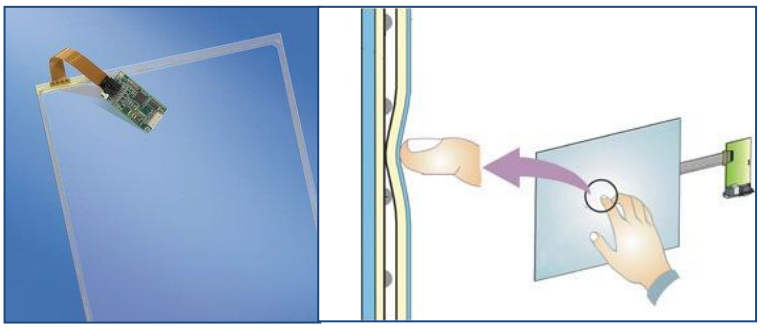


FIGURA 2.5 - Detalle de una pantalla táctil resistiva (www.micmovilidad.es/noticias/Paginas/091209_PantallasTactiles.aspx)

Los paneles capacitivos por otro lado, constan de una capa cubierta habitualmente con óxido de indio y estaño, que es capaz de conducir una corriente eléctrica continua a través de sí misma. Esta capa puede considerarse un sensor, que muestrea un campo de electrones controlado con precisión tanto en el eje vertical como en el horizontal, es decir, adquiere carga como si de un condensador se tratase. Cuando el campo eléctrico de dicho sensor es alterado de su estado de referencia por un elemento con carga, como puede ser el dedo de una persona, se mide esta distorsión y se procesa, permitiendo así determinar las coordenadas de la pulsación.

El último hito en cuanto a interfaces táctiles fue alcanzado por Microsoft con la presentación de su prototipo de ordenador multitáctil *Surface*. El concepto que se empezó a desarrollar a partir de 2001 por Steven Bathiche y Andy Wilson, es una computadora capaz de ser manejada únicamente gracias a la tecnología multitactil por varios usuarios al mismo tiempo, prescindiendo completamente de teclado y ratón.

Cuenta con una estructura basada en una disposición de pantalla horizontal, destinada a su colocación en mesas a modo de escritorio virtual, y que permite entre otras funciones el reconocimiento de dispositivos externos como móviles, cámaras de fotos, reproductores de música, etc. Además de otras adicionales, como el escaneo de documentos y/o fotografías, siendo su manejo muy intuitivo y sencillo.

Su funcionamiento se basa en las interfaces táctiles capacitivas comentadas anteriormente, en conjunción con varias cámaras capaces de procesar la imagen de los objetos en contacto con la superficie, y un proyector que mostraría el entorno grafico sobre esta.

2.2.3 MÁS GRADOS DE LIBERTAD

En la actualidad han aparecido nuevos dispositivos de control para interfaces basados en movimiento. La mayor parte de ellos destinados a un entorno lúdico con planteamientos ligeramente diferentes y más novedosos. Permitiendo no sólo desplazamientos en superficies planas, sino la interacción del usuario con entornos tridimensionales, además de otras funcionalidades mas complejas.

Un claro ejemplo de esto es el guante de datos **[3]**. Este periférico equipado con sensores electrónicos actúa como dispositivo de entrada, transmitiendo la información de los movimientos de la mano y dedos del usuario con múltiples grados de libertad. Está enfocado sobre todo en el ámbito de la realidad virtual.

En 1983, el Dr. G. Grimes desarrolló el primer guante de datos en los Laboratorios Bell de AT&T, en EE.UU. Denominado "Digital Data Entry Glove", estaba equipado con sensores de flexión en los dedos, sensores táctiles en las yemas y sensores de posición espacial.

Tres años después, la empresa americana VPL comercializaba *VPL Data Glove*, el primer guante de datos comercial, que había sido desarrollado por encargo de la NASA. En él los movimientos y flexiones de los dedos eran medidos por cables de fibra óptica en el dorso de la mano, con dos cables para cada dedo, excepto el pulgar. Mediante un diodo luminoso se enviaba luz a través de los cables de fibra, que se encargaba de medir una fotocélula en el otro extremo. Un ordenador convertía los impulsos eléctricos así generados en los movimientos visualizados de la mano.

Tras adquirir la patente, el fabricante de juguetes Mattel presentaba en el año 1989 el primero de estos métodos de control inercial destinado al público general. *Power Glove*, ideado por Grant Goddard y Samuel Cooper Davis en el departamento de investigación Abrams Gentile Entertainment (AGE), fue comercializado con licencia de Nintendo, como un accesorio para su consola de videojuegos *Nintendo Entertainment System*.

FIGURA 2.6 - NINTENDO Power Glove (http://en.wikipedia.org/)

Este permitía al jugador el realizar las mismas acciones que un mando de control tradicional, pero con los movimientos de su mano. Para ello contaba con galgas extensiometricas de carbón en los dedos, que con su compresión al cerrar la mano transmitían la misma señal que la de la pulsación de un botón de acción. Para la detección de los desplazamientos contaba con unos altavoces ultrasónicos que transmitían series de pulsos intermitentes a unos micrófonos receptores que eran colocados alrededor del monitor. Sin embargo no era capaz de detectar movimientos en el eje transversal y contaba con una precisión muy inferior al *DataGlove*.

Aunque no consiguió un gran éxito comercial, este tipo de dispositivos aún poseen demanda en el mercado actualmente. Ejemplos de ello son la empresa Essential Reality, la cual mejoró y amplió las funciones del *Power Glove* presentando el *P5 Glove* con interfaz USB en octubre de 2002. O el más reciente *CyberGlove II* de la compañía Immersion aparecido en 2005, con conexión inalámbrica a través de Bluetooth.

En la década de los 90 la compañía SEGA también intentó introducir en el mercado con menor fortuna un método diferente de control inercial. Su dispositivo *Activator* **[4]**, ideado por Assaf Gurner aparecido en 1993 y pese a su original planteamiento, no consiguió ofrecer una interacción suficientemente simple, dinámica e intuitiva como para triunfar significativamente en ventas. El funcionamiento de este dispositivo se basó en el reconocimiento de los movimientos del usuario por medio de la reflexión de la radiación

infrarroja, proyectada en distintos planos perpendiculares desde una plataforma octogonal ubicada en el suelo, con posibilidad de detección del cruce del cuerpo del usuario con el haz a varias alturas.

FIGURA 2.7 - SEGA Activator

El concepto de control inercial mediante interacción con un dispositivo tangible, tardaría en ser retomado hasta 2006, año en el que la compañía Nintendo lanzo al mercado su nueva plataforma de hardware para videojuegos con un nuevo periférico de este tipo como método principal de control. *Wiimote* (o *Wii Rimokon* en Japón) desarrollado en colaboración con Gyration Inc., pretendía introducir de nuevo un método de control inercial en el mercado del ocio electrónico. Su forma externa era semejante a la de un método de control tradicional (Control PAD), con la salvedad de sensar también sus desplazamientos y transmitirlos al entorno de juego.

En cuanto a su funcionamiento interno, mediante el uso de un acelerómetro ADXL330, Wiimote es capaz de detectar los movimientos realizados por el brazo del usuario en tres ejes. Un sensor óptico PixArt, también detecta los haces infrarrojos producidos por diez LEDs incorporados a un elemento llamado barra sensora, permitiendo usar el Wiimote también como dispositivo de señalamiento preciso en la pantalla, a modo de puntero. Por último, en una ampliación posterior, se incorporo una mayor precisión en la medida de movimientos y posición del mando, mediante el añadido de sensor magnético en el pack de expansión *Wiimotion Plus*.

FIGURA 2.8 - Detalle de Nintendo Wiimote (http://hackawii.com/wii-eeprom-hacking/)

Destinado al ámbito del ocio electrónico, se incluyeron como novedad componentes que proporcionan una mayor sensación de inmersión: Un motor de vibración, y un pequeño altavoz proporcionan una respuesta o feedback dependiendo de las situaciones que se suceden dentro del juego.

Pese a su fuerte orientación lúdica, se han estudiado posibles aplicaciones de este control enfocadas a entornos académicos. Las principales razones que han decantado a los investigadores a su empleo son su bajo coste principalmente, su accesibilidad en cuanto a la realización de unos drivers compatibles para su conexión con PCs u otras plataformas de forma inalámbrica mediante Bluetooth, y la posibilidad de uso de su memoria EEPROM interna.

Los campos de aplicación en los que se ha valorado emplear este sistema de control o modificaciones de éste son los entornos domóticos (tanto con o sin el uso de tecnología de realidad aumentada), el control de robots (**H**uman **R**obot **I**nteraction o HRI), aplicaciones musicales, u otras que potencian el uso de su capacidad de detección de luz infrarroja.

2.2.4 RECREACION DE MOVIMIENTOS A TRAVES DE LA IMAGEN

Los últimos interfaces inerciales están marcados fuertemente por la fotogrametría. Una técnica por la cual es posible determinar las propiedades geométricas de los objetos y las situaciones espaciales de seres vivos a partir de imágenes fotográficas.

Sus aplicaciones son numerosas: cartografía, arquitectura, biomecánica en diversos campos como la medicina, ergonomía o deporte, investigación policial (reconstrucción de accidentes o reconocimiento facial), realización de audiovisuales (cine y efectos especiales), y un largo etc.

En cuanto a control inercial enfocado al mercado del público general, podemos destacar la plataforma *Kinect*, aparecida en el reciente 2010. Ésta traslada parte de estos avances en el procesado de imagen al ámbito del ocio electrónico.

El periférico *Kinect,* desarrollado por Microsoft Research, se presenta como alternativa a los métodos de control inerciales desarrollados por Sony y Nintendo, y diferenciándose de éstos, prescinde totalmente de un elemento físico como mando. Comparte cierta similitud con el sistema *Eye Toy* aparecido en 2008, al basar casi por completo su funcionamiento en el procesado de la información recibida a través de cámaras, pero las diferencias en cuanto a su tecnología y posibilidades son muy pronunciadas.

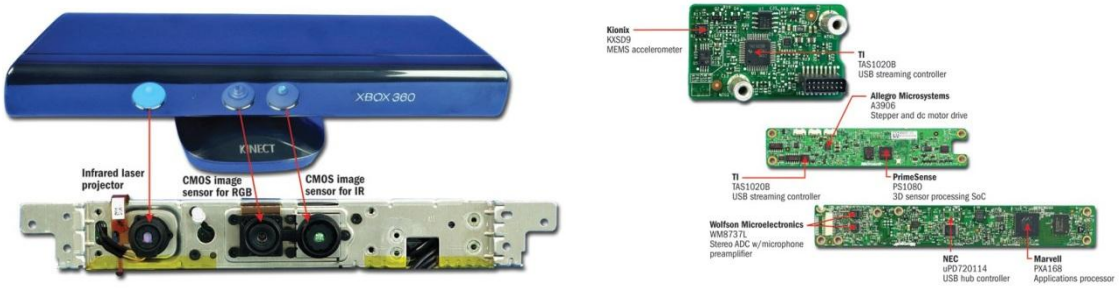


FIGURA 2.9 - Detalle del Microsoft Kinect (http://www.neoteo.com/como-es-kinect-por-dentro)

Para realizar estas funciones el hardware de *Kinect* cuenta con un sensor de profundidad, compuesto por un proyector de haz infrarrojo combinado con un sensor CMOS MT9M001 de Aptina Imaging que operando conjuntamente, permiten conocer la distribución de los elementos de la habitación dentro un rango de varios metros con cualquier condición de luz ambiental. En su funcionamiento detecta los reflejos del haz infrarrojo, dentro de un patrón de puntos y segmentos, correspondientes a partes de la morfología del propio usuario, y genera mapas que informan sobre la intensidad de los anteriormente citados puntos y segmentos, así como de los cambios en su posición relativa. Para la entrada de información de video RGB se utiliza una cámara a color, trabajando como apoyo al sensor MT9M112, lo que permite también captar y reproducir dicha imagen tal y como la vería un ojo humano**.** Ambos dispositivos poseen una resolución significativa (1,3 mega-pixeles).

La captación de señales del entorno se completa con cuatro micrófonos, que en combinación permiten captar las ordenes de voz emitidas por el usuario sin problemas de eco, y aislando posibles fuentes de ruido externo.

Por último el chip PS1080 de PrimeSense interpreta todas las señales dando lugar al llamado procesamiento sensorial, es decir, se encargará de todos los controles del sistema, del proyector de infrarrojos, de los procesos de la información que cada cámara recoge y de las entradas de audio.

Al igual que ocurre con Wiimote, la fácil accesibilidad a este periférico, unido a su bajo coste y las posibilidades que es capaz de brindar, han animado tanto a particulares como a entornos académicos a experimentar con él. Los campos a los que se destinaria su uso serian de nuevo los entornos domoticos, la robótica, además del reconocimiento avanzado de imágenes.

Esta tendencia en cuanto al uso del reconocimiento de imágenes de cara a ser empleadas como interfaces inerciales, es denotada fuertemente si tenemos en cuenta otros proyectos académicos muy destacables, como 6th Sense o Mouseless, ambos desarrollados en el **I**nstituto **T**ecnológico de **M**assachusetts (MIT), y los productos presentados por otras compañías de entretenimiento, como Move de Sony. Interfaz que combina las características de los periféricos de sus competidoras Nintendo y Microsoft, empleando como método de control un mando sensor inercial, junto con procesado de imágenes conjuntamente.

2.3 PARALELISMOS Y CONCLUSIÓN

Pese a los múltiples planteamientos diferentes adoptados para el diseño de una mejor interfaz presentados en el apartado anterior, es apreciable como en su composición todas ellos contienen elementos conceptuales similares, aunque desarrollados con tecnologías diferentes.

Se resaltan en la Figura [**2.10**], estando la interacción estructurada como un sistema de control clásico realimentado:

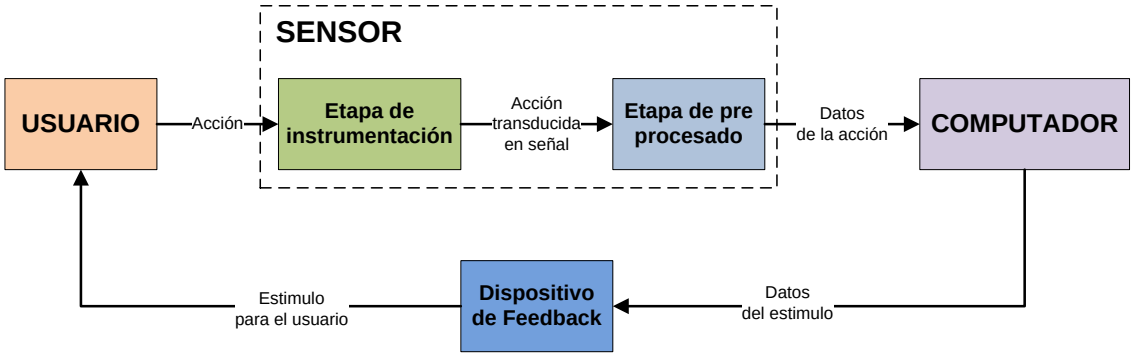


FIGURA 2.10 - Diagrama de control en lazo cerrado de un sistema de control basado en la interacción hombre-maquina

De entre todos ellos cabe resaltar la importancia del elemento sensor, sobre el cual se han podido observar múltiples menciones concretas durante el desarrollo de este capítulo.

Los elementos sensores conectan la cadena de realimentación iniciada por la acción del usuario con la maquina, detectando dicho estimulo y convirtiéndolo en una señal medible con posibilidad de ser trasmitida al computador. Y dado que la tecnología empleada en la construcción de estos componentes, delimita completamente como debe ser realizada la interacción por el usuario, son por tanto cruciales en el diseño de una interfaz.

Mayor capacidad de captación y precisión en el sensado de información, permite por tanto una interacción más rica y variada por parte del usuario. La interfaz a su vez presenta una mayor intuitividad y simpleza, siendo su modo de empleo natural y cercano a la persona. Por tanto se cumplen gran parte de los objetivos expuestos por la disciplina HCI para el desarrollo de un mejor interfaz.

2.4 REQUISITOS FUNCIONALES

A partir de estas premisas que resumimos a continuación, para ser competitivo en el mercado actual, un dispositivo de control para una interfaz inercial debe cumplir:

- Ser capaz al menos de captar los movimientos del usuario con suficiente precisión en los seis grados básicos de libertad posibles para un cuerpo rígido (rotaciones respecto a los tres ejes y desplazamientos lineales a lo largo de los mismos).
- Cumplir los criterios de ergonomía que permitan que la interacción del usuario sea cómoda y confortable. En el caso de actuar mediante contacto directo con un elemento físico este debe de ser de dimensiones y peso reducidas, y adaptarse a la fisonomía del usuario.
- Contar con una tecnología de transmisión de información inalámbrica, permitiendo así una mayor libertad de movimiento sin restricciones impuestas por el uso de cableado.
- Reducir lo máximo posible su consumo energético y aumentar en el mayor grado posible su autonomía.

Además de los enunciados, pueden contemplarse otros capaces de mejorar la experiencia de su uso y aportar una mayor versatilidad, tales como:

- Que el dispositivo de control cuente con elementos adicionales capaces de transmitir una respuesta o feedback ante las acciones realizadas por el usuario.
- Permitir una interacción conjunta con otros dispositivos de control sean de naturaleza inercial o no.
- La evaluación del entorno en el cual se desarrolla la interacción modificando los parámetros de la misma según el marco contextual.
- La catalogación de objetos e identificación de entornos no pertenecientes a la interfaz, pero que posteriormente puedan ser reconocidos por este e incluidos como elementos adicionales de control.

La gran mayoría de los sistemas de control inerciales disponibles en el mercado actualmente cumplen los primeros objetivos enumerados. No sucede así con los criterios adicionales expuestos y aun mas significativamente con estos últimos. La evaluación del entorno y el contexto en el cual lleva a cabo la interacción por parte del usuario, es por ejemplo una vía con múltiples posibilidades de desarrollo en cuanto a sistemas de realidad aumentada. Así como también la catalogación e inclusión de elementos externos al propio interfaz para que formen parte de este.

En este proyecto se abordara la inclusión de un elemento enfocado a la consecución de estos criterios. Un lector RFID embebido en un control inercial portátil, ofrece multitud de nuevas funcionalidades a aplicaciones pertenecientes a entornos industriales, ambientes demóticos, ayuda a la accesibilidad para discapacitados, ocio electrónico, etc.

CAPITULO 3. DESCRIPCION DEL DISEÑO DEL HARDWARE

A continuación se justifican las decisiones tomadas para el desarrollo del hardware del prototipo del dispositivo de control inercial elaborado para este proyecto. Los criterios se basan en los requisitos funcionales expuestos en la sección 2.4 de esta memoria.

En la Figura [3.1], se muestran los principales bloques funcionales que componen el dispositivo.

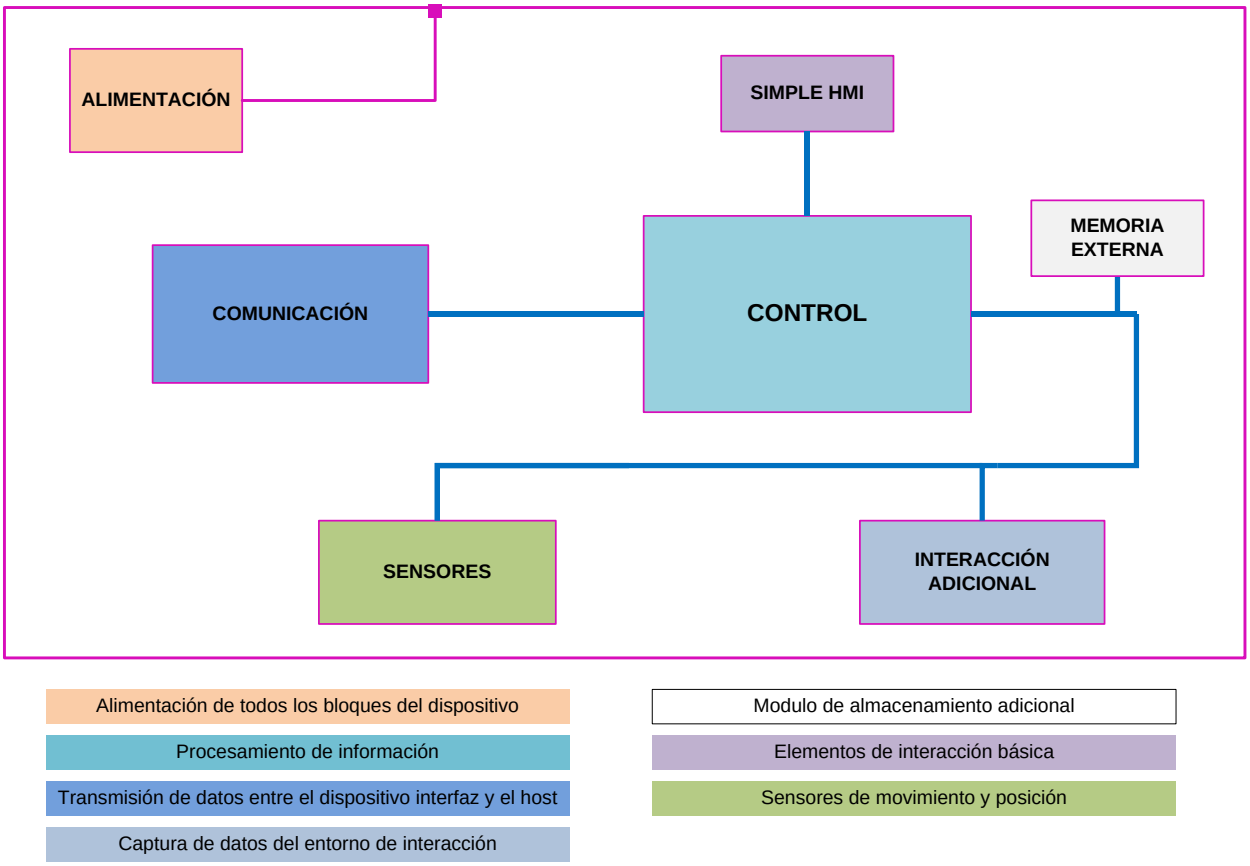


FIGURA 3.1 - Diagrama de bloque general

El proceso seguido durante la elección de los componentes seleccionados para cada bloque se detalla a continuación.

3.1 COMUNICACIÓN

Dada la naturaleza de este proyecto como dispositivo móvil, es necesario implementar un sistema de comunicaciones que permita una transmisión de información en ambos sentidos con el sistema huésped (host), generalmente un computador, de una manera eficaz pero a la vez cómoda y versátil para el usuario. La tendencia actual impone una sistema de comunicación de conexión sin cables, tal y como se señalo en la sección 2.4 de esta memoria.

De entre las tecnologías de comunicación inalámbricas RF empleadas en la actualidad; Wi-Fi, UWB, Bluetooth, ZigBee, **[5]** se ha optado por esta última como solución al cumplimiento de los requisitos del proyecto, principalmente porque este se presenta como el protocolo más eficiente en el consumo de energía. Ofrece los menores valores de potencia disipada y cuenta con un ratio mayor de gasto energético respecto a la tasa de datos en la transmisión que las otras opciones disponibles.

Siendo ZigBee y Bluetooth de entre todas las dos tecnologías más enfocadas a aplicaciones con bajos requerimientos en cuanto a tráfico de datos y alimentación, se puede apreciar en las Figura [**3.2**], y [**3.3**], que Bluetooth se ve superado en prestaciones por el protocolo empleado en este proyecto, tanto en transmisión (TX) como en recepción (RX).

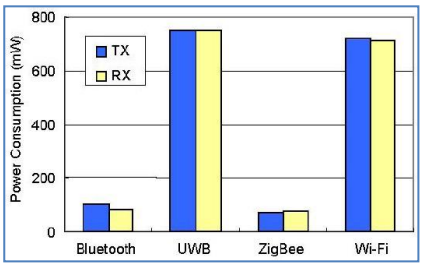


FIGURA 3.2 - Comparación de consumo [4]

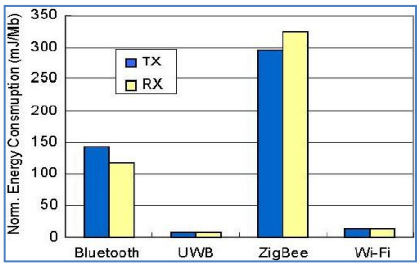


FIGURA 3.3 - Consumo frente a transmisión [4]

ZigBee ofrece además una menor complejidad en su protocolo, sin sacrificar en aspectos como la seguridad en las comunicaciones (posee un mecanismo de encriptación CTR y autentificación CBC-MAC también llamado CCM), comprobación de errores de transmisión (cuenta con CRC de 16bit), o en la versatilidad a la hora de estructurar la red permitiendo múltiples topologías y contando en este aspecto con el mayor número posible de nodos en la misma simultáneamente. Estas y otras características se pueden contrastar con las alternativas existentes en la Figura [**3.4**].

Standard	Bluetooth	UWB	ZigBee	Wi-Fi
IEEE spec.	802.15.1	802.15.3a *	802.15.4	802.11a/b/g
Frequency band	2.4 GHz	3.1-10.6 GHz	868/915 MHz; 2.4 GHz	2.4 GHz; 5 GHz
Max signal rate	1 Mb/s	110 Mb/s	250 Kb/s	54 Mb/s
Nominal range	10 m	10 m	10 - 100 m	100 m
Nominal TX power	0 - 10 dBm	-41.3 dBm/MHz	(-25) - 0 dBm	15 - 20 dBm
Number of RF channels	79	(1-15)	1/10; 16	14 (2.4 GHz)
Channel bandwidth	1 MHz	500 MHz - 7.5 GHz	0.3/0.6 MHz; 2 MHz	22 MHz
Modulation type	GFSK	BPSK, QPSK	BPSK (+ ASK), O-QPSK	BPSK, QPSK COFDM, CCK, M-QAM
Spreading	FHSS	DS-UWB, MB-OFDM	DSSS	DSSS, CCK, OFDM
Coexistence mechanism	Adaptive freq. hopping	Adaptive freq. hopping	Dynamic freq. selection	Dynamic freq. selection, transmit power control (802.11h)
Basic cell	Piconet	Piconet	Star	BSS
Extension of the basic cell	Scatternet	Peer-to-peer	Cluster tree, Mesh	ESS
Max number of cell nodes	8	8	> 65000	2007
Encryption	E0 stream cipher	AES block cipher (CTR, counter mode)	AES block cipher (CTR, counter mode)	RC4 stream cipher (WEP), AES block cipher
Authentication	Shared secret	CBC-MAC (CCM)	CBC-MAC (ext. of CCM)	WPA2 (802.11i)
Data protection	16-bit CRC	32-bit CRC	16-bit CRC	32-bit CRC

* Unapproved draft.
• Acronyms: ASK (amplitude shift keying), GFSK (Gaussian frequency SK), BPSK/QPSK (binary/quardrature phase SK), O-QPSK (offset-QPSK), OFDM (orthogonal frequency division multiplexing), COFDM (coded OFDM), MB-OFDM (multiband OFDM), M-QAM (M-ary quadrature amplitude modulation), CCK (complementary code keying), FHSS/DSSS (frequency hopping/direct sequence spread spectrum), BSS/ESS (basic/extended service set), AES (advanced encryption standard), WEP (wired equivalent privacy), WPA (Wi-Fi protected access), CBC-MAC (cipher block chaining message authentication code), CCM (CTR with CBC-MAC), CRC (cyclic redundancy check).

FIGURA 3.4 - Especificaciones técnicas de las diferentes tecnologías analizadas [4]

Si se estima la complejidad de los protocolos en base a las tramas de datos necesarias para el establecimiento de la comunicación entre los distintos elementos de la red en la capas física (número de primitivas y eventos del Host/Controller Interface en el caso de Bluetooth, o el numero de primitivas MAC/PHY para UWB, ZigBee, y otros protocolos Wi-Fi), se observa en la Figura [**3.5**], que en el caso de la solución adoptada para este proyecto, esta cuenta con una comunicación mucho más directa y reducida, siendo la principal ventaja su menor consumo para una misma longitud de mensaje transmitido, ofreciendo a su vez, una mayor accesibilidad para el desarrollo de aplicaciones (fácil integración con pocos componentes electrónicos y desarrollo de capas de la interfaz abierto).

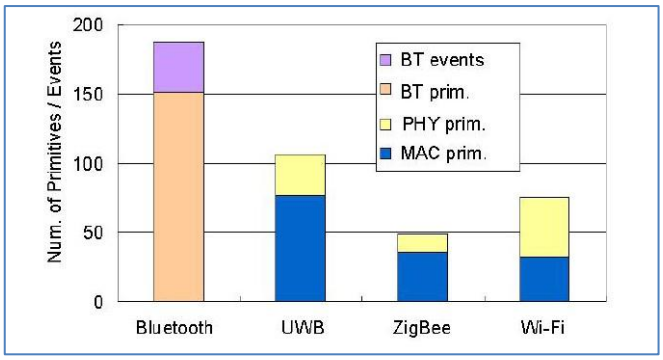


FIGURA 3.5 - Comparativa de número de primitivas por protocolo [4]

Por último, tal y como se puede constatar en la Figura [**3.4**], su único punto negativo es la baja tasa de transmisión de ofrece (250kb/s), aunque no supone en ningún caso un obstáculo, cumpliendo las especificaciones requeridas para la realización de este proyecto.

Actualmente multitud de empresas suministran componentes ZigBee orientados a un manejo a bajo nivel, como a alto nivel con paquetes o conjuntos de desarrollo especializados. Pero, dado que es preciso que el componente seleccionado cuente con un tamaño reducido (incluida la antena) y un bajo consumo, unido al hecho de que estos últimos no siempre ofrecen una buena compatibilidad con dispositivos de otras marcas, además de contar con ciertas restricciones en el desarrollo de aplicaciones bajo este estándar, se ha optado como solución por un dispositivo de bajo nivel.

Como opciones de este perfil disponibles en el mercado europeo cabrían destacarse los siguientes productos; *ZigBit* de Atmel, *Pixie* de *FlexiPanel*, *XBee* de Digi International, *ETRX3* de Telegesis, de los cuales pueden observarse algunas de sus características en la Figura [**3.6**], compartiendo todos ellos una banda de frecuencia de trabajo similares (2400 MHz) y una orientación a dispositivos portátiles de bajo consumo.

Product	MCU core	Interfacing	TX Power	Tx Current	Rx Current	Sleep Current	Size (mm)	Antenna
ZigBit 2.4 GHz	8-bit ATmega 1281v	UART, SPI, I²C, JTAG	3 dBm	18 mA	19 mA	6 µA	18.8 x 13.5	RF output
XBee DigiMesh 2.4	Freescale MC13193	UART	0 dBm	45 mA	50 mA	10 uA	24 x 27	integrated, U.FL connector, RPSMA connector
ETRX3	Ember EM250	SIF, UART, I²C, SPI	3 dBm	25mA	30 mA	2 uA	19 x 25	integrated, U.FL connector or 50Ω pad
Pixie	PIC18LF4620	UART, SPI	0 dBm	25mA	25mA	2 uA	19 x 52	integrated

FIGURA 3.6 - Comparativa de diferentes módulos Zigbee comerciales

Para el desarrollo de este proyecto se ha seleccionado finalmente el modulo ETRX357 de las series EM3000 (ETRX3) en respuesta a la exigencia planteada por la dirección de empleo de un producto de *Telegesis.*

Este componente empleado con anterioridad en otros desarrollos realizados por el equipo de investigación HOWLab, ofrece unas muy buenas características de corriente consumida respecto de la potencia emitida, dimensiones reducidas sin necesidad de conexión de antena externa y la posibilidad de trabajar con distintas configuraciones de comunicación.

FIGURA 3.7 - ETRX 357 (http://www.semix.co.il/)

Por encima de esto, la razón de más peso de cara a su elección es que este además con la posibilidad de trabajar con la actualización *Zigbee Pro*, la cual ofrece características mejoradas no incluidas en las especificaciones de la *Zigbee Alliance 2006*, como por ejemplo:

- Mejor soporte para redes de gran extensión.
- La capacidad de dividir mensajes largos y permitir la interacción con otros protocolos y sistemas.
- La posibilidad de que los dispositivos cambien de un canal a otro de manera dinámica en el caso de que aparezcan interferencias.
- Una mejor y más optimizada gestión de los dispositivos.
- Localización grupal que agiliza el tráfico en las grandes redes.
- Mayor seguridad.
- El uso de la recolección centralizada de datos.

El modulo ETRX357 integra el chip EM250 especifico para comunicaciones ZigBee/802.15.4 y su manejo es realizado mediante comandos AT definidos en un protocolo propio de Telegesis. Éstos son transmitidos en comunicación serie asíncrona RS-232 a través de la UART del microcontrolador PIC18F24J11, sin necesidad de elementos pasivos adicionales para su conexionado. Aunque se ha implementado un puerto en la PCB de conexión para la reprogramación de su firmware si esta se precisase.

3.2 CAPTURA DE MOVIMIENTOS

Tal y como se observó en el capítulo 2 de esta memoria, existen multitud de formas de abordar la elaboración de interfaces inerciales dependiendo del uso de las diferentes tecnologías que existen actualmente en el mercado.

La orientación de este proyecto es la de desarrollar un dispositivo capaz de capturar los movimientos de la mano del usuario tanto longitudinales como angulares en un entorno espacial tridimensional, tal y como queda planteado en los requisitos funcionales expuestos en la sección 2.4. Además debe contar é con un tamaño, peso y consumos reducidos para una mayor portabilidad, por lo que se descartan sistemas que precisen de periféricos externos adicionales y cuya disposición haya de mantenerse fija, tales como los empleados en sistemas con tecnología de captura de imágenes.

El objetivo final es la creación de un controlador confortable, que sea manejado o se encuentre adherido al usuario de un modo similar al de un guante de adquisición de datos, pero con una mayor libertad de movimiento y de adaptación a distintos entornos. En este caso, la utilización de sensores inerciales que puedan ser fácilmente integrados en un dispositivo de control único, es el enfoque óptimo.

Por tanto es la necesaria de la inclusión de tres sensores:

- Un acelerómetro que permita detectar las variaciones longitudinales del movimiento del control.
- Un giróscopo que permita detectar la inclinación y movimientos angulares ejercidos al control.
- Un magnetómetro o brújula que permita al sistema unos marcar ejes de coordenadas fijos y un origen para el elemento de control.

Por otro lado se valoran los siguientes criterios en conjunto para los integrados citados:

- Que estos dispongan preferiblemente de un modulo de conversión interno, transmitiendo los datos sensados de forma digital en vez de analógica.
- El empleo para ello de una comunicación serie síncrona I2C.
- Fácil accesibilidad, versatilidad en cuanto a sus modos de trabajo y un buen soporte de ayuda para el desarrollo de aplicaciones propias.
- Baja tensión de alimentación y consumo.
- Reducidas dimensiones.

De entre todas las soluciones comerciales existentes en el mercado, la adoptada en este proyecto para la detección de movimientos es la utilización del modulo SEN-10183 9DOF Stick comercializado por Sparkfun, el cual integra en dimensiones muy reducidas estos tres citados componentes. Seleccionada por sus características técnicas de entre los productos ofrecidos por este proveedor por la exigencia planteada por la dirección de proyecto.

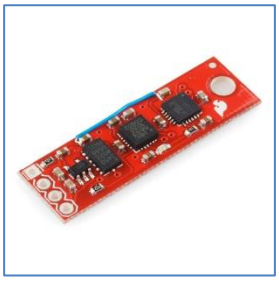


FIGURA 3.8 - Modulo de sensores para captación de movimiento (http://www.sparkfun.com)

Con este producto en una pequeña PCB, se dispone por tanto del acelerómetro digital ADXL345, el magnetómetro HMC5843 y el giróscopo ITG-3200, un regulador de tensión lineal a 3.3v, así como de todos los componentes pasivos necesarios para una configuración básica de funcionamiento de todos los dispositivos con un precio menor al que obtendríamos adquiriendo los distintos integrados por separado.
A continuación se muestran con un mayor detalle las características técnicas de los componentes que integran este modulo sensor, comprobándose que cumplen con las especificaciones requeridas para este diseño.

3.2.1 ADXL345

Este acelerómetro de 3 ejes de *Analog Devices* cuenta con un tamaño realmente reducido (3mm x 5mm x 1mm), con un consumo que puede llegar a alcanzar los 0,1uA[1] en Stand By, y 40uA en modo de medición con una alimentación limite de 2,5v. Esto junto con su elevada resolución de hasta 13bits y su amplio rango de medida de 16g (con 4msg/LSB como factor de escala), hacen de él una opción muy viable para el desarrollo de dispositivos móviles en los que prime el ahorro de energía, pero sin sacrificar aspectos como la precisión.

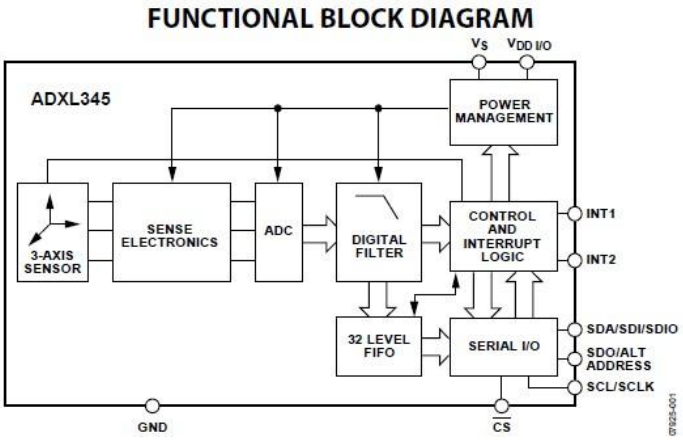


FIGURA 3.9 - Diagrama interno funcional del ADX345 (Datasheet del fabricante)

Otras de sus características son la inclusión de un buffer interno tipo FIFO de 32 niveles que permite almacenar amplias muestras de datos, minimizando de este modo la interacción con el host (microcontrolador PIC18F24J11 en este caso), permitiendo una mayor versatilidad en la comunicación de ambos dispositivos. Los diferentes modos de trabajo que permite y sus particularidades serán presentados con mayor detalle en la sección … .

El ADXL345 permite además una gran libertad en cuanto al ajuste de su rango y precisión de conversión, contando con características tan destacables como la posibilidad de permanecer en un estado de reposo y consumo mínimo del cual puede despertar por variaciones de aceleración en cualquiera de los ejes, detecciones por paso simple y doble, detección de caída libre y una muy buena robustez frente a golpes.

[1] *NOTA: Téngase que estos son los valores presentados por el fabricante para unas condiciones de operación muy concretas y que no siempre se ajustan a las situaciones de trabajo generales. Pueden observarse en la tabla de consumos (sección…) tanto los valores teóricos estimados para la configuración de este proyecto, como los experimentales medidos en el laboratorio.*

3.2.2 HMC5843

El HMC5843 de *Honeywell* se posiciona como una de las mejores opciones en cuanto a sensores magnéticos existentes en el mercado, gracias al empleo de su tecnología AMR (**A**nisotropic **M**agneto**r**esistive). Las características de presentación son un rango ampliamente configurable desde las decenas de micro-gauss hasta 6 gauss, con una corrección de Offset muy precisa y una resolución de conversión digital de hasta 12bits.

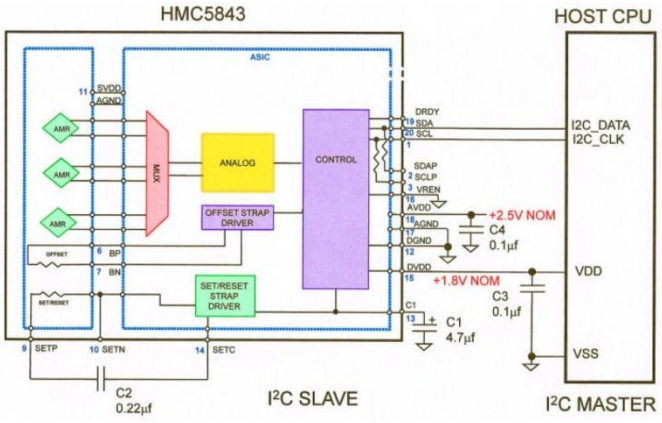


FIGURA 3.10 - Diagrama interno funcional del HMC5843 (Datasheet del fabricante)

Por otra parte, sus reducidas dimensiones (4mm x 4mm x 1mm), y sus bajos requerimientos de alimentación y consumo también hacen de él una opción excelente para la realización de aplicaciones móviles. Mínimos de corriente en 2,5uA[2] en modo Sleep, 240uA en Idle, y 0,8mA son valores reseñables de cara a obtener una buena autonomía sin un gran sobredimensionamiento de la batería, que es lo que se pretende obtener en el diseño de este proyecto.

En cuanto a sus modos de funcionamiento permite una medición continua, o la toma de muestras intermitentes bien por interrupción mediante una orden recibida a través del host, o por el evento producido por un reloj interno configurable mediante software. Por último cabe destacar que este cuenta con un modo de auto testeo para una correcta calibración para hasta un rango de 4 gauss sin necesidad campos magnéticos externos.

3.2.3 ITG-3200

Este integrado comercializado por *Invensense* se presenta como el primer giróscopo integrado en un único chip optimizado para el ámbito de los controladores para juegos y de aplicaciones remotas en 3D. Este componente ofrece como características más relevantes una gran estabilidad frente a cambios en la temperatura ambiente y la alimentación que simplifican enormemente la calibración, una fuerte disminución frente a los ruidos de baja frecuencia para un control más preciso.

[2] *Téngase que estos son los valores presentados por el fabricante para unas condiciones de operación muy concretas y que no siempre se ajustan a las situaciones de trabajo generales. Pueden observarse en la tabla de consumos (**sección**…) tanto los valores teóricos estimados para la configuración de este proyecto, como los experimentales medidos en el laboratorio.*

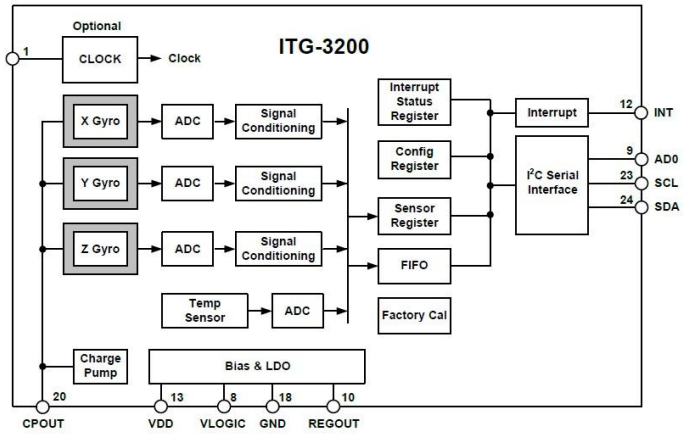


FIGURA 3.11 - Diagrama interno funcional del ITG-3200 (Datasheet del fabricante)

Como otros datos técnicos a enumerar, este dispositivo cuenta con conversores analógicos-digitales de 16bits (hasta 14.375 LSBs por °/seg para un fondo de escala de ±2000°/seg), un filtro paso bajo ajustable mediante software, un modulo de comunicación síncrona I2C que permite trabajar a 400Kb/s, un sensor de temperatura y un propio oscilador interno.

Las reducidas dimensiones del ITG-3200 (4mm x 4mm x 0,9mm), junto con sus bajos niveles de consumo (5uA[3] en STAND BY y 6,5mA en su modo operación), una fuerte resistencia frente a los choques (10000g), y las prestaciones anteriormente citadas, lo muestran como un dispositivo fuertemente orientado a las aplicaciones electrónicas móviles.

3.2.4 CONSIDERACIONES FINALES

Los tres componentes incluidos en el SEN-10183 cuentan con módulos conversión propia, transmisión digital de las señales sensadas y módulos de comunicación síncrona compatibles con los protocolos SPI e I2C, cumpliendo con todos los requisitos funcionales establecidos.

Las dimensiones finales del Stick son muy reducidas para los componentes que lo integran (32x10mm), montando componentes pasivos con encapsulado tipo 0402, siendo acorde con lo demandado para la realización de este proyecto.

Además de sus características técnicas ya citadas, es destacable también el hecho de que la disposición de estos componentes en una placa externa a la principal, permite la ventaja de continuar con la misma línea de producción a largo plazo ante la no existencia o revisión de los componentes empleados en este diseño en un futuro, ofreciendo la posibilidad de su sustitución por otros siempre y cuando se mantenga una conexión compatible I2C, denotando la modularidad del diseño de este proyecto y su patente orientación al hardware libre.

[3] *NOTA: Téngase que estos son los valores presentados por el fabricante para unas condiciones de operación muy concretas y que no siempre se ajustan a las situaciones de trabajo generales. Pueden observarse en la tabla de consumos (apartado … del capítulo …) tanto los valores teóricos estimados para la configuración de este proyecto, como los experimentales medidos en el laboratorio.*

3.3 AMPLIACIÓN DEL MARCO DE INTERACCIÓN

En este proyecto se plantea como novedad dentro del ámbito de los interfaces de control inerciales la inclusión de un sistema capaz de identificar y catalogar elementos externos a este, de tal modo que formen parte del propio sistema, ampliando el marco y la riqueza de la interacción del usuario.

Existen dos vías principalmente para alcanzar este fin, una parte del almacenamiento de las propiedades y características de los objetos externos a la interfaz dentro del propio sistema de control, siendo estos reconocidos mediante técnicas de captura de imagen. Sin embargo, la utilización de estos métodos trae consigo desventajas tales como un mayor consumo energético y dimensiones, que por lo general no favorece a que el producto final sea portátil.

La vía alternativa se basa en almacenar en los propios objetos a incluir en la interfaz la información de sus características, siendo actualmente la tecnología más completa para este fin la de identificación por radiofrecuencia o RFID (**R**adio **F**requency **ID**entification). Esta permite mediante la adhesión de las llamadas TAGs o etiquetas RFID en los objetos, las cuales pueden encontrarse en multitud de formatos, no solamente el almacenamiento de información que pueda ser interpretada por la interfaz de control, sino la modificación de esta información y que incluso sea posteriormente reconocida por un nuevo sistema interfaz, con todas las posibilidades que esto conlleva.

Para la elección del receptor/emisor RFID se ha tenido en consideración como aspecto principal, la posibilidad de que este sea compatible con las múltiples variantes empleadas por los distintos fabricantes en cuanto a TAGs y sus protocolos de comunicación empleados en cada caso, dada la gran variedad existente en este campo. Dado que su propósito es el de ser parte de un sistema embebido, se ha centrado la búsqueda en módulos comerciales de bajo a medio nivel, en los cuales sea provisto por el fabricante un soporte y las herramientas básicas necesarias para realizar tanto aplicaciones como una capa de firmware y middleware propios.

Además de esto, para la elección del modulo RFID multipropósito se ha tenido en cuenta los factores relacionados de frecuencia, rango y potencia de transmisión.

Considerando el primero de los factores, las posibilidades de transmisión existentes a día de hoy son dos principalmente. El uso de de acoplamiento inductivo posibilita trabajar en un rango de 100Khz a 30Mhz, mientras que en sistemas basados en microondas este se eleva de 2,45 a 5,8Ghz usando sistemas electromagnéticos o de microondas.

Una alta frecuencia de la onda portadora permite ventajas tales como un mayor alcance con una menor potencia radiada y un tamaño menor necesario en la antena. Sin embargo estos sistemas precisan de una complejidad y sofisticación en cuanto a su electrónica implementada, que traen consigo un mayor tamaño, demanda de energía y coste.

Si optamos por trabajar a una frecuencia inferior para un mismo alcance, ha de tenerse en consideración que no es posible transmitir con una potencia superior delimitada como máxima por los organismos reguladores de cada región, y que el margen de maniobra en cuanto a las frecuencias preestablecidas a

su vez no es demasiado amplio por estar delimitadas las bandas de trabajo para una no interferencia con otros sistemas de comunicaciones. En Europa la norma emitida por el organismo regulador ETSI (**E**uropean **T**elecomunications **S**tandars **I**nstitute) delimita la utilización de la frecuencia UHF para un rango de 869,4-869,65 MHz con un una potencia de transmisión máxima de hasta 2W. Sin embargo frecuencias LF (125-134Khz y 140-148,5Khz) así como HF (13,56Mhz) pueden ser utilizadas globalmente sin necesidad de licencia ya que trabajan dentro de la banda ISM (**I**ndustrial - **S**cientific - **M**edical).

Por ello, dada la orientación del desarrollo que nos ocupa, y al ser únicamente necesario un alcance bajo, se ha optado por emplear la tecnología de acoplamiento inductivo para una frecuencia estandarizada de 13,56Mhz, trabajando con unas pequeñas dimensiones de antena y baja potencia de emisión/recepción, estimando unos valores mínimos necesarios de entorno a 150mW para alcances no superiores a 10cm.

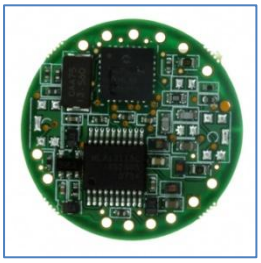


FIGURA 3.12 - Skyemodule M1-Mini (http://www.digikey.com/)

De entre las soluciones comerciales existentes se ha optado finalmente por emplear el modulo Skyemodule M1-Mini de SkyeTek, el cual ofrece un amplio abanico de compatibilidad con los diferentes protocolos de transmisión existentes; ISO15693, ISO1443A, ISO18000-3. Lo que a su vez lo posibilita para operar con casi cualquier tipo de TAG existente en el mercado, tal y como se puede observar en la Figura [**3.13**].

Protocol	Manufacturer	Product	Memory	Read	Write	Anti-Collision
ISO-15694	Texas instruments	Tag-It HF-I	2048	Yes	Yes	Yes
ISO-15694	Philips	I.Code SLI (SL2)	1024	Yes	Yes	Yes
ISO-15694	Infineon	my-d SRF55V series	2.5k-10k	S - ID Only / P - Yes	S - No / P - Yes	Yes
ISO-15694	ST Microelectronics	LR512	512	Yes	Yes	Yes
ISO-14445	Philips	Mifare	1024-4096	ID Only	No	No
ISO-14445	Philips	Mifare Ultralight	512	Yes	Yes	No
Propietary	Philips	I.Code (SL1)	1024	Yes	Yes	Yes
Propietary	Inside Contacless	PicoTag	2048-16384	Yes	Yes	No

*Compability listed refer to most recen firmware version at the time of this document, other protocols are available in previus firmware versions

FIGURA 3.13 - Protocolos compatibles con el Skyemodule M1-Mini

En sus características técnicas consta que este es capaz de una potencia de emisión de hasta 180mW para alcances de 9cm con un consumo máximo de 60mA con antena interna, lo que unido a sus dimensiones reducidas (25mm de diámetro) indica que el M1-Mini cumple perfectamente los requisitos funcionales planteados.

Otro factor importante determinante para su utilización es la posibilidad que plantea en una de sus versiones de trabajar con una comunicación I2C, además de la versatilidad que presenta este modulo para la creación de aplicaciones propias.

3.4 SIMPLE HMI

Se ha planteado la inclusión de elementos de identificación visual en este diseño, de tal manera que estos provean de forma rápida información de los estados de ejecución del sistema. Además se estima la utilidad de disponer de un elemento de control como un pulsador, de tal modo que el usuario pueda modificar de una forma sencilla algunos de los modos de funcionamiento del interfaz.

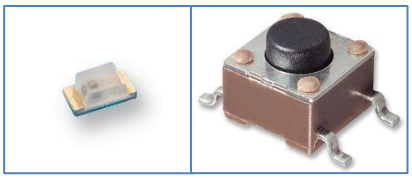


FIGURA 3.14 - LED LGQ971 y SWITCH FM4JSMA (farnell.com)

Siguiendo los criterios básicos seguidos durante toda la toma de decisiones, tales como la búsqueda de componentes de reducido tamaño y bajo consumo, finalmente se ha optado por la búsqueda de componentes SMD con las características eléctricas específicas.

Por lo que finalmente para ello se ha optado por incluir dos diodos LED LGQ971 a modo de indicadores para el primer caso, con las siguientes características:

- Encapsulado SMD tipo 0603 con un tamaño de 0.9 x 1.1mm
- Alimentación a 2.2v con una capacidad de corriente máxima de 20mA para una luminosidad máxima de 10mcd

Mientras que para el control básico la solución adoptada es la de incluir un pulsador FSM4JSMA de dimensiones 1.4 x 3.5mm. Ambos componentes pueden apreciarse en la figura [3.x], siendo su modo de funcionamiento explicado con más profundidad en la sección 4.x de esta memoria.

3.5 CONTROL

En primera instancia, es necesario determinar que elemento será el encargado de intercomunicar los distintos bloques funcionales incluidos en el diseño del prototipo y así como dar soporte físico al software de la interfaz.

La mejor opción para el caso que nos ocupa es el empleo de un microcontrolador (uC o MCU). Un microcontrolador es un circuito integrado programable con capacidades de almacenamiento y procesamiento de información, así como de control de periféricos de entrada y salida. Además, estos dispositivos cuentan actualmente con periféricos adicionales integrados tales como conversores A/D, moduladores PWM (**P**ulse **W**ith **M**odulation), timers RTC (**R**eal **T**ime **C**lock), periféricos de

comunicaciones entre distintos protocolos o controladores para displays entre otros. Por ello se han convertido en la solución perfecta para el desarrollo de sistemas embebidos de todo tipo.

Para la elección del microcontrolador, se han planteado los siguientes requisitos funciónales básicos como punto de partida:

- Baja tensión de alimentación y consumo reducido.
- Posibilidad de remapeado de puertos, dimensiones de encapsulado reducidas y patillaje ajustado a las salidas/entradas requeridas.
- Contar con al menos una línea de comunicación serie síncrona I2C.
- Contar con al menos una línea de comunicación serie asíncrona UART.
- Contar con al menos dos timers con pre escalado configurable.
- Poseer un sistema de interrupciones suficientemente versátil y completo para el desarrollo necesario del firmware.

Una vez alcanzados los objetivos originales, se contemplan:

- Capacidad de trabajo a una frecuencia suficientemente rápida, con una buena tasa de instrucciones por segundo, de tal modo que sea posible atender correctamente a las comunicaciones con los múltiples dispositivos externos del sistema (Zigbee 256kb/s, I2C 400kb/s) con un margen amplio para la ejecución simultanea de otros procesos requeridos por la interfaz. Una buena estabilidad en la frecuencia del reloj interno para una mejor precisión en la base de tiempos de los timers también es deseable.
- Memoria de programa suficiente como para albergar el código requerido para este proyecto, así como para permitir posibles ampliaciones o actualizaciones si estas se precisaran, y memoria RAM suficiente para poder almacenar toda la información temporal de los procesos necesaria para la ejecución del código.

Estos últimos requisitos serán ajustados, siempre y cuando sean suficientes para el correcto soporte del firmware, buscando la mejor relación de coste frente a las características ofrecidas por cada microcontrolador, según las posibles opciones existentes en el mercado.

Dado que no es necesario un complejo procesado de la información, la arquitectura de este dispositivo no es una clara restricción. Se descartaran inicialmente por lo tanto microcontroladores orientados a estas funciones, tales como DSP o microcontroladores de 16 o 24bits.

Esta premisa unida a la exigencia planteada por la dirección de proyecto de empleo de un microcontrolador *Microchip*, ha enfocado la búsqueda de este dispositivo a los existentes en la línea *eXtreme Low Power (XLP)* de 8bits. Los cuales cuentan con unas características muy competitivas en productos orientados a dispositivos móviles de bajo consumo.

Puede observarse en la Figura [**3.15**], la información detallada de los microcontroladores de características semejantes a lo precisado en este proyecto, donde se ha seguido un criterio de selección basado en una buena relación en calidad y prestaciones frente al precio.

Product Family	Architecture	5K $ Pricing	Flash (KB)	EEPROM (Bytes)	RAM (KB)	CPU Speed (MHz, MIPS)	LowPower	Comparators	ADC Channels	ADC Bits	Total UART	SPI	I2C	USB	Ethernet	LIN	CAN	Total Timers	Input Capture	PWM Channels	Parallel Port	Segment LCD	Supply Voltage
PIC16F1518	8	1.01	28	0	1.00	[20,5]	XLP		17	10	1	1	1			Yes		3	2	2		0	1.8 to 5.5
PIC16F1519	8	1.37	28	0	1.00	[20,5]	XLP		28	10	1	1	1			Yes		3	2	2		0	1.8 to 5.5
PIC18F24J11	8	1.65	16	0	3.70	[48,12]	XLP	2	10	10	2	2	2					5	2	2		0	2 to 3.6
PIC18F25J11	8	1.79	32	0	3.70	[48,12]	XLP	2	10	10	2	2	2					5	2	2		0	2 to 3.6
PIC18F24J50	8	1.86	16	0	3.70	[48,12]	XLP	2	10	10	2	2	2	Device				5	2	2		0	2 to 3.6
PIC18F44J11	8	1.95	16	0	3.70	[48,12]	XLP	2	13	10	2	2	2					5	2	2	PMP	0	2 to 3.6
PIC18F25J50	8	2.00	32	0	3.70	[48,12]	XLP	2	10	10	2	2	2	Device				5	2	2		0	2 to 3.6
PIC18F26J11	8	2.07	64	0	3.70	[48,12]	XLP	2	10	10	2	2	2					5	2	2		0	2 to 3.6
PIC18F45J11	8	2.09	32	0	3.70	[48,12]	XLP	2	13	10	2	2	2					5	2	2	PMP	0	2 to 3.6
PIC18F44J50	8	2.16	16	0	3.70	[48,12]	XLP	2	13	10	2	2	2	Device				5	2	2	PMP	0	2 to 3.6
PIC18F26J13	8	2.21	64	0	3.70	[48,12]	XLP	3	10	12	2	2	2					8	10	10		0	2 to 3.6
PIC18F26J50	8	2.28	64	0	3.70	[48,12]	XLP	2	10	10	2	2	2	Device				5	2	2		0	2 to 3.6
PIC18F45J50	8	2.30	32	0	3.70	[48,12]	XLP	2	13	10	2	2	2	Device				5	2	2	PMP	0	2 to 3.6
PIC18F46J11	8	2.37	64	0	3.70	[48,12]	XLP	2	13	10	2	2	2					5	2	2	PMP	0	2 to 3.6
PIC18F26J53	8	2.42	64	0	3.70	[48,12]	XLP	3	13	12	2	2	2	Device				8	10	10		0	2 to 3.6
PIC18F27J13	8	2.45	128	0	3.70	[48,12]	XLP	3	10	12	2	2	2					8	10	10		0	2 to 3.6
PIC18F46J13	8	2.52	64	0	3.70	[48,12]	XLP	3	13	10	2	2	2					8	10	10	PMP	0	2 to 3.6
PIC18F46J50	8	2.58	64	0	3.70	[48,12]	XLP	2	13	10	2	2	2	Device				5	2	2	PMP	0	2 to 3.6
PIC18F27J53	8	2.66	128	0	3.70	[48,12]	XLP	3	10	12	2	2	2	Device				8	10	10		0	2 to 3.6
PIC18F46J53	8	2.73	64	0	3.70	[48,12]	XLP	3	13	12	2	2	2	Device				8	10	10	PMP	0	2 to 3.6
PIC18F47J13	8	2.76	128	0	3.70	[48,12]	XLP	3	13	12	2	2	2					8	10	10	PMP	0	2 to 3.6

FIGURA 3.15 - Comparativa de diferentes microcontroladores XLP de Microchip (extraída de microchip.com)

El microcontrolador seleccionado finalmente para la realización de este proyecto es el PIC18F24J11, capaz de trabajar hasta a 48Mhz a 12 millones de instrucciones por segundo con dos osciladores externos, y alcanzando una cifra de 16Mhz sin la inclusión de ninguno de estos. Prestaciones muy amplias respecto de las necesidades planteadas para el desarrollo de este proyecto.

Cuenta con memoria Flash de 16KB para memoria de programa y SRAM de 3776 bytes, con lo que se estima que se obtendrá una capacidad suficiente para el almacenamiento del código de la versión del firmware realizado, así como también un margen amplio para futuras revisiones o actualizaciones.

Son destacables sus características de consumo siendo estas realmente reducidas. Para su funcionamiento precisa únicamente de una corriente en modo DEEP SLEEP (periféricos apagados, sin ejecución de código, CPU OFF, respuesta lenta) de 850nA[4], de 2,3uA en IDLE (periféricos encendidos sin ejecución de código, CPU OFF), y de 6,2uA en su modo de operación predeterminado (periféricos encendidos con ejecución de código).

Como periféricos integrados dentro del propio encapsulado se dispone de:

- Dos módulos mejorados de comparación, captura y modulación PWM (ECCP). Con capacidad de generación de hasta cuatro salidas, polaridad seleccionable, programación de tiempos muertos, autoreseteo y autoapagado, y control de direccionamiento de los pulsos.
- Dos puertos series síncronos capaces de actuar como maestro (MSSP). Permitiendo trabajar en comunicación SPI de tres hilos en cualquiera de los cuatro modos posibles, con direccionamiento

[4] **NOTA: Téngase que estos son los valores presentados por el fabricante para unas condiciones de operación muy concretas y que no siempre se ajustan a las situaciones de trabajo generales. Pueden observarse en la tabla de consumos (sección…) tanto los valores teóricos estimados para la configuración de este proyecto, como los experimentales medidos en el laboratorio*

de memoria directo (DMA) para hasta 1024 bytes. Así como bajo el estándar I2C en los modos de maestro o esclavo.

- Un puerto para comunicación en paralelo de 8bits capaz de trabajar tanto de maestro como de esclavo.
- Dos comparadores Rail to Rail de entrada multiplexada.
- Un conversor analógico digital con ancho de canal de 10 a13bits, autocalibrable y capaz de trabajar durante los modos de reposo del microcontrolador.
- Una unidad de medida de tiempo de carga (CTMU) que ofrece soporte para el conexionado con sensores capacitivos de presión tales como pantallas táctiles.
- Dos módulos USART mejorados de comunicación asíncrona con soporte para RS-485, RS-232 y LIN/J2602, con Auto Baud Detect y Auto Wake Up con el Start bit.

En este desarrollo, el microcontrolador trabaja con su propio oscilador interno a 8Mhz contando únicamente con los elementos externos pasivos que se pueden observar en las Figura [**3.16**].

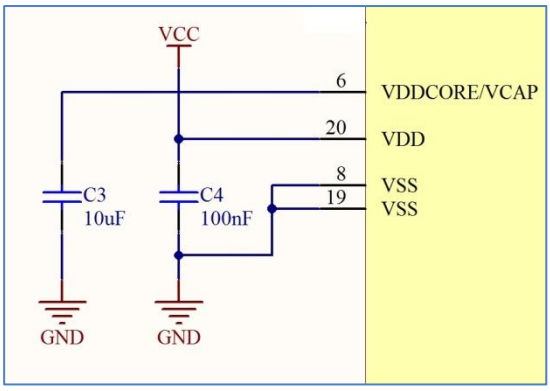


FIGURA 3.16 - Red pasiva de filtrado de la alimentación y estabilización del PIC18F24J11

Cabe señalar que el dimensionado de los condensadores C3 y C4 corresponde a las recomendaciones realizadas en el datasheet del dispositivo para una correcta estabilización y filtrado de la alimentación del microcontrolador PIC18F24J11.

Para su reprogramación se ha dispuesto de un puerto de comunicaciones en la PCB con conexionado a los terminales PGD, PGC y MCLR, permitiendo así futuras revisiones y actualizaciones del código firmware con una mayor comodidad y rapidez sin la necesidad de realizar modificaciones en el hardware.

3.6 MEMORIA

En multitud de aplicaciones es necesario el almacenamiento de datos de forma estable y segura entre distintas ejecuciones sin contar con alimentación. Aunque no esté del todo definida la utilidad final completa de este desarrollo es interesante plantear la posibilidad de que esto se precise, mas teniendo en cuenta aplicaciones de productos similares basados en sensores como registradores de datos (data logger).

La mejor opción para la reducción de tamaño final del diseño parte de la selección de un dispositivo con mayor almacenamiento, pero por lo general se suele dar una limitación en cuanto a la capacidad de memoria no volátil existente en los microcontroladores de gama media, dado el coste que supone su integración a tal nivel. Por ello se ha planteado la inclusión en este proyecto de un dispositivo de almacenamiento de memoria externo al dispositivo de procesamiento.

Las dos principales tecnologías disponibles en la actualidad para este tipo de memoria son EEPROM, FLASH y NVRAM **[6]**, sobre las cuales se cita a continuación un resumen con sus principales características.

EEPROM o E2PROM: Se presentan como la ultima iteración de las memorias PROM, pero salvando ampliamente las limitaciones existentes en cuanto a escrituras y borrados múltiples, precisando únicamente para ello de la aplicación de tensiones en la misma, al contrario de lo que sucedía en las EPROM en las que se requería de una fuente de luz ultravioleta.

Gracias a la utilización de tecnologías CMOS (**M**etal **O**xide **S**ilice), los niveles tensión mínimos para la realización de estos procesos han disminuido su magnitud enormemente hasta el punto de no precisar de programadores, así como su tamaño contando con capacidades que oscilan actualmente entre los 128bits y 1Mb.

Otras de las principales ventajas que proporciona el uso de este tipo de memoria es la posibilidad de borrados y escrituras de palabras individuales (una palabra o Word equivale a 2 bytes / 16bits), ofrecen latencias bastante bajas de escritura y lectura presentando el menor consumo de todas, y cuentan con una muy buena robustez de almacenamiento de datos a largo plazo, permitiendo una gran cantidad de ciclos de escritura/borrado.

FLASH: Son una variante de las EEPROM enfocadas al almacenamiento de una mayor cantidad de datos (desde 512Kb hasta 16Gbit) a un precio y dimensiones equivalentes, y a frecuencias de trabajo más elevadas que las de su tecnología predecesora (de 20Mhz de la EEPROM a 100Mhz en Flash). Por ello son capaces de realizar lecturas y escrituras de grupos mayores de datos en un menor tiempo, lo que ha provocado que su uso se haya extendido enormemente, desbancando casi completamente a otras tecnologías en multitud de ámbitos de aplicación, como son el almacenamiento de información en dispositivos multimedia.

Por el contrario no son capaces de trabajar a menores latencias que las EEPROM con tiempos de acceso de 50ns para FLASH frente a 20ns de estas últimas, lo que unido a que no permiten escrituras y lecturas

de datos de forma individual ha permitido que coexistan ambas tecnologías, cada una con una orientación diferenciada.

NVRAM: Existen soluciones comerciales que integran en un único encapsulado una memoria RAM, frecuentemente SRAM con valores de capacidad que oscilan entre los 8bit y los 4Mb, junto con pequeños sistemas de batería de muy reducido tamaño, con la función de proveer la energía que permita el mantenimiento de los datos en la memoria ante la ausencia de la propia alimentación del sistema.

Este tipo de memoria cuenta con todas las ventajas propias de una memoria SRAM, siendo muchísimo más rápidas que cualquiera de sus competidoras, capaces de trabajar a elevadas frecuencias y con tiempos de acceso mínimos de 5ns. Sin embargo el que no sean capaces de almacenar los datos más allá del límite permitido por la alimentación implementada no las hace idóneas para aplicaciones en las que se precise de una larga vida útil.

Atendiendo a los criterios básicos de este diseño, se ha enfocado la búsqueda de una memoria que cumpla con las siguientes características:

- Permita una comunicación serie síncrona I2C.
- Permita una baja tensión de alimentación.
- Cuente con unas dimensiones, consumo y precio reducidos.

Finalmente se ha seleccionado el modulo de memoria EEPROM 24AA512-I/SM de *Microchip*, dado que este tipo de memorias se ajustarían más a las posibles funcionalidades que se requerirían (accesos rápidos y mantenimiento de la información a largo plazo con un mínimo consumo), contando así con 512Kb adicionales de memoria no volátil.

3.7 ALIMENTACIÓN

Dado que la orientación del proyecto está enfocada a un dispositivo móvil de bajo consumo, los principales criterios a seguir para la elección de un sistema de alimentación se enfocan principalmente en conseguir:

- Un tamaño y peso reducido que permitan al usuario un mayor confort y libertad de movimiento.
- Una capacidad de carga alta, que otorgue al dispositivo de la mayor autonomía posible.
- Una eficiencia energética elevada.
- Precio reducido

Sin embargo estos factores se contraponen, por lo que es necesario estimar un punto de equilibrio entre los mismos, determinado principalmente por el uso de las diferentes tecnologías existentes, tanto en baterías como en sistemas de alimentación y regulación.

A continuación se resumen brevemente los cuatro tipos de baterías mas empleadas en el mercado actualmente **[7]**; Níquel-Cadmio(NiCd), Níquel e Hidruro metálico(NiMh), Plomo (Lead-Acid) y por ultimo Litio-Ion, junto con algunas de sus principales características.

Lead-acid (Plomo): Son las más económicas y adecuadas para aplicaciones donde el consumo de energía es alto. Cuentan con una gran capacidad de almacenamiento de carga y son capaces de proveer una corriente elevada, sin embargo su elevado peso y dimensiones impiden que sean viables para pequeños dispositivos. Sin embargo son ampliamente usadas en aplicaciones que precisen por ejemplo arranque de motores como automóviles, sillas de ruedas, o para sistemas de almacenamiento de carga de emergencia como sistemas de alimentación ininterrumpida.

NiCd: Permiten su recarga, pero cuentan con una moderada densidad energética (por lo general 50 Wh/kg) . Su uso está enfocado a aplicaciones donde se precise de una larga vida de la batería, ya que aceptan una considerable cantidad de ciclos de carga-descarga. Son capaces de proveer corrientes instantáneas elevadas, y es posible trabajar con ellas en un amplio margen de temperaturas. Sufren un efecto memoria muy pronunciado y en su composición contienen metales tóxicos.

NiMh: También recargables, proporcionan una densidad energética algo mayor que las baterías de NiCd (por lo general 70 Wh/kg), pero por el contrario su vida útil es menor que la de estas últimas. Sus características aparte de lo citado no difieren del tipo anterior, pero estas en cambio no contienen metales tóxicos en su composición.

Li-Ion: Este tipo de baterías recargables ofrecen la mayor densidad energética de las cuatro, aun con reducido peso (115 Wh/Kg), habiendo sustituido a las basadas en Níquel en prácticamente la mayoría de aplicaciones. Cuentan con un efecto memoria muy bajo por lo que pueden cargarse partiendo de distintos estados sin que ello merme su vida media. Por el contrario sufren mucho ante las descargas, debiéndose evitar su descarga completa. Por lo general no son capaces de proveer picos fuertes de corriente y precisan de medios de protección tanto para su carga como para su descarga.

Debido a que todos los elementos incluidos en este diseño precisan de una alimentación continua prácticamente similar, en un margen que parte desde los 3,3v hasta los 3.6v, con una demanda muy baja de corriente y sin fluctuaciones pronunciadas, se ha optado por el uso de una batería tipo Li-on.

El cálculo del consumo promedio de cada componente se ha calculado en base a la siguiente aproximación:

$$I_{PROMEDIO} = I_{BAJO\ CONSUMO} \times T1_{OPERACION\%} + I_{OPERACION\ NORMAL} \times T2_{OPERACION\%}$$
$$T1 + T2 = Ttotal$$

Por lo que en base a la estimación de consumo lineal que se puede observar en las tablas de la Figura [**3.18**]. El consumo previsto por ejemplo para el microcontrolador PIC18F24J11 en el caso más desfavorable, trabajando a una frecuencia de 8Mhz con tiempos correspondientes al modo de operación de streaming de sensores inerciales; Ttotal = 20ms, T1= 12ms, T2 = 8ms:

$$I_{PROMEDIO(MAX)} = 2{,}06 \times 0{,}6 + 7{,}4 \times 0{,}4 = 4.196mA$$

MODO SLEEP		
Componente	Consumo Tipico (uA)	Consumo Maximo (uA)
PIC18F24J11		
Sleep (VDD=3.3v 25º) (Micro en estado de sueño a la espera de reactivacion)	3.3	0.02
Sparkfun 9DOF Stick		
ADXL345 (Sleep mode)	40	40
HMC5843 (Sleep Mode)	110	110
PS-ITG-3200 (Sleep Mode)	5	5
Skyetek M1 (Sleep Mode)	60	60
ETRX35 (Quiescent current, including internal RC oscillator)	1	1
MEMORY CMOS 24LC512	1	5
TOTALES	220.3	221.02

MODO DE BAJA ENERGIA		
Componente	Consumo Tipico (mA)	Consumo Maximo (mA)
PIC18F24J11		
FOSC = 8 MHz, RC_RUN mode, Internal RC Oscillator (VDD=3.3v 25º)	1.01	2.06
Sparkfun 9DOF Stick		
ADXL345 (Data rate > 100 Hz)	0.145	0.145
HMC5843 (Measurement Mode)	0.9	0.9
PS-ITG-3200 (Normal Operating Current)	6.5	6.5
Skyetek M1 (Idle mode)	15	15
ETRX35 (Receive current consumption)	35.5	37.5
MEMORY CMOS 24LC512	0.4	5
	Consumo Tipico Activo (mA)	Consumo Maximo Activo (mA)
TOTALES	59.455	67.105

TODOS LOS COMPONENTES EN FUNCIONAMIENTO		
Componente	Consumo Tipico (mA)	Consumo Maximo (mA)
PIC18F24J11		
FOSC = 16 MHz (PRI_RUN mode,4 MHz Internal Oscillator with PLL) (VDD=3.3v 25º)	5.2	7.4
Sparkfun 9DOF Stick		
ADXL345 (Data rate > 100 Hz)	0.145	0.145
HMC5843 (Measurement Mode)	0.9	0.9
PS-ITG-3200 (Normal Operating Current)	6.5	6.5
Skyetek M1 (Active Mode)	60	60
ETRX35 (Transmit current consumption)	35.5	41.5
MEMORY CMOS 24LC512	0.4	5
	Consumo Tipico Activo (mA)	Consumo Maximo Activo (mA)
TOTALES	108.645	121.445

FIGURA 3.18 - Tabla de valores de consumo de corriente teóricos proporcionados por los fabricantes

La tabla incluida en la Figura [**3.19**],expone el consumo promedio esperado dependiendo del modo de operación, tras realizar las operaciones previamente citadas:

Dados los cálculos realizados, para asegurar una para una autonomía de unas 20 horas a pleno funcionamiento en la situación de mayor consumo, se precisara de una batería capaz de ofrecer valores de entre 1100 y 1300mAh.

Modo de operación	Consumo promedio (mA)	Vida util de la bateria* (horas)
Sistema en espera	11.456	87.29
Streaming de datos de los sensores inerciales	38.369	26.06
Escaneo de TAGs RFID	33.956	29.45
Streaming de sensores inerciales y escaneo RFID	61.487	16.26

*Para una bateria de 1000mAH

Modo de operación	Minimo requerido (mA)	Maximo (mA)
Microcontrolador	2.06	7.4
Zigbee	37.5	41.5
RFID	15	60
Sensores	7.545	7.545

Modo de operación	Microcontrolador%	Zigbee%	RFID%	Sensores%
Sistema en espera	0.1	0.3	0	0
Streaming de datos de los sensores inerciales	0.4	0.8	0	1
Escaneo de TAGs RFID	0.1	0.5	1	0
Streaming de sensores inerciales y escaneo RFID	0.7	1	1	1

FIGURA 3.19 - Medias de la corriente consumida estimadas en base a el tiempo de funcionamiento promedio

Tal y como se ha citado con anterioridad, es imprescindible minimizar la pérdida de potencia perdida la adaptación de la tensión de alimentación, por lo que la tensión de la batería deberá ser muy cercana a la tensión nominal que necesitan los componentes del circuito. Por lo que teniendo en cuenta las características del diseño, el empleo de reguladores conmutados queda descartado por su tamaño y por el encarecimiento que supondría para el producto final, y de reguladores lineales por la poca eficiencia de estos, siempre y cuando contemos con una batería cuya tensión sea lo suficientemente estable y por si misma próxima al nivel solicitado.

Se ha optado por incluir el modelo ER14250 de *Eve*, el cual se corresponde con una batería de 3,6v 1200mAh con formato estandarizado 1/2AA (diámetro 14,5 x 25,4mm, 8,8g) por ser el único tipo de batería capaz de ofrecer ese voltaje, dentro de un margen de precio y dimensiones optimas.

FIGURA 3.20 - ER14250 (farnell.com)

De este modo no se precisa de elementos de adaptación de tensión, empleando únicamente como protección un diodo Schottky modelo BAT754A de encapsulado SOT-23, el cual presenta una muy baja tensión de forward (Vf) de únicamente 260mV.

3.8 CONSIDERACIONES FINALES

En la Figura [3.21], se muestra el diagrama de bloques del resultado final, después de la selección de componentes realizada.

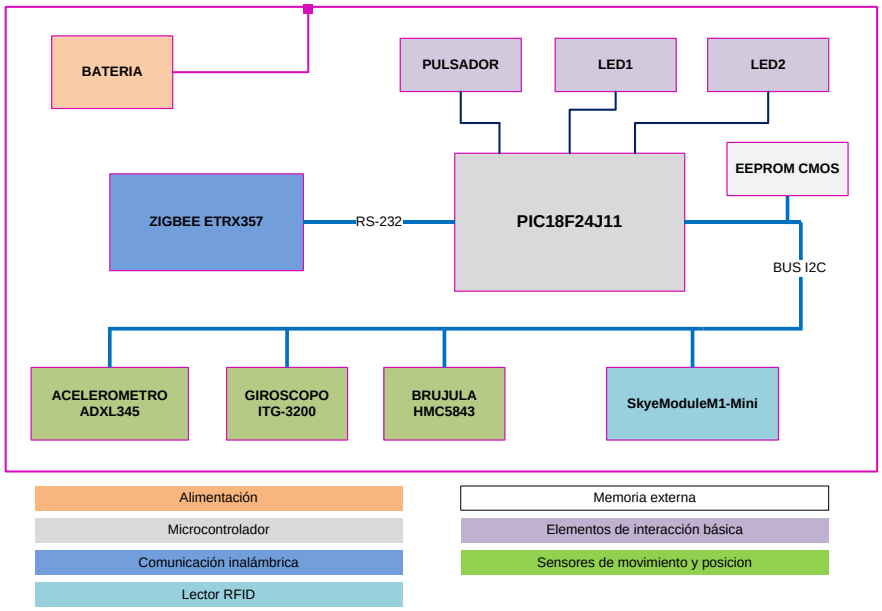


FIGURA 3.21 - Diagrama de bloques con los componentes seleccionados

Resumen de las soluciones adoptadas finalmente:

- Uso combinado de tres sensores integrados para la captura de movimientos: acelerómetro ADXL345, giróscopo ITG-3200 y brújula HMC5843.
- Empleo del protocolo de comunicaciones ZigBee para la transmisión inalámbrica, mediante un modulo de hardware especifico y compatible con este método de comunicación, el ETRX357 de *Telegesis*.
- Reducción de sus dimensiones empleando casi exclusivamente componentes de montaje superficial del menor tamaño según las condiciones posibles de fabricación a nuestro alcance (0603, SOIC, SOT,…), realizando un diseño eficiente del ruteado de la PCB en dos caras economizando al máximo espacio disponible.
- Ajuste de los consumos tanto en la elección de los componentes como por programación, para el empleo de un sistema de alimentación de batería reducida y liviana, además de una mayor autonomía.
- Por último, la introducción de un nuevo elemento en un interfaz inercial, el lector RFID M1-Mini de *Skyetek*.

Cabe también señalar que durante el detallado de la realización de este hardware se pretende incidir en el concepto de modularidad con una orientación abierta. Este proyecto pretende ser dotado con la posibilidad de que sean aplicadas futuras ampliaciones y/o modificaciones de una manera rápida y versátil. Este hecho se expondrá no obstante con más detalle en los siguientes apartados referentes a los distintos bloques que componen este diseño, así como durante el siguiente capítulo.

CAPITULO 4. DESCRIPCIÓN DEL DISEÑO DEL FIRMWARE

Este capítulo tiene como objeto el exponer los procesos seguidos y las soluciones adoptadas durante la elaboración del firmware implementado en este proyecto, así como el resultado final alcanzado. Para una mayor comprensión del mismo se presentara tomando como punto de referencia diferentes enfoques.

- Su funcionamiento general como maquina de estados.
- Su funcionamiento interno como diagrama de flujo de un algoritmo aproximado.
- Las capas de abstracción de la programación.
- La organización de su sistema de archivos.

Para su realización se han seguido criterios de programación modular y estructurada, realizando el código íntegramente en lenguaje C, optimizándolo en la mayor medida posible para el empleo del microcontrolador PIC18F24J11 como soporte de su ejecución.

Las herramientas empleadas han sido principalmente el IDE o entorno de desarrollo integrado MPLAB junto con el compilador PCWHD.

4.1 DESCRIPCION GENERAL

En una primera aproximación al sistema, se ha planteado una estructura del mismo equivalente al de una maquina de estados, cuyo diagrama aparece en la Figura [**4.1**], como medio para ilustrar el funcionamiento global del dispositivo.

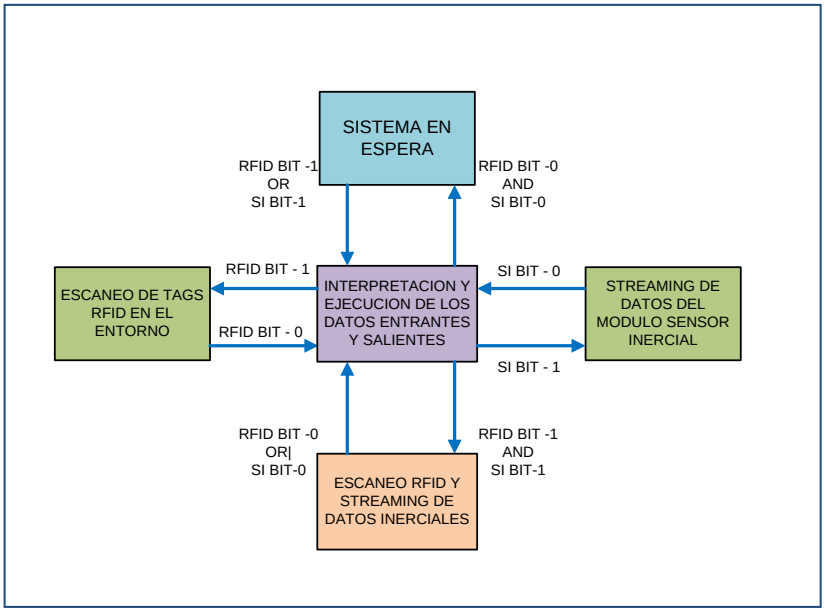


FIGURA 4.1 - Maquina de estados

Se estima por tanto que este consta de cuatro modos principales, que son:

Sistema en espera: Todos los dispositivos a excepción del microcontrolador, modulo de comunicaciones Zigbee y elementos de interfaz simple, se encuentran en un estado de reposo correspondiente a su principal modo de bajo consumo definido por su hardware.

El Z-Mota MOVE 2.0 no ofrece en este caso ninguna funcionalidad excepto la de recibir información a través del modulo Zigbee o mediante la modificación del estado del pulsador, y procesarla. Lo que le permite al sistema realizar una transición a cualquiera de los otros tres estados disponibles según la actuación del usuario, así como la modificación de la configuración básica en los citados modos de operación.

El sistema se posiciona siempre en este estado tras una puesta en funcionamiento o reset, ya sea este realizado mediante software o hardware, por lo que puede considerársele el modo base o modo por defecto.
El sistema se posicionara siempre en este estado tras las tareas de identificación de los comandos de control a través del host recibidos y su posterior ejecución, captación de datos de los sensores y escaneo de TAGs RFID, regresando automáticamente al estado de funcionamiento anterior en base a la frecuencia de operación marcada en este último.

Streaming de datos de los sensores inerciales: Parte del estado de sistema en espera, manteniendo las funcionalidades ya citadas, junto con la activación del conjunto de sensores inerciales formado por ADXL345, HMC5843 e ITG-3200.

Si no se ha realizado ninguna reconfiguración previa, estos comenzaran a realizar lecturas con la periodicidad marcada por defecto, de datos relativos a la posición y movimiento del Z-Mota MOVE 2.0, enviando dichos datos al sistema host mediante el protocolo Zigbee.

Escaneo de TAGs RFID: Parte del estado de sistema en espera, manteniendo las funcionalidades ya citadas por este, junto con la activación del lector RFID Skyemodule M1-Mini.

Si no se ha realizado ninguna reconfiguración previa, este realizara una búsqueda periódica de etiquetas RFID (TAGs) compatibles dentro de su campo de acción, enviando la identificación de estas al sistema host solo ante una detección satisfactoria.

Escaneo de TAGs RFID y Streaming de datos de los sensores inerciales: Como su nombre indica, en este estado se encuentran activados tanto el conjunto de sensores ADXL345, HMC5843 e ITG-3200 como el lector RFID Skyemodule M1-Mini , obteniendo el usuario todas las funcionalidades que provee el ZMota MOVE al mismo tiempo.

Serán por lo tanto detectadas etiquetas RFID dentro del campo del lector, a la vez que son realizadas lecturas de los datos relativos a la posición y movimiento del control inercial, siendo transmitidos ambos al sistema host mediante el protocolo Zigbee.

Es posible realizar un cambio en el modo de operación del dispositivo ya sea mediante comunicación Zigbee o por medio de una modificación en el estado del pulsador en cualquiera de los modos de operación expuestos.

Se emplearan los dos Leds de señalización para denotar el modo de operación, según la tabla expuesta en la Figura [**4.2**]. Los cambios de modo de operación mediante el pulsador se realizaran de forma secuencial, siguiendo el orden marcado por dicha tabla.

Modo de operación	LED 1	LED 2
Sistema en espera	0	0
Streaming de datos de los sensores inerciales	0	1
Escaneo de TAGs RFID	1	0
Streaming de sensores inerciales y escaneo RFID	1	1

FIGURA 4.2 - Estado de los LED según modo de funcionamiento

Una actuación prolongada sobre el pulsador de control posiciona el sistema en un modo de reposo profundo desconectado todos los dispositivos.

Conociendo los modos de operación, puede abordarse la realización de un diagrama de bloques inicial que describa de forma aproximada los procesos internos requeridos por el dispositivo para alcanzar el funcionamiento establecido, tal y como se observa en la Figura [**4.3**].

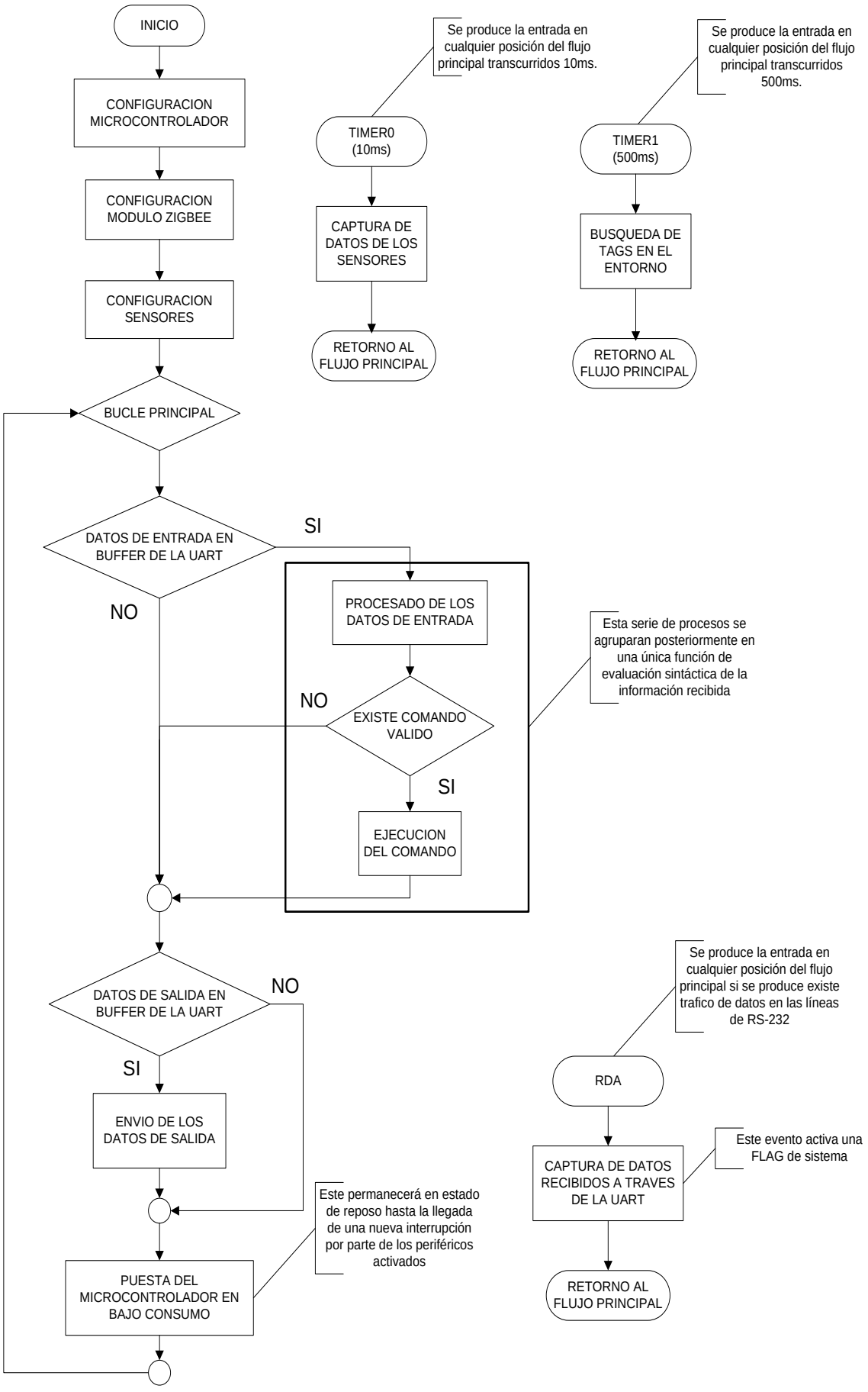


FIGURA 4.3 - Diagrama de flujo general del software implementado en el Z-Mota Move 2.0

4.2 ESTRUCTURACION DEL FIRMWARE

En el siguiente paso, el firmware del dispositivo es planteado de nuevo desde una diferente perspectiva, tomando en esta ocasión como punto de referencia los niveles de abstracción empleados para su desarrollo, tal y como se observa en la Figura [4.4].

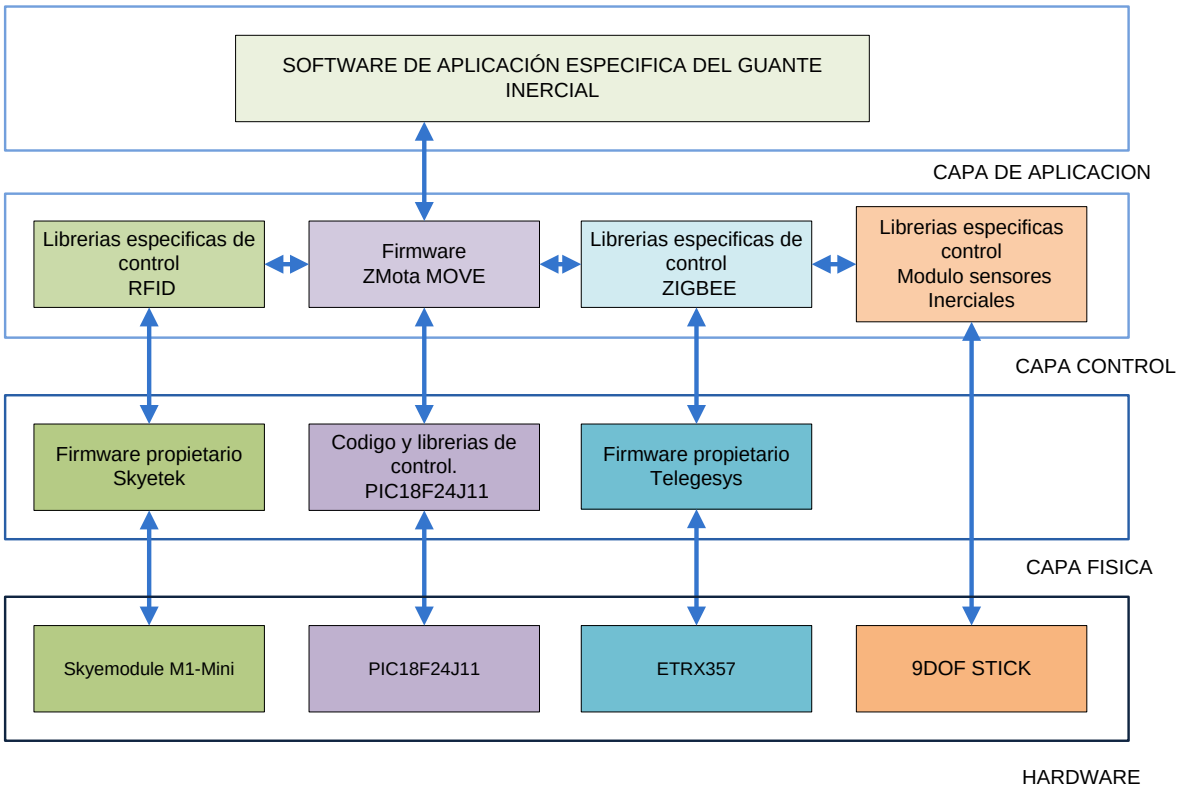


FIGURA 4.4 - Estructura de las capas de aplicación del software implementado

Realizar esta división permite abordar tanto el desarrollo como las modificaciones posteriores, ya sean estructurales y de funcionamiento general o especificas dentro de un proceso concreto, de manera aislada. Es decir, pueden elaborarse independientemente el código de cada bloque, siendo solo necesario definir los datos que cada función precisara tanto de entrada como de salida.

Cada una de las capas determinadas en este proyecto se ha enfocado a la realización de tareas a diferentes niveles, siendo sus características las resumidas a continuación.

HARDWARE: Se encuentra en el nivel más bajo y está compuesta por el conjunto de componentes electrónicos capaces de ofrecer una funcionalidad mínima.

CAPA FISICA: Sirve de enlace entre la capa de hardware y la de control. Su objeto es el de agrupar conjuntos de operaciones directas sobre el hardware de tal modo que estos sean percibidos en la capa superior como una sola.

Estas operaciones son realizadas a niveles de registros, bits individuales y líneas de control, estando condicionadas fuertemente por las especificaciones y características de los propios componentes.

CAPA DE CONTROL: Enlaza la capa física con la capa de aplicación. Permite actuar sobre los procesos de la capa física con un nivel mayor de abstracción, lo que ofrece la posibilidad de realizar operaciones sobre el hardware sin tener una comprensión completa de su funcionamiento interno.

Las librerías engloban las funciones especificas para el manejo de un componente concreto, mientras que el firmware del Z-Mota Move 2.0 sirve de nexo entre estas, el código implementado para el control del procesador y la capa superior de aplicación.

CAPA DE APLICACIÓN: Por ultimo la capa de aplicación sirve como soporte para la realización de los procesos finales que determinan el funcionamiento del Z-Mota Move 2.0, apoyándose las funciones que su firmware propio le ofrece para ello.

De forma práctica se ha realizado una organización del sistema de archivos que conforma el diseño completo, asignando las diferentes competencias a cada uno de los mismos según los criterios enunciados en la estructura anteriormente presentada, tal y como puede apreciarse en la Figura [**4.5**].

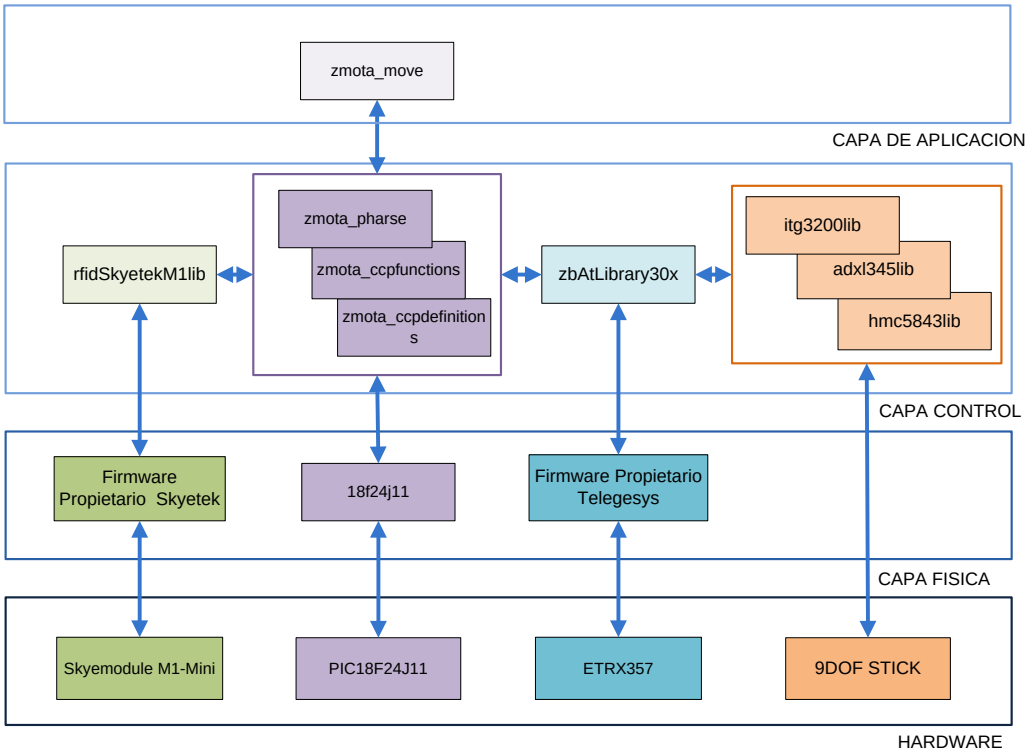


FIGURA 4.5 - Ubicación de los distintos archivos según sus competencias

El contenido de estos archivos será explicado con más detalle en los distintos apartados destinados a mostrar las soluciones individuales adoptadas para la realización de este proyecto. Además de encontrarse disponible la totalidad del código para su consulta en la sección F de esta memoria dentro de anexos.

La dependencia de comunicación entre las funciones contenidas dentro de los distintos archivos es indicada con más detalle en la Figura [**4.6**].

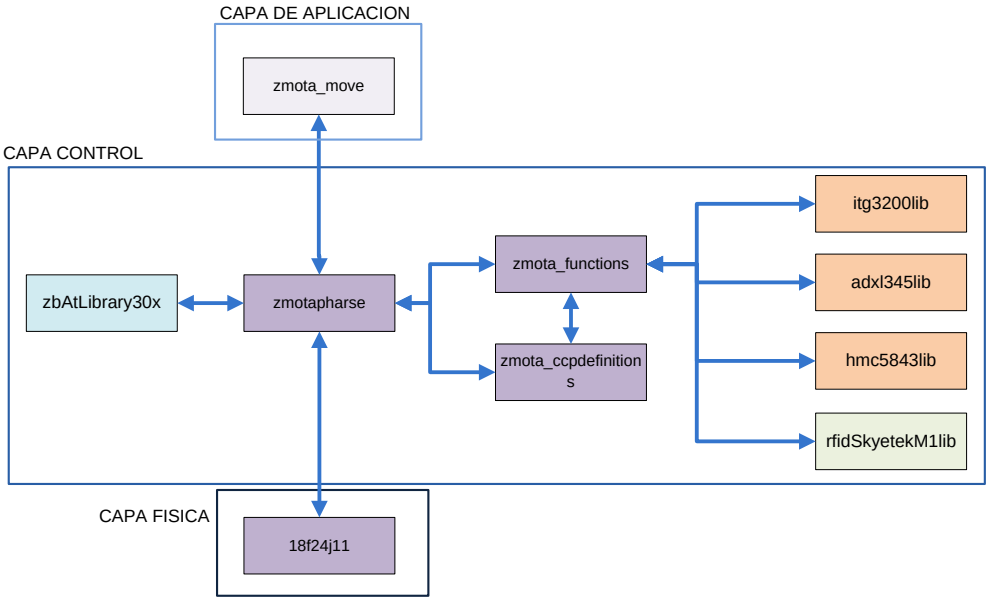


FIGURA 4.6 - Jerarquía del software implementado en el Z Mota Move

Lo cual significa que una función solo podrá transmitir información con otra siguiendo el camino definido en este diagrama. Por lo que por ejemplo, una función de la capa de aplicación, no podrá comunicarse con el hardware de otro modo que no sea a través de funciones establecidas en las capas intermedias hasta llegar a este.

En la realización de este proyecto ha sido necesario realizar partiendo de cero el código de:

- El software de aplicación que conforma la capa de aplicación
- El software correspondiente a la implementación del perfil Zigbee OSGi4AmI
- El software de control del microcontrolador PIC18F24J11
- Las librerías de control para el modulo RFID Skyemodule M1-Mini
- Las librerías de control para el modulo del conjunto de sensores ADXL345, HMC5843 e ITG-3200

Mientras que han sido proporcionadas por HOWLab las librerías de control del modulo ZIGBEE ETRX357.

4.3 PROTOCOLO DE COMUNICACIONES CCP

OSGi4AmI es una ontología para integración de dispositivos en sistemas basados en Inteligencia Ambiental (AmI), su nombre es debido a que se ha implementado como una colección de interfacesJava para su uso en desarrollo con tecnología OSGi (**O**pen **S**ervices **G**ateway **I**nitiative).

Define una formalización para los dispositivos, de tal modo que estos pueden ser utilizados por las aplicaciones de forma transparente, independientemente del fabricante del dispositivo, permitiendo centrarse únicamente en la funcionalidad que requiere la aplicación desarrollada.

Es decir, OSGi4AmI proporciona una representación virtual de los dispositivos presentes en el entorno físico, de forma que puedan ser utilizados por las aplicaciones sin necesitar nada que no sea el conocer las características funcionales que ofrecen, dejando explícitamente a un lado detalles de su funcionamiento y configuración interna.

DEVICE COMMANDS	0x00	GET_TYPE
	0x01	GET_CLUSTERS
	0x02	PING
	0x03	RESET
	0x04	ECHO
	0x05	GET_MODE
	0x06	SET_MODE
	0x07	GET_ID
	0x08	SET_ID
	0x09	GET_DESCRIPTION
	0x0A	SET_DESCRIPTION
	0x0B	GET_O4A_END_POINTS
	0x0C	SET_O4A_END_POINTS
	0x0D	DELETE_O4A_END_POINTS
	0x0E	GET_ANNOUNCE
DEVICE EVENTS	0x00	DEV_ACK
	0x01	NOT_SUPPORTED
	0x02	DEV_ERROR
	0x03	DEV_TYPE
	0x04	DEV_CLUSTERS
	0x05	DEV_ECHO
	0x06	DEV_ID
	0x07	DEV_DESCRIPTION
	0x08	DEV_MODE
	0x09	DEV_END_POINTS
	0x0A	DEV_ANNOUNCE

FIGURA 4.7 - Comandos y eventos definidos en el clúster DEVICE (Documentación aportada por HOWLab)

Para conseguir esto, se contemplan atributos y funcionalidades comunes a dispositivos diferentes, agrupándolos en los llamados clúster. Los cuales a su vez engloban un conjunto de operaciones específicas llamados comandos, y sus correspondientes respuestas o eventos según el tipo de hardware definido. Un ejemplo de lo citado puede observarse en la tabla de la Figura [**4.7**] para los comandos y eventos correspondientes al clúster Device.

Para identificar a los diferentes dispositivos se les asigna end points, los cuales permiten una comunicación individual dentro del entorno OSGi4AmI, pudiendo ser elementos independientes o parte integrante de otros sistemas.

Todos los dispositivos compatibles se ajustan a la definición de dispositivo básico o BaseDevice y en función de sus características, pueden ajustarse además a la definición de Sensor, Actuator, SimpleHMI, según naturaleza.

OSGi4AmI propone las bases para definir formalmente un dispositivo, pero da la libertad para definir nuevos dispositivos, incluso individuales, combinando las funcionalidades contempladas por los distintos clúster.

En este proyecto la comunicación del host con los distintos dispositivos del entorno se realizara mediante Zigbee. Siguiendo las estructuras de los mensajes un protocolo de comunicaciones CCP (**C**luster-Based **C**ommunication **P**rotocol), compartiendo una estructura similar a la ontología OSGi4AmI, tal y como puede apreciarse en la Figura [**4.8**],

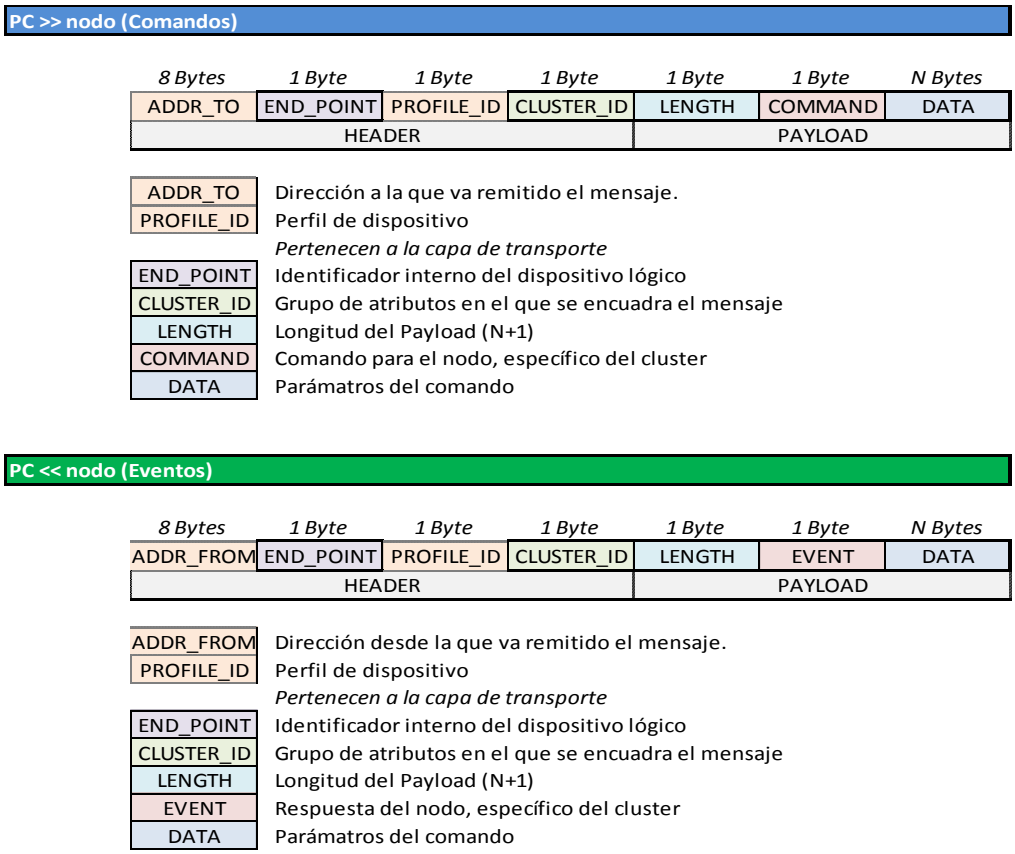


FIGURA 4.8 - Estructura de un mensaje de forma acorde al perfil HOWLab OSGi4AmI (Documentación aportada por HOWLab)

En el desarrollo del software de este proyecto, un perfil tipo OSGi4AmI ha sido codificado dentro de la capa de control en los siguientes archivos:

"zmota_pharse.c", el cual gestiona el trafico de mensajes del perfil y del que caben resaltarse principalmente sus funciones:

> ***"ccp_parse_input"*** - encargada de analizar la sintaxis del mensaje recibido a través del host (PC) y en caso de identificación satisfactoria de un comando valido, el lanzamiento del mismo

dentro de la función correspondiente ubicada en el archivo *" zmota_functions.c",* o en caso contrario proceder con la devolución del consiguiente error.

"ccp_build_response"- encargada de montar el mensaje que será enviado al host con la información del evento resultante de la acción realizada por la anterior función.

"zmota_functions.c", archivo que contiene las llamadas a las funciones que integran las librerías de control del lector RFID *"rfidSkyetekM1lib.c",* así como a las librerías correspondientes a los tres componentes que componen el modulo de captación de movimiento *"adxl345lib.c", "itg3200lib.c"* y *"hmc5843lib.c"*, traduciendo tanto el comando como los parámetros recibidos por el host tal y como las precisan.

Además de otras necesarias para atender las peticiones seleccionadas a través de *"ccp_parse_input",* para operaciones del dispositivo base, englobadas dentro de los clúster DEVICE y POWER.

Y por ultimo ***"zmota_ccpdefinitions.h",*** que contiene las variables, definiciones y constantes que son empleadas por las funciones ubicadas en los archivos *" zmota_pharse.c"* y *" zmota_functions.c"*.

Para el tratamiento de la información contenida en el header y payload de dichos mensajes dentro de estas dos funciones, así como en las implementadas en el archivo ***" zmota_functions.c",*** se hace uso de dos estructuras, *"ccp_buffer_in"* y *"ccp_buffer_out".*

Estas contienen como elementos individuales los correspondientes campos equivalentes al formato indicado en la Figura [**4.8**], así como un bit de estado o flag que indica la existencia de datos de parámetros de un tamaño variable, junto con un espacio correspondiente para los datos sin estructura fija para su almacenamiento siempre que sea requerido.

La identificación de los comandos dentro del mensaje se realiza campo por campo, junto con una comprobación de la correspondencia de su longitud con el valor determinado por length, contando el usuario con confirmaciones y señalizaciones de error que ayudan en gran medida a una rápida adaptación al uso del perfil CCP, orientado a alcanzar la compatibilidad con OSGi4AmI. Estos procedimientos pueden verse de forma esquemática en el diagrama de flujo de la Figura [**4.9**].

Los clúster, así como sus comandos y eventos correspondientes incluidos en el Z-Mota Move 2.0 son los siguientes: DEVICE, POWER, SENSOR, SENSOR AUTO REFRESH, SIMPLE HMI INPUT, RFID READER.

Pero dada la versatilidad ofrecida por el perfil CCP, existe la posibilidad de poder ampliar en un futuro las funcionalidades del sistema final, permitiendo un mayor número de comandos por clúster, o incluso aumentar el número de estos últimos mediante la integración en el hardware de nuevos componentes, tal y como se pretende conseguir en futuras revisiones.

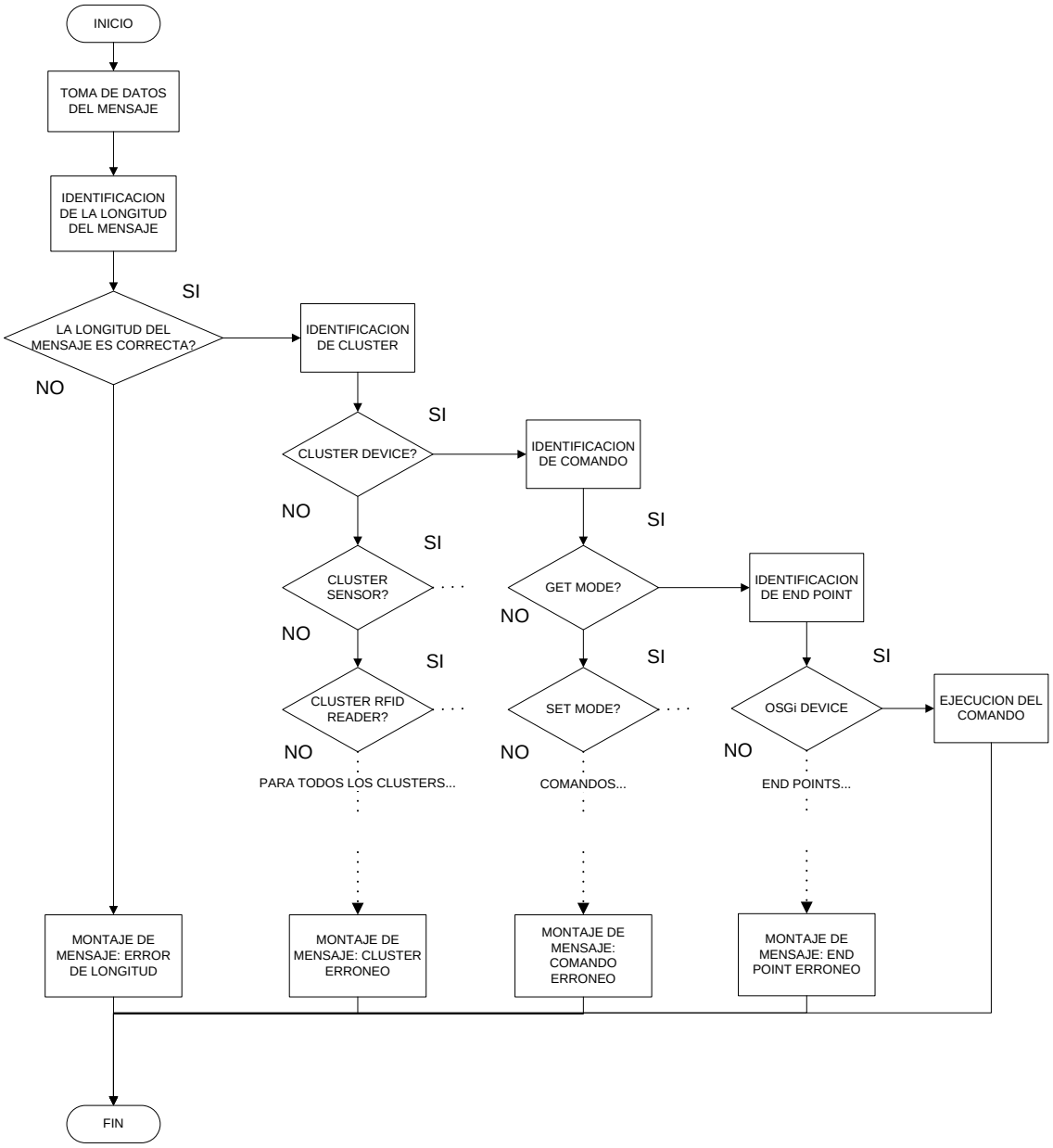


FIGURA 4.9 - Diagrama de flujo de los procesos realizados por la función "ccp_pharse_input"

La comunicación a través Zigbee se realiza mediante una llamada a *"zbSendDataToZigbeeSink"* desde ***"zmota_pharse.c"***, función de envío de datos contenida en la librería *"zbAtLibrary30x.c"* utilizando como medio un array de dato especifico para las recepciones y envíos a través de la UART del microcontrolador, siendo cedida su referencia mediante el traspaso de su dirección como puntero entre las funciones ***"ccp_parse_input.c"*** y ***"ccp_build_response"*** dependiendo del sentido de los citados mensajes.

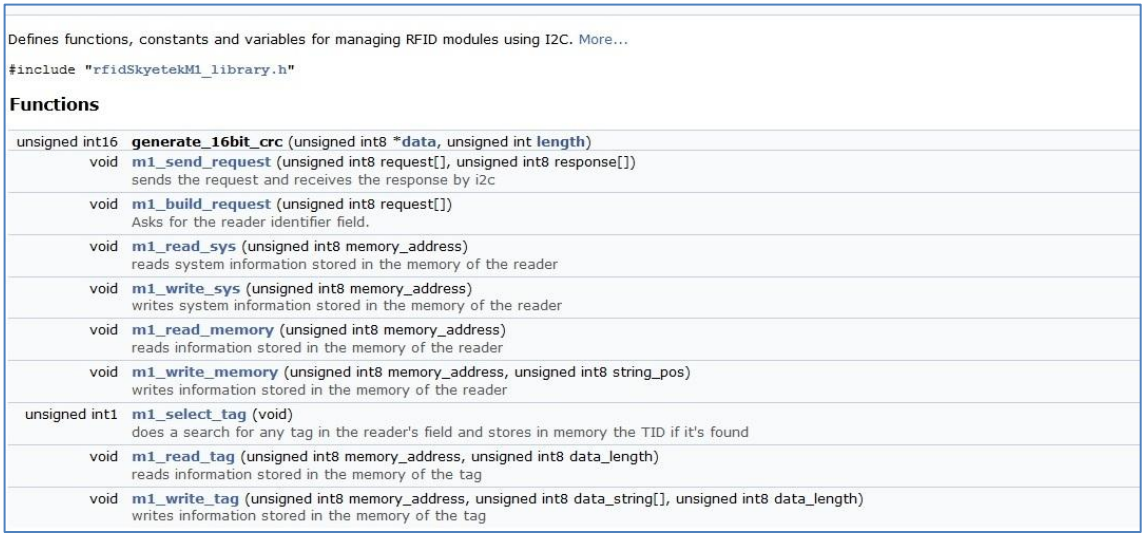
4.4 DESARROLLO DE LAS LIBRERIAS DE CONTROL RFID

Ciertos componentes integrados en el diseño final del Z-Mota Move 2.0 ya disponen de un firmware propio, los cuales ofrecen mayor o menor versatilidad para el desarrollo de aplicaciones tanto de bajo y medio nivel. Por tanto, una gran parte del diseño de software se centra en compatibilizar o adaptar las funcionalidades que estos nos ofrecen.

El lector RFID Skyemodule M1-Mini de Skyetek cuenta con un protocolo que delimita todos los aspectos necesarios para realizar una comunicación satisfactoria con el mismo, ya sea mediante I2C, SPI o conexión directa con el host mediante USB, con un formato de datos en ASCII o binario. Para más Información relativa al citado protocolo puede consultarse la sección C dentro de anexos.

La interacción con el lector Skyemodule M1-Mini se ha implementado en este proyecto para operar mediante comunicación series síncrona bajo el estándar I2C. Los procesos necesarios han sido codificados dentro de la capa de control como librerías específicas en los siguientes archivos:

"rfidSkyetekM1lib.c", cuyas funciones pueden observarse la Figura [**4.10**]. Gestiona el trafico I2C y construye los mensajes según el protocolo de Skyetek dependiendo del tipo de comando solicitado. Este cuenta con un juego de funciones que abarcan todas las operaciones disponibles realizables en el lector a bajo nivel, pero de entre ellas cabe resaltar las dos que sirven de base:



Defines functions, constants and variables for managing RFID modules using I2C. More...

`#include "rfidSkyetekM1_library.h"`

Functions

unsigned int16	**generate_16bit_crc** (unsigned int8 *data, unsigned int length)
void	**m1_send_request** (unsigned int8 request[], unsigned int8 response[]) sends the request and receives the response by i2c
void	**m1_build_request** (unsigned int8 request[]) Asks for the reader identifier field.
void	**m1_read_sys** (unsigned int8 memory_address) reads system information stored in the memory of the reader
void	**m1_write_sys** (unsigned int8 memory_address) writes system information stored in the memory of the reader
void	**m1_read_memory** (unsigned int8 memory_address) reads information stored in the memory of the reader
void	**m1_write_memory** (unsigned int8 memory_address, unsigned int8 string_pos) writes information stored in the memory of the reader
unsigned int1	**m1_select_tag** (void) does a search for any tag in the reader's field and stores in memory the TID if it's found
void	**m1_read_tag** (unsigned int8 memory_address, unsigned int8 data_length) reads information stored in the memory of the tag
void	**m1_write_tag** (unsigned int8 memory_address, unsigned int8 data_string[], unsigned int8 data_length) writes information stored in the memory of the tag

FIGURA 4.10 - Funciones incluidas dentro de "rfidSkyetekM1lib.c", (captura de la documentación realizada para la librería RFID elaborada con Doxygen)

"m1_build_request " - encargada de la construcción de los mensajes en un array de datos que posteriormente será tratado por ***"m1_send_request",*** escribiendo los campos correspondientes a la información necesaria para una correcta comunicación.

"m1_send_request" - encargada del envío de los datos empaquetados previamente a través de I2C, mediante llamadas a operaciones de hardware a través del código de la capa física del microcontrolador.

"rfidSkyetekM1lib.h", que contiene las variables, definiciones y constantes que son empleadas por las funciones ubicadas en el archivo *" rfidSkyetekM1lib.c "*.

En su funcionamiento, las librerías RFID se apoyan esencialmente en dos estructuras de datos, *"m1_buffer"* y *"m1_request_fields",* tal y como se puede apreciar en la Figura [**4.11**],

FIGURA 4.11 - Estructuras "m1_buffer" y "m1_request_fields", (captura de la documentación realizada para la librería RFID elaborada con Doxygen)

"m1_buffer" es empleada como almacenamiento temporal de todos y cada uno de los campos necesarios para la formación de los mensajes de salida según el protocolo, así como también de los de entrada.

"m1_request_fields" señala mediante bits de estado o flags, la necesidad de la inclusión o no de los diferentes campos según lo impuesto por el tipo de comando RFID empleado.

De este modo el resto de las funciones codificadas para la implementación de cada uno de los comando RFID disponible con este hardware basan su funcionamiento, solo operaran mediante la modificación de los valores requeridos, aportando los datos a transmitir y tomando los datos recibidos a través del lector o TAG. De modo aclaratorio se expone en la Figura [**4.12**], un diagrama de flujo aproximado de los procesos seguidos para la ejecución de uno de los comandos RFID.

En su estado actual, las librerías de control RFID realizadas, permiten la ejecución de todas las operaciones dispuestas por el firmware de SKYETEK para el lector RFID M1 MINI independientemente de su implementación final o no en la capa de aplicación del software Z-Mota Move 2.0.

Además se ha trabajado fuertemente por la implementación de funcionalidades no aseguradas por el fabricante, como son la realización de inventarios de TAGs automatizados empleando la comunicación I2C con el host, existiendo la previsión de llegar a alcanzar su realización en futuras revisiones. Puede consultarse más información sobre el realizado código junto con sus comentarios para una mayor comprensión del desarrollo llevado a cabo en este proyecto en la sección F.

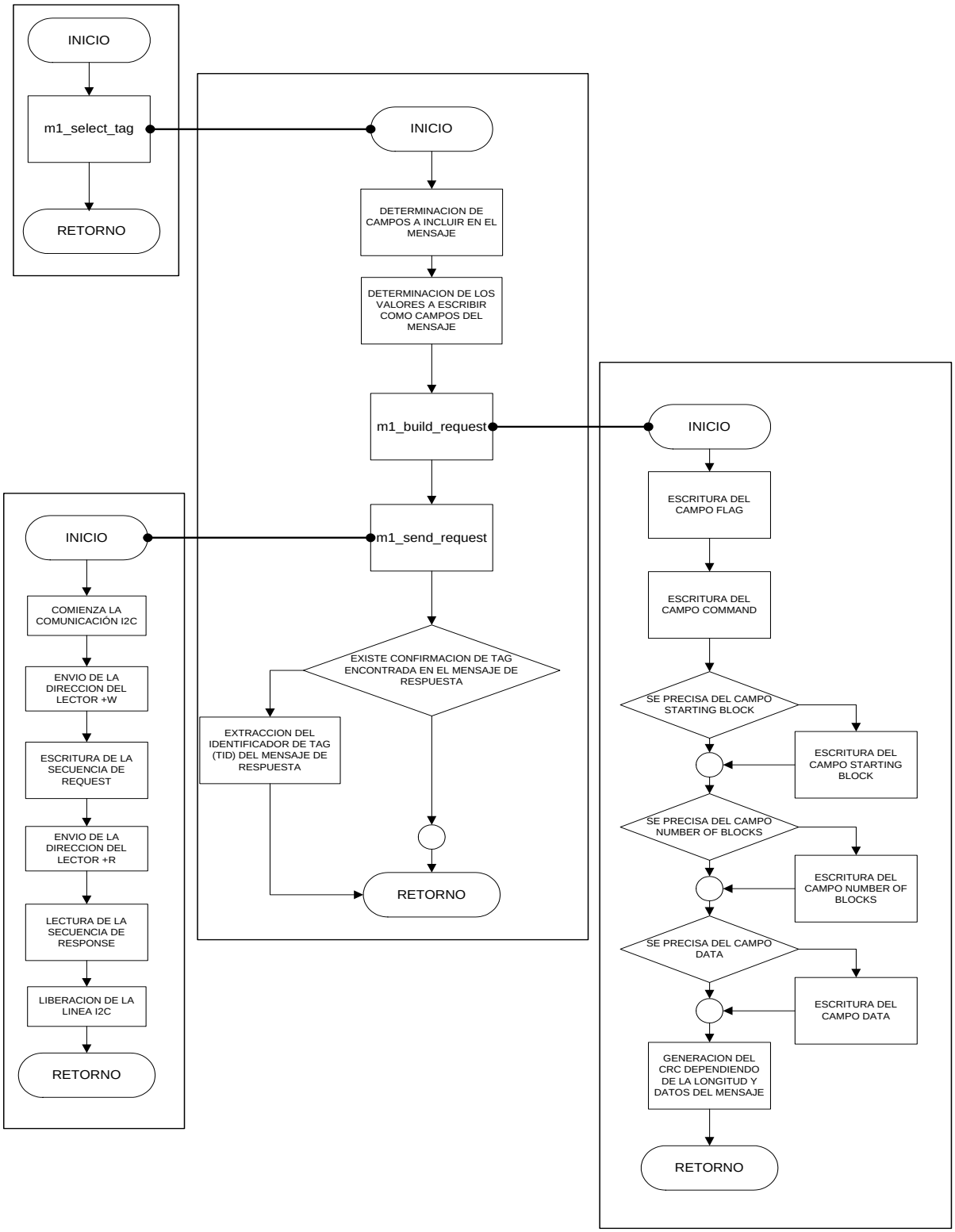


FIGURA 4.12 - Diagrama de flujo de procesos requeridos para la ejecución de SELECT_TAG

4.5 DESARROLLO DE LAS LIBRERIAS DE CONTROL DEL MODULO DE SENSORES INERCIALES

El control realizado para los distintos componentes que conforman el modulo de sensores inerciales comparte ciertas características comunes. Estas además de sus particularidades individuales serán expuestas a continuación para una mayor comprensión del desarrollo del software.

Los procesos necesarios han sido codificados dentro de la capa de control como librerías especificas en los archivos, siguiendo el procedimiento de distribución que ubica a las funciones en el que cuenta con extensión ***".c"*** así como las variables, definiciones y constantes que son empleadas en él ***".h"***

La correspondencia de librerías con su dispositivo se corresponde con las siguientes:

- ***"adxl345lib.c"y "adxl345lib.h"*** , se centran en el control del acelerómetro ADXL345.
- ***"itg3200lib.c"y "itg3200lib.h"*** , se centran en el control del acelerómetro ITG-3200.
- ***"hmc5843lib.c"y " hmc5843lib.h"*** , se centran en el control del acelerómetro HMC5843.

El control de los diferentes sensores se realiza mediante la escritura de ciertos registros de su memoria interna mediante una transmisión conforme al estándar I2C.

Al contrario de lo que sucedía en los casos del modulo Zigbee y RFID, esta comunicación se realiza de manera directa, sin la necesidad de una conformación de los datos en base a ningún protocolo propietario. Por ello cabe destacar que en la codificación realizada se han construido dos funciones tipo.

Estas toman el nombre de *"..._ read_register "* y *"..._ write_register"* precedidas de las siglas de su dispositivo sensor y ejecutan las operaciones necesarias para la comunicación entre sensores y microcontrolador. Se ha incluido información complementaria acerca de los procesos de comunicación I2C en la sección B dentro de anexos.

La funcionalidad básica de las librerías de control de sensores es la de automatizar los accesos a las muestras de datos de aceleración, inclinación, y campo magnético que estos proveen. Para la toma de estos datos se han implementado funciones con una estructura similar, pero respetando individualmente las particularidades de la lectura de las muestras que ofrecen los sensores.

Estas han sido definidas como: ***"adxl_capture_axis_data", "itg_capture_gyros_data", "hmc_capture_channel_data".***

Todas ellas almacenan los datos recibidos por una muestra única de los sensores en el momento de su llamada, tanto un una estructura correspondiente a sus tres valores de medida (ejes), como en un array.
Esto permite que pueda ser implementados nuevos procesos que requieran la realización de varias tomas de muestras de datos y su transmisión en paquetes, tal y como sucede en la aplicación desarrollada Z-Mota Move 2.0.

En las librerías se incluyen funciones que realizan de forma automática la configuración del funcionamiento de los dispositivos sensores. Han sido realizadas evaluando la forma de trabajo optima para cada uno de ellos individualmente, y siendo definidas como: ***"adxl_configure_axis_data", "itg_configure _gyros_data", "hmc_configure _channel_data".***

Tal y como se desarrolla en la sección D dentro de anexos, el manejo de los sensores está ligado a la modificación de ciertas posiciones de su memoria interna. Si la configuración realizada por las funciones presentadas no fuese la requerida para una determinada aplicación, dentro de los archivos ***".h"*** se incluyen las definiciones de algunos otros patrones de configuración comunes.

Además, se han realizado estructuras de datos predefinidas para la modificación de los bits correspondientes dentro de los registros específicos de configuración de una manera más accesible, de tal forma que se puede ejercer un control total del funcionamiento de los diferentes sensores de una manera rápida y clara.

Un ejemplo claro de esto se ofrece en la representación del diagrama de flujo parcial en la Figura [**4.13**], como ejemplo para una reconfiguración de ADXL345.

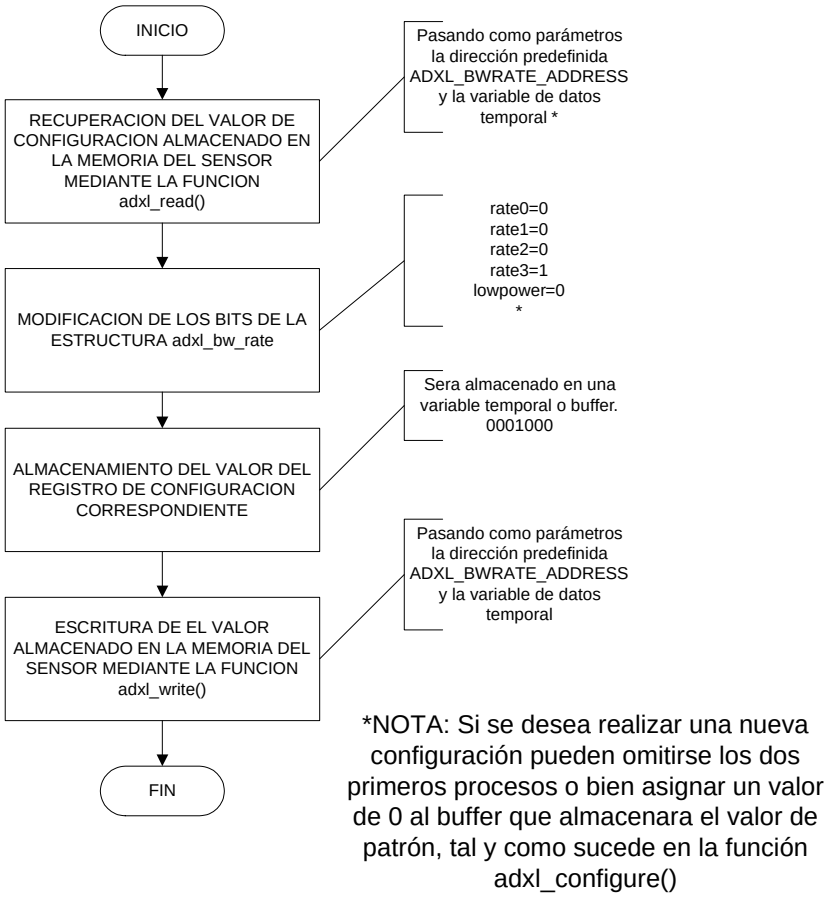


FIGURA 4.13 - Diagrama de flujo del proceso de modificación de un resgistro de configuración del ADXL345

CAPITULO 5 DESCRIPCIÓN DEL DISEÑO DE LA PCB

5.1 Criterios de diseño

Durante este capítulo se describen los criterios y decisiones tomadas para la realización de la placa de circuito impreso que dará soporte a los diferentes componentes que conforman el diseño del Z-Mota Move 2.0.

Para el diseño de la PCB se ha utilizado como herramienta la suite de diseño Altium Designer Build 8. La fabricación de la misma ha corrido a cargo de los maestros de taller del Departamento de Ingeniería Electrónica y Comunicaciones de EINA (**E**scuela de **In**geniería y **A**rquitectura, Universidad de Zaragoza)

En un proyecto de estas características la mayor restricción esta impuesta por la necesidad de que las dimensiones finales del mismo sean los reducidas posible, ya que tal y como es un control inercial que permanecerá adherido a la mano del usuario final, no debe resultar inconfortable en su uso ya sea por forma, peso o tamaño.

Tal y como se ha expuesto de forma reiterada en diferentes puntos del capítulo 2, se ha realizado una selección de componentes SMD del menor tamaño posible siempre y cuando su precio no desaconsejase mantener ese criterio. Por lo tanto tenemos que:

- En componentes pasivos resistivos se ha empleado un encapsulado 0603, así como en los LED.
- En componentes pasivos capacitivos se ha empleado un encapsulado 0804.
- En el diodo de se corresponde con un SOT-23.
- Mientras que en los integrados de mayor patillaje se ha optado por el empleo de SOIC, 28 para el microcontrolador y 8 para la memoria.

Todos los demás componentes que poseen medidas no estandarizadas poseen un encapsulado SMD. Puede apreciarse un detalle de la relación de tamaños para los componentes pasivos en la Figura [**5.1**].

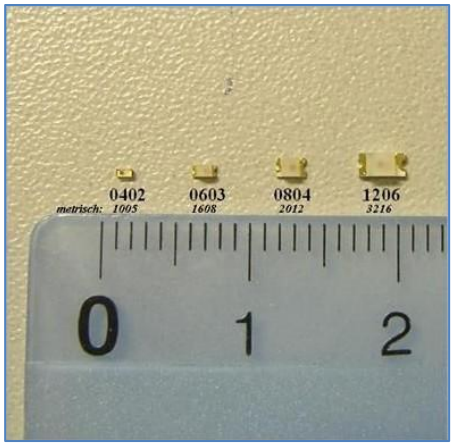


FIGURA 5.1 - Comparativa de dimensiones SMD

Para el ruteado de la placa de circuito impreso, dada la tecnología de fabricación disponible, se han impuesto las siguientes reglas:

- Posibilidad de fabricación a dos caras
- Ancho de pista mínimo de pista 0,3mm
- Distancia mínima entre pistas 0,3mm
- Posibilidad de utilización de vías con paso metalizado
- Diámetro de taladrado mínimo de 0,2mm

A partir de los mínimos, los principales criterios adoptados para la realización de la placa han sido:

- Anchura mínima de las pistas de alimentación de 0,5mm
- Ruteo en pistas de alimentación en base a conseguir un retorno lo más corto posible, evitando realizar bucles amplios.
- Los condensadores de filtrado de los integrados deberán estar lo más próximos posible a ellos.
- Realización de plano de masa en ambas caras, a ser posible rodeando completamente esta para un mejor aislamiento.
- Evitar el trazado de pistas cerca de los bordes de la placa.
- Evitar el trazado de pistas que transcurran de forma paralela distancias considerables. En caso no poder cumplirse este criterio, es recomendable trazar una salvaguarda conectada a masa entre las mismas para evitar interferencias tipo crosstalk.
- Evitar cambios de dirección bruscos en el trazado (menores que 45º) en el sentido estimado para la corriente de línea. Es recomendable realizar uniones tipo Y.
- Las terminaciones a pads siempre serán directas (sin ángulos), a ser posible empleando un diseño tipo lagrima a razón de evitar reflexiones.
- Posicionado de los componentes optimo para realizar un ruteo corto.
- Colocación de conectores en lugares fácilmente accesible siendo recomendable su ubicación en los bordes de la placa.

Existen además consideraciones particulares dadas las características de los componentes seleccionados en este proyecto:

Tal y como recomienda el fabricante en su documentación técnica, se deberá asegurar que no existe ningún componente ni cobre bajo la antena del ETRX357, siendo recomendable dejar libre un espacio mayor para un cierto margen de seguridad,

El modo de conexión y la disposición del modulo de sensores de Sparkfun sobre la placa de circuito impreso impone la restricción de que no puedan ubicarse elementos visuales o susceptibles de manipulación en su misma cara. Por lo que se descarto completamente la colocación de los indicadores LED así como el pulsador. Aunque debido a su separación si es posible la de pequeños componentes pasivos, así como la del conector de programación.

Siendo este junto con el modulo Zigbee los componentes de mayor volumen, se intento una disposición de componentes y un dimensionado de la placa ajustado a los valores límite impuestos por estos. Para un mayor aprovechamiento del tamaño se estimaron dos formatos inicialmente, en perfil alargado o

totalmente cuadrado, optándose en última instancia por este último en base a la previsión de una mayor confortabilidad y ergonomía del producto final.

5.2 Descripción general

En total se realizaron 4 versiones, pudiendo observarse las distintas variaciones de las mismas en la Figura [5.2], así como su resultado final reflejado en las Figura [5.3].

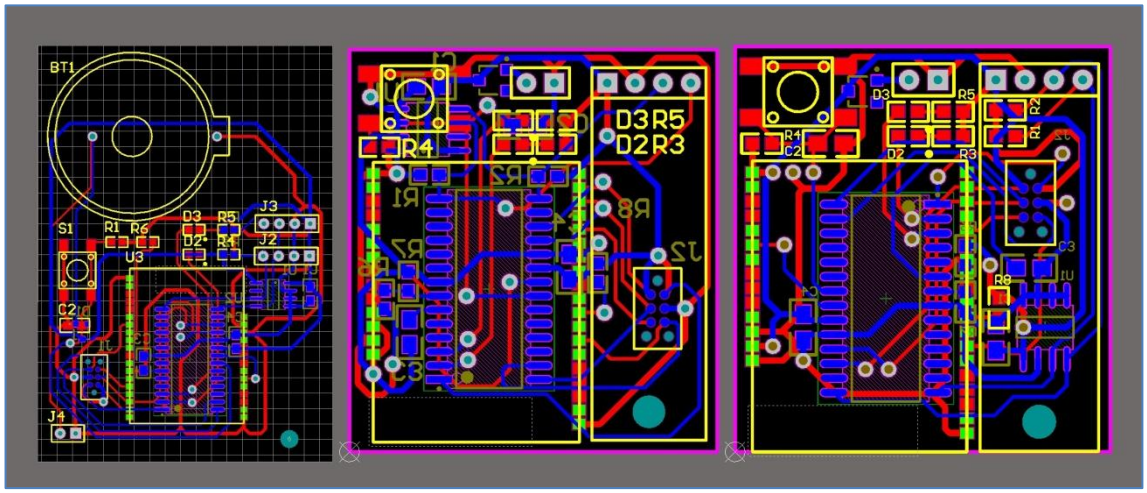
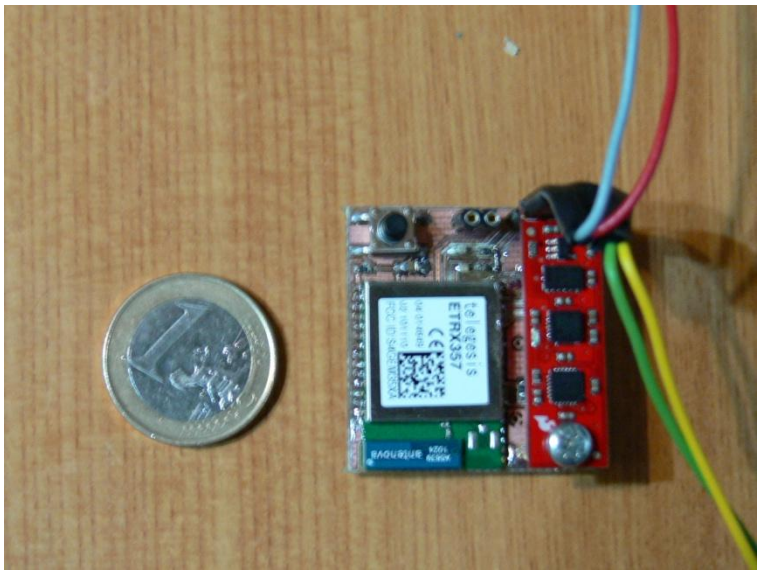


FIGURA 5.2 - Diferentes progresos en el diseño de la PCB

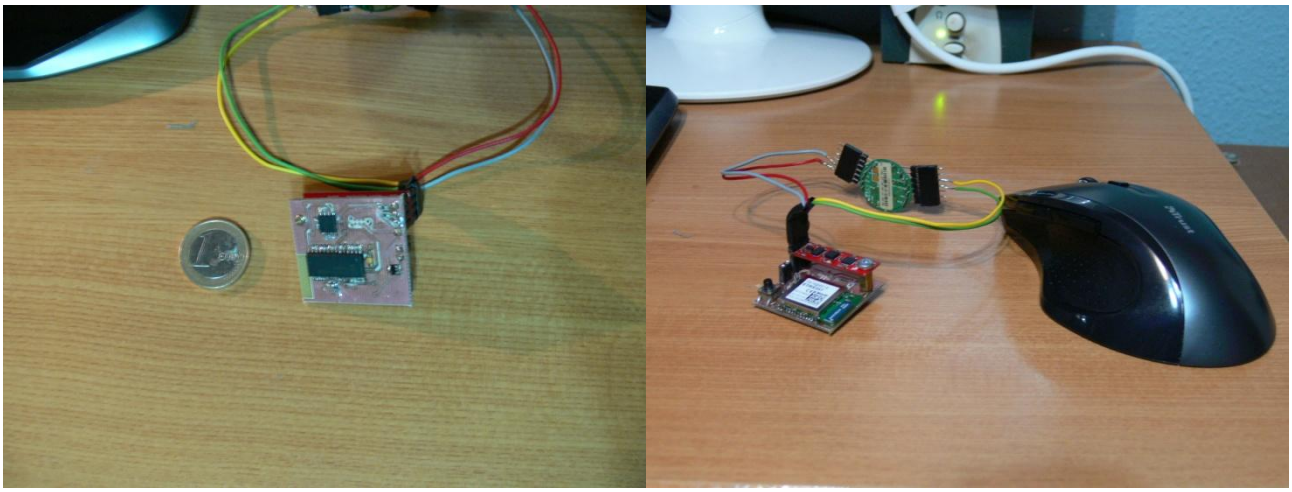
FIGURAS 5.3 - Diseño final de la PCB

Las dimensiones finales corresponden a 32,9mm x 35,5mm con una profundidad no superior a 10mm

El resultado final de la fabricación del primer prototipo puede observarse en las imágenes de la figura [**5.4**] y [**5.5**] .

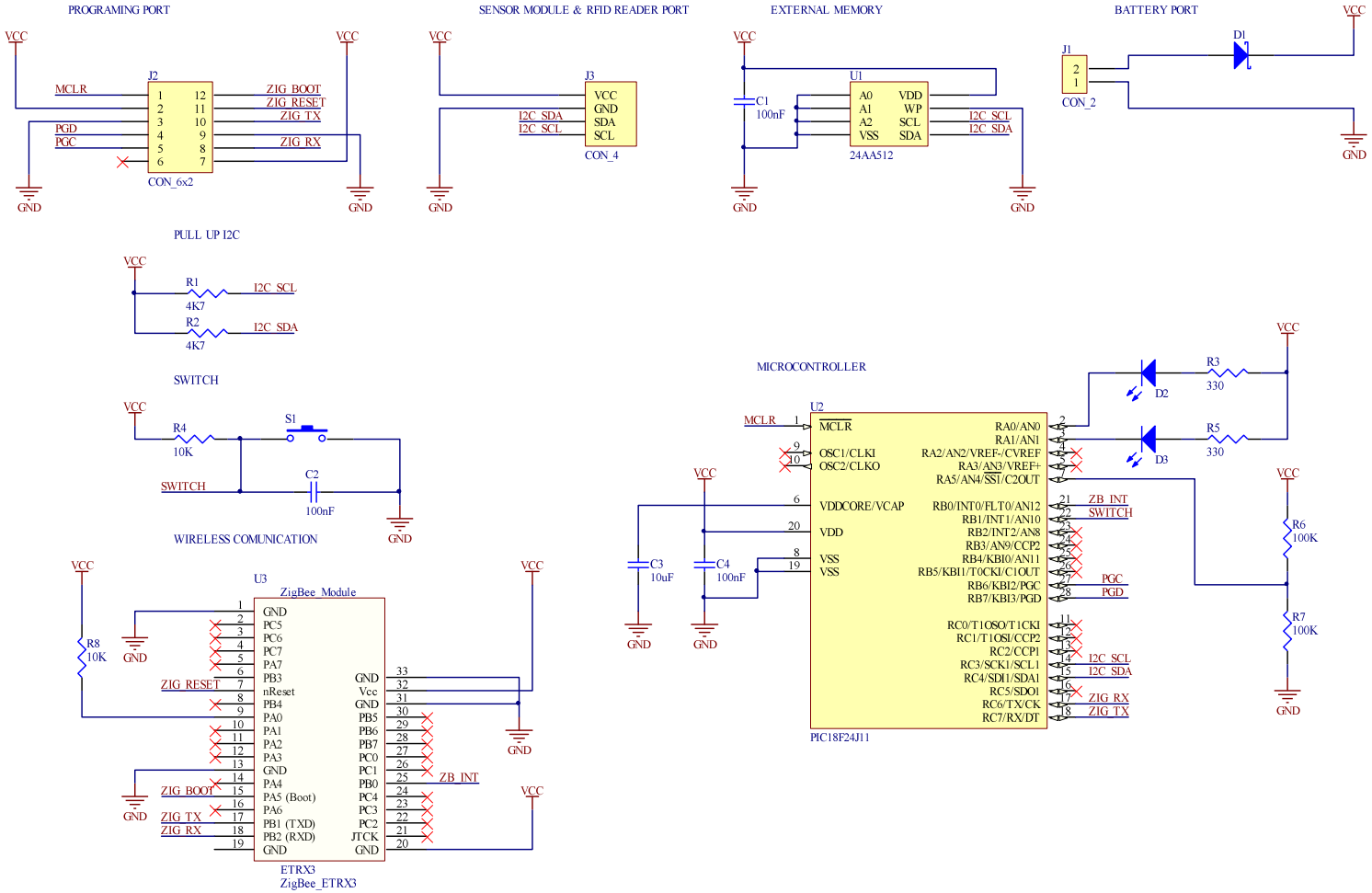


FIGURAS 5.4 - Frontal Z-Mota Move 2.0



FIGURAS 5.5 -Trasera e imagen general del Z-Mota Move 2.0

5.3 ESQUEMA GENERAL DEL CIRCUITO



5.4. DESGLOSE DE PRECIOS DE FABRICACIÓN

Fabricante	Código Farnell	Descripcion	Precio unitaro	Precio u. pedido +250	Precio u. pedido +1000	Cantidad por PCB	Precio por PCB 1 pedido	Precio por PCB +250 pedidos	Precio por PCB +1000 pedidos
VISHAY DRALORIC	1469749	Resistencia, 0603 THICK FILM 10 KOHM ,100mW, 5%	0,019	0,014	0,011	2	0,038	0,028	0,022
VISHAY DRALORIC	1692522	Resistencia, 0603 THICK FILM, 330OHM ,100mW, 5%	0,018	0,018	0,011	2	0,036	0,036	0,022
VISHAY DRALORIC	1739170	Resistencia, 0603 THICK FILM, 4K7 OHM, 100mW, 5%	0,034	0,032	0,028	2	0,068	0,064	0,056
TELEGESIS	1854234	Modulo Zigbee TELEGESIS, ETRX357, CHIP ANT	20,66	20,23	20,23	1	20,66	20,23	20,23
TE CONNECTIVITY / ALC	3801305	SPST SWITCH, 24V, 50mA	0,32	0,27	0,25	1	0,32	0,27	0,25
OSRAM	1226372	DIODE LED, SMD, GREEN, 10MCD, VF 2.2, IFMAX 20MA	0,092	0,057	0,051	2	0,184	0,114	0,102
NXP	1081203	DIODE, SCHOTTKY, SINGLE, VF 260mV	0,098	0,072	0,05	1	0,098	0,072	0,05
MULTICOMP	1759166	Condensador ceramico MLCC, 0805, X7R, 25V, 100NF	0,011	0,011	0,006	3	0,033	0,033	0,018
MICROCHIP	1331301	Memoria 512K I2C CMOS Serial EEPROM	2,63	1,5	1,5	1	2,63	1,5	1,5
MICROCHIP	1706302	Microcontrolador RISC Alto Rendimiento con Tecnologia nanoWatt, 16K	2,9	2,25	2,25	1	2,9	2,25	2,25
AVX	1658720	Condensador ceramico, 0805, CASE R, 4V, 10UF	0,157	0,157	0,157	1	0,157	0,157	0,157
EVE	1365935	EVE - ER14250 - BATTERY, LITHIUM, 1/2AA	3,4	2,6	2,6	1	3,4	2,6	2,6
SPARKFUN	NF (SEN-10183)	9 Degrees of Freedom - Sensor Stick	70,88	55,19	55,19	1	70,88	55,19	55,19
SKYETEK	NF	SkyeModule M1-mini	67,6519	61,3	61,3	1	67,6519	61,3	61,3
NF	NF	Conector de programación	0	0	0	1	0	0	0
NF	NF	Connector de 2 pines	0	0	0	1	0	0	0
NF	NF	Connector de 4 pines	0	0	0	2	0	0	0
						Precio total por PCB	169,0559	143,844	143,747
	*Los precios de la presente tabla estan detallados en euros (€)						Precio total +250	35961	
								Precio total +1000	143747

Puede observarse que la disminución más abrupta del precio de fabricación se produce para un pedido de 250 unidades, donde la diferencia alcanza aproximadamente 30€.

CAPITULO 6 CONCLUSIONES FINALES

En relación a los objetivos generales planteados en un inicio, el cumplimiento de los mismos ha sido alcanzado en los siguientes grados.

Se ha realizado un estudio previo acerca de los diferentes dispositivos de control para interfaces inerciales, para posteriormente realizar una selección de todos los componentes hardware necesarios para el desarrollo de un dispositivo que cumpla con los requisitos funcionales planteados.

Se ha realizado el diseño del esquema electrónico y el diseño de la placa de circuito impreso que sirva como soporte y la fabricación de una primera placa prototipo que integra los componentes seleccionados.

Se ha completado la programación de las librerías de control del lector RFID Skyemodule M1-Mini, así como las librerías de control del conjunto de sensores inerciales ADXL345, HMC5843 e ITG-3200, abarcando la mayor parte de sus funcionalidades. Sin embargo existe la posibilidad de incluir posibles mejoras en las estas, como el añadido de un sistema de inventario automatizado de TAGS o la implementación de un sistema anticolisión mas optimizado para la comunicación I2C, en el caso del lector RFID.

También se ha programado un firmware que permite que la comunicación ZigBee implementada en el dispositivo sea compatible con un protocolo basado en clúster (CCP), junto con una aplicación software que permite la comprobación de la integración y la funcionalidad todos los elementos que conforman el diseño del Z-Mota Move 2.0.

Tanto las librerías como el firmware han sido testeados exhaustivamente en un prototipo basado en una placa de pruebas, realizándose pruebas iniciales en el hardware del prototipo basado en el diseño documentado en la presente memoria.

Por lo tanto se puede asegurar de que se ha desarrollado un periférico en un estado altamente funcional, capaz de actuar como control dentro de un interfaz inercial, gracias a la implementación de sensores específicos de captura de movimiento y medida de posición, con la ventaja añadida de la implementación de un sistema RFID. En conjunción con la capacidad de comunicación inalámbrica mediante Zigbee, con unas reducidas dimensiones, y una autonomía considerable.

Las librerías de control tanto para el modulo RFID Skyemodule M1-Mini como para los componentes sensores ADXL345, HMC5843 e ITG-3200, son totalmente accesibles y permiten, no solo mejores y más complejas aplicaciones futuras para el Z-Mota Move 2.0 sino además, la implementación de estos componentes en otros diseños reduciendo enormemente el trabajo necesario para su integración y uso.

Ha sido solicitada ya información respecto a este dispositivo y mostrado el interés por alumnos de otras especialidades de integrar el Z-Mota Move 2.0 en sus proyectos.

A nivel académico y título personal, durante la realización de este proyecto se ha obtenido una formación y metodología de trabajo solida, orientada a abordar el proceso de desarrollo de un dispositivo con las características planteadas.

Ampliando enormemente los conocimientos de los que se disponía antes del inicio de este proyecto en multitud de facetas tales como:

- sensores e instrumentación
- la programación de microcontroladores y uso de sus periféricos
- el diseño y la realización de placas de circuito impreso
- la elaboración de documentación técnica

Así como adquiriendo conocimientos totalmente nuevos en sistemas de comunicación inalámbricos de todo tipo. Especialmente Zigbee y RFID.

Tomado aun mayor contacto con herramientas informáticas tales como Altium Designer, PCWHD, MPLAB, Office (Word, Excel, Visio). Y por ultimo mejorando la pericia en la soldadura de componentes SMD de reducido tamaño y en procedimientos de testeo, detección y corrección de fallos de funcionamiento.

ANEXO A - CALCULOS SIMPLE HMI

La elección del valor de las resistencias de polarización R4 y R5 está determinada para una corriente de visualización suficientemente clara de los LED LGQ971, con un reducido consumo de corriente durante su tiempo de encendido. Se ha calculado su valor en referencia a la tabla provista en el datasheet de OSRAM mostrada en la Figura [**A.1**], donde se expresa la iluminación como función de la corriente If máxima (20mA) para un límite superior correspondiente a 10mcd (milicandelas).

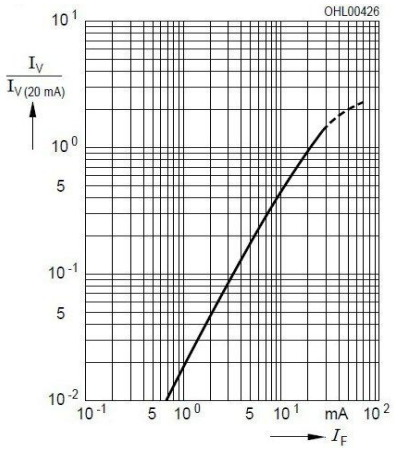


FIGURA A.1 - Grafica de luminosidad frente a Iforward

$$If = \frac{Vcc - Vf}{R} = \frac{3{,}3v - 2{,}2v}{330\Omega} = 3{,}33 \times 10^{-3}A$$

Dando como lugar a una luminosidad aproximada de:

$$L = \frac{Iv}{Iv(20ma)} \times L_{MAX} = 0{,}09 \times 10mcd \approx 1mc$$

La configuración del pulsador de control conectado a la línea RB1 del puerto B del microcontrolador puede apreciarse en las Figura [**A.2**].

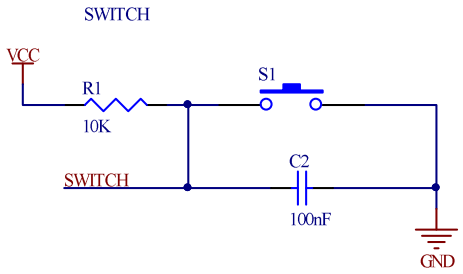


FIGURA A.2 - Esquema de conexionado del pulsador

El valor de R1 está estimado para una corriente de descarga lo más limitada posible (<0,5mA), casi inapreciable en cuanto a su consumo. La configuración con el condensador C2 corresponde a una red RC que permite un establecimiento de los niveles lineal evitando el efecto de rebotes que podría suponer una identificación falsa de pulsación por parte del microcontrolador.

$$t_{respuesta} = R \times C = 10K\Omega \times 10nF = 1ms$$

ANEXO B - COMUNICACIÓN I2C

La configuración de la línea I2C **[8]** necesaria para una correcta comunicación con todos los dispositivos conectados en el bus se detalla en la Figura [**B.1**], su disposición está basada según los criterios establecidos en las especificaciones del propio protocolo de comunicaciones (Versión 2.1 Enero 2000).

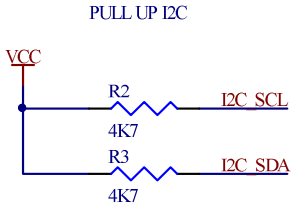


FIGURA B.1 – Resistencias Pull up empleadas en bus I2C

Siendo el valor de resistencia mínima necesaria para R2 y R3 superior a aproximadamente 1Kohm debido a la alimentación a 3,3v empleada (ver Figura [**B.3**]), y la resistencia máxima permitida en función del número de dispositivos conectados al bus con un límite máximo de 400pF tal y como está estipulado en las especificaciones del protocolo, se ha ajustado el valor de sendas R2 y R3 para permitir una comunicación correcta con todos los dispositivos presentes en las mejores condiciones de transmisión.

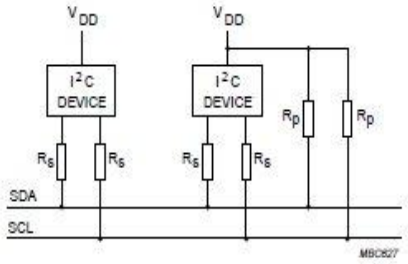


FIGURA B.2 - Conexionado bus I2C genérico

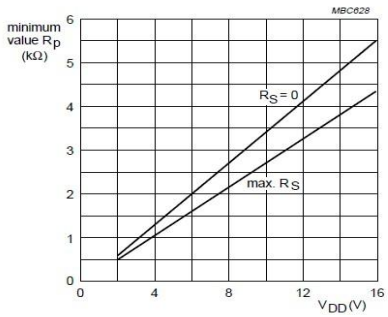


FIGURA B.3 - Grafica de valores de Rp frente a tensión de bus Vdd

$$Rp \geq \frac{Vdd - Vol}{Iol} \geq \frac{3{,}3 - 0{,}4}{3 \times 10^{-3}} \approx 1K\Omega$$

Si estimamos que la suma de las capacidades de los mismos se encuentra en torno a un máximo de 100pF (en torno a 20pf por dispositivo aproximadamente) el cual determina el límite de trabajo para la velocidad más rápida con una transmisión a 400 Kb/s, la resistencia no deberá en ningún caso superar un valor cercano a 8Kohm, tal y como se puede observar en la grafica de la Figura [**B.4**], por lo que el valor de R2 y R3 queda justificado para un margen de amplio de seguridad según las condiciones más restrictivas.

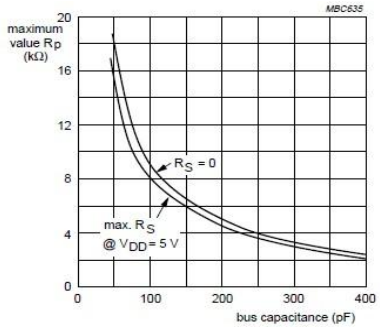


FIGURA B.4 - Mínimo valor de Rp frente la capacidad total en el I2C

La comunicación I2C con los distintos dispositivos deberá ser ejecutada tal y como se describe en el cronograma de las Figura [B.5] y [B.6], empleando un tamaño de dirección 7bits+1, con el microcontrolador actuando como Master (El dispositivo que actúa como maestro marca el tiempo de reloj en la línea SCL):

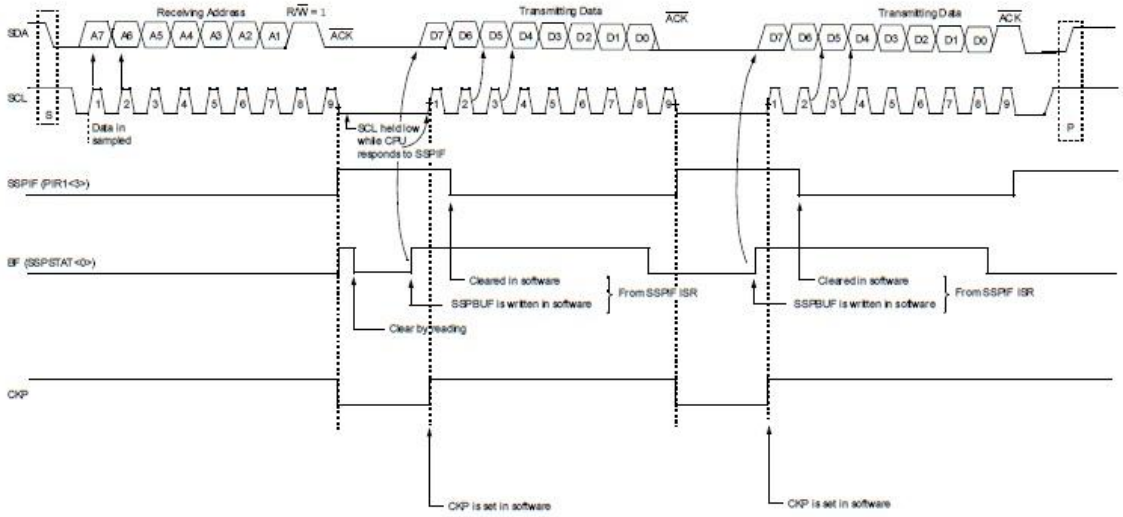


FIGURA B.5 - Cronograma de transmisión Maestro-Esclavo para dirección de 7bits+1 (THE I 2C-BUS SPECIFICATION)

Proceso de transmisión:

1. Inicia una condición de Start llevando la líneas SDA y SCL a 0 (Nivel Lógico correspondiente a GND)
2. Envió de la dirección del dispositivo seguida del bit R/W (0 para escritura)
3. Si el dispositivo esclavo identifica su dirección fuerza un pulso (0) en la línea SDL (ACK)
4. Se enviaran los datos byte a byte a través de la línea en relación a un bit por cada pulso de reloj, liberando la línea SDL por un tiempo mínimo según las características de los componentes conectados al bus entre cada dato.
5. Se concluye la comunicación con una condición de Stop llevando las líneas SDA y SCL a 1 (Nivel Lógico correspondiente a Vcc)

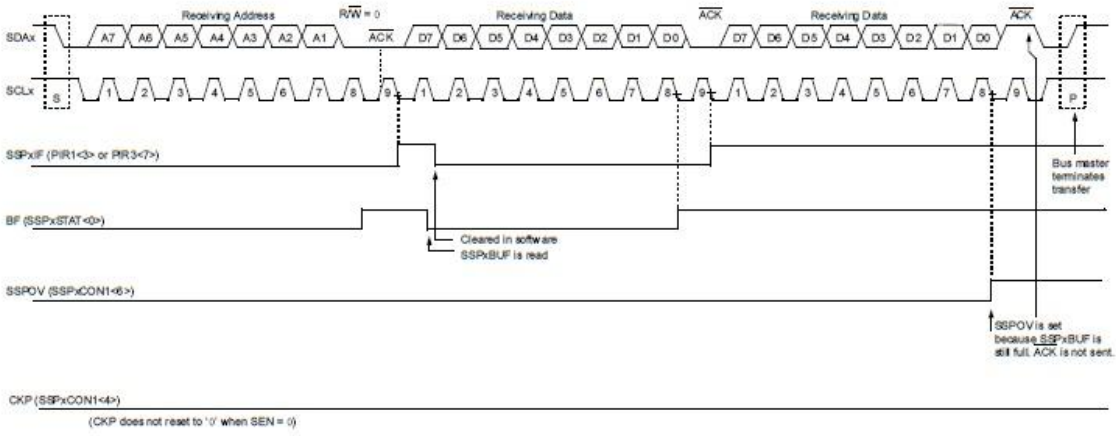


FIGURA B.6 - Cronograma de transmisión Esclavo-Maestro para dirección de 7bits+1 (THE I 2C-BUS SPECIFICATION)

Proceso de recepción:

1. Inicia una condición de Start llevando la líneas SDA y SCL a 0 (Nivel Lógico correspondiente a GND)
2. Envió de la dirección del dispositivo seguida del bit R/W (1 para escritura)
3. Si el dispositivo esclavo identifica su dirección fuerza un pulso (0) en la línea SDL (ACK)
4. Se recibirán los datos byte a byte a través de la línea en relación a un bit por cada pulso de reloj, liberando la línea SDL por un tiempo mínimo según las características de los componentes conectados al bus entre cada dato.
5. Se concluye la comunicación con una condición de Stop llevando las líneas SDA y SCL a 1 (Nivel Lógico correspondiente a Vcc)

Un ejemplo visual puede observarse en la Figura [**B.7**], para un inicio de transmisión en sentido maestro a esclavo:

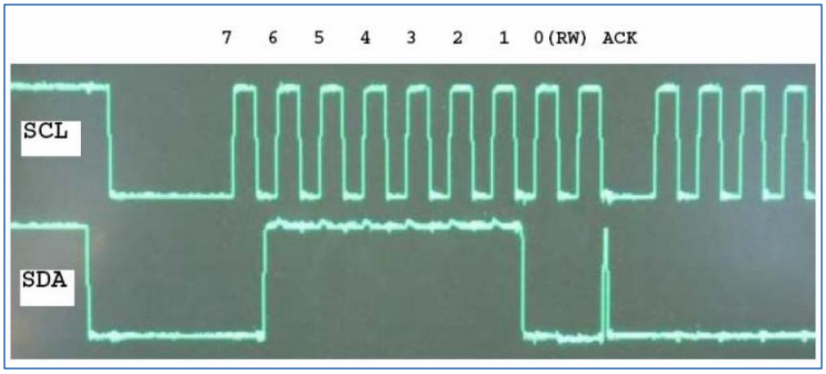


FIGURA B.7 - Detalle de inicio de una comunicación I2C

Ha sido programada a través de las librerías de control de Microchip ofrecidas como apoyo al desarrollo de aplicaciones para el microcontrolador PIC18F24J11, cuyas funciones realizan la conmutación automática de las líneas necesarias SDA y SCL bajo los tiempos requeridos para las diferentes velocidades de transmisión, así como el envió de datos serie a través de la línea SDA gracias a un registro de desplazamiento interno del propio PIC18F24J11 englobado en el MSSP MODULE (**M**ASTER **S**YNCHRONOUS **S**ERIAL **P**ORT).

Las citadas funciones son.

***"i2c_start"* -** Inicio de la transmisión

***"i2c_stop"* -** Conclusión del proceso de transmisión.

***"i2c_write"* -** Escribe un dato tamaño byte en la línea I2C. Si el dato es correctamente recibido por el modulo este forzara un ACK sobre la línea SDL.

***"i2c_load"* -** Capta un dato tamaño byte de la línea I2C. Puede añadirse como argumento el estado del bit ACK forzado por el microcontrolador en la línea.

Con algunos dispositivos es recomendable realizar una lectura en vacío desde el máster, forzando el bit ACK a 0, para liberar el bus.

También cabe resaltar no realizar el numero de lecturas correctas puede dar lugar a un bloqueo de algunos dispositivos en su funcionamiento como esclavo. Por lo que es recomendable gestionar envió y recepción mediante la definición y comprobación de un byte de longitud dentro del protocolo usado para la transmisión.

Estos bloqueos en el bus suponen a su vez la imposibilidad en algunos casos de la ejecución de nuevas instrucciones en los microcontroladores de algunos dispositivos, y es un factor importante a tener en consideración por los efectos que puede tener en el funcionamiento de estos.

Para más información acerca del estándar I2C se sugiere la consulta del documento **"THE I 2C-BUS SPECIFICATION"** mencionado a su vez en la bibliografía proporcionada.

ANEXO C - PROTOCOLO SKYETEK

A continuación se incluye información complementaria acerca del protocolo de comunicaciones empleado por el Skyemodule M1-Mini [9].

La documentación técnica ofrecida por el fabricante especifica que los mensajes deben ser construidos tal y como puede observarse en la Figura [C.1].

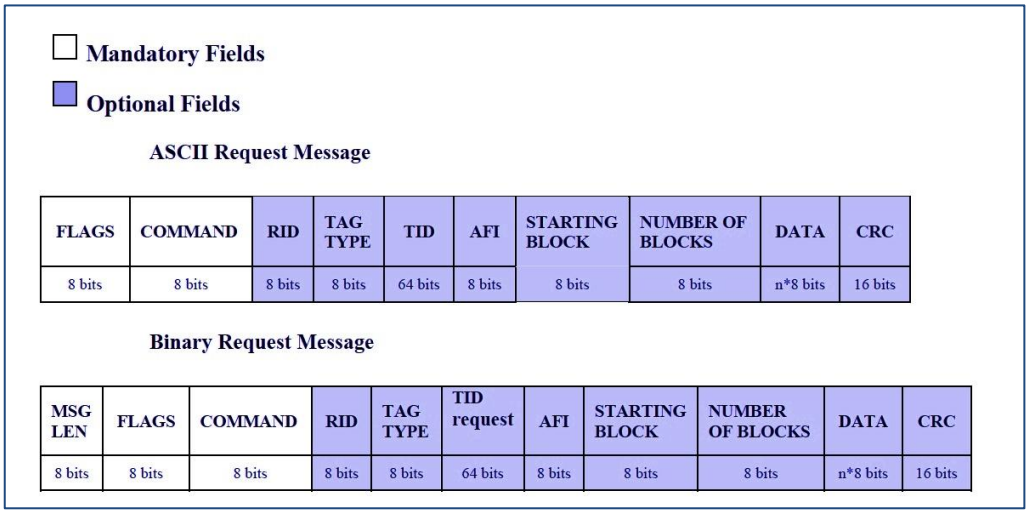


FIGURA C.1 - Estructura de los mensajes de forma acorde al protocolo RFID Skyetek (SkyeTek_ProtocolV2_071102)

Una vez recibido el carácter especial <STX>, correspondiente al valor hexadecimal 0x02, el lector pasara a interpretar los campos incluidos en el mensaje, cuyas definiciones son:

- MSG LEN: En este campo se incluye el número de bytes que compone el mensaje completo, siendo solo necesario su uso en comunicación binaria, ya que en una comunicación ASCII los mensajes quedan delimitados por los caracteres especiales retorno de carro <CF> (valor 0x0D hexadecimal) y salto de línea <LF> (valor 0x0A hexadecimal).

- FLAGS: El valor incluido dentro del campo Flags modifica ciertos comportamientos de los comandos transmitidos.

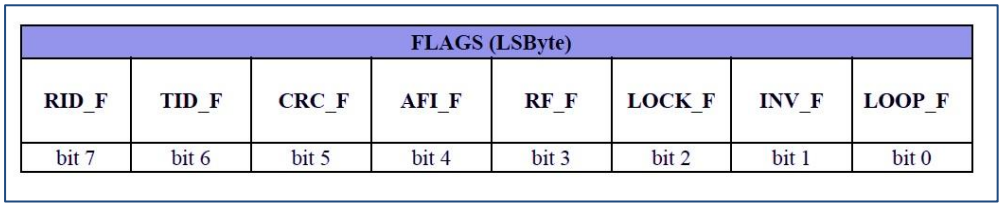


FIGURA C.2 - Bits del campo FLAG (SkyeTek_ProtocolV2_071102)

Los bits de control de la Figura [C.2].permiten:

- RID_F - Indica si está presente o no el campo RID dentro del mensaje

- TID_F - Indica si está presente o no el campo TID dentro del mensaje
- CRC_F - Activa la comprobación de redundancia cíclica para los mensajes, debiéndose incluir por tanto los valores correspondientes al final del cuerpo del mensaje en su campo correspondiente.
- AFI_F - Indica si está presente o no el campo AFI dentro del mensaje
- RF_F - El transmisor permanece en estado activo después de la recepción del comando. Esto permite el poder seguir transmitiendo energía a la TAG correspondiente, manteniéndola activa a la espera de nuevos mensajes.
- LOCK_F - Permite marcar ciertos bloques de la TAG impidiendo nuevas escrituras en ellos.
- INV_F - Activa el modo inventario para comandos tipo Select_TAG de tal manera que el lector realiza peticiones a todas las TAGS que se encuentren en el entorno.
- LOOP_F - Activa el modo en el cual el lector repite el comando de manera cíclica manteniéndose a la espera de confirmación en la respuesta.

- COMMAND: Este campo especifica el tipo de comando y el objetivo del mismo.

Como se puede observar en la Figura [C.3], existen tres operaciones escritura, lectura y petición (Select), teniendo esta ultima como único objetivo las TAGs existentes dentro del entorno.

Table 6 – COMMAND Field

COMMAND Type				COMMAND Target			
reserved (set to 0)	Write_Bit	Read_Bit	Sel_Bit	reserved (set to 0)	Tag_Bit	Sys_Bit	Mem_Bit
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

FIGURA C.3 - Bits de configuración del campo COMMAND (SkyeTek_ProtocolV2_071102)

Un comando de Select Tag se procesa por el hardware tanto del lector como una búsqueda de TAGs compatibles dentro del entorno del mismo, y una posterior identificación de las mismas siendo estas activadas por el campo magnético del lector.

La manera en la que se establecerá este dialogo puede modificarse mediante la activación de bits específicos del campo Flag, tal y como se ha explicado anteriormente.

Las operaciones de escritura y lectura pueden llevarse a cabo tanto en la memoria de las TAGs, como en la memoria interna del lector, siendo esta subdividida en dos apartados denominados simplemente memoria en el primer caso, y memoria de sistema en el segundo.

La memoria de sistema contiene la información identificativa del propio lector como son su número de serie, versión de firmware e RID, así como datos de su configuración como son la tasa de baudios de

transmisión en comunicación serie, modo de operación, y primer comando a ejecutar tras reinicio. Estos datos son almacenados en entre las posiciones 0x00 y 0x12 de la memoria del lector.

Se encuentra reservada con carácter interno el uso de las posiciones que parten desde la 0x05 hasta la 0x7F, siendo el resto disponible para su libre uso.

- RID, Reader **ID**: Este permite la comunicación con varios lectores desde un mismo host según si identificado asignado.

- TAG TYPE: Dentro de los distintos tipos de TAGs disponibles, este campo indicara específicamente una comunicación con etiquetas de una sola tecnología en concreto, o bien permitirá la posibilidad de detectar el tipo de TAG existente en el entorno.

- TID, **T**AG **ID**: Almacenara el identificador de la TAG para operaciones de petición, escritura y lectura individuales.

- AFI: Sus siglas corresponden al acrónimo **A**plication **F**ield **I**dentifier, tecnología que presentan las TAGs tipo ICode1 e ISO15693. Este campo solo debe ir incluido en comandos de petición (Select) destinados a la comunicación con este tipo de etiquetas.

- STARTING BLOCK: La memoria de las etiquetas puede diferir de una tecnología a otra en su organización interna. La unidad mínima empleada para la comunicación entre estas, el lectores y el host es denominada como bloque (Block), el cual tiene una longitud de bits variable que puede oscilar entre 1 byte para la memoria de sistema del lector hasta los 8 bytes por bloque en las etiquetas tipo PicoTag . El uso de este campo en la comunicación determina cuántos de estos bloques se transmitirán en un único mensaje.

- CRC: Por ultimo este campo almacena los valores del CRC que determinaran si la integridad del mensaje está intacta. El método de comprobación de errores corresponde al CCITT KERMIT de 16bits.

ANEXO D - CONFIGURACION DEL BLOQUE DE SENSORES INERCIALES

En los siguientes apartados se resumen los aspectos individuales contemplados para el funcionamiento particular de cada dispositivo. No se profundizara en todas las posibilidades ofrecidas por cada uno de los componentes, pudiendo encontrar una información ampliada en la consulta de sus respectivas hojas de características.

D1 ADXL345

Este sensor de aceleración cuenta con una pila de datos FIFO (**F**irst **I**nput **F**irst **O**utput) de 32 niveles, en la que se basa para ofrecer diferentes modos de operación. De forma concreta estos son:

Bypass Mode: La pila de datos permanece inactiva.

FIFO Mode: Las medidas de datos correspondientes a los tres ejes se almacenan en la pila hasta alcanzar un tamaño de muestra previamente configurada, momento en el que se activa una interrupción especifica (Watermark). La pila continua almacenando datos hasta ocupar todo el tamaño disponible.

El dispositivo sigue operando, permitiendo funciones como la detección de paso (Tap Detection), sin embargo el uso de la FIFO permanece bloqueado.

Stream Mode: Su funcionamiento es prácticamente similar al FIFO Mode, con la salvedad de que tras la acumulación de datos en la pila por encima del tamaño de muestra preconfigurado y activación de la Watermark. Siguen acumulándose datos en la misma descartando los datos más antiguos cuando esta se desborda.

Trigger Mode: En este modo la FIFO acumula todas las muestras posibles que le permite su tamaño, descartando los datos previamente almacenados para la inclusión de nuevos tras el desbordamiento. Cuando un evento de disparo (Trigger Event) previamente habilitado ocurre, la FIFO mantiene el valor de las últimas n-muestras determinadas por su configuración y envía un pulso a una de las líneas INT1 o INT2 a selección del usuario.

La puesta en funcionamiento en los diferentes modos se establece mediante la modificación del registro de memoria 0x38 FIFO_CTL tal y como se indica en la tabla de la Figura [**D.1**].

D7	D6	D5	D4	D3	D2	D1	D0
FIFO_MODE		Trigger	Samples				

FIFO_MODE Bits

These bits set the FIFO mode

Table 19. FIFO Modes

Setting			
D7	**D6**	**Mode**	**Function**
0	0	Bypass	FIFO is bypassed.
0	1	FIFO	FIFO collects up to 32 values and then stops collecting data, collecting new data only when FIFO is not full.
1	0	Stream	FIFO holds the last 32 data values. When FIFO is full, the oldest data is overwritten with newer data.
1	1	Trigger	When triggered by the trigger bit, FIFO holds the last data samples before the trigger event and then continues to collect data until full. New data is collected only when FIFO is not full.

FIGURA D.1 - Registro FIFO_CTL (Datasheet del fabricante)

Puede accederse a cualquiera de los niveles de la FIFO para la toma de las muestras almacenadas, siendo ubicadas entre las direcciones 0x32 y 0x37. Estos se almacenan en un formato de 16bits en complemento a 2, divididos en dos registros de 1byte. Este proceso ha sido implementado en la función ***"itg_capture_gyros_data"*** pudiéndose acceder a las muestras como un único bloque de 16bits, si estas van a ser procesadas internamente por el microcontrolador, o bien en un array para su empaquetado y transmisión.

Ajustar el offset de las muestras puede realizarse manualmente mediante la escritura en los registros 0x1E, 0x1F y 0x20 identificables como OFSX, OFSY, OFSZ dependiendo de las necesidades de la aplicación.

Es posible configurar tanto el formato de los datos almacenados en la pila modificando los bits del registro 0x31 DATA_FORMAT, así como la resolución y rango de medida de los sensores tal y como se muestra en la Figura [**D.2**],

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

FIGURA D.2 - Registro DATA_FORMAT (Datasheet del fabricante)

La configuración de los bits de rango (Range), permiten secuencialmente marcar los valores de 2,4,8 y 16g, mientras que la activación del bit FULL_RES conmuta entre la resolución con los rangos de escala del conversor de 10bits incluido en el ADXL345 anteriormente citados, o una resolución fija de 4mg/LSB con hasta 13bits en la conversión.

Register 0x2C

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	LOW_POWER	Rate			

Register 0x2D

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

FIGURA D.3 - Registros BW_RATE y POWER_CTL (Datasheet del fabricante)

Por último cabe destacar que es posible modificar la frecuencia de muestreo, así como el modo de funcionamiento para un consumo más eficiente de los modos de energía modificando la información contenida en los registros 0x2C BW_RATE y 0x2D POWER_CTL. En la Figura [**D.3**] puede observarse como están distribuidos los bits de control en ambos.

La frecuencia de trabajo puede establecerse para valores de que oscilan desde los 6,25 a los 3200Hz tal y como queda reflejado en la Figura [**D.4**],junto con una estimación de la corriente consumida para los dos principales modos de energía.

Current Consumption vs. Data Rate

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (µA)
3200	1600	1111	145
1600	800	1110	100
800	400	1101	145
400	200	1100	145
200	100	1011	145
100	50	1010	145
50	25	1001	100
25	12.5	1000	65
12.5	6.25	0111	55
6.25	3.125	0110	40

Current Consumption vs. Data Rate, Low Power Mode

Output Data Rate (Hz)	Bandwidth (Hz)	Rate Code	I_{DD} (µA)
400	200	1100	100
200	100	1011	65
100	50	1010	55
50	25	1001	50
25	12.5	1000	40
12.5	6.25	0111	40

FIGURA D.4 - Tablas de consumo frente a frecuencias de muestreo (Datasheet del fabricante)

Se ha configurado un Output Data Rate por defecto de 100Hz, mientras que el fondo de escala se corresponde con 4g, pudiendo variarse estos parámetros según los requerimientos de la aplicación modificando los valores especificados dentro de la función ***"adxl_configure"***.

Los modos de energía seleccionables son los que se detallan a continuación:

POWER OFF: El dispositivo se encuentra totalmente apagado.

BUS DISABLED: El dispositivo se encuentra en modo de espera, pero la comunicación no está habilitada.

BUS ENABLED: El dispositivo se encuentra en modo de espera, la comunicación está habilitada permitiendo la activación mediante trafico I2C o SPI.

MEASSUREMENT: El modo normal de funcionamiento con sensores y comunicación activados.

Además de lo citado, existe la posibilidad de implementar en futuras revisiones funcionalidades interesantes gracias a la versatilidad que proporciona este sensor, tales como detecciones de caída libre, de actividad, y medidas de aceleración paso a paso simples y dobles.

D.2 ITG-3200

El sensor ITG-3200 no ofrece tanta complejidad en su diseño como el acelerómetro ADXL345, ya que únicamente es capaz de trabajar en un modo continuo de toma de muestras y realizar una transmisión en stream de datos al host. Por otra parte precisa de una configuración manual de aspectos a bajo nivel relativos al modo de operación del sensor.

Las tomas de muestras de datos del giróscopo están sampleados internamente para 1 o 8Khz (Finternal) dependiendo de la configuración de las posiciones DLPF_CFG en el registro 0x16 DLPF FULL SCALE, cuya disposición queda reflejada en la Figura [**D.5**], así como también la frecuencia del filtro paso bajo empleado con valores que parten de un mínimo de 5Hz hasta un máximo de 256Hz.

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
16	22	-			FS_SEL		DLPF_CFG			00h

DLPF_CFG

DLPF_CFG	Low Pass Filter Bandwidth	Internal Sample Rate
0	256Hz	8kHz
1	188Hz	1kHz
2	98Hz	1kHz
3	42Hz	1kHz
4	20Hz	1kHz
5	10Hz	1kHz
6	5Hz	1kHz
7	Reserved	Reserved

FIGURA D.5 - Registro DLPF_CFG (Datasheet del fabricante)

La modificación de los bits del registro 0x15 SMLPRT_DIV (Sample Rate Divider) definen, según el estado de los bits del DLP_CFG, como son almacenadas las muestras en la memoria interna del ITG-3200 tras el filtrado para su lectura. Ver Figura [**D.6**].

Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
15	21	SMPLRT_DIV								00h

FIGURA D.6 - Registro SMLPRT_DIV (Datasheet del fabricante)

La frecuencia de muestreo efectiva *(Fsample)* se calcula con el dato del introducido en base a la ecuación:

$$Fsample = \frac{Finternal}{(divider + 1)}$$

Se ha configurado un Sample Rate por defecto de 100Hz, mientras que el fondo de escala se corresponde con 2000 grados/seg, pudiendo variarse estos parámetros según los requerimientos de la aplicación modificando los valores especificados dentro de la función ***" hmc_configure"***.

Cabe resaltar también, que la ganancia del sistema y su fase varía en función de la frecuencia del filtro empleada, tal y como queda detallado en la Figura [**D.7**].

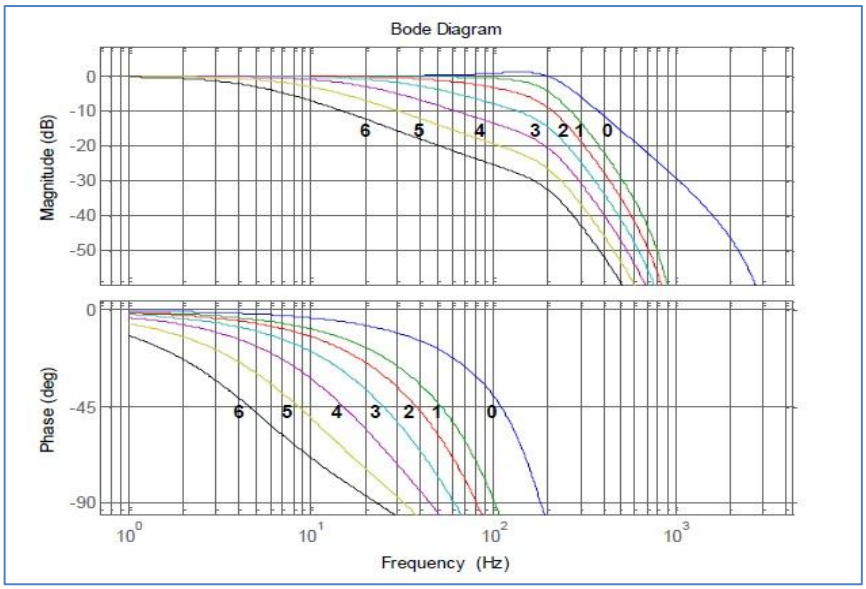
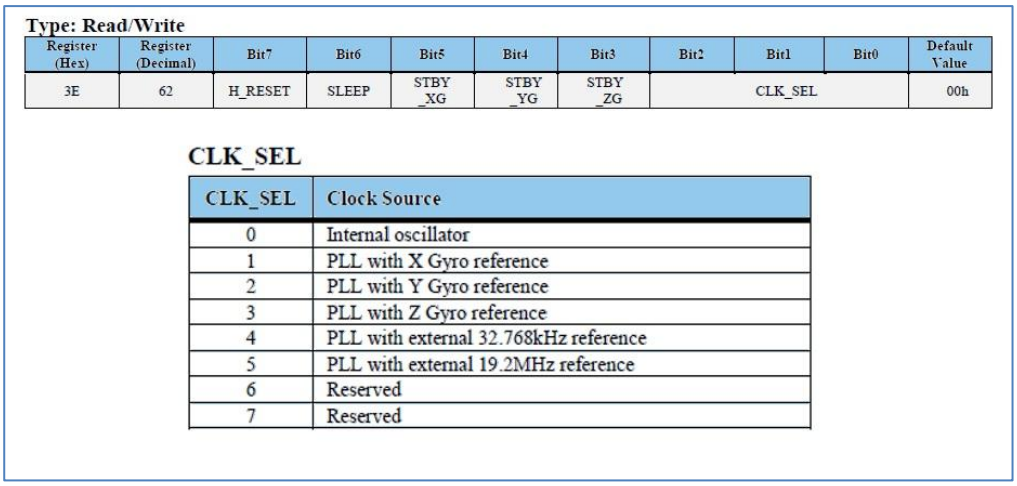


FIGURA D.7 - Frecuencia del filtro paso bajo frente a ganancia y fase (Datasheet del fabricante)

Por lo que si en el desarrollo de aplicaciones se considera oportuno su activación para el filtrado de vibraciones ambientales, habrá que considerar la pequeña variación en la medida resultante de cara a una correcta calibración inicial.

Los dos modos de funcionamiento disponible para este dispositivo (Activo y Reposo) pueden ser conmutados mediante la modificación del bit 6 (Sleep) del registro 0x3E PWR_MGM. También puede realizarse un reset del mismo, poner en estado de reposo el muestreo de los ejes de forma individual, así como modificar la fuente de reloj del sistema. La disposición de estos bits de control puede observarse en la Figura [**D.8**].



Type: Read/Write

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Default Value
3E	62	H_RESET	SLEEP	STBY _XG	STBY _YG	STBY _ZG	CLK_SEL			00h

CLK_SEL

CLK_SEL	Clock Source
0	Internal oscillator
1	PLL with X Gyro reference
2	PLL with Y Gyro reference
3	PLL with Z Gyro reference
4	PLL with external 32.768kHz reference
5	PLL with external 19.2MHz reference
6	Reserved
7	Reserved

FIGURA D.8 - Registro PWR_MGM (Datasheet del fabricante)

La lectura de los datos muestreados del sensor se realiza accediendo a los registros ubicados entre las direcciones 0x1D y 0x22 denominados GYRO_(eje)_(H o L) siendo los datos almacenados en un formato de 16bits en complemento a 2, divididos en dos registros de 1byte. Este proceso ha sido implementado en la función ***" itg_capture_gyros_data"*** pudiéndose acceder a las muestras como un único bloque de 16bits.

El sensor ITG-3200 además presenta interrupciones programables y cuenta con un sensor de temperatura, del cual se ha implementado una función de lectura ***" itg_capture_temp_data"***.

D.3 HMC 5843

Este componente propone cuatro modos de funcionamiento seleccionables mediante la activación de los bits MD correspondientes dentro del registro MR 0x02 (Mode Register) tal y como se especifica en la Figura [**D.9**]:

MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
(0)	(0)	(0)	(0)	(0)	(0)	MD1 (1)	MD0 (0)

Location	Name	Description
MR7 to MR2	0	These bits must be cleared for correct operation.
MR1 to MR0	MD1 to MD0	Mode Select Bits. These bits select the operation mode of this device.

MD1	MD0	Mode
0	0	Continuous-Conversion Mode. In continuous-conversion mode, the device continuously performs conversions an places the result in the data register. RDY goes high when new data is placed in all three registers. After a power-on or a write to the mode or configuration register, the first measurement set is available from all three data output registers after a period of $2/f_{DO}$ and subsequent measurements are available at a frequency of f_{DO}, where f_{DO} is the frequency of data output.
0	1	Single-Conversion Mode. When single-conversion mode is selected, device performs a single measurement, sets RDY high and returned to sleep mode. Mode register returns to sleep mode bit values. The measurement remains in the data output register and RDY remains high until the data output register is read or another conversion is performed.
1	0	Idle Mode. Device is placed in idle mode.
1	1	Sleep Mode. Device is placed in sleep mode.

FIGURA D.9 - Registro MR (Datasheet del fabricante)

Continuous-Measurement Mode: Durante este modo, el sensor realiza continuamente mediciones de campo magnético con la periodicidad programada, almacenando las muestras obtenidas en los registros correspondientes de su memoria. Entre los muestreos el propio sensor de forma automática reduce su demanda de corriente entrando en un estado de espera (Iddle Mode) para un consumo más eficiente de energía.

Single-Measurement Mode: Tal y como su nombre indica, el dispositivo realiza una única medida para después adoptar el modo de funcionamiento de espera (Iddle Mode).

Idle Mode - El muestreo y la conversión es detenida, no así el reloj de sistema. El valor de los datos de la última muestra es mantenido. La comunicación I2C permanece habilitada tanto para recepción como transmisión de datos.

Sleep Mode - El reloj interno permanece detenido así como casi la totalidad de las funcionalidades del dispositivo, a excepción del bus I2C, atendiendo solamente a este en recepción para permitir una salida de este estado mediante dicha comunicación.

Off Mode - Dispositivo totalmente apagado

La configuración de las medidas realizadas por el magnetómetro HMC543 es realizada mediante la modificación de los valores de dos registros denominados 0x00 CRA, 0x01 CRB (Configuration Register A y B).

En el registro de configuración A se establecen los valores de muestreo de datos y de corriente de polarización tal y como queda reflejado en la Figura [**D.10**].

CRA7	CRA6	CRA5	CRA4	CRA3	CRA2	CRA1	CRA0
(0)	(0)	(0)	DO2 (1)	DO1 (0)	DO0 (0)	MS1 (0)	MS0 (0)

DO2	DO1	DO0	Minimum Data Output Rate (Hz)
0	0	0	0.5
0	0	1	1
0	1	0	2
0	1	1	5
1	0	0	10 (default)
1	0	1	20
1	1	0	50
1	1	1	Not used

MS1	MS0	Mode
0	0	Normal measurement configuration (default). In normal measurement configuration the device follows normal measurement flow. Pins BP and BN are left floating and high impedance.
0	1	Positive bias configuration. In positive bias configuration, a positive current is forced across the resistive load on pins BP and BN.
1	0	Negative bias configuration. In negative bias configuration, a negative current is forced across the resistive load on pins BP and BN.
1	1	This configuration is not used.

FIGURA D.10 - Registro CRA (Datasheet del fabricante)

La configuración de los bits MS1 y MS2 permite establecer una corriente de polarización empleada para modificar el offset del sensor. Lo cual permite generar un campo magnético que contrarreste al ambiental.

El registro de configuración B permite por otra parte el establecimiento del rango de conversión con valores que se encuentran entre 0.7 Ga a 6.5Ga para el fondo de escala para un resolución de 280 a 1620 mili gauss por LSB. Ver tabla de la Figura [**D.11**].

Se ha configurado un Output Data Rate por defecto de 50Hz, mientras que el fondo de escala se corresponde con 4Ga, pudiendo variarse estos parámetros según los requerimientos de la aplicación modificando los valores especificados dentro de la función ***"hmc_configure"***.

CRB7	CRB6	CRB5	CRB4	CRB3	CRB2	CRB1	CRB0
GN2 (0)	GN1 (0)	GN0 (1)	(0)	(0)	(0)	(0)	(0)

GN2	GN1	GN0	Sensor Input Field Range:	Gain (counts/ milli-gauss)	Output Range
0	0	0	±0.7Ga	1620	0xF800–0x07FF (-2048–2047)
0	0	1	±1.0Ga (default)	1300	0xF800–0x07FF (-2048–2047)
0	1	0	±1.5Ga	970	0xF800–0x07FF (-2048–2047)
0	1	1	±2.0Ga	780	0xF800–0x07FF (-2048–2047)
1	0	0	±3.2Ga	530	0xF800–0x07FF (-2048–2047)
1	0	1	±3.8Ga	460	0xF800–0x07FF (-2048–2047)
1	1	0	±4.5Ga	390	0xF800–0x07FF (-2048–2047)
1	1	1	±6.5Ga (Not Recommended)	280	0xF800–0x07FF (-2048–2047)

FIGURA D.11 - Registro CRB (Datasheet del fabricante)

La lectura de los datos muestreados del sensor se realiza accediendo a los registros ubicados entre las direcciones 0x03 y 0x08 denominados Data Output (eje) (M o L)SB Register siendo los datos almacenados en un formato de 16bits en complemento a 2, divididos en dos registros de 1byte. Este proceso ha sido implementado en la función ***" hmc_capture_channel_data"*** pudiéndose acceder a las muestras como un único bloque de 16bits para los tres ejes individualmente para su tratamiento si este fuese preciso, además de ser almacenados en una array para el empaquetamiento de varias muestras y su transmisión.

ANEXO E - COMUNICACIÓN ZIGBEE

Para el desarrollo de este proyecto, las herramientas empleadas para la establecer la conexión Zigbee necesaria entre el Z-Mota Move 2.0 y el host (PC) son:

- Las librerías ***"zbAtLibrary30x"*** para el modulo ZIGBEE ETRX357 integrado dentro del dispotivo de control inercial.
- El terminal de comunicación serie Telegesis Terminal (ver Figura [**E.1**]) junto con el modulo ETRX2USB, corriendo bajo el sistema operativo Windows.

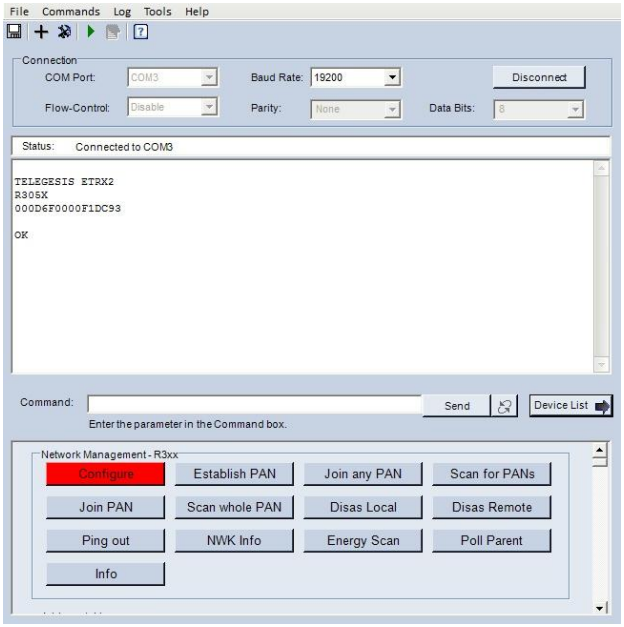


FIGURA E.1 - Detalle de la interfaz de control Telegesis

Al precisar únicamente una red reducida, limitada a los dos componentes citados, no cabe hablar de ninguna topología específica, actuando los dispositivos como:

- Host - Como coordinador y router (COO) estableciendo la red PAN (**P**ersonal **A**rea **N**etwork)
- Z-Mota Move 2.0 - Como SED (**S**leepy **E**nd **D**evice).

La transmisión de datos se realiza mediante comandos AT siguiendo el protocolo establecido por Telegesis **[10]**.

Las convenciones de este sistema proponen una comunicación basada en la transmisión de caracteres ASCII, iniciando la comunicación con los caracteres <AT+> y finalizándola con los caracteres especiales <CR> (retorno de carro) y <LF> (salto de línea) como terminadores.

Dialogo de ejemplo para la lectura del registro de memoria interna del ETRX357 correspondiente con la dirección 0x00 :

COMANDO EJECUTADO ***ATS00?<CR>***
RESPUESTA EN TERMINAL ***<CR><LF>FFFF<CR><LF>***
RESPUESTA EN TERMINAL ***<CR><LF>OK<CR><LF>***

La sintaxis para los parámetros definida es la que se presenta en la Figura [**C.2**],

XX	8-bit hexadecimal number. Valid characters are 0-9, a-f and A-F
XXXX	16-bit hexadecimal number. Valid characters are 0-9, a-f and A-F
n	Number from 0-9
s	Sign
b	Bit (0 or 1)
c	character
<PID>	16-bit hexadecimal PAN ID (0000 to FFFF)
<EPID>	64-bit hexadecimal extended PAN ID
<channel>	decimal channel (802.15.4 channel 11-26)
<password>	8 character password
<EUI64>	64-bit IEEE 802.15.4 address in hexadecimal
<ioread>	32-bit hexadecimal number representing the reading of S1A
<data>	Custom Data
<ClusterList>	A list of 16 bit cluster identifiers in hexadecimal representation
<FirmwareRevision>	The Firmware Revision Number

FIGURA E.2 - Tabla de parámetros y su sintaxis en la comunicación AT de Telegesis (TG-ETRXn-R305-Commands)

Es necesario conocer que tanto el ETRX357 como el ETRX2USB cuentan con una FIFO de 128bytes dedicada al almacenamiento y recepción de los mensajes, por queda determinada la limitación de longitud de los mismos al valor citado de 128 caracteres.

Cabe la posibilidad de realizar transmisiones binarias de datos, para la cual es necesario del mismo modo que en ASCII seguir el convenio de comandos AT establecido para el inicio de la comunicación. Sin embargo estas transmisiones se realizaran en dos mensajes, no siendo necesario el empleo de los caracteres terminadores.

Es reseñable además el hecho de que puedan realizarse también operaciones remotas dentro de los nodos de una red Zigbee mediante comandos específicos, empleando diferentes identificadores <PID><EPID><EUI64>, ofreciendo de ese modo un abanico amplio de posibles funcionalidades en los diseños.

Como aspecto a importante a tener en consideración durante la configuración de la red, hay que recordad que para la comunicación del Z-Mota Move 2.0 con el host, se proporciona la función ***"zbSendDataToZigbeeSink"*** dentro de la librería ***"zbAtLibrary30x"***. Por lo que para que llegue a ser efectiva ha de configurarse el host como Sink (sumidero) tras el establecimiento de una nueva PAN.

Para más información acerca del protocolo de Telegesis se sugiere la consulta del documento "**TG-ETRXn-R305-Commands"** mencionado a su vez en la bibliografía proporcionada.

ANEXO F - CODIGO FUENTE

F.1 File: rfidSkyetekM1_library.c

```
/*!\file rfidSkyetekM1_library.c
   \brief Defines functions, constants and variables for managing RFID modules using I2C
 *
*
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
 * Code lines mark with //** can be modified according to customer's requirements.
 */

#include "rfidSkyetekM1_library.h"

////////////////////////////////////////////////////////////////////////////////////
// LOW LEVEL LAYER
//
////////////////////////////////////////////////////////////////////////////////////

/*! \fn generate_16bit_crc (unsigned char *data, int16 length, int16 pattern)
 *  \brief calculates the crc of the message
 *  \param *data is a pointer to the array over wich the CRC is to be calculated
 *  \param length is the number of bytes in the array pointed to by *data
*/
unsigned int16 generate_16bit_crc( unsigned int8 *data, unsigned int length )
{
  unsigned int i;   // Byte counter
  unsigned int j;   // Bit counter
  unsigned int16 crc;   // Calculation of CRC

  restart_wdt();  // Resets the Watch Dog timer

  crc = CRC_PRESET; // Preset value

  for(i = 0; i < length; i++)  // For all bytes of the string
    {
      crc ^= *data++;

      for(j = 0; j < 8; j++) // Test each bit in the byte
        {
          if(crc & 0x0001)
            {
              crc >>= 1;
              crc ^= CRC_CCITT_KERMIT; // Polynomial function applied
            }
          else
            crc >>= 1;
        }
    }
  return(crc);  /* returns calculated crc (16 bits) */
}

////////////////////////////////////////////////////////////////////////////////////
```

```
/*! \fn void void m1_send_request (unsigned int8 request[],unsigned int8 response[])
 *  \brief sends the request and receives the response by i2c
 *  \param request[] is a pointer to the array is to be sent
 *  \param response[] is a pointer to the array is to be received
*/
void m1_send_request ( unsigned int8 request[],unsigned int8 response[] )
{
   unsigned int count = 0; // Array counter
   unsigned int length = 0; // length of the I2C message without STX byte

   restart_wdt(); // Resets the Watch Dog timer

// Request block

   i2c_start(); // Starts the communication
   i2c_write(M1_ADDRESS_WRITE); // Send the Skyetek M1 I2C address for writing

   i2c_write(request[0]); // Write stx
   i2c_write(request[1]); // Write length
   length = request[1];

   for (count=2; count < (length+2); count++) // Write the remaining bytes of the
response
      i2c_write(request[count]);

   i2c_stop();      // Stop condition

// Response block

   i2c_start();     // Start condition
   i2c_write(M1_ADDRESS_READ); // Send the Skyetek M1 I2C address for reading

   response[0] = i2c_read(1); // Read stx
   response[1] = i2c_read(1); // Read length
   length = response[1];

   for (count = 2; count < (length+2); count++) // Read the remaining bytes of the
response
      response[count] = i2c_read(1);

   i2c_read(0); // Releases the clock line with a void reading
   i2c_stop();      // Stop condition
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void m1_build_request (unsigned int8 request[])
 *  \brief Asks for the reader identifier field
 *  \param request[] is a pointer to the array is to be sent
*/
void m1_build_request(unsigned int8 request[])
{
   unsigned int8 count=2; // Byte counter
   unsigned int8 bytes_stored=0; // Number of bytes stored

   unsigned int16 crc;   // CRC in a two bytes unique variable

   restart_wdt(); // Resets the Watch Dog timer

// STX is included in the request stored to be sent
   request[0] = STX;

// Basic fields
   request[count++] = m1_buffer.flag;
   request[count++] = m1_buffer.command;

// Tag fields
   if (m1_request_fields.type == 1)
      request[count++] = m1_buffer.type;
   bytes_stored = count;
   if (m1_request_fields.tid == 1)
      for (count = bytes_stored; count < bytes_stored+M1_MAX_TID_LENGHT; count++)
         request[count] = m1_buffer.tid[count-bytes_stored];

// Read/Write fields
   if (m1_request_fields.starting_block == 1)
```

```
      request[count++] = m1_buffer.starting_block;
   if (m1_request_fields.number_of_blocks==1)
      request[count++] = m1_buffer.number_of_blocks;

// Data field
   bytes_stored = count;
   if (m1_request_fields.data == 1)
   {
      for (count = bytes_stored;count <(
bytes_stored+(m1_buffer.number_of_blocks*m1_bytexblock)); count++)
         request[count] = m1_buffer.data[count-bytes_stored];
      m1_bytexblock = 1; // Reset the ratio
   }

// Lengt and CRC calculation
   m1_buffer.length = (count);
   request[1] = m1_buffer.length;

   crc = generate_16bit_crc(&m1_request_string[1], (count-1) ); // Calculates the CRC

   m1_buffer.crc[0] = (crc&0xFF00) >> 8;
   request[count++] = m1_buffer.crc[0]; // High CRC byte to the corresponding position
of the string

   m1_buffer.crc[1] = (crc&0x00FF);
   request[count++] = m1_buffer.crc[1]; // Low CRC byte to the corresponding position of
the string
}

////////////////////////////////////////////////////////////////////////////////
// MIDDLE LEVEL LAYER
//
////////////////////////////////////////////////////////////////////////////////

/*! \fn void m1_read_sys ( unsigned int8 memory_address )
 *  \brief reads system information stored in the memory of the reader
 *  \param memory_address is the memory address where it performs the read
 */
void m1_read_sys ( unsigned int8 memory_address )
{
   unsigned int8 count = 0; // Byte counter
   unsigned int8 length = 0; // Response message length

   restart_wdt(); // Resets the Watch Dog timer

// Fields to include
   m1_request_fields = 0x00; // Reset the request fields
   m1_request_fields.starting_block = 1;
   m1_request_fields.number_of_blocks = 1;

// Fields values
   m1_buffer.flag = M1_FLAG_CRC;
   m1_buffer.command = M1_COMMAND_READ_SYS;
   m1_buffer.starting_block = memory_address;
   m1_buffer.number_of_blocks = 0x01; //Limited value by protocol

// Build the request and send
   m1_build_request( m1_request_string );
   m1_send_request( m1_request_string,m1_response_string );

// Stores the response data in PIC's memory (can be more than a byte)
   m1_user_data_length=0;
   length = m1_response_string[1];
   for(count = 3; count < length; count++)
   {
      m1_user_data_string[count-3] = m1_response_string[count];
      m1_user_data_length++;
   }
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void m1_write_sys ( unsigned int8 memory_address )
 *  \brief writes system information stored in the memory of the reader
 *  \param memory_address is the memory address where it performs the write
*/
void m1_write_sys ( unsigned int8 memory_address )
```

```
{
   restart_wdt(); // Resets the Watch Dog timer

// Fields to include
   m1_request_fields = 0x00; // Reset the request fields
   m1_request_fields.starting_block = 1;
   m1_request_fields.number_of_blocks = 1;
   m1_request_fields.data = 1;

// Fields values
   m1_buffer.flag = M1_FLAG_CRC;
   m1_buffer.command = M1_COMMAND_WRITE_SYS;
   m1_buffer.starting_block = memory_address;
   m1_buffer.number_of_blocks = 0x01; // Limited value by protocol
   m1_buffer.data[0] = m1_user_data_string[0];

// Build the request and send
   m1_build_request(m1_request_string);
   m1_send_request(m1_request_string,m1_response_string);
}

////////////////////////////////////////////////////////////////////////////////
/*! \fn void m1_read_memory (unsigned int8 memory_address)
 *  \brief reads information stored in the memory of the reader
 *  \param memory_address is the memory address where it performs the read
*/
void m1_read_memory ( unsigned int8 memory_address )
{
   restart_wdt(); // Resets the Watch Dog timer

// Fields to include
   m1_request_fields = 0x00; // Reset the request fields
   m1_request_fields.starting_block = 1;
   m1_request_fields.number_of_blocks = 1;

// Fields values
   m1_buffer.flag = M1_FLAG_CRC;
   m1_buffer.command = M1_COMMAND_READ_MEM;
   m1_buffer.starting_block = memory_address;
   m1_buffer.number_of_blocks = 0x01; // Can only read one block per access (Don´t
change)

// Build the request and send
   m1_build_request(m1_request_string);
   m1_send_request(m1_request_string,m1_response_string);
}

////////////////////////////////////////////////////////////////////////////////
/*! \fn void m1_write_memory ( unsigned int8 memory_address, unsigned int8 string_pos )
 *  \brief writes information stored in the memory of the reader
 *
 *  \param memory_address is the memory address where it performs the write
 *  \param string_pos is the position in which the corresponding data is saved for
multiple writes
 *
*/
void m1_write_memory ( unsigned int8 memory_address, unsigned int8 string_pos )
{
   restart_wdt(); // Resets the Watch Dog timer

// Fields to include
   m1_request_fields = 0x00; // Reset the request fields
   m1_request_fields.starting_block = 1;
   m1_request_fields.number_of_blocks = 1;
   m1_request_fields.data = 1;

// Fields values
   m1_buffer.flag = M1_FLAG_CRC;
   m1_buffer.command = M1_COMMAND_WRITE_MEM;
   m1_buffer.starting_block = memory_address;
   m1_buffer.number_of_blocks = 0x01; // Can only read one block per access (Don´t
change)
   m1_buffer.data[string_pos] = m1_user_data_string[string_pos];

// Build the request and send
   m1_build_request(m1_request_string);
   m1_send_request(m1_request_string,m1_response_string);
```

```
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void m1_select_tag (void)
 *  \brief does a search for any tag in the reader's field and stores in memory the TID
if it's found
*/
unsigned int1 m1_select_tag (void)
{
   unsigned int1 tag_detected = 0;

   unsigned int8 count; // Byte counter
   unsigned int8 first_tid_byte; // Relative position of the first TID byte in the
response array

   restart_wdt(); // Resets the Watch Dog timer

// Fields to include
   m1_request_fields = 0x00; // Reset the request fields
   m1_request_fields.type = 1;
   m1_request_fields.tid = 0;

// Fields values
   m1_buffer.flag = M1_FLAG_CRC;
   m1_buffer.command = M1_COMMAND_SELECT_TAG;
   m1_buffer.type = M1_TYPE_AUTODETECT;

// Build the request and send
   m1_build_request(m1_request_string);
   m1_send_request(m1_request_string,m1_response_string);

// Stores the response code
   m1_buffer.response_code = m1_response_string[2];

// Stores the tag id and type if have been detected
   if ( m1_buffer.response_code == M1_COMMAND_SELECT_TAG) // If SELECT_TAG pass
   {
      first_tid_byte = m1_response_string[1]-M1_MAX_TID_LENGHT; // stores the position
of the first TID byte
      for (count = 0; count < M1_MAX_TID_LENGHT; count++)
         m1_buffer.tid[count] = m1_response_string[count+first_tid_byte]; // TID reading
begins at a relative position calculated from the end of the response array

      m1_buffer.type = m1_response_string[3]; // tag type always has the same position
in the array

      return tag_detected = 1;
   }
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void m1_read_tag (unsigned int8 memory_address,unsigned int8 data_length)
 *  \brief reads information stored in the memory of the tag
 *  \param memory_address is the memory address where it performs the read
 *  \param data_length is the number of blocks to read (usually four or more bytes for
any type of tag)
*/
void m1_read_tag (unsigned int8 memory_address,unsigned int8 data_length)
{
   unsigned int8 count = 0; // Byte counter
   unsigned int8 length = 0; // Response message length

   restart_wdt(); // Resets the Watch Dog timer

// Fields to include
   m1_request_fields=0x00; // Reset the request fields
   m1_request_fields.type=1;
   m1_request_fields.tid=1;
   m1_request_fields.starting_block=1;
   m1_request_fields.number_of_blocks=1;

// Fields values
   m1_buffer.flag=M1_FLAG_CRC_RID;
   m1_buffer.command=M1_COMMAND_READ_TAG;
   m1_buffer.starting_block=memory_address;
```

```
   m1_buffer.number_of_blocks=data_length;

// Build the request and send
   m1_build_request(m1_request_string);
   m1_send_request(m1_request_string,m1_response_string);

// Stores the response data in PIC's memory (can be more than a byte)
   m1_user_data_length=0;
   length = m1_response_string[1];
   for(count = 3; count < length; count++)
   {
      m1_user_data_string[count-3] = m1_response_string[count];
      m1_user_data_length++;
   }
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void m1_write_tag (unsigned int8 memory_address,unsigned int8
data string[],unsigned int8 data length)
 *  \brief writes information stored in the memory of the tag
 *  \param data_string[] is a pointer to the data string that stores the data to write
in the memory of the tag
 *  \param memory_address is the memory address where it performs the write
 *  \param data_length is the number of blocks to write (usually four or more bytes for
any type of tag)
*/
void m1_write_tag (unsigned int8 memory_address,unsigned int8 data_string[],unsigned
int8 data_length)
{
   unsigned int8 count; // Byte counter

   restart_wdt(); // Resets the Watch Dog timer

// Conversion of blocks to bytes
   if (m1_buffer.type==(M1_TYPE_ISO15693||M1_TYPE_ICODESL1||M1_TYPE_GEMWAVEC210))
      m1_bytexblock=4;

// Fields to include
   m1_request_fields=0x00; // Reset the request fields
   m1_request_fields.type=1;
   m1_request_fields.tid=1;
   m1_request_fields.starting_block=1;
   m1_request_fields.number_of_blocks=1;
   m1_request_fields.data=1;

// Fields values
   m1_buffer.flag=M1_FLAG_CRC_RID;
   m1_buffer.command=M1_COMMAND_WRITE_TAG;
   m1_buffer.starting_block=memory_address;
   m1_buffer.number_of_blocks=data_length;
   for (count=0;count<(m1_buffer.number_of_blocks*m1_bytexblock);count++)
      m1_buffer.data[count]=data_string[count];

// Build the request and send
   m1_build_request(m1_request_string);
   m1_send_request(m1_request_string,m1_response_string);
}
```

F.2 File: rfidSkyetekM1_library.h

```
/*!\file rfidSkyetekM1 library.h
   \brief Introduces functions, constants and variables for managing RFID modules using
I2C
 *
 * For more details refer to "SkyeTek_ProtocolV2_071102.pdf"
 *
*
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
 * Code lines mark with //** can be modified according to customer's requirements.
*/

////////////////////////////////////////////////////////////////////////////////
// DEFINITIONS
//
////////////////////////////////////////////////////////////////////////////////


// CRC

/*!\def CRC_PRESET
 * \brief preset value for calculating CRC
*/
#define CRC_PRESET 0x0000

/*!\def CRC_CCITT_KERMIT
 * \brief value of the polynomial for the CRC calculation
*/
#define CRC_CCITT_KERMIT 0x8408

////////////////////////////////////////////////////////////////////////////////

// DATA DEFINITIONS

/*!\def M1_TID_LENGHT
 * \brief maximum number of Tags ID can be stored in the pic memory
*/
#define M1_MAX_TID_LENGHT  8 //**

/*!\def M1_MAX_DATA
 * \brief maximum number of bytes can be transferred in a i2c communication
*/
#define M1_MAX_DATA  30 //**

/*!\def M1_MAX_READERS
 * \brief maximum number of RID can be stored in the pic memory
*/
#define M1_MAX_READERS  2 //**

/*!\def M1_MAX_TAGS
 * \brief maximum number of Tags ID can be stored in the pic memory
*/
#define M1_MAX_TAGS  3 //**

////////////////////////////////////////////////////////////////////////////////

// SKYETEK M1_I2C_DEFINITIONS
```

```
/*!\def M1_ADDRESS_WRITE
 * \brief Skyetek M1 I2C address (0x3F) with the eight bit reset (1-read/0-write)
*/
#define M1_ADDRESS_WRITE 0x7E

/*!\def M1_ADDRESS_READ
 * \brief Skyetek M1 I2C address (0x3F) with the eight bit set (1-read/0-write)
*/
#define M1_ADDRESS_READ 0x7F

/*!\def STX
 * \brief start byte defined by the Skyetek protocol to start any comunication
*/
#define STX 0x02

////////////////////////////////////////////////////////////////////////////////

// TAG TYPES

/*!\def M1_TYPE_AUTODETECT
 * \brief any compatible tag type
*/
#define M1_TYPE_AUTODETECT   0x00

/*!\def M1_TYPE_ISO15693
 * \brief specific tag type
 * \four bytes for each block
*/
#define M1_TYPE_ISO15693   0x01

/*!\def M1_TYPE_ICODESL1
 * \brief specific tag type
 * \four bytes for each block
*/
#define M1_TYPE_ICODESL1   0x02

/*!\def M1_TYPE_TAGITHF
 * \brief specific tag type
 * \four bytes for each block
*/
#define M1_TYPE_TAGITHF   0x03

/*!\def MM1_TYPE_ISO14443A
 * \brief specific tag type
 * \sixteen bytes for each block
*/
#define M1_TYPE_ISO14443A   0x04

/*!\def M1_TYPE_PICOTAG
 * \brief specific tag type
 * \eight bytes for each block
*/
#define M1_TYPE_PICOTAG   0x06

/*!\def M1_TYPE_GEMWAVEC210
 * \brief specific tag type
 * \
*/
#define M1_TYPE_GEMWAVEC210   0x08

/*!\def M1_TYPE_MIFAREULTRALIGHT
 * \brief specific tag type
 * \four bytes for each block
*/
#define M1_TYPE_MIFAREULTRALIGHT   0x0A

/*!\def M1_TYPE_LRI64
 * \brief specific tag type
 * \
*/
#define M1_TYPE_LRI64   0x0B


////////////////////////////////////////////////////////////////////////////////

// RFID COMMANDS
```

```
// TAG commands

/*!\def M1_COMMAND_SELECT_TAG
 * \brief
*/
#define M1_COMMAND_SELECT_TAG   0x14


// Read commands

/*!\def M1_COMMAND_READ_MEM
 * \brief
*/
#define M1_COMMAND_READ_MEM   0x21

/*!\def M1_COMMAND_READ_SYS
 * \brief
*/
#define M1_COMMAND_READ_SYS   0x22

/*!\def M1_COMMAND_READ_TAG
 * \brief
*/
#define M1_COMMAND_READ_TAG   0x24


// Write commands

/*!\def M1_COMMAND_WRITE_MEM
 * \brief
*/
#define M1_COMMAND_WRITE_MEM   0x41

/*!\def M1_COMMAND_WRITE_SYS
 * \brief
*/
#define M1_COMMAND_WRITE_SYS   0x42

/*!\def M1_COMMAND_WRITE_TAG
 * \brief
*/
#define M1_COMMAND_WRITE_TAG   0x44

////////////////////////////////////////////////////////////////////////////////

// RFID FLAG TYPES

/*!\def M1 FLAG CRC
 * \brief bits in the flag field that must be set for crc use
*/
#define M1_FLAG_CRC   0x20


/*!\def M1 FLAG CRC&RID
 * \brief bits in the flag field that must be set for crc and reader id use
*/
#define M1_FLAG_CRC_RID   0x60


////////////////////////////////////////////////////////////////////////////////
// VARIABLES
//
////////////////////////////////////////////////////////////////////////////////

// SKYETEK M1 VARIABLES AND STRUCTURES

// Tag management variables

/*! \var m1_tag_detected;
 *  \brief sets the value of bytes per block
*/
unsigned int8 m1_bytexblock=1; // Setting the value of bytes per block conversion
according to the type of label

////////////////////////////////////////////////////////////////////////////////
```

```
// Request and response structures

/*! \var struct m1_flag
 *  \brief structure that indicates which flags must be enabled to modify the behavior
of the command
*/
struct m1_flag
{
   /*! see protocol documentation for more information */
   unsigned int1 loop_f:1;
   /*! see protocol documentation for more information */
   unsigned int1 inv_f:1;
   /*! see protocol documentation for more information */
   unsigned int1 lock_f:1;
   /*! see protocol documentation for more information */
   unsigned int1 rf_f:1;
   /*! see protocol documentation for more information */
   unsigned int1 afi_f:1;
   /*! see protocol documentation for more information */
   unsigned int1 crc_f:1;
   /*! see protocol documentation for more information */
   unsigned int1 tid_f:1;
   /*! see protocol documentation for more information */
   unsigned int1 rid_f:1;
};

/*! \var struct m1_buffer
 *  \brief structure that contain the fields necessary to send a request to the reader
as indicated by the protocol also acts as a buffer storing data from the last operation
 *
*/
struct
{
   /*! total message length */
   unsigned int8 length;
   /*! modification flag register bits that define the scope of application of the
message */
   struct m1_flag flag;
   /*! command to perform */
   unsigned int8 command;
   /*! response code received to perform */
   unsigned int8 response_code;
   /*! reader ID field */
   unsigned int8 rid;
   /*! specifies the tag type with wich the RF module attempts to communicate */
   unsigned int8 type;
   /*! tag ID field */
   unsigned int8 tid[M1_MAX_TID_LENGHT];
   /*! application field identifier */
   unsigned int8 afi;
   /*! first target memory block to be written or read */
   unsigned int8 starting_block;
   /*! number of memory blocks to read or write */
   unsigned int8 number_of_blocks;
   /*! data field */
   unsigned int8 data[M1_MAX_DATA];
   /*! cyclic redundancy check field (*) */
   unsigned int8 crc[2];
} m1_buffer;

/*! \var struct m1_request_fields
 *  \brief structure that indicates which fields must be included in the request
*/
struct
{
   /*! the field reader identification will be included */
   unsigned int rid:1;
   /*! the field tag type will be included */
   unsigned int type:1;
   /*! the field tag identification will be included */
   unsigned int tid:1;
   /*! the field application field identifier will be included */
   unsigned int afi:1;
   /*! the field starging block will be included */
   unsigned int starting_block:1;
   /*! the field number of blocks will be included */
   unsigned int number_of_blocks:1;
```

```
   /*! the field data will be included */
   unsigned int data:1;
   /*! the field crc will be included */
   unsigned int crc:1;
} m1_request_fields;

////////////////////////////////////////////////////////////////////////////////
////////////

// Request and response arrays

/*! \var int8 m1_request_string
 *  \brief stores the data received by I2C
*/
unsigned int8 m1_request_string[M1_MAX_DATA];

/*! \var int8 m1_response_string
 *  \brief stores the data received by I2C
*/
unsigned int8 m1_response_string[M1_MAX_DATA];

/*! \var int8 m1_user_data_string
 *  \brief stores the string user-defined data to be read or written in TAG or System
memory
*/
unsigned int8 m1_user_data_string[M1_MAX_DATA];

/*! \var int8 m1_user_data_length
 *  \brief stores the number of bytes contained in m1_user_data_string
*/
unsigned int8 m1_user_data_length;
```

F.3 File: adxl1345lib.c

```
/*! \file adxl345lib.c
 * \brief Introduces functions for the control of the accelerometer ADXL345
 *
*
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
 */

#include "adxl345lib.h"

////////////////////////////////////////////////////////////////////////////////
// LOW LEVEL LAYER
//
////////////////////////////////////////////////////////////////////////////////

/*! \fn void adxl_read_register ( unsigned int8 register, unsigned int8 data )
 *  \brief read a byte of data from ADXL 345 by I2C
 *
 *  \param reg_address stores the address to be readed
 *  \param *data is a pointer to the variable that stores the data will be readed
*/
void adxl_read_register ( unsigned int8 reg_address, unsigned int8 *data )
{
   restart_wdt(); // Resets the Watch Dog timer

// Request block

   i2c_start(); // Starts the communication
   i2c_write(ADXL_ADDRESS_WRITE); // Send write address
   i2c_write(reg_address); //


// Response block

   i2c_start();     // Start condition
   i2c_write(ADXL_ADDRESS_READ); // Send read address

   *data = i2c_read(1); //

   i2c_read(0); // Releases the clock line with a void reading
   i2c_stop();  // Stop condition
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void adxl_i2c_read_register ( unsigned int8 register, unsigned int8 data )
 *  \brief write a byte of data from ADXL 345 by I2C
 *
 *  \param reg_address stores the address to be written
 *  \param *data is a pointer to the variable that stores the data will be written
*/
void adxl_write_register ( unsigned int8 reg_address, unsigned int8 data )
{
   restart_wdt(); // Resets the Watch Dog timer

// Request block
```

```
   i2c_start(); // Starts the communication
   i2c_write(ADXL_ADDRESS_WRITE); // Send write address
   i2c_write(reg_address); //
   i2c_write(data); //
   i2c_stop();      // Stop condition

}

////////////////////////////////////////////////////////////////////////////////
// MIDDLE LEVEL LAYER
//
////////////////////////////////////////////////////////////////////////////////

/*! \fn void adxl_sleep ( )
 *  \brief puts the device into sleep or wake up
 *
 * \param state 0 - normal mode, 1 - sleep mode
 *
*/
void adxl_sleep (unsigned int state )
{

   unsigned int8 buffer; // Temporarily stores the configuration data to send

   adxl_power_ctl.sleep=state;
   buffer=adxl_power_ctl;
   adxl_write_register ( ADXL_POWERCTL_ADDRESS, buffer ); // Write the value in memory

}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void adxl_configure ( )
 *  \brief configure registers of the sensor for a desired mode of operation
 *
*/
void adxl_configure ( )
{
   unsigned int8 buffer; // Temporarily stores the configuration data to send

   restart_wdt(); // Resets the Watch Dog timer

// Configure Data Rate & Banwidth
   adxl_bw_rate=0; // Clear the pattern
   adxl_bw_rate.rate0=0;
   adxl_bw_rate.rate1=0;
   adxl_bw_rate.rate2=0;
   adxl_bw_rate.rate3=1; // Configures the output data rate to 100Hz
   adxl_bw_rate.low_power=0; // Low Power not selected
   buffer=adxl_bw_rate;
   adxl_write_register ( ADXL_BWRATE_ADDRESS, buffer ); // Write the value in memory

   delay_ms(10);
// Configure the Power control Register
   adxl_power_ctl=0; // Clear the pattern
   adxl_power_ctl.measure_mode=1; // (0 standby mode / 1 measurement mode)
   buffer=adxl_power_ctl;
   adxl_write_register ( ADXL_POWERCTL_ADDRESS, buffer ); // Write the value in memory

   delay_ms(10);
// Configure the Data Format Register
   adxl_data_format=0; // Clear the pattern
   adxl_data_format.rangehigh=0;
   adxl_data_format.rangelow=1;  // Configures the range of measurement to 4g
   buffer=adxl_data_format;
   adxl_write_register ( ADXL_DATA_FORMAT_ADDRESS, buffer ); // Write the value in
memory

   delay_ms(10);
// Configure the Operation Mode
   adxl_write_register ( ADXL_FIFO_CTL_ADDRESS, ADXL_STREAM_MODE ); // Write the value
in memory
}

////////////////////////////////////////////////////////////////////////////////
```

```
/*! \fn void adxl_capture_axis_data ( unsigned int8 *data_string )
 *  \brief gets sensor data and stores in PIC memory
 *
*/
void adxl_capture_axis_data ( unsigned int8 *data_string )
{
   unsigned int8 buffer_high;
   unsigned int8 buffer_low;

   restart_wdt(); // Resets the Watch Dog timer

// Get X-Axis Data
   adxl_read_register ( ADXL_DATA_X1_ADDRESS, &buffer_high ); // Read high register
   adxl_read_register ( ADXL_DATA_X0_ADDRESS, &buffer_low ); // Read low register

// Mount data and store (String)
   *data_string=buffer_high;data_string++;
   *data_string=buffer_low;data_string++;

// Mount data and store (Single)
   adxl_axis.x = (buffer_high) << 8;
   adxl_axis.x += (buffer_low);


// Get Y-Axis Data
   adxl_read_register ( ADXL_DATA_Y1_ADDRESS, &buffer_high ); // Read high register
   adxl_read_register ( ADXL_DATA_Y0_ADDRESS, &buffer_low ); // Read low register

// Mount data and store (String)
   *data_string=buffer_high;data_string++;
   *data_string=buffer_low;data_string++;

// Mount data and store (Single)
   adxl_axis.y = (buffer_high) << 8;
   adxl_axis.y += (buffer_low);

// Get Z-Axis Data
   adxl_read_register ( ADXL_DATA_Z1_ADDRESS, &buffer_high ); // Read high register
   adxl_read_register ( ADXL_DATA_Z0_ADDRESS, &buffer_low ); // Read low register

// Mount data and store (String)
   *data_string=buffer_high;data_string++;
   *data_string=buffer_low;data_string++;

// Mount data and store (Single)
   adxl_axis.z = (buffer_high) << 8;
   adxl_axis.z += (buffer_low);
}

////////////////////////////////////////////////////////////////////////////////////////
```

F.4 File: adxl1345lib.h

```
/*! \file stick9DOF.h
 * \brief Introduces constants and variables for the control of the accelerometer
ADXL345
 *
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
                                                    _____________________\n
 * Author: Person who modifies the firmware\n
 * Update: For explaining a change in firmware\n
 * ____________________________________________________________________\n
 *
 */

////////////////////////////////////////////////////////////////////////////////////
// DEFINITIONS
//
////////////////////////////////////////////////////////////////////////////////////

// ADXL 345 I2C DEFINITIONS

/*!\def ADXL_ADDRESS_WRITE
 * \brief ADXL 345 address () with the eight bit reset (1-read/0-write)
*/
#define ADXL_ADDRESS_WRITE 0xA6

/*!\def M1 ADDRESS READ
 * \brief ADXL 345 address () with the eight bit set (1-read/0-write)
*/
#define ADXL_ADDRESS_READ 0xA7


////////////////////////////////////////////////////////////////////////////////////


// ADXL 345 REGISTERS

/*!\def ADXL DEVID ADDRESS
 * \brief holds a fixed device ID
*/
#define ADXL_DEVID_ADDRESS 0x00

/*!\def ADXL OFFSET (Axis)n ADDRESS
 * \brief user-set offset adjustments in twos complement format with a scale factor of
15.6 mg/LSB (that is, 0x7F = +2 g)
*/
#define ADXL OFFSET X0 ADDRESS 0x1E
#define ADXL OFFSET Y0 ADDRESS 0x1F
#define ADXL_OFFSET_Z0_ADDRESS 0x20

/*!\def ADXL_BWRATE_ADDRESS
 * \brief
*/
#define ADXL_BWRATE_ADDRESS 0x2C

/*!\def ADXL_POWERCTL_ADDRESS
 * \brief
*/
```

```
#define ADXL_POWERCTL_ADDRESS 0x2D

/*!\def ADXL_DATA_FORMAT
 * \brief
*/
#define ADXL_DATA_FORMAT_ADDRESS 0x31

/*!\def ADXL_DATA_(Axis)n_ADDRESS
 * \brief
*/
#define ADXL_DATA_X0_ADDRESS 0x32
#define ADXL_DATA_X1_ADDRESS 0x33

#define ADXL DATA Y0 ADDRESS 0x34
#define ADXL_DATA_Y1_ADDRESS 0x35

#define ADXL_DATA_Z0_ADDRESS 0x36
#define ADXL_DATA_Z1_ADDRESS 0x37

/*!\def ADXL_FIFO_CTL_ADDRESS
 * \brief
*/
#define ADXL_FIFO_CTL_ADDRESS 0x38

////////////////////////////////////////////////////////////////////////////////////

// ADXL 345 OPERATION MODES

/*!\def ADXL_STREAM_MODE
 * \brief
*/
#define ADXL_STREAM_MODE 0x80

////////////////////////////////////////////////////////////////////////////////////
// VARIABLES
//
////////////////////////////////////////////////////////////////////////////////////

/*! \var int8 adxl_id
 *  \brief stores the device id
*/
unsigned int8 adxl_id;

/*! \var struct adxl_bw_rate
 *  \brief
*/
struct
{
   /*! Rate 0 */
   unsigned int1 rate0;
   /*! Rate 1 */
   unsigned int1 rate1;
   /*! Rate 2 */
   unsigned int1 rate2;
   /*! Rate 3 */
   unsigned int1 rate3;

   ////////////////////////////////////////////////////////////////////
   // D3 // D2 // D1 // D0 //   Bandwidth Hz //  Output Data Rate Hz  //
   ////////////////////////////////////////////////////////////////////
   // 1  // 1  // 1  // 1  //     1600       //        3200          //
   // 1  // 1  // 1  // 0  //      800       //        1600          //
   // 1  // 1  // 0  // 1  //      400       //         800          //
   // 1  // 1  // 1  // 0  //      200       //         400          //
   // 1  // 0  // 1  // 1  //      100       //         200          //
   // 1  // 0  // 0  // 0  //       50       //         100          //
   // 1  // 0  // 1  // 1  //       25       //          50          //
   // 1  // 0  // 1  // 0  //       12.5     //          25          //
   // 0  // 1  // 0  // 1  //        6.25    //          12.5        //
   // 0  // 1  // 0  // 0  //        3.125   //           6.25       //
   ////////////////////////////////////////////////////////////////////

   /*! Sleep Bit */
   unsigned int1 low_power; // 1 selects reduced power operation, which has somewhat
higher noise

   /*! Not used */
```

```
   unsigned int1 notused5;
   /*! Not used */
   unsigned int1 notused6;
   /*! Not used */
   unsigned int1 notused7;

}adxl_bw_rate;

/*! \var struct adxl_power_ctl
 *  \brief
*/
struct
{
   /*! Wake up Low */
   unsigned int1 wkuplow;
   /*! Wake up High */
   unsigned int1 wkuphigh;

   // Frequency of readings in sleep mode

   ////////////////////////////////
   // D1 // D0 //   Frecuency  //
   ////////////////////////////////
   // 0  // 0  //     8Hz      //
   // 0  // 1  //     4Hz      //
   // 1  // 0  //     2Hz      //
   // 1  // 1  //     1Hz      //
   ////////////////////////////////

   /*! Sleep Bit */
   unsigned int1 sleep; // 0 - normal mode, 1 - sleep mode
   /*! Measure Bit */
   unsigned int1 measure_mode; // 0 - standby mode, 1 - measurement mode
    /*! Autosleep */
   unsigned int1 autosleep; // Switch to sleep mode when inactivity is detected
   /*! Link Bit */
   unsigned int1 link; // This bit serially links the activity and inactivity functions
   /*! Not used */
   unsigned int1 notused6;
   /*! Not used */
   unsigned int1 notused7;

}adxl_power_ctl;


/*! \var struct adxl_data_format
 *  \brief
*/
struct
{
   /*! Range Low */
   unsigned int1 rangelow;
   /*! Range High */
   unsigned int1 rangehigh;

   ////////////////////////////////////
   //  High  /  Low   /   Meaning    //
   ////////////////////////////////////
   //    0   /    0   /     2G       //
   //    0   /    1   /     4G       //
   //    1   /    0   /     8G       //
   //    1   /    1   /    12G       //
   ////////////////////////////////////


   /*! Justify Bit */
   unsigned int1 justify; // 1 in the justify bit selects left (MSB) justified mode, 0
selects right justified mode with sign extension
   /*! FULL RES Bit */
   unsigned int1 fullres; // 1 - full resolution mode, where the output resolution
increases with the g range set by the range bits to maintain a 4 mg/LSB scale factor
                          // 0 - 10-bit mode, where the range bits determine the maximum
g range and scale factor

   /*! Not used */
   unsigned int1 notused;
   /*! INT_INVERT Bit */
```

```
   unsigned int1 invert; // 0 - sets the interrupts to active high, 1 - sets the
interrupts to active low
   /*! SPI Bit */
   unsigned int1 spi;  // 1 - device to 3-wire SPI mode, 0 - device to 4-wire SPI mode
   /*! SELF TEST Bit */
   unsigned int1 test;  // applies a self-test force to the sensor, causing a shift in
the output data

}adxl_data_format;


/*! \var struct adxl_axis
 *  \brief
*/
struct
{
   /*! ... */
   unsigned int16 x;
   /*! ... */
   unsigned int16 y;
   /*! ... */
   unsigned int16 z;

}adxl_axis;
```

F.5 File: hmc5843lib.c

```
/*! \file hmc5843lib.c
 * \brief Introduces functions for the control of the magnetometer HMC5843
 *
*
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
 */

#include "hmc5843lib.h"

////////////////////////////////////////////////////////////////////////////////////
// LOW LEVEL LAYER
//
////////////////////////////////////////////////////////////////////////////////////

/*! \fn void hmc_read_register ( unsigned int8 register, unsigned int8 data )
 *  \brief read a byte of data from  HMC 5843 by I2C
 *
 *  \param reg_address stores the address to be readed
 *  \param *data is a pointer to the variable that stores the data will be readed
*/
void hmc_read_register ( unsigned int8 reg_address, unsigned int8 *data )
{
   restart_wdt(); // Resets the Watch Dog timer

// Request block
   i2c_start(); // Starts the communication
   i2c_write(HMC_ADDRESS_WRITE); // Send write address
   i2c_write(reg_address); //

   i2c_stop();      // Stop condition

// Response block
   i2c_start();     // Start condition
   i2c_write(HMC_ADDRESS_READ); // Send read address

   *data = i2c_read(1); //

   i2c_read(0); // Releases the clock line with a non ack reading
   i2c_stop();      // Stop condition
}

////////////////////////////////////////////////////////////////////////////////////

/*! \fn void hmc_read_register ( unsigned int8 register, unsigned int8 data )
 *  \brief write a byte of data from  HMC 5843 by I2C
 *
 *  \param reg_address stores the address to be written
 *  \param *data is a pointer to the variable that stores the data will be written
*/
void hmc_write_register ( unsigned int8 reg_address, unsigned int8 data )
{
   restart_wdt(); // Resets the Watch Dog timer

// Request block

   i2c_start(); // Starts the communication
```

```
   i2c_write(HMC_ADDRESS_WRITE); // Send write address
   i2c_write(reg_address); //
   i2c_write(data); //
   i2c_stop();      // Stop condition
}


////////////////////////////////////////////////////////////////////////////////
// MIDDLE LEVEL LAYER
//
////////////////////////////////////////////////////////////////////////////////

/*! \fn void hmc_sleep ( )
 *  \brief puts the device into sleep or wake up
 *
 * \param state 0 - normal mode, 1 - iddle mode 2 - sleep mode
 *
*/
void hmc_sleep (unsigned int state )
{

   switch (state)
   {
      case 0:
         hmc_write_register ( HMC_MODE_ADDRESS, HMC_CONTINUOUS_MODE ); // Write the
value in memory
      break;

      case 1:
         hmc_write_register ( HMC_MODE_ADDRESS, HMC_IDLE_MODE ); // Write the value in
memory
      break;

      case 2:
         hmc_write_register ( HMC_MODE_ADDRESS, HMC_SLEEP_MODE ); // Write the value in
memory
      break;
   }
}


/*! \fn void hmc_configure ( )
 *  \brief configure registers of the sensor for a desired mode of operation
 *
*/
void hmc_configure ( )
{
   unsigned int8 buffer; // Temporarily stores the configuration data to send

   restart_wdt(); // Resets the Watch Dog timer

// Configure Register A
   hmc_configuration_a=0; // Clear the pattern

   hmc_configuration_a.do0=0;
   hmc_configuration_a.do1=0;
   hmc_configuration_a.do2=1; // Output Data Rates ( 1 0 0 by default for 10 Hz )

   hmc_configuration_a.ms0=0;
   hmc_configuration_a.ms1=0; // Measurement Mode ( 0 0 by default for Normal
measurement configuration )

   buffer=hmc_configuration_a;
   hmc_write_register ( HMC_CONFIGURATION_A_ADDRESS, buffer ); // Send the value to the
device

// Configure Register B
   hmc_write_register ( HMC_CONFIGURATION_B_ADDRESS, HMC_1000MGA );  // Set the gain to
1300 and the range to 1.0 Ga

// Configure Mode Register
   hmc_write_register ( HMC_MODE_ADDRESS, HMC_CONTINUOUS_MODE );
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void hmc_capture_channel_data ( unsigned int8 *data_string )
```

```
 *  \brief gets sensor data and stores in PIC memory
 *
*/
void hmc_capture_channel_data ( unsigned int8 *data_string )
{

   unsigned int8 buffer_high;
   unsigned int8 buffer_low;

// Get X-Channel Data

   hmc_read_register ( HMC_DATA_XMSB_ADDRESS, &buffer_high ); // Read high register
   hmc_read_register ( HMC_DATA_XLSB_ADDRESS, &buffer_low ); // Read low register

// Mount data and store (String)

   *data_string=buffer_high;data_string++;
   *data_string=buffer_low;data_string++;

// Mount data and store (Single)

   hmc_channel.x = (buffer_high) << 8;
   hmc_channel.x += (buffer_low);


// Get Y-Channel Data

   hmc_read_register ( HMC_DATA_YMSB_ADDRESS, &buffer_high ); // Read high register
   hmc_read_register ( HMC_DATA_YLSB_ADDRESS, &buffer_low ); // Read low register

// Mount data and store (String)

   *data_string=buffer_high;data_string++;
   *data_string=buffer_low;data_string++;

// Mount data and store (Single)

   hmc_channel.y = (buffer_high) << 8;
   hmc_channel.y += (buffer_low);


// Get Z-Channel Data

   hmc_read_register ( HMC_DATA_ZMSB_ADDRESS, &buffer_high ); // Read high register
   hmc_read_register ( HMC_DATA_ZLSB_ADDRESS, &buffer_low ); // Read low register

// Mount data and store (String)

   *data_string=buffer_high;data_string++;
   *data_string=buffer_low;data_string++;

// Mount data and store (Single)

   hmc_channel.z = (buffer_high) << 8;
   hmc_channel.z += (buffer_low);


}

////////////////////////////////////////////////////////////////////////////////////////
```

F.6 File: hmc5843.h

```
/*! \file hmc5843.h
* \brief Introduces constants and variables for the control of the magnetometer HMC5843
*
* ..
* ..
*
* Author: ..\n
* Copyright: .. \n
* Date: .. \n
* Update record:\n
* _____________________________________________________\n
* Author: Person who makes a change in firmware\n
* Update: For explaining a change in firmware\n
* _____________________________________________________\n
*/


////////////////////////////////////////////////////////////////////////////////
// DEFINITIONS
//
////////////////////////////////////////////////////////////////////////////////

// HMC 5843 I2C DEFINITIONS

/*!\def HMC ADDRESS WRITE
 * \brief HMC 5843 address () with the eight bit reset (1-read/0-write)
*/
#define HMC_ADDRESS_WRITE 0x3C

/*!\def M1 ADDRESS READ
 * \brief HMC 5843 address () with the eight bit set (1-read/0-write)
*/
#define HMC_ADDRESS_READ 0x3D

////////////////////////////////////////////////////////////////////////////////

// HMC 5843 REGISTERS

/*!\def HMC_CONFIGURATION_A_ADDRESS
 * \brief HMC 5843 Configuration Register A
*/
#define HMC_CONFIGURATION_A_ADDRESS 0x00

/*!\def HMC CONFIGURATION B ADDRESS
 * \brief HMC 5843 Configuration Register B
*/
#define HMC_CONFIGURATION_B_ADDRESS 0x01

/*!\def HMC MODE ADDRESS
 * \brief HMC 5843 Mode Register
*/
#define HMC_MODE_ADDRESS 0x02

/*!\def HMC DATA (Axis)xSB ADDRESS
 * \brief Data Output (Axis) xSB Register wich x mentions L or M (Less significant or
most significant)
*/
#define HMC DATA XMSB ADDRESS 0x03
#define HMC_DATA_XLSB_ADDRESS 0x04

#define HMC DATA YMSB ADDRESS 0x05
#define HMC_DATA_YLSB_ADDRESS 0x06

#define HMC DATA ZMSB ADDRESS 0x07
#define HMC_DATA_ZLSB_ADDRESS 0x08

/*!\def HMC STATUS REGISTER ADDRESS
 * \brief HMC 5843 Status Register
*/
#define HMC_STATUS_REGISTER_ADDRESS 0x09

////////////////////////////////////////////////////////////////////////////////

// HMC 5843 OPERATION MODES
```

```
/*!\def HMC_STREAM_MODE
 * \brief
*/
#define HMC_CONTINUOUS_MODE 0x00

/*!\def HMC_STREAM_MODE
 * \brief
*/
#define HMC_SINGLE_MODE 0x01

/*!\def HMC_STREAM_MODE
 * \brief
*/
#define HMC_IDLE_MODE 0x02

/*!\def HMC_STREAM_MODE
 * \brief
*/
#define HMC_SLEEP_MODE 0x03

////////////////////////////////////////////////////////////////////////////////

// HMC 5843 SENSOR INPUT FIELD RANGE (Values for register Configuration B)


/*!\def HMC 700MGA
 * \brief Output Range 0xF800-0x07FF(-2048-2047 ) Gain 1620
*/
#define HMC_700MGA 0x00

/*!\def HMC 1000MGA
 * \brief Output Range 0xF800-0x07FF(-2048-2047 ) Gain 1300 (DEFAULT)
*/
#define HMC_1000MGA 0x20

/*!\def HMC 1500MGA
 * \brief Output Range 0xF800-0x07FF(-2048-2047 ) Gain 970
*/
#define HMC_1500MGA 0x40

/*!\def HMC 2000MGA
 * \brief Output Range 0xF800-0x07FF(-2048-2047 ) Gain 780
*/
#define HMC_2000MGA 0x60

/*!\def HMC 3200MGA
 * \brief Output Range 0xF800-0x07FF(-2048-2047 ) Gain 530
*/
#define HMC_3200MGA 0x80

/*!\def HMC 3800MGA
 * \brief Output Range 0xF800-0x07FF(-2048-2047 ) Gain 460
*/
#define HMC_3800MGA 0xA0



////////////////////////////////////////////////////////////////////////////////
// VARIABLES
//
////////////////////////////////////////////////////////////////////////////////

/*! \var int8 hmc_id
 *  \brief device id
*/
unsigned int8 hmc_id;


/*! \var struct adxl_power_ctl
 *  \brief
*/
struct
{
   /*! Measurement Configuration Bit 0 */
   unsigned int1 ms0;
   /*! Measurement Configuration Bit 1 */
```

```
   unsigned int1 ms1;

   /*! Data Output Rate Bit 0 */
   unsigned int1 do0;
   /*! Data Output Rate Bit 1 */
   unsigned int1 do1;
   /*! Data Output Rate Bit 2 */
   unsigned int1 do2;

   ////////////////////////////////////////////////////////////////
   //  DO2   /  DO1   /  DO0   /  Minimum Data Output Rate (Hz)   //
   ////////////////////////////////////////////////////////////////
   //    0   /    0   /    0   /                       0.5        //
   //    0   /    0   /    1   /                         1        //
   //    0   /    1   /    0   /                         2        //
   //    0   /    1   /    1   /                         5        //
   //    1   /    0   /    0   /                        10        //
   //    1   /    1   /    1   /                        20        //
   //    1   /    1   /    0   /                        50        //
   ////////////////////////////////////////////////////////////////

   /*! Not used */
   unsigned int1 notused5;
   /*! Not used */
   unsigned int1 notused6;
   /*! Not used */
   unsigned int1 notused7;
}hmc_configuration_a;


/*! \var struct hmc_satatus_register
 *  \brief
*/
struct
{
   /*! Ready Bit */
   unsigned int1 ready; //  Set when data is written to all six data registers
   /*! Data output register lock */
   unsigned int1 lock;  //  This bit is set when this some but not all for of the six
data output registers have been read
   /*! Regulator Enabled Bit. */
   unsigned int1 ren;   //  This bit is set when the internal voltage regulator is
enabled.
   /*! Not used */
   unsigned int1 notused3;
   /*! Not used */
   unsigned int1 notused4;
   /*! Not used */
   unsigned int1 notused5;
   /*! Not used */
   unsigned int1 notused6;
   /*! Not used */
   unsigned int1 notused7;
}hmc_satatus_register;


/*! \var struct hmc_channel
 *  \brief
*/
struct
{
   /*! ... */
   unsigned int16 x;
   /*! ... */
   unsigned int16 y;
   /*! ... */
   unsigned int16 z;
}hmc_channel;
```

F.7 File: itg3200lib.c

```
/*! \file itg3200lib.c
 * \brief Introduces functions for the control of the gyroscope ITG3200
 *
 *
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
 */

#include "itg3200lib.h"

////////////////////////////////////////////////////////////////////////////////
// LOW LEVEL LAYER
//
////////////////////////////////////////////////////////////////////////////////

/*! \fn void itg_read_register ( unsigned int8 register, unsigned int8 data )
 *  \brief read a byte of data from ITG 3200 by I2C
 *
 *  \param reg_address stores the address to be readed
 *  \param *data is a pointer to the variable that stores the data will be readed
*/
void itg_read_register ( unsigned int8 reg_address, unsigned int8 *data )
{

// Resets the Watch Dog timer
   restart_wdt();

// Request block

   i2c_start(); // Starts the communication

   i2c_write(ITG_ADDRESS_WRITE); //

   i2c_write(reg_address); //


// Response block

   i2c_start();     // Start condition

   i2c_write(ITG_ADDRESS_READ); // Send read address

   *data = i2c_read(1); //

   i2c_read(0); // Releases the clock line with a non ack reading

   i2c_stop();      // Stop condition

}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void itg_write_register ( unsigned int8 register, unsigned int8 data )
 *  \brief write a byte of data from ITG 3200 by I2C
 *
 *  \param reg_address stores the address to be written
 *  \param *data is a pointer to the variable that stores the data will be written
```

```
*/
void itg_write_register ( unsigned int8 reg_address, unsigned int8 data )
{

// Resets the Watch Dog timer
   restart_wdt();

// Request block

   i2c_start(); // Starts the communication

   i2c_write(ITG_ADDRESS_WRITE); // Send write address

   i2c_write(reg_address);

   i2c_write(data); //

   i2c_stop();      // Stop condition

}

////////////////////////////////////////////////////////////////////////////////
// MIDDLE LEVEL LAYER
//
////////////////////////////////////////////////////////////////////////////////

/*! \fn void itg_configure ( )
 *  \brief configure registers of the sensor for a desired mode of operation
 *
*/
void itg_configure ( )
{

// Resets the Watch Dog timer
   restart_wdt();

// Configure the Sample Rate Divider Register

   itg_write_register ( ITG_SAMPLE_RATE_REG, ITG_SAMPLE_RATE_VALUE ); //Set the sample
rate to 100 hz

// Configure the DLPF, Full Scale Pass Filter Register

   itg_write_register ( ITG_SCALE_FILTER_CONF_REG, ITG_SCALE_FILTER_CONF_VALUE ); //Set
the gyroscope scale for the outputs to +/-2000 degrees per second


}


////////////////////////////////////////////////////////////////////////////////

/*! \fn void adxl_capture_temp_data ( )
 *  \brief gets sensor data and stores in PIC memory
 *
*/
void itg_capture_temp_data ()
{

   unsigned int8 buffer_high;
   unsigned int8 buffer_low;

// Resets the Watch Dog timer
   restart_wdt();

// Get Temperature Data

   itg_read_register ( ITG_TEMPH_REG, &buffer_high ); // Read high register
   itg_read_register ( ITG_TEMPL_REG, &buffer_low ); // Read low register

// Mount data and store

   itg_temp = (buffer_high) << 8;
   itg_temp += (buffer_low);


}
```

```
////////////////////////////////////////////////////////////////////////////////

/*! \fn void itg_capture_axis_data ( unsigned int8 *data_string )
 *  \brief gets sensor data and stores in PIC memory
 *
*/
void itg_capture_gyros_data (unsigned int8 *data_string )
{

   unsigned int8 buffer_high;
   unsigned int8 buffer_low;

// Resets the Watch Dog timer
   restart_wdt();

// Get X-Gyros Data

   itg_read_register ( ITG_GYRO_XH_REG, &buffer_high ); // Read high register
   itg_read_register ( ITG_GYRO_XL_REG, &buffer_low ); // Read low register

// Mount data and store (String)

   *data_string=buffer_high;data_string++;
   *data_string=buffer_low;data_string++;

// Mount data and store (Single)

   itg_gyros.x = (buffer_high) << 8;
   itg_gyros.x += (buffer_low);


// Get Y-Gyros Data

   itg_read_register ( ITG_GYRO_YH_REG, &buffer_high ); // Read high register
   itg_read_register ( ITG_GYRO_YL_REG, &buffer_low ); // Read low register

// Mount data and store (String)

   *data_string=buffer_high;data_string++;
   *data_string=buffer_low;data_string++;

// Mount data and store (Single)

   itg_gyros.y = (buffer_high) << 8;
   itg_gyros.y += (buffer_low);


// Get Z-Gyros Data

   itg_read_register ( ITG_GYRO_ZH_REG, &buffer_high ); // Read high register
   itg_read_register ( ITG_GYRO_ZL_REG, &buffer_low ); // Read low register

// Mount data and store (String)

   *data_string=buffer_high;data_string++;
   *data_string=buffer_low;data_string++;

// Mount data and store (Single)

   itg_gyros.z = (buffer_high) << 8;
   itg_gyros.z += (buffer_low);


}

////////////////////////////////////////////////////////////////////////////////
```

F.8 File: itg_3200.h

```
/*! \file itg3200.h
 * \brief Introduces constants and variables for the control of the gyroscope ITG3200
 *
 *
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
 */


////////////////////////////////////////////////////////////////////////////////
// DEFINITIONS
//
////////////////////////////////////////////////////////////////////////////////

// PS-ITG 3200 I2C DEFINITIONS

/*!\def ITG_ADDRESS_WRITE
 * \brief ITG 3200 address (0x69) with the eight bit reset (1-read/0-write)

*/
#define ITG_ADDRESS_WRITE 0xD0

/*!\def ITG ADDRESS READ
 * \brief ITG 3200 address (0x69) with the eight bit set (1-read/0-write)
*/
#define ITG_ADDRESS_READ 0xD1


////////////////////////////////////////////////////////////////////////////////

// PS-ITG 3200 I2C REGISTERS

/*!\def WHO_AM_I_REG
 * \brief Contains the I2C address of the device
*/
#define WHO_AM_I_REG    0x00

/*!\def ITG_SAMPLE_RATE_REG
 * \brief Sample Rate Divider
*/
#define ITG_SAMPLE_RATE_REG    0x15

/*!\def ITG SCALE FILTER CONF REG
 * \brief Configures several parameters related to the sensor acquisition as DLPF, Full
Scale Pass Filter
*/
#define ITG_SCALE_FILTER_CONF_REG    0x16

/*!\def ITG TEMP REG
 * \brief 8bit register of 16bit temperature output data where L means: H - High L - Low
*/
#define ITG TEMPH REG 0x1B
#define ITG_TEMPL_REG 0x1C

/*!\def ITG_GYRO_(Axis)L_REG
 * \brief 8bit register of (Axis)L gyro 16bit output data where L means: H - High L -
Low
```

```
*/
#define ITG_GYRO_XH_REG 0x1D
#define ITG_GYRO_XL_REG 0x1E
#define ITG_GYRO_YH_REG 0x1F
#define ITG_GYRO_YL_REG 0x20

#define ITG_GYRO_ZH_REG 0x21
#define ITG_GYRO_ZL_REG 0x22

/*!\def ITG_POWER_MANAGEMENT_REG
 * \brief This register is used to manage the power control, select the clock source,
and to issue a master reset to the device.
*/
#define ITG_POWER_MANAGEMENT_REG    0x3E

////////////////////////////////////////////////////////////////////////////////////

// PS-ITG 3200 CONFIGURATION VALUES

/*!\def ITG_SCALE_FILTER_CONF_VALUE ( 0x03 by default )
 * \brief Parameters of the sensor acquisition as DLPF and Full Scale Pass Filter
 * \remarks The gyros outputs are sampled internally at either 1kHz or 8kHz, determined
by the DLPF_CFG setting (see register 22).
 * \par Fsample = Finternal / (divider+1), where Finternal is either 1kHz or 8kHz
*/
#define ITG_SCALE_FILTER_CONF_VALUE    0x18

/*!\def ITG_SAMPLE_RATE_VALUE
 * \brief Parameter of the Sample Rate Divider ( 0x00 by default )
*/
#define ITG_SAMPLE_RATE_VALUE   0x09

//    FS_SEL ( B4-B3 ) // 3 - 2000°/sec
//
//    DLPF_CFG ( B0-B2 )
//    ////////////////////////////////////////////////////////////////////
//    / DLPF_CFG  // Low Pass Filter Bandwidth // Internal Sample Rate  //
//    ////////////////////////////////////////////////////////////////////
//    /  0        //          256Hz            //        8kHz           //
//    /  1        //          188Hz            //        1kHz           //
//    /  2        //          98Hz             //        1kHz           //
//    /  3        //          42Hz             //        1kHz           //
//    /  4        //          20Hz             //        1kHz           //
//    /  5        //          10Hz             //        1kHz           //
//    /  6        //          5Hz              //        1kHz           //
//    ////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////
// VARIABLES
//
////////////////////////////////////////////////////////////////////////////////////

/*! \var int8 itg_id
 *  \brief device id
*/
unsigned int8 itg_id;


/*! \var struct itg_temp
 *  \brief
*/
unsigned int16 itg_temp;


/*! \var struct itg_gyros
 *  \brief
*/
struct
{
   /*! ... */
   unsigned int16 x;
   /*! ... */
   unsigned int16 y;
   /*! ... */
   unsigned int16 z;

}itg_gyros;
```

F.9 File: zmota_Application.c

```
/*! \file itg3200.h
 * \brief Introduces constants, variables and functions for ...
 *
 *
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
 */


////////////////////////////////////////////////////////////////////////////////
// DEFINITIONS
//
////////////////////////////////////////////////////////////////////////////////

// PS-ITG 3200 I2C DEFINITIONS

/*!\def ITG_ADDRESS_WRITE
 * \brief ITG 3200 address (0x69) with the eight bit reset (1-read/0-write)

*/
#define ITG_ADDRESS_WRITE 0xD0

/*!\def ITG ADDRESS READ
 * \brief ITG 3200 address (0x69) with the eight bit set (1-read/0-write)
*/
#define ITG_ADDRESS_READ 0xD1

////////////////////////////////////////////////////////////////////////////////

// PS-ITG 3200 I2C REGISTERS

/*!\def WHO_AM_I_REG
 * \brief Contains the I2C address of the device
*/
#define WHO_AM_I_REG    0x00

/*!\def ITG SAMPLE RATE REG
 * \brief Sample Rate Divider
*/
#define ITG_SAMPLE_RATE_REG    0x15

/*!\def ITG SCALE FILTER CONF REG
 * \brief Configures several parameters related to the sensor acquisition as DLPF, Full
Scale Pass Filter
*/
#define ITG_SCALE_FILTER_CONF_REG    0x16

/*!\def ITG TEMP REG
 * \brief 8bit register of 16bit temperature output data where L means: H - High L - Low
*/
#define ITG_TEMPH_REG 0x1B
#define ITG_TEMPL_REG 0x1C

/*!\def ITG GYRO (Axis)L REG
 * \brief 8bit register of (Axis)L gyro 16bit output data where L means: H - High L -
Low  */
#define ITG_GYRO_XH_REG 0x1D
```

```
#define ITG_GYRO_XL_REG 0x1E

#define ITG GYRO YH REG 0x1F
#define ITG_GYRO_YL_REG 0x20

#define ITG_GYRO_ZH_REG 0x21
#define ITG_GYRO_ZL_REG 0x22

/*!\def ITG POWER MANAGEMENT REG
 * \brief This register is used to manage the power control, select the clock source,
and to issue a master reset to the device.
*/
#define ITG_POWER_MANAGEMENT_REG    0x3E

////////////////////////////////////////////////////////////////////////////////////

// PS-ITG 3200 CONFIGURATION VALUES

/*!\def ITG SCALE FILTER CONF VALUE ( 0x03 by default )
 * \brief Parameters of the sensor acquisition as DLPF and Full Scale Pass Filter
 * \remarks The gyros outputs are sampled internally at either 1kHz or 8kHz, determined
by the DLPF_CFG setting (see register 22).
 * \par Fsample = Finternal / (divider+1), where Finternal is either 1kHz or 8kHz
*/
#define ITG_SCALE_FILTER_CONF_VALUE    0x18

/*!\def ITG SAMPLE RATE VALUE
 * \brief Parameter of the Sample Rate Divider ( 0x00 by default )
*/
#define ITG_SAMPLE_RATE_VALUE   0x09

//    FS_SEL ( B4-B3 ) // 3 - 2000°/sec
//
//    DLPF_CFG ( B0-B2 )
//    ////////////////////////////////////////////////////////////////////
//    / DLPF CFG  // Low Pass Filter Bandwidth // Internal Sample Rate  //
//    ////////////////////////////////////////////////////////////////////
//    /  0        //          256Hz           //        8kHz           //
//    /  1        //          188Hz           //        1kHz           //
//    /  2        //          98Hz            //        1kHz           //
//    /  3        //          42Hz            //        1kHz           //
//    /  4        //          20Hz            //        1kHz           //
//    /  5        //          10Hz            //        1kHz           //
//    /  6        //          5Hz             //        1kHz           //
//    ////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////////
// VARIABLES
//
////////////////////////////////////////////////////////////////////////////////////

/*! \var int8 itg_id
 *  \brief device id
*/
unsigned int8 itg_id;


/*! \var struct itg_temp
 *  \brief
*/
unsigned int16 itg_temp;


/*! \var struct itg_gyros
 *  \brief
*/
struct
{
   /*! ... */
   unsigned int16 x;
   /*! ... */
   unsigned int16 y;
   /*! ... */
   unsigned int16 z;

}itg_gyros;
```

F.10 File: zmota_Application.c

```
/*! \file zmota_Application.c
 * \brief specific application software for testing the glove inertial Z Mota Move 2.0
on breadboard prototype
 *
*
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
 */

#include "zmota_Parse.c"

#int RTCC // 1s (also Timer0)
void  RTCC_isr(void)
{
   restart_wdt(); // Resets the Watch Dog timer
   if (peripheral_state.rfid) // Checks the peripherals states
   {
      flags.timer0=1; // Sets the flag by timer0 interruption

      disable_interrupts(INT_TIMER2);
      call_select_tag(); // Performs a search for tags in the environment
   }
}

#int TIMER1 // 262ms
void  TIMER1_isr(void)
{
   flags.timer1=1; // Sets the flag by timer1 interruption
// Makes the LEDs flashes according to operation mode
   activate_LEDS();
}

// Timer 2 interruption
#int TIMER2 // 10ms
void  TIMER2_isr(void)
{
// Checks the peripheral state
   if (peripheral_state.acel)
   {
      flags.timer2=1; // Sets the flag by timer2 interruption

      disable_interrupts(INT_RTCC);
      read_inertial_sensors (); // Takes the samples from the inertial sensor block
   }
}

#int EXT
void  EXT_isr(void)
{
   flags.ext=1; // Sets the flag by button pressed
// Sets the mode according to the mode_loop (0x00 to 0x04)
   switch(mode_selector_loop)
   {
      case DEV_MODE_ALL_OFF:
         mode_selector_loop++; // Rotates the mode value
         set_zmota_mode(DEV_MODE_ALL_OFF); // Sets the mode
```

```
            break;

            case DEV_MODE_RFID_ONLY:
               mode_selector_loop++; // Rotates the mode value
               set_zmota_mode(DEV_MODE_RFID_ONLY); // Sets the mode
            break;

            case DEV_MODE_ACEL_ONLY:
               mode_selector_loop++; // Rotates the mode value
               set_zmota_mode(DEV_MODE_ACEL_ONLY); // Sets the mode
            break;

            case DEV_MODE_ALL_ON :
               mode_selector_loop++; // Rotates the mode value
               set_zmota_mode(DEV_MODE_ALL_ON ); // Sets the mode
            break;

            default:
               mode_selector_loop=0; // Reset the mode value to defect mode
            break;
      }
   }

   #int RDA
   void  RDA_isr(void)
   {
      disable_interrupts(INT_TIMER2);
      read_UART(); // Reading UART
   }

   void main()
   {
      disable_all_interrupts(); // Disables the interrupts
      setup_wdt(WDT_OFF); // Disables the Watch Dog

   // Initialization starts
      output_bit(PIN_D0,1); // Puts OFF the LED indicator
      initialize_PIC(); // PIC 18F45K20 configuration
      initialize_Zigbee (); // Zigbee configuration
      initialize_inertial_sensors (); // ADXL345, HMC5843 & ITG3200 setup
      clear_all_interrupts(); // Clear the interrupts flags
      enable_all_interrupts(); // Enables the interrupts
      setup_wdt(WDT_ON); // Enables the Watch Dog
      set_zmota_mode(DEV_MODE_ALL_OFF); // Sets the default mode of operation
      clear_flags(); // Clear flags
      output_bit(PIN_D0,0); // Puts ON the LED indicator
   // End of the initialization

   // Main loop
      while(TRUE)
      {
         restart_wdt(); // Resets the Watch Dog timer

      // If there are data received from UART, looks for valid command
         if (flags.uart_incoming_data)
         {
            ccp_parse_input ( uart_buffer.lenght_data, uart_buffer.data );
            flags.uart_incoming_data = 0; // Reset the UART Flag after checking
            enable_interrupts(INT_RTCC);
            enable_interrupts(INT_TIMER2);
         }

      // If there are response data, mounts it and sends through UART
         if (flags.uart_outgoing_data)
         {
            ccp_send_response ( &uart_buffer.lenght_data, uart_buffer.data );
            flags.uart_outgoing_data = 0; // Reset the UART Flag after checking
            enable_interrupts(INT_RTCC);
            enable_interrupts(INT_TIMER2);
         }

         clear_UART_FIFO(); // Cleans UART FIFO
         iddle_PIC (); // Puts the PIC in IDLE Mode

      }

   }
```

F.11 File: zmota_Base_18F45K20.c

```
/*! \file zmota_Base_18F45K20.h
 * \brief specific application software for testing the glove inertial Z Mota Move 2.0
on breadboard prototype
 *
*
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
 */

#include <18F45K20.h>

#device ICD=TRUE
#device adc=8

#FUSES WDT128                       //Watch Dog Timer uses 1:128 Postscale
#FUSES INTRC                    //Internal RC Osc
#FUSES NOPROTECT                //Code not protected from reading
#FUSES BROWNOUT                 //Reset when brownout detected
#FUSES BORV22                   //Brownout reset at 2.2V
#FUSES PUT                      //Power Up Timer
#FUSES NOCPD                    //No EE protection
#FUSES STVREN                   //Stack full/underflow will cause reset
#FUSES NODEBUG                  //No Debug mode for ICD
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or B5(PIC18) used
for I/O
#FUSES NOWRT                    //Program memory not write protected
#FUSES NOWRTD                   //Data EEPROM not write protected
#FUSES IESO                     //Internal External Switch Over mode enabled
#FUSES FCMEN                    //Fail-safe clock monitor enabled
#FUSES PBADEN                   //PORTB pins are configured as analog input channels on
RESET
#FUSES NOWRTC                   //configuration not registers write protected
#FUSES NOWRTB                   //Boot block not write protected
#FUSES NOEBTR                   //Memory not protected from table reads
#FUSES NOEBTRB                  //Boot block not protected from table reads
#FUSES NOCPB                    //No Boot Block code protection
#FUSES LPT1OSC                  //Timer1 configured for low-power operation
#FUSES MCLR                     //Master Clear pin enabled
#FUSES NOXINST                  //Extended set extension and Indexed Addressing mode
disabled (Legacy mode)
#FUSES NODELAYINTOSC

#use delay(clock=8000000,restart_wdt)
#use
rs232(baud=19200,parity=N,xmit=PIN C6,rcv=PIN C7,bits=8,restart_wdt,STREAM=UART_ZIGBEE)
#use i2c(Master,Fast,sda=PIN_C4,scl=PIN_C3,restart_wdt)


#include <string.h>

////////////////////////////////////////////////////////////////////////////
// FUNCTION STATEMENTS
//
////////////////////////////////////////////////////////////////////////////

//MAIN DEVICE
```

```
void initialize_PIC(); // executes all the routines necessary for the commissioning of
the device Mota Move Z
void iddle_PIC (); // changes the operating mode of the PIC to a state of energy saving

void disable_all_interrupts();
void clear_all_interrupts();
void enable_all_interrupts();

void clear_flags();

// PERIPHERALS

void read_UART(); // takes the data from the UART
void clear_UART_FIFO(); // clears the UART_FIFO to enable the entry of new data

void activate_LEDS(); //


////////////////////////////////////////////////////////////////////////////////
// DEFINITIONS
//
////////////////////////////////////////////////////////////////////////////////

// RS-232

/*!\def TIMEOUT UART
 * \brief time limit for the execution of the Rs-232 interruption
*/
#define UART_TIMEOUT 0xFF

/*!\def MAX BUFFER LENGTH
 * \brief maximum number of bytes can be stored in the Rs-232 buffer array
*/
#define UART_MAX_BUFFER_LENGTH  40

////////////////////////////////////////////////////////////////////////////////
//VARIABLES
//
////////////////////////////////////////////////////////////////////////////////

// PIC´s REGISTERS **(only for PIC 18F45K20 or similar architecture)

/*! \var struct RCSTA
 *  \brief structure PIC 18F45K20 RCSTA register
 *
*/
struct  {
   /*! Ninth bit of Received Data */
   int8 RX9D:1;
   /*! Overrun Error bit */
   int8 OERR:1;
   /*!  Framing Error bit */
   int8 FERR:1;
   /*! Address Detect Enable bit */
   int8 ADDEN:1;
   /*! Continuous Receive Enable bit */
   int8 CREN:1;
   /*! Single Receive Enable bit */
   int8 SREN:1;
   /*! 9-bit Receive Enable bit */
   int8 RX9:1;
   /*! Serial Port Enable bit */
   int8 SPEN:1;
} RCSTA;
#byte RCSTA = 0x0FAB  //** address of the RCSTA register

/*! \var struct OSCCON
 *  \brief structure PIC 18F45K20 OSCCON register
 *
*/
struct  {
   /*! ... */
   unsigned int SCS0:1;
   /*! ... */
   unsigned int SCS1:1;
   /*! ... */
   unsigned int IOFS:1;
```

```
   /*! ... */
   unsigned int OSTS:1;
   /*! ... */
   unsigned int IRCF:3;
   /*! ... */
   unsigned int IDLEN:1;
} OSCCON;
#byte OSCCON = 0xFD3

////////////////////////////////////////////////////////////////////////////////

// SYSTEM

/*! \var struct rs232_buffer
 *  \brief structure that stores the data obtained through UART
 *
*/
struct
{
   /*! stores the RS-232 data received */
   unsigned int8 data[UART_MAX_BUFFER_LENGTH];
   /*! indicates how many data has been stored in the buffer string */
   unsigned int8 lenght_data;
}uart_buffer;


/*! \var struct flags
 *  \brief structure that stores the data obtained through UART
 *
*/
struct
{
   /*! indicates that the button has been pressed */
   int1 ext;
   /*! indicates the existence of an interruption made by timer0 */
   int1 timer0;
   /*! indicates the existence of an interruption made by timer1 */
   int1 timer1;
   /*! indicates the existence of an interruption made by timer2  */
   int1 timer2;
   /*! indicates the existence of data received via RS-232 */
   int1 uart_incoming_data;
   /*! indicates the existence of data to send via RS-232 */
   int1 uart_outgoing_data;
}flags;


/*! \var struct peripheral_state
 *  \brief structure that stores the
 *
*/
struct
{
   /*! indicates the activation or deactivation of the readings made by inertial sensor
block  */
   int1 acel;
   /*! indicates the activation or deactivation of the readings made by RFID reader  */
   int1 rfid;
}peripheral_state;


/*! \var int8 mode_selector_loop
 *  \brief
*/
unsigned int8 mode_selector_loop=0;

/*! \var int1 ledx_state
 *  \brief LED status
*/
unsigned int1 led1_state=0;
unsigned int1 led2_state=0;


////////////////////////////////////////////////////////////////////////////////
// FUNCTIONS
//
////////////////////////////////////////////////////////////////////////////////
```

```
/*! \fn void initialize_PIC( )
 *  \brief executes all the routines necessary for the commissioning of the device Mota
Move Z
 *
*/
void initialize_PIC()
{
// PIC Peripherals' setup
   setup_adc_ports(NO_ANALOGS|VSS_VDD);
   setup_adc(ADC_CLOCK_DIV_2|ADC_TAD_MUL_0);
   setup_psp(PSP_DISABLED);
   setup_comparator(NC_NC_NC_NC);
// Timers' setup
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_32); // 1s
   setup_timer_1(T1_INTERNAL|T1_DIV_BY_8); // 262ms
   setup_timer_2(T2_DIV_BY_16,83,15); //10ms
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void iddle_PIC ()
 *  \brief changes the operating mode of the PIC to a state of energy saving
 *
*/
void iddle_PIC ()
{
// Configure oscilator register
   OSCCON.SCS0=0;
   OSCCON.SCS1=1;
   OSCCON.IDLEN=1;

   setup_wdt(WDT_OFF); // Disables the Watch Dog
   sleep(); // Activates the mode
   setup_wdt(WDT_ON); // Enables the Watch Dog
}

////////////////////////////////////////////////////////////////////////////////


/*! \fn void disable_all_interrupts()
 *  \brief
 *
 *
*/
void disable_all_interrupts( )
{
   disable_interrupts(GLOBAL);
   disable_interrupts(INT_RDA);
   disable_interrupts(INT_EXT);
   disable_interrupts(INT_RTCC);
   disable_interrupts(INT_TIMER1);
   disable_interrupts(INT_TIMER2);
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void clear_interrupts()
 *  \brief
 *
*/
void clear_all_interrupts( )
{
   clear_interrupt(INT_RDA);
   clear_interrupt(INT_EXT);
   clear_interrupt(INT_RTCC);
   clear_interrupt(INT_TIMER1);
   clear_interrupt(INT_TIMER2);
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void enable_interrupts()
 *  \brief
 *
*/
void enable_all_interrupts( )
```

```
{

   enable_interrupts(INT_EXT);
   enable_interrupts(INT_RDA);
   enable_interrupts(INT_RTCC);
   enable_interrupts(INT_TIMER1);
   enable_interrupts(INT_TIMER2);
   enable_interrupts(GLOBAL);

}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void clear_flags()
 *  \brief
 *
*/
void clear_flags()
{
   delay_ms(10);
   flags.ext = 0;
   flags.timer0 = 0;
   flags.timer1 = 0;
   flags.timer2 = 0;
   flags.uart_incoming_data = 0;
   flags.uart_outgoing_data = 0;
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void read_UART()
 *  \brief takes the data from the UART
 *
*/
void read_UART()
{
   unsigned int8 count = 0; // Counter
   unsigned int8 time_over = 0; // Attempts counter

   restart_wdt(); // Resets the Watch Dog timer

// Reading Process
   while (kbhit(UART_ZIGBEE) || time_over < UART_TIMEOUT) // While there is data in the
UART and the time limit hasn´t been exceeded
   {
      if ( kbhit(UART_ZIGBEE) ) // If there is data in the UART
      {
         time_over = 0; // Reset the attempts counter
         uart_buffer.data[count] = fgetc(UART_ZIGBEE); // Takes the incoming data and
stores it in memory
         count++; // Icreases the counter
      }
      else time_over++; // Increases the time counter
   }
   flags.uart_incoming_data = 1; // Set UART flag for new data check
   uart_buffer.lenght_data = count; // Stores the legth of the new data in UART

}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void clear_UART_FIFO()
 *  \brief clears the UART_FIFO to enable the entry of new data
 *
*/
void clear_UART_FIFO()
{
   if (RCSTA.OERR) // If exist a Overrun Error (if there are garbage data in the USART
buffer)
   {
      RCSTA.CREN = 0; // Turns off continuous USART data receiving
      delay_ms(10);
      RCSTA.CREN=1; // Allows contiuos USART data receiving again
   }
}

////////////////////////////////////////////////////////////////////////////////
```

```
/*! \fn void activate_LEDS( )
 *  \brief leds routine for Simple HMI
 *
*/
void activate_LEDS()
{
// Checks the peripheral state
   if (peripheral_state.rfid)
   {
      // LED indicator flashing every 0,5 sec if the PIC is running in normal operation
      led1_state=!led1_state; // LED status changes
      output_bit(PIN_D0,led1_state); // Set the LED indicator
   }
   else
   {
      led1_state=0; // LED status off
      output_bit(PIN_D0,led1_state);  // Turns off the LED indicator
   }

// Checks the peripheral state
   if (peripheral_state.acel)
   {
   // LED indicator flashing every 0,5 sec if the PIC is running in normal operation
      led2_state=!led2_state; // LED status changes
      output_bit(PIN_D2,led2_state); // Set the LED indicator
   }
   else
   {
      led2_state=0; // LED status off
      output_bit(PIN_D2,led2_state);  // Turns off the LED indicator
   }
}
```

F.12 File: zmota_Parse.c

```
/*! \file zmota_Parse.c
 * \brief Introduces functions for CCP
 *
*
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n */

#include "zmota_Base_18F45K20.h"

#include <string.h>


#include "zmota CcpDefinitions.h"
#include "zmota_CcpFunctions.c"



////////////////////////////////////////////////////////////////////////////////
// FUNCTIONS
//
////////////////////////////////////////////////////////////////////////////////

/*! \fn void ccp_send_ack ( unsigned int8 ack )
 *  \brief responds with an acknowledgment for the specified command
 *
 *
*/
void ccp_send_ack ( unsigned int8 ack )
{

   // Return the main fields in the response

      ccp_buffer_out.end_point=ccp_buffer_in.end_point;
      ccp_buffer_out.cluster_id=ccp_buffer_in.cluster_id;
      ccp_buffer_out.event=ack; // Response with ack
      ccp_buffer_out.data_flag=1; // There are data in this response
   // Response with the cluster and the command sent
      ccp_buffer_out.data[0]=ccp_buffer_in.cluster_id;
      ccp_buffer_out.data[1]=ccp_buffer_in.command;
      ccp_buffer_out.length=3; // Updates the length field

   // Refresh flags
      flags.uart_outgoing_data = 1; // Sets the flag for sending data through the UART

}

/*! \fn void ccp_build_response ( unsigned int8 *output_lenght, unsigned int8 output[] )
 *  \brief
 *
 *
*/
void ccp_send_response ( unsigned int8 *output_lenght, unsigned int8 output[] )
{

   unsigned int8 count; // Array counter

// Puts the main fields
```

```
   output[0]=ccp_buffer_out.end_point;
   output[1]=ccp_buffer_out.cluster_id;
   output[2]=ccp_buffer_out.length;
   output[3]=ccp_buffer_out.event;

// Puts the data fields
   if (ccp_buffer_out.data_flag)
   {
      for (count=0;count<(ccp_buffer_out.length-1);count++)
      {
         output[4+count]=ccp_buffer_out.data[count];
      }
   }

// Updates the length
   *output_lenght=ccp_buffer_out.length + 3;

// Sends the response
   zbSendDataToZigbeeSink(*output_lenght, output);

}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void ccp_parse_input ( unsigned int8 input_lenght, unsigned int8 input[] )
 *  \brief parses the data received by uart and launch the correspondent command
 *
 *  \param input_lenght is the number of bytes in the array pointed to by input[]
 *  \param input[] is a pointer to the array to parse looking for a valid command
 *
*/
void ccp_parse_input ( unsigned int8 input_lenght, unsigned int8 input[] )
{

   int1 parse_error_flag=1; //  Indicates the existence of an unexpected error in the
parse function
   unsigned int8 count; // Array counter

   restart_wdt(); // Resets the Watch Dog timer

// Gets the main fields
   ccp_buffer_in.end_point=input[0];
   ccp_buffer_in.cluster_id=input[1];
   ccp_buffer_in.length=input[2];
   ccp_buffer_in.command=input[3];

// Gets the data fields if the exists
   if (ccp_buffer_in.length>1)
   {
      for (count=0;count<(ccp_buffer_in.length-1);count++)
      {
         ccp_buffer_in.data[count]=input[4+count];
      }
   }

// Message length check
   if (input_lenght == (ccp_buffer_in.length + 3))
   {
   // Cluster check
      switch (ccp_buffer_in.cluster_id)
      {
      //Basic Cluster_____________________________________________________________

      //Cluster DEVICE 0x00
         case DEVICE:

         // Command check
            switch (ccp_buffer_in.command)
            {
            //Command GET TYPE 0x00
               case GET_TYPE: // Response DEV_TYPE or DEV_ERROR

               // Resets the error flag if the command is implemented
                  parse_error_flag=0;

               //End Point check
               switch (ccp_buffer_in.end_point)
```

```
                {
                // End Point O4A BASE DEVICE 0x10
                   case O4A_BASE_DEVICE:
                      ccp_command_get_type(O4A_BASE_DEVICE); // executes the command
                   break;

                // End Point O4A_RFID_DEVICE 0x11
                   case O4A_RFID_DEVICE:
                      ccp_command_get_type(O4A_RFID_DEVICE); // executes the command
                   break;

                // End Point O4A_9DOF_DEVICE 0x12
                   case O4A_9DOF_DEVICE:
                      ccp_command_get_type(O4A_9DOF_DEVICE); // executes the command
                   break;

                // If not find endpoint returns DEV_ERROR
                   default:
                      restart_wdt(); // Resets the Watch Dog timer
                      ccp_send_error(DEV_ERROR, ERROR_NOT_SUPPORTED); // ERROR
                    break;
                }
                break;

             //Command CLUSTERS 0x01
             // case GET_CLUSTERS:


                break;
             //Command PING 0x02
                case PING:


                break;
             //Command RESET 0x03
                case RESET:


                break;
             //Command GET_MODE 0x05
                case GET_MODE: // Response DEV_MODE or DEV_ERROR

                // Resets the error flag if the command is implemented
                   parse_error_flag=0;

                //End Point check
                   switch (ccp_buffer_in.end_point)
                   {
                   // End Point O4A BASE DEVICE 0x10
                      case O4A_BASE_DEVICE:
                         ccp_command_get_mode(); // executes the command

                      break;
                   // If not find endpoint returns DEV_ERROR

                      default:
                      restart_wdt(); // Resets the Watch Dog timer
                      ccp_send_error(DEV_ERROR, ERROR_NOT_SUPPORTED); // ERROR
                      break;
                   }



                break;
             //Command SET MODE 0x06
                case SET_MODE: // Response DEV_ACK or DEV_ERROR

                  parse_error_flag=0;// Resets the error flag if the command is
implemented

                //End Point check
                   switch (ccp_buffer_in.end_point)
                   {
                   // End Point O4A_BASE_DEVICE 0x10
                      case O4A_BASE_DEVICE:
                         ccp_command_set_mode(); // executes the command
```

```
                  break;
               // If not find end point returns DEV_ERROR
                  default:
                     restart_wdt(); // Resets the Watch Dog timer
                     ccp_send_error(DEV_ERROR,ERROR_NOT_SUPPORTED); // ERROR
                  break;
               }

            break;
         //Command GET ID 0x07
            case GET_ID:
            delay_ms(1);

            break;
         //Command SET ID 0x08
            case SET_ID:
            delay_ms(1);

            break;
         //Command GET DESCRIPTION 0x09
            case GET_DESCRIPTION:
            delay_ms(1);

            break;
         //Command SET DESCRIPTION 0x0A
            case SET_DESCRIPTION:
            delay_ms(1);

            break;
         //Command GET O4A END POINTS 0x0B
          //case GET_O4A_END_POINTS:


            break;
         //Command SET_O4A_END_POINTS 0x0C
          //case SET_O4A_END_POINTS:


            break;
         //Command GET_ANNOUNCE 0x0E
            case GET_ANNOUNCE:
            delay_ms(1);

            break;
         }
      break;

   //Cluster POWER 0x02
      case POWER:

      break;

   //Sensor Cluster_____________________________________________________________

   //Cluster SENSOR 0x10
      case SENSOR:

      // Command check
         switch (ccp_buffer_in.command)
         {
            //Command GET SENSOR VALUE 0x00
            case GET_SENSOR_VALUE: // Response SENSOR_VALUE or none

            // Resets the Watch Dog timer
                  restart_wdt();

            // Resets the error flag if the command is implemented
                  parse_error_flag=0;

            // Executes the command
                  read_inertial_sensors ();


            break;
         }

      break;
```

```
        //Cluster SENSOR AUTO REFRESH  0x11
           case SENSOR_AUTO_REFRESH:
           delay_ms(1);
           break;

        //RFID Cluster_____________________________________________________________

        //Cluster RFID READER 0x20
           case RFID_READER:

          // Command check
              switch (ccp_buffer_in.command)
              {
              //Command RFID SELECT TAG 0x10
                 case RFID_SELECT_TAG: //Response RFID_TAG_DETECTED or none

                 //End Point check
                    switch (ccp_buffer_in.end_point)
                    {

                    // End Point O4A_RFID_DEVICE    0x11
                       case O4A_RFID_DEVICE:

                    // Resets the Watch Dog timer
                       restart_wdt();

                    // Resets the error flag if the command is implemented
                       parse_error_flag=0;

                    // Executes the command
                       call_select_tag();
                    }
                 break;

              // If not find end point returns DEV_ERROR
              default:

                 restart_wdt();

              // ERROR
                 ccp_send_error(DEV_ERROR,ERROR_NOT_SUPPORTED);


                 break;


              }
           break;


        //Simple HMI Input Cluster_____________________________________________________

        //Cluster SIMPLE HMI INPUT  0x30
           case SIMPLE_HMI_INPUT:
           delay_ms(1);
           break;

        //Cluster FIRMWARE CLUSTER  0xF0
           case FIRMWARE_CLUSTER:
           delay_ms(1);
           break;
        }
  // Send an error if the length doesn't match
     }
     else
     {
        ccp_send_error(DEV_ERROR,ERROR_FRAME);
     }

  // if there is an unexpected error
     if (parse_error_flag==1)
        ccp_send_error(DEV_ERROR,ERROR_UNKNOW);

  }
```

```
////////////////////////////////////////////////////////////////////////////////////

/*! \fn void ccp_send_error ( unsigned int8 error, unisgned int8 error_type )
 *  \brief responds with an error for the specified command
 *
 *  \param error
 *
*/
void ccp_send_error ( unsigned int8 error, unsigned int8 error_type )
{

   // Return the main fields in the response

      ccp_buffer_out.end_point=ccp_buffer_in.end_point;
      ccp_buffer_out.cluster_id=ccp_buffer_in.cluster_id;

   // Refresh flags

      flags.uart_outgoing_data = 1; // Sets the flag for sending data through the UART
      ccp_buffer_out.data_flag=1; // There are data in this response

   // Response with error

      ccp_buffer_out.event=error;

   // Response with the cluster and the command sent

      ccp_buffer_out.data[0]=ccp_buffer_in.cluster_id;
      ccp_buffer_out.data[1]=ccp_buffer_in.command;

   // Response with the error type

      ccp_buffer_out.data[2]=error_type;

   // Updates the length field

      ccp_buffer_out.length=4;


}
```

F.13 File: zmota_CcpDefinitions.h

```
/*! \file zmota_CcpDefinitions.h
 * \brief Introduces definitions, constants and variables for an OSGI4AmI Zigbee
comunication
 *
*
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
 */

////////////////////////////////////////////////////////////////////////////////
// FUNCTION STATEMENTS
//
////////////////////////////////////////////////////////////////////////////////
//------------------------------------------------------------------------------
// zmota_Parse.c //

void ccp_send_ack ( unsigned int8 ack );
void ccp_send_error ( unsigned int8 error, unsigned int8 error_type );

void ccp_build_response ( unsigned int8 *output_lenght, unsigned int8 output[] );
void ccp_parse_input ( unsigned int8 input_lenght, unsigned int8 input[] );

//------------------------------------------------------------------------------
// zmota_CcpFunctions.c //

// MAIN DEVICE

void set_zmota_mode ( unsigned int8 mode );

// PERIPHERALS

void initialize_Zigbee ( );
void initialize_inertial_sensors ( );

void read_all_inertial_sensors ( );
void RFID_call_select_tag ();

// CLUSTER: BASE ---------------------------------------------------------------
void ccp_command_get_type ( );
void ccp_command_get_mode ( );
void ccp_command_set_mode ( );


////////////////////////////////////////////////////////////////////////////////
// DEFINITIONS
//
////////////////////////////////////////////////////////////////////////////////

// GENERAL

/*! \def DATA_MAX
 *  \brief
 */
#define DATA_MAX  40


/*! \def NUM_ENDPOINTS
```

```
 *  \brief
 */
#define NUM_ENDPOINTS   3



/*! \const unsigned int8 base clusters
 *  \brief clusters available in the base device
 *
*/
//const unsigned int8 base_clusters[6]       = {  0x00,          0x02,          0x10,
0x11,          0x30,         0x0F0};
  //                                              DEVICE        POWER          SENSOR
SENSOR         SIMPLE        FIRMWARE_CLUSTER
  //
AUTO REFRESH     HMI INPUT

/*! \const unsigned int8 base clusters
 *  \brief clusters available in the base device
 *
*/
//const unsigned int8 rfid_clusters[2]       = {  0x02,          0xF0};
  //                                              POWER          FIRMWARE_CLUSTER
  //

/*! \const unsigned int8 base clusters
 *  \brief clusters available in the 9DOF device **** in future versions will be divided
into an endpoint for sensor
 *
*/
//const unsigned int8 ninedof clusters[3]    = {  0x02,          0x10,          0x11};
  //                                              POWER          SENSOR         SENSOR
  //                                                                         AUTO REFRESH


/*! \var struct ccp_endpoints
 *  \brief list of endpoints including their characteristics
 *
*/
const unsigned int8 ccp_endpoints [NUM_ENDPOINTS][5]                ={{ 0x10,
0x01                    0x01                    0x01               0x02}
//
O4A_BASE_DEVICE       O4A_DEVICE              O4A_DEVICE                IMPLEMENT
VERSION
                                                                        { 0x11,
0x80                    0x01                    0x01               0x01}
//
O4A RFID_DEVICE       O4A_DEVICE_RFID_READER  O4A_DEVICE                IMPLEMENT
VERSION
                                                                        { 0x12,
0x10                    0x01                    0x01               0x01}};
//
O4A 9DOF_DEVICE       O4A_DEVICE_SENSOR       O4A_DEVICE_ACCELEROMETER  IMPLEMENT
VERSION


// END POINTS -----------------------------------------------------------------

/*! \def O4A_BASE_DEVICE
 *  \brief OSGi4Ami Base Device. Device base from which they derive the rest of embedded
devices in OSGi4AMI
 */
#define O4A BASE_DEVICE    0x10  //0x01 O4A_DEVICE                      0x01 O4A_DEVICE
0x01     0x02

/*! \def O4A_RFID_DEVICE
 *  \brief OSGi4Ami RFID Device.
 */
#define O4A RFID DEVICE    0x11  //0x80 O4A_DEVICE_RFID_READER          0x80
O4A_DEVICE_RFID_READER      0x01     0x01

/*! \def O4A__DEVICE
 *  \brief OSGi4Ami 9 degrees of freedom inertial module  Device.
 */
#define O4A 9DOF DEVICE    0x12  //0x10 O4A_DEVICE_SENSOR               0x20
O4A_DEVICE_ACCELEROMETER    0x01     0x01
```

```
// CATEGORIES AND TYPES ------------------------------------------------------

#define O4A_DEVICE           0x01
#define O4A_DEVICE_SENSOR    0x10
      #define  O4A_DEVICE_ACCELEROMETER   0x20


#define O4A_DEVICE_SIMPLE_HMI_INPUT   0x30

#define O4A_DEVICE_RFID_READER        0x80 //**** undefined category

////////////////////  SET OF PARAMETER
////////////////////////////////////////////////////////////////////////

// CLUSTER: BASE -------------------------------------------------------------

/*! \def DEVICE_MODE
 *   \brief
 */
#define  DEV_MODE_ALL_OFF              0x00   //
#define  DEV_MODE_RFID_ONLY            0x01   //
#define  DEV MODE ACEL ONLY            0x02   //
#define  DEV_MODE_ALL_ON               0x03   //

/*! \def ERROR_CODE
 *  \brief
 */
#define  ERROR RESERVED                0x00   // Reserved
#define  ERROR UNKNOW                  0x01   // Undefined Error
#define  ERROR NOT SUPPORTED           0x02   // Command not supported
#define  ERROR NOT IMPLEMENTED         0x03   // Command not implemented
#define  ERROR_FRAME                   0x04   // Malformed message
#define  ERROR_DEVICE                  0x05   // Device Error
#define  ERROR BUFFER                  0x06   // Buffer Error
#define  ERROR CONFIGURATION           0x07   //
#define  ERROR TELEGESIS REGISTER      0x08   //
#define  ERROR_NO_DEBUG_AVAILABLED     0xF0   // Debug not allowed


/*! \def POWER_MODE
 *  \brief
 */
#define  POWER_MODE_ACTIVE             0x00   //
#define  POWER_MODE_IDLE               0x01   //
#define  POWER_MODE_SLEEP              0x02   //


// CLUSTER: SIMPLE HMI INPUT -------------------------------------------------

 /*! \def SHMI_INPUT_EVENT_TYPE
*   \brief Events generated by a Simple Input HMI
*/
#define SHMI IET NOTPRESSED            0x00  //Not pressed
#define SHMI_IET_PRESSED               0x01  //Pressed
#define SHMI_IET_SHORT_PRESSED         0x02  //Short pressed
#define SHMI_IET_LONG_PRESSED          0x03  //Long pressed



////////////////////  SET COMMANDS AND CLUSTER
////////////////////////////////////////////////////////////

#define DEVICE                                    0x00     // Set of operations
associated to the base of OSGi4AMI
      //Commands-----
      #define GET_TYPE                      0x00     // ************DONE***************
    //#define GET CLUSTERS                  0x01     //
      #define PING                          0x02     //
      #define RESET                         0x03     //
    //#define ECHO                          0x04     //
      #define GET_MODE                      0x05     // Endpoint:O4A_DEVICE /
Cluster:DEVICE Event:DEV_MODE / Data: [0] DEVICE_MODE ************DONE***************
      #define SET MODE                      0x06     // ************DONE***************
      #define GET ID                        0x07     //
      #define SET_ID                        0x08     //
```

```
      #define GET_DESCRIPTION                0x09     //
      #define SET_DESCRIPTION                0x0A     //
    //#define GET O4A END POINTS             0x0B     //
    //#define SET O4A END POINTS             0x0C     //
    //#define DELETE 04A END POINTS          0x0D     //
      #define GET_ANNOUNCE                   0x0E     //
      //Events-----
      #define DEV ACK                        0x00     // Endpoint:O4A_DEVICE /
Cluster:DEVICE Event:DEV ACK / Data: [0]CLUSTER_ID [1]COMMAND
************DONE***************
      #define NOT_SUPPORTED                  0x01     //
      #define DEV_ERROR                      0x02     // Endpoint:O4A_DEVICE /
Cluster:DEVICE Event:DEV ERROR / Data: [0]CLUSTER_ID [1]COMMAND [2] TYPE OF ERROR
************DONE***************
      #define DEV TYPE                       0x03     // Endpoint:O4A DEVICE /
Cluster:DEVICE Event:DEV_TYPE / Data: [0] ENDPOINT [1] CATEGORY [2] TYPE [3] IMPLEMENT
[4] VERSION
    //#define DEV CLUSTERS                   0x04     //
    //#define DEV ECHO                       0x05     //
      #define DEV ID                         0x06     //
      #define DEV DESCRIPTION                0x07     //
      #define DEV_MODE                       0x08     //
      #define DEV_O4A_END_POINTS             0x09     //
      #define DEV_ANNOUNCE                   0x0A     //


#define POWER                                0x02     // Set of operations related
to energy management
      //Commands-----
      #define GET POWER VALUE                0x00     //
      #define GET POWER REFRESH RATE         0x01     //
      #define SET POWER REFRESH RATE         0x02     //
      #define GET POWER MODE                 0x04     //
      #define SET_POWER_MODE                 0x05     //
      //Events-----
      #define POWER VALUE                    0x00     //
      #define POWER REFRESH RATE             0x01     //
      #define POWER LOW BATTERY              0x02     //
      #define POWER_MODE                     0x03     //


#define SENSOR                               0x10    // Set of operations
associated with the sensor
       //Commands-----
      #define GET_SENSOR_VALUE               0x00     // ************DONE***************
      #define GET_SENSOR_CONFIG              0x01     //
      #define SET SENSOR_CONFIG              0x02     //
      //Events-----
      #define SENSOR VALUE                   0x00     // ************DONE***************
      #define SENSOR_CONFIG                  0x01     //

#define SENSOR AUTO REFRESH                  0x11     // Joint operations with the
automatic refresh of the value measured by the sensor
      //Commands-----
      #define GET SENSOR REFRESH RATE        0x00     //
      #define SET_SENSOR_REFRESH_RATE        0x01     //
      //Events-----
      #define SENSOR_REFRESH_RATE            0x00     //


#define RFID READER                          0x20     // Set of operations
associated with the RFID Reader
      //Commands-----
      #define RFID SELECT TAG                0x00     // ************DONE***************
      #define RFID READ MEM                  0x01     //
      #define RFID READ SYS                  0x02     //
      #define RFID_READ_TAG                  0x03     //
      #define RFID_WRITE_MEM                 0x04     //
      #define RFID WRITE SYS                 0x05     //
      #define RFID WRITE_TAG                 0x06     //
      //Events-----
      #define RFID TAG DETECTED              0x00     // Endpoint:O4A RFID DEVICE /
Cluster:O4A_DEVICE_RFID_READER Event:RFID_TAG_DETECTED / Data: (TID 8 Bytes)
************DONE***************
      #define RFID READ MEM PASS             0x01     //
      #define RFID READ SYS PASS             0x02     //
      #define RFID_READ_TAG_PASS             0x03     //
```

```
      #define RFID_WRITE_MEM_PASS            0x04     //
      #define RFID WRITE SYS PASS            0x05     //
      #define RFID WRITE TAG PASS            0x06     //
      #define RFID ERROR                     0x00     // Endpoint:04A RFID DEVICE /
Cluster:04A DEVICE RFID READER Event:RFID ERROR / Data: [0]CLUSTER_ID [1]COMMAND [2]
TYPE OF ERROR ************DONE***************

#define SIMPLE HMI INPUT                           0x30     // Set of operations
associated with the input device HMI Simple
      //Commands-----
      #define GET_SIMPLE_HMI_INPUT_STATUS    0x00     //
      #define GET_SIMPLE_HMI_INPUT_CONFIG    0x01     //
      #define SET SIMPLE_HMI_INPUT_CONFIG    0x02     //
       //Events-----
      #define SIMPLE HMI INPUT STATUS UPDATED 0x00    //
      #define SIMPLE_HMI_INPUT_CONFIG        0x01     //

#define FIRMWARE CLUSTER                           0xF0     // Joint operations with the
firmware management and low level control
      //Commands-----
      #define FIRMWARE CHANGE NET            0x00     //
      #define FIRMWARE_RESTORE_DEFAULT_NET   0x01     //
      #define FIRMWARE_KAMSTRUP_BRIGE        0x02     //
      //Events-----
      #define FIRMWARE_KAMSTRUP              0x00     //


////////////////////////////////////////////////////////////////////////////////
// VARIABLES
//
////////////////////////////////////////////////////////////////////////////////


/*! \var struct ccp_buffer_in
 *  \brief structure for the storage of input data for the CCP protocol
*/
struct
{
   /*! Field that stores the end_point value */
   unsigned int8  end_point;
   /*! Field that stores the cluster_id value */
   unsigned int8  cluster_id;
   /*! Field that stores the length value */
   unsigned int8  length;
   /*! Field that stores the command value */
   unsigned int8  command;
   /*! Field that stores the data information */
   unsigned int8  data[DATA_MAX];
   /*! Field that stores the number of data bytes */
   unsigned int8  data_length;
   /*! Indicates whether there is information in the data field */
            int1  data_flag;
   /*! Field that stores active mode of operation */
   unsigned int8  active_mode;
}ccp_buffer_in;


/*! \var struct ccp_buffer_out
 *  \brief structure structure for the storage of input data for the CCP protocol
*/
struct
{
   /*! Field that stores the end_point value */
   unsigned int8  end_point;
   /*! Field that stores the cluster_id value */
   unsigned int8  cluster_id;
   /*! Field that stores the length value */
   unsigned int8  length;
   /*! Field that stores the event value */
   unsigned int8  event;
   /*! Field that stores the data information */
   unsigned int8  data[DATA_MAX];
   /*! Indicates whether there is information in the data field */
            int1  data_flag;
}ccp_buffer_out;
```

F.14 File: zmota_CcpFunctions.c

```
/*! \file zmota_CcpFunctions.c
* \brief Introduces functions for an OSGI4AmI Zigbee comunication and control of the Z
Mota Move peripherals
*
*
 *  \author Eduardo Molina Tomas \n
 *
 *   Copyright 2011,2012 HOWLab. Universidad de Zaragoza.\n
 *   This file is part Z-Mota Move 2.0. Interfaz basado en la monitorización del
movimiento e identificación por radiofrecuencia.\n
 *
 *   Z-Mota Move 2.0. Interfaz basado en la monitorización del movimiento e
identificación por radiofrecuencia is free software: you can redistribute it and/or
modify\n
 *   it under the terms of the GNU General Public License as published by\n
 *   the Free Software Foundation, either version 3 of the License, or\n
 *   (at your option) any later version.\n
 *
 *   AccelerationStreaming is distributed in the hope that it will be useful,\n
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of\n
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the\n
 *   GNU General Public License for more details.\n
 *\n
 *   You should have received a copy of the GNU General Public License\n
 *   along with AccelerationStreaming.  If not, see <http://www.gnu.org/licenses/>.\n *
*/

#include "zbAtLibrary30x.c"
#include "rfidSkyetekM1_library.c"
#include "adxl345lib.c"
#include "hmc5843lib.c"
#include "itg3200lib.c"

////////////////////////////////////////////////////////////////////////////////
// OPERATIONAL FUNCTIONS
//
////////////////////////////////////////////////////////////////////////////////

/*! \fn unsigned int8 rfid_clear_buffer ( unsigned int8 string1[], unsigned int8
string_length )
 *  \brief writes zero in all elements of the string
 *
 *  \param unsigned int8 string1[] string to clear
 *  \param unsigned int8 string_lenght number of elements in the string to clear
 *
*/
void clear_string ( unsigned int8 string1[], unsigned int8 string_length )
{

// Resets the Watch Dog timer
   restart_wdt();

   unsigned int8 count=0; // Counter

   for (count = 0; count < string_length; count++)
   {

      string1[count]=0;

   }

}

////////////////////////////////////////////////////////////////////////////////
// MIDDLEWARE FUNCTIONS
//
////////////////////////////////////////////////////////////////////////////////


/*! \fn void set_manual_mode ( )
 *  \brief
 *
 *  \param unsigned int8 mode - operation mode to change
 *
*/
```

```
void set_zmota_mode (unsigned int8 mode)
{
   restart_wdt(); // Resets the Watch Dog timer

// Sets manually the required fields
   ccp_buffer_in.end_point=O4A_BASE_DEVICE;
   ccp_buffer_in.cluster_id=DEVICE;
   ccp_buffer_in.command=SET_MODE;
   ccp_buffer_in.data[0]=mode;

   ccp_command_set_mode (); // Calls to function
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void ccp_command_get_type ( )
 *  \brief returns the information of the type of device that is located in the
corresponding endpoint
 *
 *  \param endpoint_table_row
 *
*/
void ccp_command_get_type ( unsigned int8 endpoint )
{
   unsigned int count = 0; // Array counter
   unsigned int endpoint_table_row=0; // Indicates the position of the device in the
table ccp_endpoints

         restart_wdt(); // Resets the Watch Dog timer

         while (endpoint!=ccp_endpoints[endpoint_table_row][0] && count<NUM_ENDPOINTS )
            endpoint_table_row++;

      // Return the main fields in the response
         ccp_buffer_out.end_point=ccp_endpoints [endpoint_table_row][0];
         ccp_buffer_out.cluster_id=DEVICE;
         ccp_buffer_out.event=DEV_TYPE;
         ccp_buffer_out.data_flag=1; // There are data in this response
      // Returns the active end points of the device in data
         for (count=0;count<5;count++)
            ccp_buffer_out.data[count]=ccp_endpoints [endpoint_table_row][count];

         ccp_buffer_out.length=6; // Updates the length field
      // Refresh flags
         flags.uart_outgoing_data = 1; // Sets the flag for sending data through the
UART
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void ccp_command_get_mode ( )
 *  \brief
 *
 *
*/
void ccp_command_get_mode ( )
{
         restart_wdt(); // Resets the Watch Dog timer

      // Return the main fields in the response
         ccp_buffer_out.end_point=O4A_BASE_DEVICE;
         ccp_buffer_out.cluster_id=DEVICE;
         ccp_buffer_out.event=DEV_TYPE;   // Response with DEV_TYPE 0x03
         ccp_buffer_out.data_flag=1; // There are data in this response
      // Return the active mode in the data field
         ccp_buffer_out.data[0]=ccp_buffer_in.active_mode;
         ccp_buffer_out.length=2;   // Updates the length field
       // Refresh flags
         flags.uart_outgoing_data = 1; // Sets the flag for sending data through the
UART
}

////////////////////////////////////////////////////////////////////////////////

/*! \fn void ccp_command_set_mode ( )
 *  \brief
 *
```

```
 *
*/
void ccp_command_set_mode ( )
{

// Return the main fields in the response
   ccp_buffer_out.end_point=O4A_BASE_DEVICE;
   ccp_buffer_out.cluster_id=DEVICE;


   restart_wdt(); // Resets the Watch Dog timer

   // MODE check
   switch (ccp_buffer_in.data[0])
   {
   // Mode of operation 0x00 DEV_MODE_ALL_OFF
      case DEV_MODE_ALL_OFF:
      // Stores the active mode
         ccp_buffer_in.active_mode=DEV_MODE_ALL_OFF;
      //Change the device operating mode
         peripheral_state.rfid=0;
         peripheral_state.acel=0;

         ccp_send_ack (DEV_ACK); // Acknowledge
      break;

   // Mode of operation 0x01 DEV_MODE_RFID_ONLY
      case DEV_MODE_RFID_ONLY:
       // Stores the active mode
          ccp_buffer_in.active_mode=DEV_MODE_RFID_ONLY;
       //Change the device operating mode
          peripheral_state.rfid=1;
          peripheral_state.acel=0;

          ccp_send_ack (DEV_ACK); // Acknowledge
      break;

    // Mode of operation 0x02 DEV_MODE_ACEL_ONLY
      case DEV_MODE_ACEL_ONLY:
       // Stores the active mode
           ccp_buffer_in.active_mode=DEV_MODE_ACEL_ONLY;
       //Change the device operating mode
           peripheral_state.rfid=0;
           peripheral_state.acel=1;

           ccp_send_ack (DEV_ACK); // Acknowledge

      break;
    // Mode of operation 0x03 DEV_MODE_ALL_ON
      case DEV_MODE_ALL_ON:
       // Stores the active mode
           ccp_buffer_in.active_mode=DEV_MODE_ALL_ON;
       //Change the device operating mode
            peripheral_state.rfid=1;
            peripheral_state.acel=1;

            ccp_send_ack (DEV_ACK); // Acknowledge
      break;

     // If not find mode of operation returns DEV_ERROR
      default:
            ccp_send_error(DEV_ERROR,ERROR_NOT_SUPPORTED); // ERROR

      break;
            }
}

////////////////////////////////////////////////////////////////////////////////
// PERIPHERALS FUNCTIONS
//
////////////////////////////////////////////////////////////////////////////////

// ZIGBEE

/*! \fn void initialize_Zigbee( )
 *  \brief executes all the routines necessary for the commissioning of the Zigbee
device
```

```
 *
*/
void initialize_Zigbee ( )
{

   restart_wdt(); // Resets the Watch Dog timer

   zbBootloader(); //Dissociate node, restore factory defaults and configure the Zigbee
device
   zbStartUpNet(); //Joins network
   strcopy (uart_buffer.data, "Communication ready!");
   zbSendDataToZigbeeSink(strlen (uart_buffer.data), uart_buffer.data);

}


////////////////////////////////////////////////////////////////////////////////


// 9DOF STICK

/*! \fn void initialize_inertial_sensors( )
 *  \brief executes all the routine necessary to set the inertial sensor block
 *
*/
void initialize_inertial_sensors ( )
{
   // Setup
   adxl_configure (); // configures registers of the ADXL345 for a desired mode of
operation
   delay_ms(10);
   hmc_configure (); // configures registers of the HMC5843 for a desired mode of
operation
   delay_ms(10);
   itg_configure (); // configures registers of the ITG3200 for a desired mode of
operation

   // Read identifiers
   adxl_read_register (ADXL_DEVID_ADDRESS , &adxl_id );  // *E5
   delay_ms(10);
   itg_read_register (WHO_AM_I_REG , &itg_id); // *69
   delay_ms(10);
   hmc_read_register(HMC_STATUS_REGISTER_ADDRESS, &hmc_id); // *07
}

/*! \fn void read_inertial_sensors ( )
 *  \brief
 *
*/
void read_inertial_sensors ( )
{
   unsigned int8 count=0; // counter
   unsigned int8 samples=1; //

   restart_wdt(); // Resets the Watch Dog timer

// Return the main fields in the response
   ccp_buffer_out.end_point=O4A_9DOF_DEVICE;
   ccp_buffer_out.cluster_id=SENSOR;
   ccp_buffer_out.event=SENSOR_VALUE; // Response with SENSOR_VALUE 0x00
   ccp_buffer_out.data_flag=1; // There are data in this response
// Read the sensors samples and stores in data
   for (count=0;count<(samples*18);count=count+18)
   {
         adxl_capture_axis_data (&ccp_buffer_out.data[count]);
         hmc_capture_channel_data (&ccp_buffer_out.data[count+6]);
         itg_capture_gyros_data (&ccp_buffer_out.data[count+12]);
   }
   ccp_buffer_out.length=(samples*18)+1; // Updates the length field

// Refresh flags
   flags.uart_outgoing_data = 1; // Sets the flag for sending data through the UART
}


////////////////////////////////////////////////////////////////////////////////
```

```
// RFID

/*! \fn void call_select_tag(  )
 *  \brief
 *
*/
void call_select_tag( )
{

   unsigned int8 count = 0;  // Counter

   restart_wdt(); // Resets the Watch Dog timer

// Clear RFID buffer strings
   clear_string ( m1_request_string, M1_MAX_DATA );
   clear_string ( m1_response_string, M1_MAX_DATA );

   setup_wdt(WDT_OFF); // Disables the Watch Dog
   m1_select_tag(); // Run the RFID command
   setup_wdt(WDT_ON); // Enables the Watch Dog

// If the request has been succesful
   if ( m1_buffer.response_code == M1_COMMAND_SELECT_TAG  )
   {
      // Return the main fields in the response
      ccp_buffer_out.end_point=O4A_RFID_DEVICE;
      ccp_buffer_out.cluster_id=RFID_READER;
      ccp_buffer_out.event=RFID_TAG_DETECTED; // Response with RFID_TAG_DETECTED 0x10
      ccp_buffer_out.data_flag=1; // There are data in this response

      // Stores the TID in the data field
      for (count = 0; count < 8; count++) // TID always have eight bytes
         ccp_buffer_out.data[count]=m1_buffer.tid[count];
      ccp_buffer_out.length=9;

   // Refresh flags
      flags.uart_outgoing_data = 1; // Sets the flag for sending data through the UART

   }

}
```

BIBLIOGRAFIA

- **[1] "HCI (human computer interaction):concepto y desarrollo" - Mari Carme Marcos - En: "El profesional de la información", 2001, junio, v. 10, n. 6, pp. 4-16.**
- **[2] "Evolución y tendencias en la interacción persona–ordenador " - Mireia Ribera Turró - En: "El profesional de la información", 2005, noviembre–diciembre, v. 15, n. 6, pp. 414-422.**
- **[3] (http://www.saturn.es/tp/article/,925507.html)**
- **[4] (http://kotaku.com/5748382/sega-tried-kinect-once-and-it-didnt-go-so-well)**
- **[5] "A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi" - Jin-Shyan Lee, Yu-Wei Su, and Chung-Chou Shen**
- **[6](http://galia.fc.uaslp.mx/~cantocar/microprocesadores/EL_Z80_PDF_S/13_TIPOS_DE_MEMORIA.PDF)**
- **[7] (http://batteryuniversity.com/learn/article/whats_the_best_battery)**
- **[8] "THE I 2C-BUS SPECIFICATION" - PHILIPS**
- **[9] "SkyeTek_ProtocolV2_071102" - SKYETEK**
- **[10] "TG-ETRXn-R305-Commands" - TELEGESIS**

Información complementaria no citada de forma explicita:

- **"RFiD Handbook - Fundamentals and Applications" - KLAUS FINKENZELLER - Editorial WILEY - ISBN 0-470-84402-7**
- **"Informe Tecnico: Protocolo Zigbee (IEEE 802.15.4)" - Javier Martín Moreno, Daniel Ruiz Férnandez (http://rua.ua.es/dspace/bitstream/10045/1109/7/Informe_ZigBee.pdf)**