Universidad de Montevideo
Facultad de Ingeniería

# Proyecto UMix

## Búsqueda de supervivientes

Borrador del Informe del
Proyecto Final de Carrera

Versión 1.2

Juan Brenes
Sofía Silva

Montevideo
17 de Diciembre de 2009

# Abstract del documento

En este documento se presenta un borrador del Informe Final del Proyecto de Fin de Carrera. Se incluye contenido teórico y práctico sobre los módulos del proyecto que se han completado hasta el momento y sobre la metodología de trabajo empleada.

# Historial de cambios

| Versión | Fecha | Descripción | Autor |
|---|---|---|---|
| 1.0 | 7 de noviembre de 2009 | Formato del documento | Sofía Silva |
| 1.1 | 5 de diciembre de 2009 | Versión inicial del documento | Sofía Silva |
| 1.2 | 17 de diciembre de 2009 | Se completaron los capítulos Marco Teórico y Desarrollo | Juan Brenes<br>Sofía Silva |
| | | | |

# Avance del Resumen Ejecutivo

## *Introducción*

Hoy en día, en el mundo suceden catástrofes muy frecuentemente. Esto genera enormes cantidades de heridos cuyo rescate suele ser una tarea ardua y compleja. La búsqueda de supervivientes implica tener suficiente personal a disposición para recorrer las áreas de catástrofe. Por otro lado, el hecho de que sean personas las que realizan la búsqueda implica un riesgo para éstas por el hecho de estar recorriendo zonas peligrosas.

En este contexto, este proyecto tiene como principal objetivo contribuir a simplificar el rescate de personas luego de catástrofes, desarrollando un sistema automatizado que realice las tareas de búsqueda de supervivientes.

## *Antecedentes*

Debido a que Uruguay no es un país en donde ocurran terremotos ni otros fenómenos naturales que provoquen importantes catástrofes, hasta el momento no se han llevado a cabo proyectos similares al proyecto UMix. Sin embargo, sí existen a nivel internacional proyectos en esta área, aunque estos son propietarios, mucho más complejos y de alto costo. Desde un principio, la idea del proyecto UMix fue mantener bajos costos y modularidad para que pueda ser mejorado por otros equipos en algún futuro.

## *Objetivos*

El objetivo general de este proyecto es contribuir al desarrollo de tecnologías que faciliten la tarea de búsqueda de supervivientes en áreas de catástrofes.

Los objetivos específicos de este proyecto son los siguientes:

- Implementar funcionalidades básicas de búsqueda de personas.

- Desarrollar un módulo que permita la comunicación inalámbrica con una computadora personal utilizando protocolos de red estandarizados. Esto permitirá la fácil administración y manipulación del móvil.

- Incluir un módulo de posicionamiento que haga uso de un dispositivo GPS estándar para determinar la ubicación del móvil en todo momento y en particular cuando se detecta un superviviente.

- Diseñar el sistema de movimiento de forma tal que la plataforma física de movimiento sea fácilmente intercambiable.

- Simular el módulo de detección de vida con un sensor de sonido.

## *Metodología*

Durante el semestre en que se cursó la Asignatura Proyecto de Telemática también se cursó Gestión de Proyectos, cuyo objetivo era poder gestionar el proyecto. Para cumplir con este objetivo se realizaron diversas tareas, algunas de las cuales concluyeron con un entregable. Estas tareas incluyeron la planificación del desarrollo del proyecto, creando un cronograma con desglose de tareas y asignación de los integrantes a las mismas; la redacción de un documento de alcance y la redacción del Plan de Proyecto.

Para la asignación de tareas se tuvo en cuenta la modularidad del proyecto, lo cual permitió dividir las tareas de forma de poder avanzar en paralelo con los distintos módulos. Es decir, que el avance de uno de los integrantes del equipo condicionaba lo mínimo posible al avance del otro integrante.

Por otro lado, el proyecto requiere de la realización de tareas técnicas para poder implementar los distintos módulos que comprende el sistema. Para llevar a cabo esta implementación se planificó la dedicación de cierto tiempo a la investigación teórica del módulo, para luego pasar a la implementación y, por último, el testeo.

Para los módulos asociados al PIC se dividió la implementación y el testeo en dos etapas: primero se implementó y testeó en una simulación (utilizando el programa Proteus) y luego se armaron los circuitos físicos y se testearon. Esta última etapa, a su vez se dividió en dos: primero se armaron los circuitos sobre protoboards y luego de verificado su correcto funcionamiento, se soldaron sobre placas pertinax.

## *Resultados*

Como resultado de esta primera etapa del proyecto, se logró un prototipo que cumple con las funcionalidades que se especificaron al comienzo del proyecto:

- Compilación de Linux para la placa BeagleBoard (BB). Se compiló el kernel de Ubuntu para el procesador utilizado por la placa BB (OMAP3530 que se basa en el procesador ARM Cortex-A8)
- Comunicación inalámbrica entre la placa BB y un PC. Conexión de la placa BB a internet a través de un modem inalámbrico, la cual permite que la misma se comunique con un PC utilizando un servidor DDNS (Dynamic DNS).

- Comunicación entre la placa BB y el PIC. Envío de directivas de movimiento de la placa BB al PIC y notificación de detección de vida y de obstáculos del PIC a la placa BB.
- Instalación de un servidor Apache con módulo para PHP en la placa BB.
- Control y monitoreo del sistema. Interfaz web que permite enviar directivas y recibir notificaciones.
- Ejecución de un programa escrito en el lenguaje C con rutinas de acceso a una base de datos SQL.
- Módulo de movimiento: Control del motor de continua y del motor paso a paso desde un PC.
- Simulación de sensor de vida: Simulación del sensor de vida con un sensor infrarrojo que detecta un tono de 1 kHz de frecuencia y notificación a la placa BB.
- Detección de obstáculos: Detección de obstáculos al hacer contacto con ellos y notificación de esta detección a la placa BB.

## *Conclusiones*

Hasta el momento se han cumplido con los objetivos fijados para la entrega del primer prototipo. Sin embargo, algunas tareas importantes para el proyecto fueron pospuestas para poder dedicar más tiempo a la documentación y a los testeos necesarios para esta entrega. El equipo asume la responsabilidad de dicha postergación y prevé que será necesario reajustar los tiempos de la planificación.

Los atrasos mencionados anteriormente se debieron principalmente a falta de desarrollo en el kernel de Linux utilizado, sobre todo en los temas relacionados a la comunicación inalámbrica y en la invocación de código SQL desde una aplicación escrita en lenguaje C.

Además, durante las tareas de testeo de circuitos se experimentaron dificultades debido a que en la práctica los circuitos no se comportan exactamente igual a como lo hacen en la simulación.

Es importante tener en cuenta que al momento de la planificación del proyecto no se tuvo en cuenta la gran demanda de tiempo requerida para la elaboración de documentos. Este fue otro motivo por el cual fue necesario postergar tareas.

Por último, se experimentaron dificultades al intentar comunicar la placa BB con el PIC debido a que los mismos manejan lógicas con distintos niveles de voltaje y fue imposible encontrar en el mercado local los circuitos integrados necesarios para solucionar este problema.

# Índice

# Índice de Figuras

# Índice de Tablas

# 1 Introducción del Proyecto Final de Carrera

## 1.1 Proyecto UMix

El presente proyecto, “Proyecto UMix – Búsqueda de supervivientes”, consiste en la investigación necesaria para desarrollar un sistema automatizado de búsqueda de supervivientes de catástrofes y en el desarrollo de un prototipo de este sistema.

Para el desarrollo de este proyecto se utilizaron diversos dispositivos de ubicación, monitoreo y comunicación ya estandarizados. Se manejan microswitches como sensores que permiten detectar obstáculos; se simulan los sensores de vida con un sensor infrarrojo que detecta un tono con una frecuencia determinada y se utilizan circuitos integrados que implementan drivers para controlar los motores del vehículo. Además, se maneja un sistema de comunicación inalámbrica estándar que permite notificar a un dispositivo cliente.

Todo el sistema fue implementado en base a software, interfaces y plataformas libres, lo cual permite la fácil adaptación de este proyecto a otros sistemas.

## 1.2 Objetivos

### 1.2.1 Objetivo general del proyecto

Contribuir al desarrollo de tecnologías que faciliten la tarea de búsqueda de supervivientes en áreas de catástrofes. Para ello se utilizarán componentes estandarizados, de fácil adquisición y se recurrirá a código open source, de forma de permitir futuros desarrollos sobre la base de este proyecto.

## 1.2.2 Objetivos específicos del proyecto

Implementar funcionalidades básicas de búsqueda de personas, es decir realizar un móvil que se desplace de forma autónoma en un área determinada e informe sobre los lugares específicos donde se encontraron posibles supervivientes. Para esto es necesario:

- Cargar un sistema operativo Linux sobre una placa de desarrollo BeagleBoard (BB). Se compiló el kernel Ubuntu adaptándolo a la placa BB.
- Desarrollar la lógica para el control de los motores y para la interacción con los sensores en un microcontrolador PIC 16F877A.
- Utilizar circuitos integrados para controlar los sensores y los motores.
- Implementar sensores de obstáculos y sensor infrarrojo.
- Utilizar un motor de continua y un motor paso a paso para la tracción y la dirección del móvil respectivamente.
- Desarrollar un módulo en la placa BB que permita la comunicación inalámbrica con una computadora personal utilizando protocolos de red estandarizados. Esto permitirá la fácil administración y manipulación del móvil. Para esto se utilizó un modem inalámbrico Huawei E160. Para conectar este dispositivo a la placa fue necesario agregar un hub USB con alimentación independiente y puerto maestro.
- Incluir un módulo de posicionamiento que haga uso de un dispositivo GPS estándar para determinar la ubicación del móvil en todo momento y en particular cuando se detecta un superviviente.
- Diseñar el sistema de movimiento de forma tal que la plataforma física de movimiento sea fácilmente intercambiable.
- Simular el módulo de detección de vida con un sensor infrarrojo.

En la Figura 1.1 - Esquema modular se presenta un esquema del proyecto, en el cual se muestran los módulos del mismo, en donde residen y cómo interactúan entre sí.

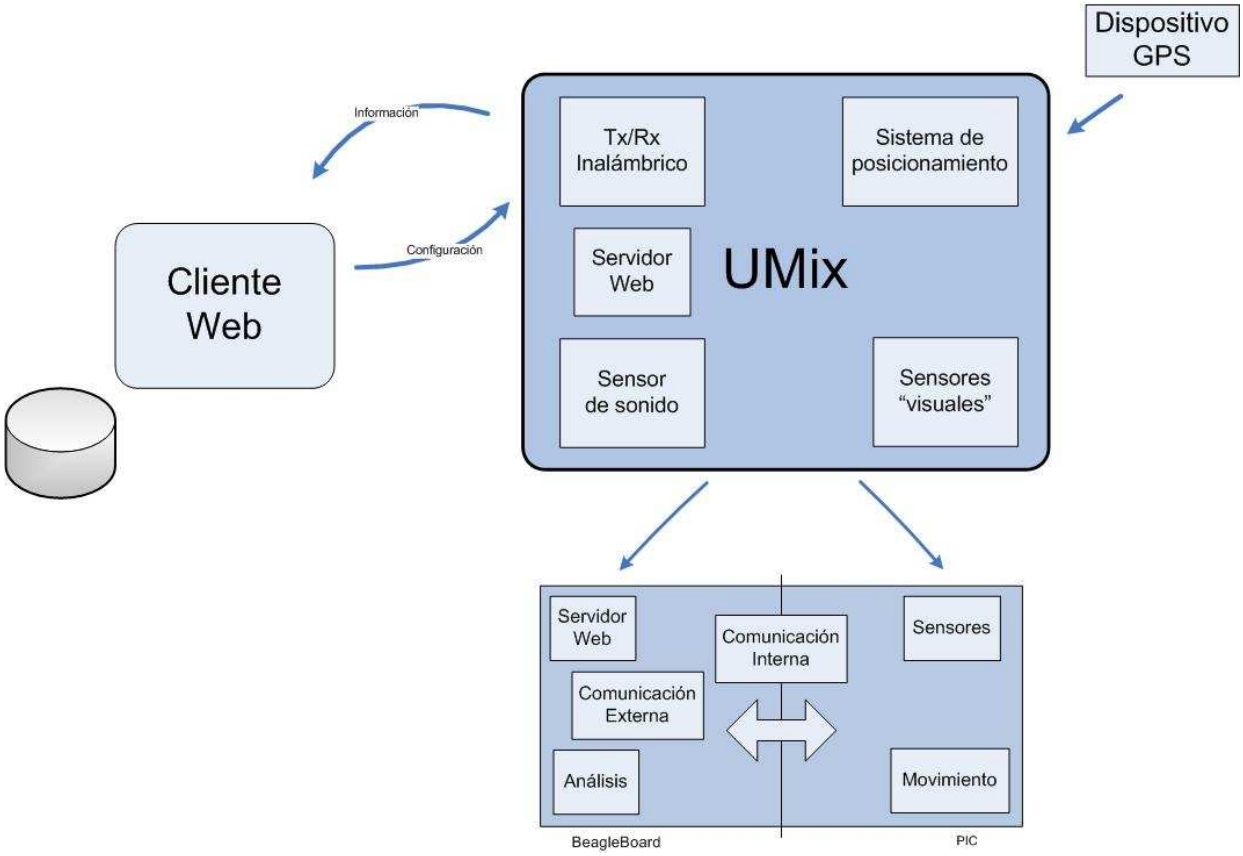


**Figura 1.1 - Esquema modular**

## 1.3 Antecedentes

Existen muchos sistemas embebidos que utilizan el kernel de Linux como base para realizar sus funciones. En particular, existen muchos proyectos de robótica implementados con dicho sistema operativo utilizando específicamente la placa BB.

El puerto de expansión de la placa BB ha sido frecuentemente utilizado como una interfaz de comunicación con un PIC.

Por otro lado, también existen proyectos que implementan interfaces de control de motores desde un PIC y manejo de sensores.

Sin embargo, no se ha encontrado ningún proyecto que integre todos estos elementos y que les dé una aplicación similar a la del proyecto UMix.

## 1.4 Justificación de impacto

UMix es un proyecto que no pretende competir con sistemas automatizados de búsqueda que utilicen alta tecnología, si no que su objetivo es brindar una solución económica, sencilla y modular que pueda adaptarse a distintas realidades. Además pretende ser la base para futuros proyectos de investigación y desarrollo que haya en la rama.

Si bien muchos de los componentes que se utilizarán en la implementación de UMix ya existen, no existe ningún proyecto hasta el momento que las vincule y las integre en una solución de estas características.

Se espera que el valor agregado por la integración de todas las tecnologías y las herramientas desarrolladas permitan que el proyecto UMix tenga un gran impacto y aceptación en el mercado.

## 1.5 Grupos de interés

- Entes Gubernamentales que participen en la búsqueda de supervivientes
- ONGs u otras organizaciones que participen en la búsqueda de supervivientes
- Fuerzas armadas, ejércitos
- Grupos sociales que habiten en zonas de accidentes o catástrofes naturales frecuentes
- Empresas dedicadas a la venta de este tipo de tecnología
- Investigadores particulares de robótica y desarrolladores de sistemas automatizados
- Desarrolladores del kernel de linux

# 2 Marco Teórico del Proyecto Final de Carrera

En este capítulo se establece un marco teórico como referencia para comprender las tareas desarrolladas dentro del proyecto y se detallan los conceptos claves utilizados.

## *2.1 Sistemas embebidos*

Un sistema embebido es una computadora de poca capacidad, destinada a realizar una tarea específica en tiempo real. Estos sistemas deben su nombre a que, por lo general, forman la parte integral del hardware de un sistema más grande.

Su diseño es, en general, muy flexible de forma que se adapta a la gran mayoría de los requerimientos de los potenciales usuarios.

Estos sistemas son controlados por un procesador central que típicamente es implementado en base a microcontroladores o procesadores digitales de señal.

La limitada función que realizan estos sistemas hace que se pueda reducir enormemente el consumo de energía y espacio físico que ocupan.

En el caso del proyecto UMix, se eligió utilizar un sistema de este tipo de forma de facilitar la lógica relacionada con el manejo de protocolos de red, manejo de dispositivos GPS y el algoritmo de movimiento del robot.

## *2.2 La placa BeagleBoard*

beagleboard.org (2009)

Para el desarrollo del proyecto UMix se decidió utilizar la placa BeagleBoard (BB), en primer lugar debido a la gran potencialidad que brindan sus componentes y debido a la posibilidad que tenía el equipo de proyecto de adquirir la misma.

Todas las características aquí detalladas fueron obtenidas del manual de referencia de la placa BB, por lo que se deberá recurrir a dicho documento en caso que se desee obtener más información. Cabe aclarar además, que muchas de las características de la placa no fueron utilizadas en el proyecto y por eso no se mencionan en este documento.

### 2.2.1 Reseña

La placa BB es una plataforma basada en el procesador OMAP3530 (que a su vez se basa en el procesador ARM Cortex-A8) y diseñada específicamente para cumplir las especificaciones relacionadas con la comunidad “Open Source”.

Su implementación se realizó teniendo como objetivo disponer de una plataforma con una combinación mínima de herramientas que permitan al usuario experimentar el poder del OMAP3530, sin pretender en ningún momento obtener una plataforma de desarrollo completa. Muchas de las capacidades del OMAP3530 no son accesibles desde la placa BB.

De todas formas, el diseño de la placa incluye diversas interfaces de comunicación que hacen que dicha plataforma sea ampliamente extensible, permitiendo así el desarrollo de diversas herramientas y aplicaciones.

## 2.2.2 Características generales de la placa

La Tabla 2-1 muestra las características de la placa:

| | Feature | |
|---|---|---|
| **Processor** | OMAP3530 ES3.0 | |
| **POP Memory** | Micron | |
| | 2Gb NAND (256MB) | 2Gb MDDR SDRAM (256MB) |
| **PMIC TPS65950** | Power Regulators | |
| | Audio CODEC | |
| | Reset | |
| | USB OTG PHY | |
| **Debug Support** | 14-pin JTAG | GPIO Pins |
| | UART | LEDs |
| **PCB** | 3.1" x 3.0" (78.74 x 76.2mm) | 6 layers |
| **Indicators** | Power | 2-User |
| | PMU | |
| **HS USB 2.0 OTG Port** | Mini AB USB connector | |
| | TPS65950 I/F | |
| | MiniAB | |
| **HS USB Host Port (Rev C2/3 Only)** | Single USB HS Port | Up to 500ma Power |
| **Audio Connectors** | 3.5mm | 3.5mm |
| | L+R out | L+R Stereo In |
| **SD/MMC Connector** | 6 in 1 SD/MMC/SDIO | 4/8 bit support, Dual voltage |
| **User Interface** | 1-User defined button | Reset Button |
| **Video** | DVI-D | S-Video |
| **Power Connector** | USB Power | DC Power |
| **Expansion Connector (Not Populated)** | Power (5V & 1.8V) | UART |
| | McBSP | McSPI |
| | I2C | GPIO |
| | MMC | PWM |
| **2 LCD Connectors (Rev C2/3 Only)** | Access to all of the LCD control signals plus I2C | 3.3V, 5V, 1.8V |

**Tabla 2-1 - Características placa BB**

## 2.2.3 Procesador

Texas Instruments (2009)

La placa BB incluye un procesador OMAP3530 versión ES3.0, empaquetado en un POP (Package On Package) de .4mm. Esta técnica de empaquetamiento consiste en situar las memorias (NAND y SDRAM) de la placa sobre el procesador, haciendo que el mismo no sea visible para el usuario.

Para conocer más detalles técnicos relacionados con este procesador se puede consultar la hoja de datos OMAP3530, referenciada en la Bibliografía de este informe.

## 2.2.4 Memoria

Como se mencionó anteriormente, la placa BB posee únicamente dos tipos de memorias:

- 2Gb NAND x 16 (256 MB)

Las memorias NAND son un tipo de memoria Flash de bajo costo y gran capacidad. Las memorias Flash por su parte se caracterizan por mantener los datos aún sin disponer de una fuente de energía, y en particular, las memorias NAND se diferencian de las otras memorias Flash por su implementación en base a la compuerta lógica a la que debe su nombre.

- 2Gb MDDR SDRAM x32 (256 MB @166 MHz)

La Mobile Double Data Rate (MDDR) es un tipo de doble transferencia de datos SDRAM, pensado para dispositivos móviles. El objetivo de esta tecnología es ahorrar espacio físico dentro de la placa y mejorar el consumo total de energía en comparación con la versión normal de este tipo de memoria.

Al ser una memoria RAM, esta memoria depende de una fuente de energía para mantener la información que guarda.

Como se mencionó anteriormente, los dos tipos de memoria citados son los únicos que vienen nativamente con la placa BB, sin embargo, es posible ampliar la memoria NAND de la placa mediante la utilización de una tarjeta SD ó mediante algún disco USB conectado mediante un HUB-USB (con alimentación independiente) al puerto USB-OTG de la placa BB. Cabe destacar que la viabilidad de esta última opción depende del sistema operativo que se desee utilizar posteriormente.

## 2.2.5 Conector SD/MMC 6 en 1

Una de las interfaces de expansión que dispone la BB es el conector SD/MMC 6 en 1, que soporta los siguientes dispositivos:

- Tarjetas WiFi

- Cámaras
- Tarjetas Bluetooth
- Módulos GPS
- Memorias SD
- Memorias MMC
- Tarjetas SDIO
- Tarjetas MMCMobile
- Tarjetas RS-MMC
- Tarjetas MiniSD

Esta interface soporta el estándar MMC4.0 (MMC+) y puede ser utilizada para bootear la placa BB desde tarjetas MMC ó SD. Todas las tarjetas de 4 ó 8 bits funcionan para esta interface, sin embargo, únicamente las de 4 bits se pueden utilizar para bootear.

La Figura 2.1 describe el conexionado de la interface.

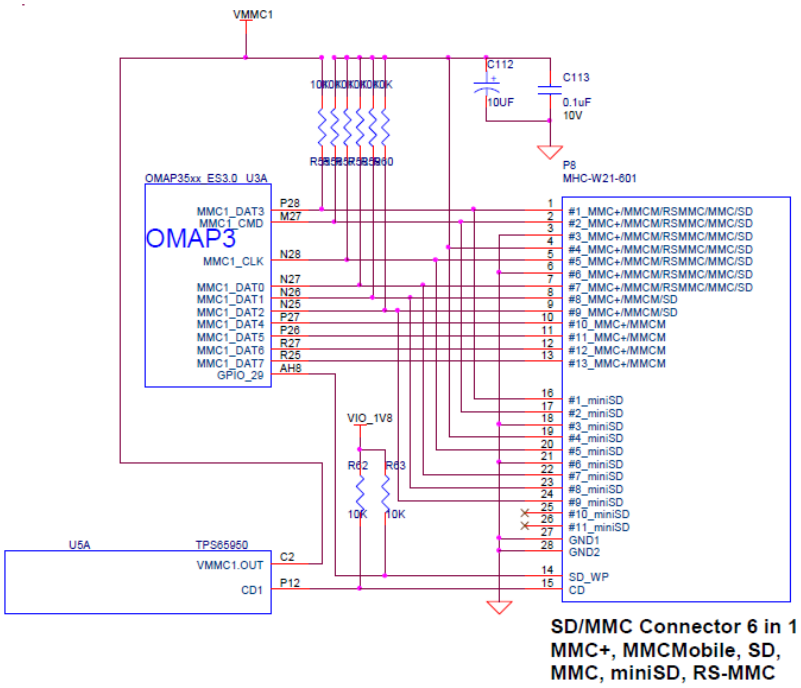


**Figura 2.1 - Conexionado conector SD/MMC 6 en 1**

### 2.2.6 Puerto USB-OTG

Hewlett-Packard Company et al. (2008)

Este puerto es la fuente primaria de alimentación y transferencia de datos prevista para la placa BB, derivando la energía de la PC o equipo donde se conecte.

El Universal Serial Bus (bus universal en serie) o Conductor Universal en Serie (CUS), abreviado comúnmente USB, es un puerto que sirve para conectar periféricos a una computadora.

El diseño del USB tenía en mente eliminar la necesidad de adquirir tarjetas separadas para poner en los puertos bus ISA o PCI, y mejorar las capacidades plug-and-play, permitiendo a esos dispositivos ser conectados o desconectados al sistema sin necesidad de reiniciar. Sin embargo, en aplicaciones donde se necesita ancho de banda para grandes transferencias de datos, o donde se necesita una latencia baja, los buses PCI o PCIe son la opción más adecuada.

La extensión USB-On-The-Go (USB-OTG) añadida al estándar USB, permite que un equipo pueda alternativamente actuar como servidor o como dispositivo. Esta nueva funcionalidad fue específicamente diseñada para equipos como la BB, cuyo rol en la comunicación USB varía continuamente dependiendo de la aplicación.

Las especificaciones técnicas de este protocolo de comunicación se encuentran detalladas en el documento **USB 3.0** referenciado en la bibliografía de este documento. Por más información se puede consultar dicho documento.

La Figura 2.2 muestra el conexionado del puerto USB-OTG.

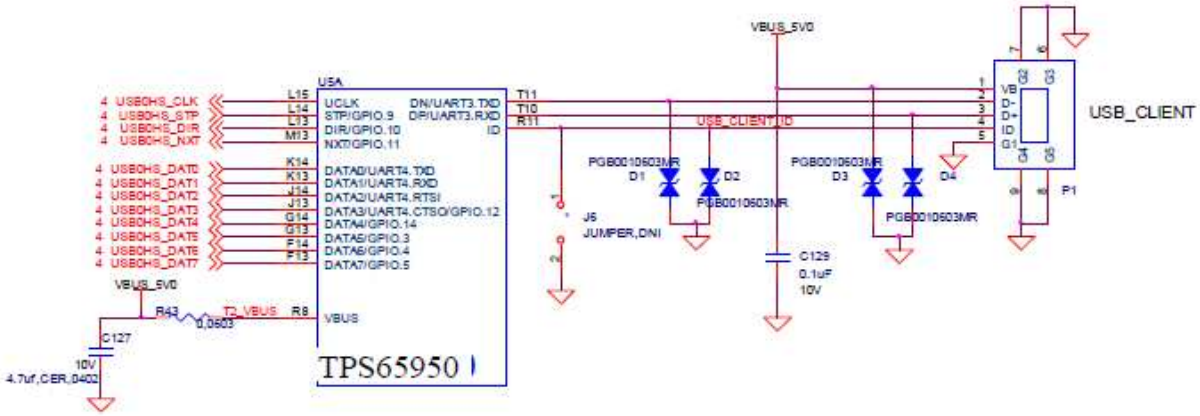


**Figura 2.2 - Conexionado puerto USB-OTG**

## 2.2.7 Puerto de Expansión

La placa BB posee un puerto de expansión diseñado para hacer aún más adaptable la aplicación de la placa. Dicho puerto está conectado con diversos pines de entrada del procesador pudiendo multiplexar varias señales. La Figura 2.3 muestra el conexionado de este puerto.

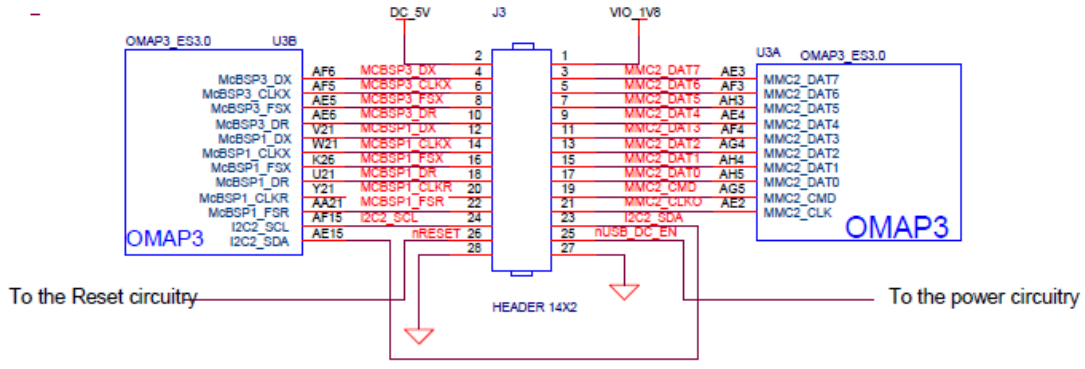


**Figura 2.3 - Conexionado puerto de expansión**

Existen varios protocolos soportados nativamente en dicho puerto, entre ellos los siguientes:

- General Port Input Output (GPIO)
- I2C
- SPI

Luego de investigar sobre estos protocolos, por la simpleza de su uso y la gran documentación existente, se decidió utilizar el protocolo GPIO.

#### 2.2.7.1 Protocolo GPIO

en.wikipedia.org (2009)

General Purpose Input/Output es un protocolo que permite establecer una interfaz con periféricos y dispositivos externos. Esta interfaz puede actuar como una entrada (para leer señales digitales de otras partes del circuito), o como una salida, para controlar otros dispositivos.

Las interfaces GPIO suelen estar organizadas en grupos, típicamente de 8 pines (un puerto GPIO), que usualmente se pueden configurar como entradas o como salidas de forma independiente.

En algunos casos, las interfaces GPIO se pueden configurar para producir interrupciones a la CPU y para ser capaces de utilizar el acceso directo a memoria para mover grandes cantidades de datos de forma eficiente hacia el dispositivo o desde el dispositivo.

### 2.2.8 Puerto RS232

En la implementación de la BB se previó la incorporación de un puerto serial que permitiera la ejecución de comandos mediante una consola. Dicho puerto utiliza el estándar TIA/EIA 232 y está conectado directamente a la UART3 del OMAP3530 de la forma en que muestra la Figura 2.4.

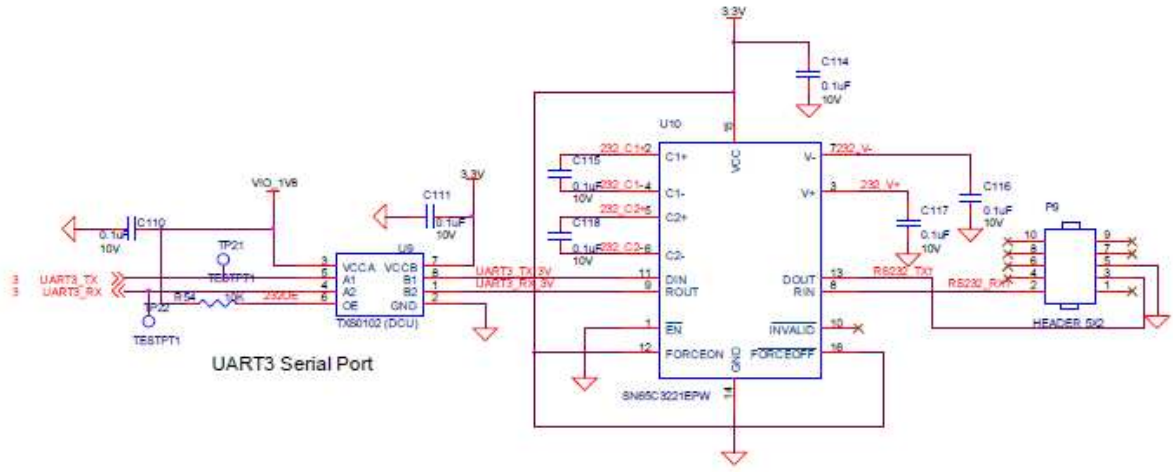


**Figura 2.4 - Puerto Serial RS232**

Este puerto se utiliza para ejecutar comandos que permitan realizar la configuración inicial del procesador para que arranque de la SD automáticamente, y para instalar luego un servidor SSH en el sistema operativo de forma de poder utilizar directamente un intérprete de comandos a través de una red TCP/IP.

## 2.2.9 Alimentación

La BB posee dos fuentes de alimentación disponibles. La primera es la correspondiente al puerto USB-OTG, en donde la placa toma como alimentación los 5 V que provee el PC ó el HUB-USB en donde se haya conectado. La segunda fuente posible es conectando directamente la BB a una fuente de alimentación independiente, como puede ser un transformador a la pared, y obtener de ahí los 5 V necesarios.

En algunas aplicaciones que necesitan un gran consumo de corriente para su correcto funcionamiento, es posible utilizar un conector en forma de Y en el puerto USB-OTG para obtener más potencia ó utilizar un pin del puerto de expansión que permite ingresar otros 5 V adicionales de energía.

La Figura 2.5 muestra el conexionado de la alimentación de la BB.

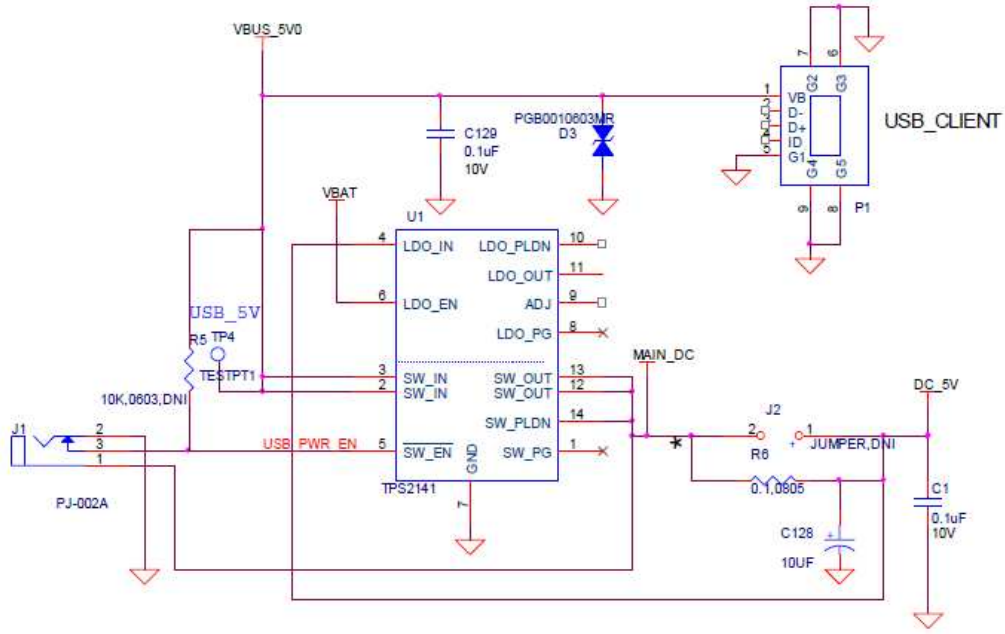


**Figura 2.5 - Conexionado alimentación BB**

## *2.3 Sistema operativo*

Una de las grandes decisiones que hay que tomar al momento de realizar proyectos de estas características es qué sistema operativo utilizar.

En principio es posible realizar el desarrollo de todo el código del sistema operativo y compilarlo, sin embargo, esta opción demanda mucho tiempo y recursos, teniendo en cuenta las funcionalidades que se le pretende dar a esta aplicación en particular. Principalmente el desarrollo de todos los protocolos de red e implementación de servidores requeriría de un tiempo que, en este caso, es inadmisible.

La segunda opción es utilizar alguno de los sistemas operativos que se encuentran en el medio y adaptarlo para que funcione en la plataforma de hardware establecida, como por ejemplo Windows ó algunas de las distribuciones de Linux.

La amplia documentación, soporte y escalabilidad del kernel de Linux, hacen que cualquier distribución del mismo tenga grandes ventajas con respecto a cualquier otro tipo de sistema operativo, cuando se trata de sistemas embebidos.

Es por estas razones que se optó por utilizar el kernel de Linux para implementar UMix y, en particular, se eligió la distribución Ubuntu por la experiencia que tiene el equipo de proyecto trabajando con dicho sistema operativo. Además, existe documentación específica de cómo adaptar Ubuntu a la BB, lo cual facilita enormemente el proceso de instalación.

### 2.3.1 Linux

A pesar de que el término “Linux” refiere únicamente al kernel implementado por Linus Torvalds, por lo general se utiliza este término para referirse a todos los sistemas operativos que hacen uso de dicho kernel.

Originalmente diseñado como proyecto para la Universidad de Helsinki y basado en el kernel Minix, propuesto por Andrew Tannenbaum, Linux ganó gran popularidad tras haber sido adoptado por el proyecto GNU como reemplazo de Hurd.

En cuanto a su arquitectura, Linux es un kernel de tipo monolítico. Esto significa que todo el sistema operativo se ejecuta dentro del espacio de memoria asignado al kernel y en modalidad de supervisor. A diferencia de otros tipos de kernel, los núcleos monolíticos definen una interfaz de alto nivel sobre el hardware del sistema, que permite la manipulación del mismo a través de llamadas al sistema.

La Figura 2.6 ilustra las diferencias entre los tipos más comunes de núcleos.

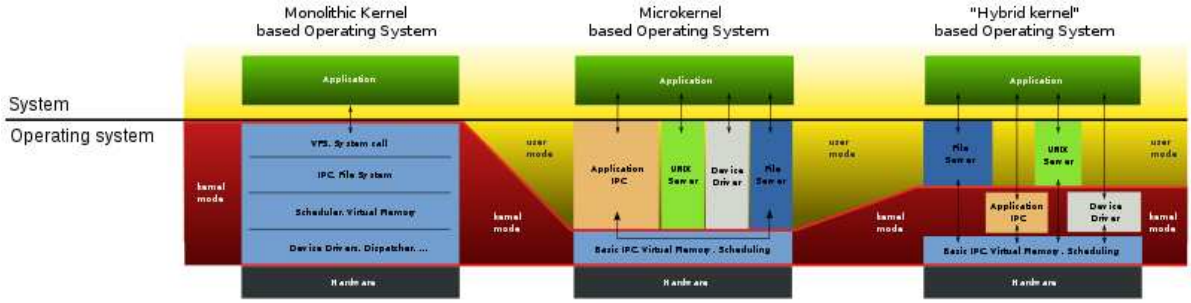


**Figura 2.6 - Diferencias entre núcleos**

La Figura 2.7 ilustra el mapa de memoria de Linux.

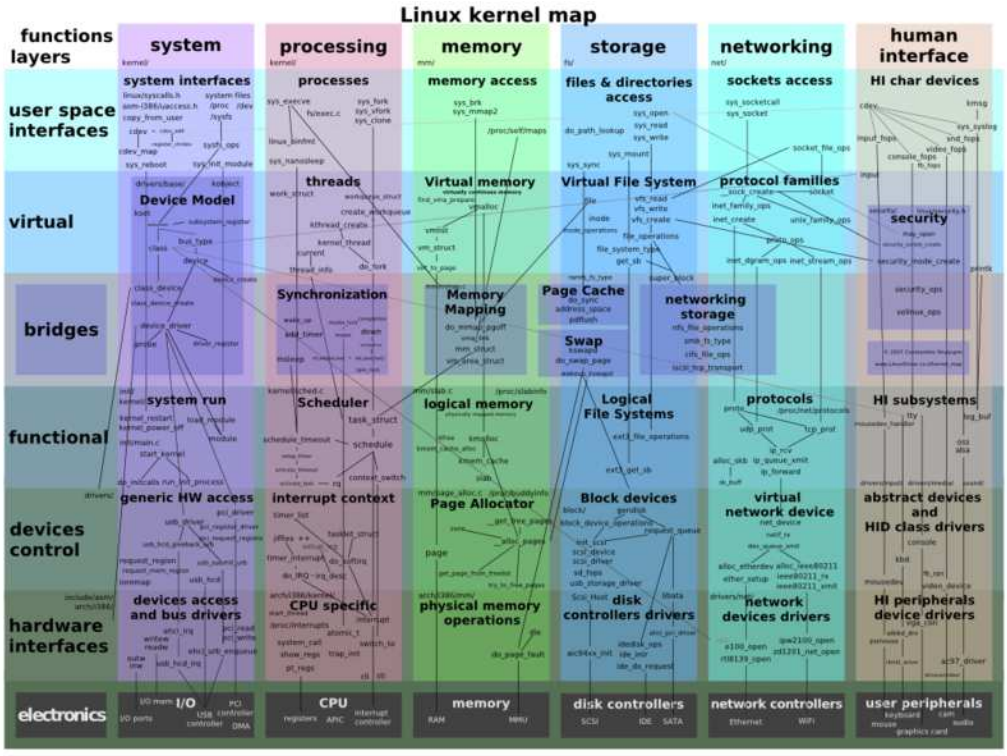


**Figura 2.7 - Mapa de memoria de Linux**

Una de las grandes ventajas de Linux con respecto a otros núcleos, es que su implementación se hizo en C, el cual es un lenguaje de programación de alto nivel. Esto hace, en primer lugar, que el código sea fácilmente interpretable por cualquier ser humano y, en segundo lugar, que sea transportable hacia casi cualquier sistema sin mayores dificultades.

### 2.3.2 Kernel de Ubuntu

Si bien el kernel de Linux está estandarizado y mantenido al día de hoy por Linus Torvalds, muchas comunidades independientes decidieron basarse en dicho kernel e incorporarle algunos cambios y aplicaciones, de forma de obtener sistemas operativos completos y funcionales para una gran variedad de sistemas.

Cada sistema operativo liberado por dichas comunidades se conoce como “distribución de linux”, como son por ejemplo: Ubuntu, Suse, Debian, RedHat, Mandriva, Slackware, etc. Debido a las libertades que ofrece la licencia GPL, bajo la cual esta licenciado Linux, cualquier persona puede registrar su propia distribución de Linux, sin necesidad de realizar ningún desembolso económico.

En particular una distribución que se ha hecho muy popular en los últimos tiempos es Ubuntu. Esta distribución tiene la ventaja de basarse en otra distribución denominada Debian, reconocida por su estabilidad, por tener una gran comunidad de soporte y desarrollo y por lanzar actualizaciones rápidamente.

En el proyecto UMix se decidió utilizar este kernel por varios motivos. En primer lugar, el kernel de Ubuntu tiene la certificación ARM Compliant, lo cual significa que el código de una de sus versiones está optimizado y testeado para los procesadores de la gama ARM, que son los que se utilizaran en el proyecto. En segundo lugar, existe mucha documentación relacionada con Ubuntu y en especial, existen manuales completos de cómo adaptarlo a la placa BB. Y en último, está la experiencia con dicho sistema operativo por parte de los desarrolladores del proyecto.

## *2.4 Comunicación*

El objetivo del proyecto UMix requiere que el móvil pueda comunicarse fácilmente con diversos dispositivos que se encuentran a gran distancia. Es por este motivo, que a la hora de su diseño se optó por incorporar al sistema la posibilidad de manejar conexiones de red estandarizadas, que permitan el pasaje de paquetes TCP/IP.

En concreto, se puso énfasis en desarrollar compatibilidad con los módems que ofrecen conexiones inalámbricas 3G, ya que esta tecnología está ampliamente difundida y es altamente probable que exista cobertura mediante dicha conexión en el área de la catástrofe.

### 2.4.1 Enlace 3G

El nombre 3G ó IMT-2000 se refiere a una familia de estándares para comunicaciones móviles definidos por la ITU (International Telecommunication Unit). Los servicios provistos por estos estándares incluyen telefonía, video-llamadas y transferencia de datos dentro de entornos móviles.

Las grandes ventajas que ofrece esta tecnología con respecto a su predecesora son las siguientes: mayor velocidad de transferencia de datos, mejor aprovechamiento espectral y nuevas funcionalidades orientadas a la confidencialidad y seguridad de las comunicaciones.

Dentro del proyecto se incluyó un módem 3G conectado a un HUB-USB que, a su vez, se conectó al puerto USB-OTG de la BB mediante su puerto maestro. El modelo de dicho módem se escogió específicamente para que fuera compatible con los drivers incorporados en el kernel de Ubuntu utilizado. De esta forma, y luego de configurar adecuadamente el software de conexión, se dispuso de una conexión inalámbrica mediante la cual acceder a las distintas interfaces de usuario y al servidor SSH que provee UMix.

## 2.4.2 Enlace Ethernet

Ethernet ó IEEE 802.3 es un estándar para redes de área local que utiliza el protocolo de acceso al medio CSMA/CD. Dentro de dicho estándar se definen los métodos de cableado, la señalización eléctrica y la forma de empaquetar la información.

Este medio de comunicación fue incorporado con el objetivo de facilitar la transferencia de información entre la BB y un PC durante el transcurso de la implementación del sistema. De cualquier forma, esta interface se dejará habilitada de modo que se pueda acceder al móvil prácticamente desde cualquier PC.

## *2.5 Microcontrolador PIC*

Microchip Technology Inc. (1997)

Para el desarrollo de la lógica de control de motores y sensores se utilizó un microcontrolador PIC 16F877A. Este microcontrolador es un dispositivo de alta performance con CPU RISC (Reduced Instruction Set Computer), la cual consta de 35 instrucciones de una palabra que se ejecutan en un único ciclo de instrucción.

El PIC 16F877A cuenta con varios periféricos. Los periféricos son lo que diferencia un microcontrolador de un microprocesador. Facilitan la interfaz con el exterior del PIC y tareas internas tales como mantener distintas bases de tiempo.

Algunos de los periféricos incluidos en este PIC son: timers, módulos de captura, comparación y modulación por ancho de pulso (PWM), puerto serial síncrono (SSP), receptor transmisor síncrono asíncrono universal (USART), puerto paralelo esclavo (PSP), conversor analógico/digital (A/D), entre otros. De todos los periféricos disponibles, en este proyecto solo se hizo uso del Timer2 y de los puertos de entrada/salida. Por este motivo, en la sección *Periféricos* de este informe solo se explicarán en detalle estos módulos.

En la Figura 2.8 se muestra el diagrama de bloques del PIC utilizado.

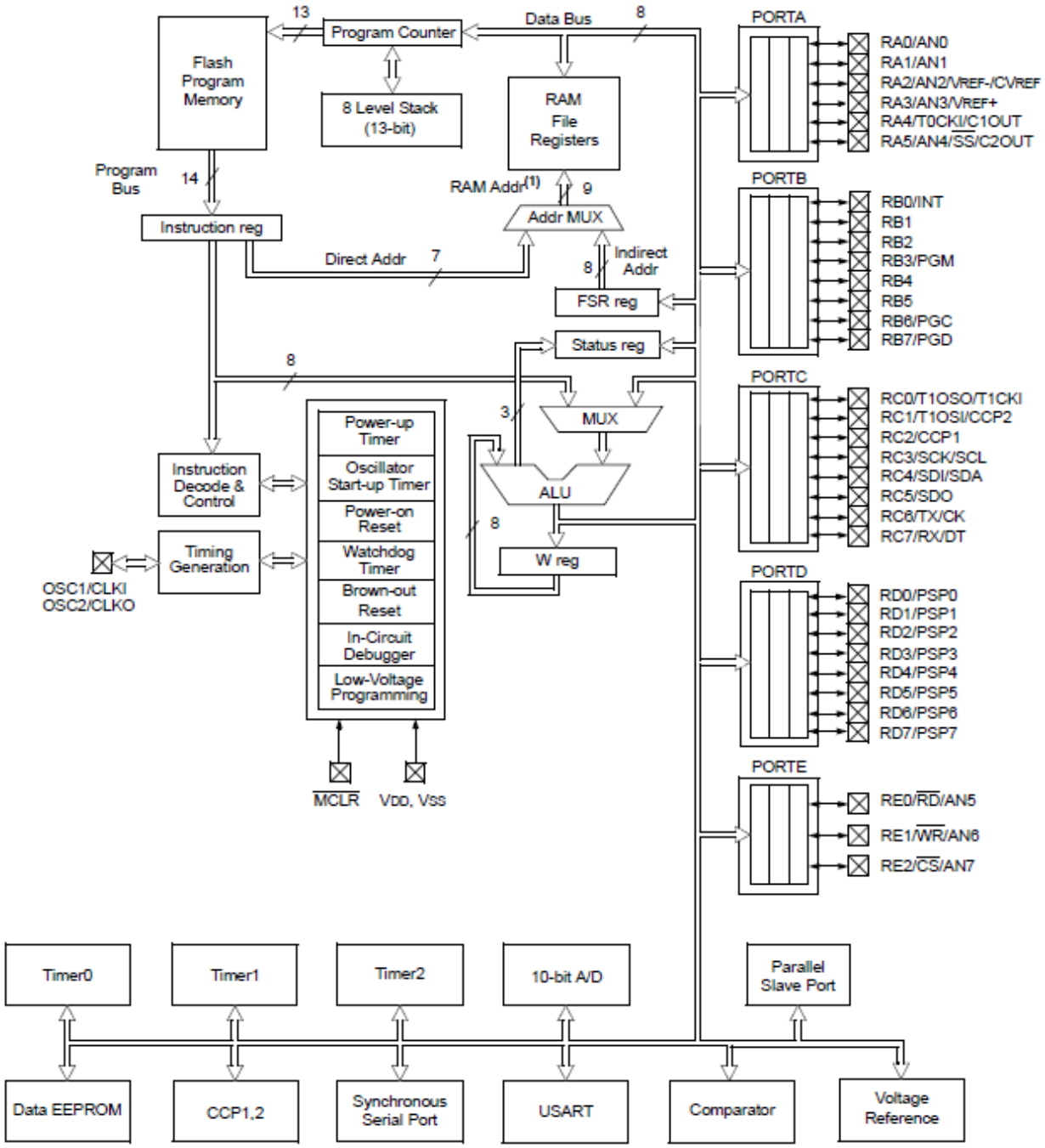


**Figura 2.8 - Diagrama de bloques PIC 16F877A**

## 2.5.1 Arquitectura

El PIC 16F877A tiene una arquitectura Harvard, por lo que tiene la memoria de programa separada de la memoria de datos y éstas son accedidas desde buses separados. Esto mejora el ancho de banda y permite que se ejecute una instrucción mientras se carga la siguiente (pipeline de instrucciones).

Otras características de la arquitectura de este PIC son las siguientes:

- Dispone de instrucciones de palabra larga e instrucciones de palabra sencilla
- Ejecuta las instrucciones en un solo ciclo de máquina
- Tiene un set de instrucciones reducido lo cual permite que sea aprendido fácilmente
- La memoria de registro de archivos o datos puede ser accedida directa o indirectamente

Su programación es sencilla pero eficiente, ya que las instrucciones son simétricas, es decir que es posible realizar cualquier operación sobre cualquier registro usando cualquier modo de direccionamiento.

Solo se utilizan dos instrucciones no orientadas a registros que son SLEEP (coloca al dispositivo en el modo de menor consumo) y CLRWDT (verifica que el chip esté operando correctamente evitando que el Watchdog Timer (WDT) se desborde y resetee el dispositivo).

## 2.5.2 CPU y ALU

La Unidad Central de Procesamiento (CPU) es responsable de usar la información en la memoria de programa (instrucciones) para controlar la operación del dispositivo. Muchas de estas instrucciones operan sobre la memoria de datos. Para operar sobre la memoria de datos es necesaria la Unidad Lógica Aritmética (ALU).

Además de realizar operaciones aritméticas y lógicas, la ALU controla los bits de estado (que se encuentran en el registro STATUS). El resultado de algunas instrucciones fuerzan los bits de estado a un valor que depende del estado del resultado.

### 2.5.3 Oscilador

El circuito interno del oscilador se usa para generar el reloj del dispositivo. El reloj del dispositivo es necesario para que el dispositivo ejecute las instrucciones y para que funcionen los periféricos. Cuatro períodos del reloj del dispositivo general un ciclo del reloj interno de instrucciones ($T_{CY}$).

El oscilador puede trabajar en uno de ocho modos distintos: Cristal de baja frecuencia (LP), Cristal/Resonador (XT), Cristal/Resonador de alta velocidad (HS), Resistencia/Capacitancia externa (RC) con o sin salida de reloj y Resistencia/Capacitancia interna de 4 MHz con o sin salida de reloj. Para seleccionar el modo del oscilador se utilizan los bits de configuración del dispositivo FOSC2, FOSC1 y FOSC0. Los diferentes modos permitirán ahorrar costos del sistema o ahorrar energía.

En este proyecto se utiliza el oscilador en modo HS para lo cual se conecta un cristal de 20 MHz en los pines OSC1 y OSC2. Se tomo la decisión de utilizar este modo debido a que se requería alta velocidad en la lógica de control de motores y sensores.

### 2.5.4 Memoria

El PIC 16F877A tiene tres bloques de memoria: memoria de programa, memoria de datos y memoria de datos EEPROM.

#### 2.5.4.1 Memoria de programa

La memoria de programa es una memoria Flash que puede ser borrada eléctricamente, es decir que puede ser borrada y reprogramada sin ser removida del circuito.

Para especificar la dirección de la instrucción a cargar para ser ejecutada se utiliza un contador de programa (PC) de 13 bits, es decir que es capaz de direccionar un espacio de memoria de programa de una palabra de 8K por 14 bits. El ancho del bus de la memoria de programa es 14 bits. Debido a que todas las instrucciones son de una palabra, un dispositivo con una memoria de programa de 8K por 14 bits tiene espacio para 8K de instrucciones.

En la posición 0000h se encuentra el vector de Reset y en la posición 0004h se encuentra el vector de interrupción. Es decir que al resetearse el PIC el PC apuntará a la dirección 0000h y al ocurrir una interrupción el PC apuntará a la dirección 0004h.

Se dispone de un stack de 8 niveles de profundidad por 13 bits de ancho, el cual permite que ocurra una combinación de hasta 8 llamadas de programa e interrupciones. El stack contiene las direcciones de retorno.

En la Figura 2.9 se muestra el mapa de la memoria de programa y el stack.

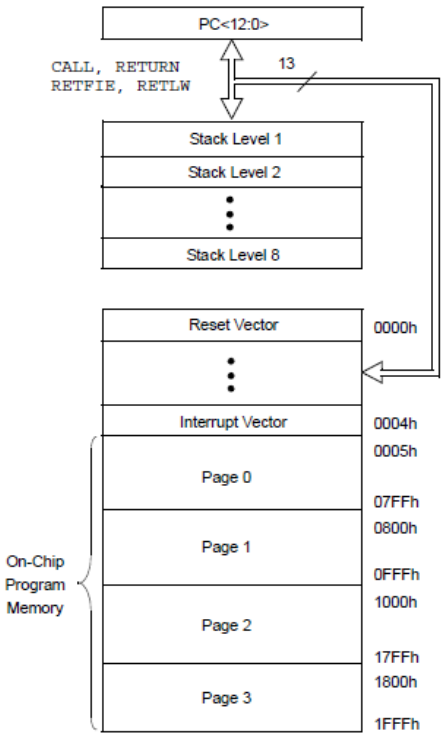


**Figura 2.9 - Mapa de la memoria de programa y stack**

### 2.5.4.2 Memoria de datos

La memoria de datos está particionada en cuatro bancos los cuales contienen los registros de propósito general (GPRs) y los registros de función especial (SFRs). Los registros de función especial son registros utilizados por la CPU y por los módulos periféricos para controlar la operación deseada del dispositivo, mientras que los registros de propósito general son el área general para almacenamiento de datos.

Toda la memoria de datos puede ser accedida tanto directa como indirectamente. El direccionamiento directo requiere el uso de los bits RP1 (bit 6 del registro STATUS) y RP0 (bit 5 del registro STATUS). El direccionamiento indirecto requiere el uso del registro de selección de archivo (FSR). El direccionamiento indirecto utiliza el bit puntero de registro indirecto (IRP) del registro STATUS para los accesos a las áreas Banco0/Banco1 o Banco2/Banco3 de la memoria de datos.

Cada banco tiene 128 bytes. Las ubicaciones más bajas de cada banco están reservadas para los registros de función especial. Luego están los registros de propósito general, implementados como RAM estática.

En la Figura 2.10 se muestra el mapa de los bancos.

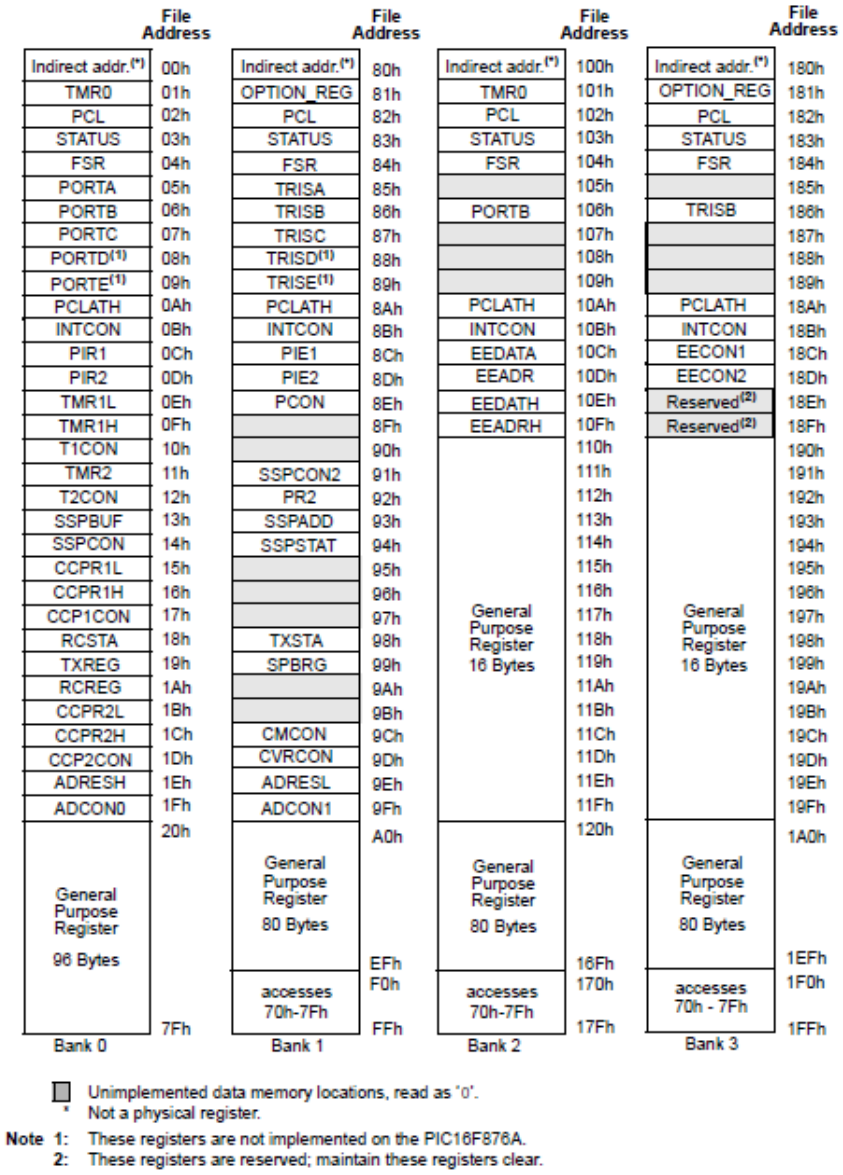


**Figura 2.10 - Mapa de bancos de memoria**

### 2.5.4.2.1 Registro STATUS

Este registro contiene el estado aritmético de la ALU, el estado de RESET y los bits de selección de banco para la memoria de datos. Debido a que la selección del banco de la memoria de datos es controlada por este registro, es necesario que esté presente en cada banco.

#### 2.5.4.2.2 Registro OPTION_REG

El registro OPTION_REG es un registro que se puede leer y escribir y que contiene varios bits de control para configurar el prescaler del Timer0/WDT, la interrupción externa INT, el Timer0 y las pull-ups del puerto B.

#### 2.5.4.2.3 Registro PCON

El registro Control de Poder (PCON) contiene flags, los cuales, junto con los bits TO y PD, permiten que el usuario pueda diferenciar entre los resets del dispositivo.

### 2.5.4.3 Memoria de datos EEPROM

La memoria de datos EEPROM se puede leer y escribir durante operación normal. Esta memoria no se mapea directamente en el espacio del archivo de registro. En cambio, es direccionada indirectamente a través de los SFRs.

Se utilizan cuatro SFRs para leer y escribir esta memoria: **EECON1** (contiene los bits de control), **EECON2** (es el registro utilizado para inicial la lectura/escritura), **EEDATA** (mantiene el dato de 8 bits a ser leído/escrito) y **EEADR** (mantiene la dirección de la ubicación de la EEPROM que está siendo accedida).

## 2.5.5 Modo SLEEP

El modo SLEEP es un modo en el que el dispositivo es puesto en su estado de menor consumo de corriente. Se apaga el oscilador del dispositivo, por lo que no avanza el reloj del sistema. Se ingresa al modo SLEEP ejecutando la instrucción **sleep**.

Si está habilitado, el WDT es puesto en “0” pero sigue corriendo, el bit PD del registro **STATUS** también es puesto en “0”, el bit TO es seteado y el driver del oscilador es

apagado. Los puertos de entrada/salida mantienen el estado que tenían antes de que se ejecutara la instrucción **sleep**.

Para menor consumo de corriente en este estado, todos los pines de entrada/salida deberían estar a $V_{DD}$ o a $V_{SS}$, sin ninguna circuitería externa drenando corriente del pin de entrada/salida y los módulos que se especifica que tienen una corriente diferencial en modo sleep deberían ser deshabilitados.

El dispositivo se puede despertar del modo SLEEP a través de uno de los siguientes eventos:

- Cualquier reset del dispositivo
- WDT (si está habilitado)
- Cualquier módulo periférico que pueda setear su flag de interrupción mientras el dispositivo se encuentra en modo SLEEP, tales como:
    - Pin de interrupción externa (RB0)
    - Pines de cambio en el puerto
    - Comparadores
    - A/D
    - Timer1
    - LCD
    - SSP
    - Captura

El primer evento va a resetear el dispositivo al despertarse. Sin embargo, los otros dos van a despertar al dispositivo y luego continuar con la ejecución del programa. Se pueden usar los bits TO y PD del registro **STATUS** para determinar la causa del reset del dispositivo. El bit PD, el cual es seteado al encenderse el dispositivo, se pone en “0” cuando se invoca la instrucción SLEEP. El bit TO se pone en “0” si el dispositivo se despertó por el WDT.

Cuando se está ejecutando la instrucción SLEEP, la siguiente instrucción es pre-cargada. Para que el dispositivo se despierte a través de un evento de interrupción, el correspondiente bit de habilitación de interrupción debe ser seteado. El dispositivo se va a despertar sin importar el estado del bit GIE. Si el bit GIE está en “0”, el dispositivo continuará la ejecución en la instrucción siguiente a la instrucción SLEEP. Si el bit GIE está en “1”, el dispositivo ejecuta la instrucción siguiente a la instrucción SLEEP y luego va al vector de interrupción (0004h). En casos en los que la ejecución de la instrucción siguiente a la instrucción SLEEP no es deseada, el usuario debería colocar un NOP luego de la instrucción SLEEP.

### 2.5.6 Interrupciones

Pueden existir distintas fuentes de interrupción:

- Interrupción del Pin INT (Interrupción externa)
- Interrupción por desborde del Timer0
- Interrupción por cambio en el Puerto B (pins RB7:RB4)
- Interrupción por cambio en el comparador
- Interrupción del puerto paralelo esclavo
- Interrupciones de la USART
- Interrupción de recepción
- Interrupción de transmisión
- Interrupción por conversión A/D completa
- Interrupción LCD
- Interrupción por escritura en la EEPROM de datos completa
- Interrupción por desborde del Timer1
- Interrupción por desborde del Timer2
- Interrupción CCP
- Interrupción SSP

Como mínimo se utiliza un registro para el control y el estado de las interrupciones. Éste es el registro **INTCON**, el cual contiene flags de interrupciones, bits para habilitar interrupciones y el bit para habilitar la interrupción global (GIE). Este último bit, si está seteado, habilita todas las interrupciones no enmascaradas o, si está en cero, deshabilita todas las interrupciones.

Además, existen registros para habilitar las interrupciones de los periféricos y registros para mantener los flags de interrupción. Estos registros son: **PIE1**, **PIR1**, **PIE2** y **PIR2**.

La instrucción “regresar de la interrupción”, RETFIE, hace que la ejecución salga de la rutina de interrupción y setea el bit GIE, lo cual permite que se ejecute cualquier interrupción pendiente.

Cuando ocurre una interrupción, se pone en 0 el bit GIE para deshabilitar cualquier otra interrupción, la dirección de retorno se inserta en el stack y se carga el PC con 0004h. Una vez en la rutina de atención de interrupción, se puede determinar la fuente de la interrupción testeando los flags de interrupción. Por lo general, estos flags deben ser puestos en 0 por software antes de volver a habilitar la interrupción global para evitar interrupciones recursivas.

En este proyecto se utilizan las fuentes de interrupción: Interrupción del Pin INT (Interrupción externa) e Interrupción por cambio en el Puerto B (pins RB7:RB4) por lo que solo estas fuentes de interrupción serán descritas con mayor detalle.

### 2.5.6.1 Interrupción externa

La interrupción externa en el pin INT es disparada por un flanco, ya sea el flanco ascendente si está seteado el bit INTEDG (bit 6 del registro OPTION_REG), o el flanco descendente si el bit INTEDG está en 0.

Cuando aparece un flanco válido en el pin INT, se setea el flag INTF (bit 1 del registro INTCON). Esta interrupción se puede habilitar/deshabilitar poniendo en 1 o en 0 el bit

INTE (bit 4 del registro INTCON). El bit INTF debe ponerse en 0 por software en la rutina de atención de interrupción antes de volver a habilitar esta interrupción.

La interrupción INT puede despertar al procesador del modo SLEEP, si el bit INTE fue seteado antes de entrar en modo SLEEP. El estado del bit GIE decide si el procesador va o no al vector de interrupción al despertarse.

### 2.5.6.2 Interrupción por cambio en el Puerto B

Cuatro pines del Puerto B, los pines RB7:RB4, tienen una función de interrupción por cambio. Solo los pines configurados como entradas pueden provocar que ocurra esta interrupción.

Los pines de entrada de RB7:RB4 son comparados con el valor que se almacenó la última vez que se leyó el Puerto B. Si alguno de estos pines no coincide con su valor anterior, se pondrá en “1” el flag RBIF (bit 0 del registro INTCON) lo cual generará una interrupción por cambio del Puerto B.

Esta interrupción puede despertar al dispositivo del modo SLEEP. Para limpiar la interrupción, el usuario puede leer o escribir en el Puerto B o bien poner en “0” el flag RBIF.

En la Figura 2.11 se muestra un diagrama de bloques de los pines RB7:RB4.

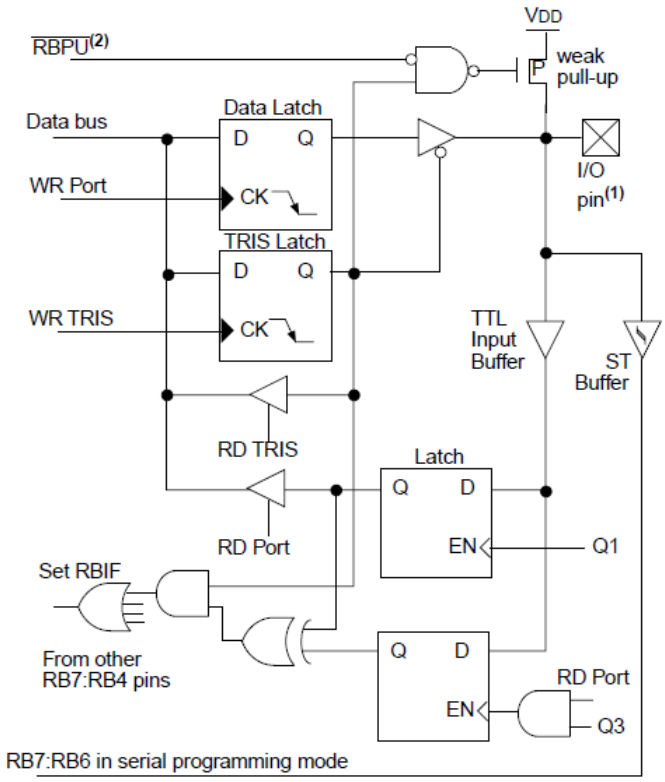


**Figura 2.11 - Diagrama de bloques pines RB7:RB4**

## 2.5.7 Periféricos

### 2.5.7.1 Puertos de entrada/salida

Los pines de entrada/salida de propósito general pueden considerarse los periféricos más simples. Permiten al PIC monitorear y controlar otros dispositivos. Para agregar flexibilidad y funcionalidad al PIC, algunos pines están multiplexados con una o varias funciones alternativas.

Para la mayoría de los puertos, la dirección de los pines de entrada/salida es controlada por el registro de dirección de datos, llamado registro TRIS. El bit x del registro TRIS controla la dirección del bit x del puerto correspondiente. Un “1” en el bit TRIS hace que el

bit correspondiente sea una entrada, mientras que un “0” hace que el bit correspondiente sea una salida.

La Figura 2.12 muestra un esquema de un típico puerto de entrada/salida.

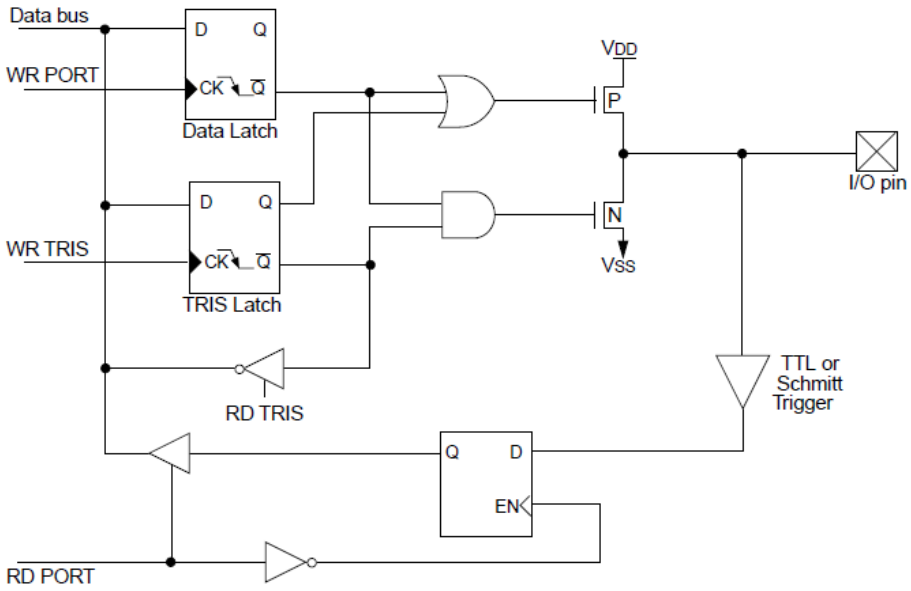


**Figura 2.12 - Típico puerto de entrada/salida**

### 2.5.7.2 Timer2

El PIC 16F877A dispone de tres timers: el Timer0, el Timer1 y el Timer2. Los tres timers tienen distintas características. En este proyecto se decidió trabajar con el Timer2 debido a que es el único de los tres que tiene pre-scaler y post-scaler. Los otros timers solo tienen pre-scaler.

El Timer2 es un timer de 8 bits. Sin embargo, si se utilizan el pre-scaler y el post-scaler en su máxima configuración, el tiempo de desborde es el mismo que el de un timer de 16 bits.

En la Figura 2.13 se muestra un diagrama de bloques del Timer2.

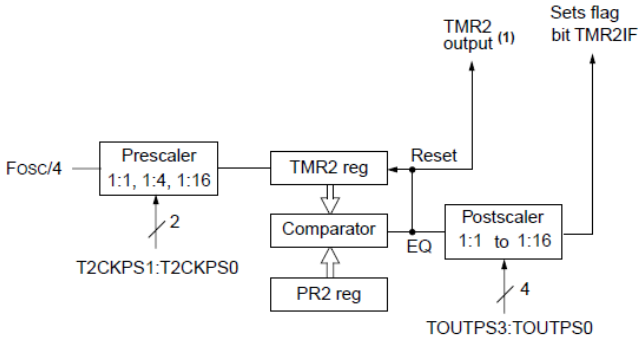


**Figura 2.13 - Diagrama de bloques Timer2**

El post-scaler cuenta cierta cantidad de veces que el registro **TMR2** haya coincidido con el registro **PR2**.

El registro de control del Timer2 es el registro T2CON. En la Figura 2.14 se muestra la estructura de este registro y los distintos valores que deben tomar sus bits para configurar el post-scaler, para configurar el pre-scaler y para encender el timer.

| U-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 | R/W-0 |
|---|---|---|---|---|---|---|---|
| — | TOUTPS3 | TOUTPS2 | TOUTPS1 | TOUTPS0 | TMR2ON | T2CKPS1 | T2CKPS0 |
| bit 7 | | | | | | | bit 0 |

bit 7 **Unimplemented:** Read as '0'

bit 6:3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits

0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON**: Timer2 On bit

1 = Timer2 is on
0 = Timer2 is off

bit 1:0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

| Legend | | |
|---|---|---|
| R = Readable bit | W = Writable bit | |
| U = Unimplemented bit, read as '0' | | - n = Value at POR reset |

**Figura 2.14 - Registro T2CON**

El módulo Timer2 utiliza como reloj de entrada el reloj del dispositivo ($F_{OSC}/4$). Con los bits T2CKPS1:T2CKPS0 se puede seleccionar la pre-escala que se le aplicará a este reloj.

El registro **TMR2** se puede leer y escribir, y se pone en "0" en todos los resets del dispositivo. El Timer2 se incrementa desde 00h hasta que coincide con **PR2** y luego se resetea a 00h en el siguiente ciclo de incremento. **PR2** es un registro que también se puede leer y escribir.

Hay cuatro bits que seleccionan el valor del post-scaler entre 1:1 y 1:16. Después de que el post-scaler se desborda, el flag de interrupción TMR2 (TMR2IF) se setea para indicar un desborde del Timer2.

## *2.6 Herramientas de desarrollo*

Para el desarrollo de la aplicación a nivel del PIC se utilizaron un conjunto de herramientas, tales como el entorno de desarrollo MPLAB, el paquete MPASM, el simulador Proteus y el programador PICSTART Plus. Existen diversas herramientas que permiten simplificar la tarea de desarrollo. En esta sección solo se describirán las herramientas que fueron utilizadas en el proyecto.

### 2.6.1 Entorno de desarrollo integrado MPLAB

Microchip Technology Inc. (2006)

Es el entorno en el cual opera todo el conjunto de herramientas de desarrollo. Esto permite que este conjunto de herramientas sea consistente, por lo que solo es necesario un mínimo conocimiento de la nueva herramienta.

Este entorno de desarrollo integra los siguientes aspectos del desarrollo:

- Edición del código fuente
- Gestión del proyecto
- Generación de código de máquina (a partir de Assembler o “C”)
- Simulación y emulación del dispositivo
- Programación del dispositivo

Este conjunto de herramientas permite el desarrollo completo de un proyecto sin salir del entorno MPLAB.

El software MPLAB, incluido en el entorno de desarrollo, permite editar archivos fuente, ya sea en lenguaje assembler o en “C”; permite compilar de forma sencilla; permite depurar la aplicación; permite ejecutar emuladores, entre otras funcionalidades.

## 2.6.2 Paquete MPASM

Microchip Technology Inc. (2005)

Para que el dispositivo opere como se desea en la aplicación específica, es necesario escribir un programa para el microcontrolador. En el caso de este proyecto se escribió un programa en lenguaje assembler, por lo que se generó un archivo *.asm*. Para grabar este programa en el PIC, primero es necesario ensamblarlo, es decir, generar un archivo de extensión *.hex*.

El paquete MPASM incluye el **MPASM Assembler, el MPLINK Object Linker y el MPLIB Librtarian**.

Es posible producir el archivo *.hex* a partir de un único archivo *.asm* con código simple, o bien es posible generar un archivo *.hex* a partir de la unión de distintos códigos objeto y otros módulos ensamblados y/o compilados. Para unir estos módulos se utiliza la herramienta MPLINK. También es posible utilizar la herramienta MPLIB para generar bibliotecas, las cuales son una colección de códigos objeto listos para ser utilizados y que se almacenan todos juntos en un único archivo con extensión *.lib*.

## 2.6.3 Simulador Proteus

El simulador Proteus es un entorno gráfico de diseño electrónico sencillo e intuitivo, el cual permite simular un proyecto antes de probarlo en la práctica. El hecho de poder realizar esta simulación permite ahorrar tiempo y dinero al poder verificar el correcto funcionamiento del proyecto antes de armarlo en el laboratorio. Sin embargo, vale la pena aclarar que en la situación real pueden surgir inconvenientes que no surgieron en la simulación, principalmente debido a la presencia de ruido.

La ventaja de la simulación es que permite depurar los circuitos que se desee armar y hacer modificaciones a medida que se vayan detectando errores o funcionamientos no esperados.

## 2.6.4 Programador PICSTART Plus

El programador PICSTART Plus es un programador de prototipos de bajo costo y fácil de utilizar, el cual se conecta a la PC a través del puerto serial. Es a través de este dispositivo que es posible grabar el programa ensamblado en el microcontrolador.

## *2.7 Control de motores*

ATE-Universidad de Oviedo
Martin, Daniel C. ( 2001-2008)
es.wikipedia.org

El móvil construido utiliza dos motores para su desplazamiento: un motor de continua para la tracción (ruedas traseras) y un motor paso a paso (PaP) para la dirección (ruedas delanteras).

### 2.7.1 Motor de continua (Tracción)

"El **motor de corriente continua** es una máquina que convierte la energía eléctrica en mecánica, principalmente mediante el movimiento rotatorio." (…) "Una máquina de corriente continua (generador o motor) se compone principalmente de dos partes, un estator que da soporte mecánico al aparato y tiene un hueco en el centro generalmente de forma cilíndrica. En el estator además se encuentran los polos, que pueden ser de imanes permanentes o devanados con hilo de cobre sobre núcleo de hierro. El rotor es generalmente de forma cilíndrica, también devanado y con núcleo, al que llega la corriente mediante dos escobillas."

Para controlar el sentido de giro del motor de continua es necesario poder invertir la polaridad del voltaje aplicado a los terminales del motor. Existen varias formas de lograr esto: una es utilizando una fuente simétrica o dos fuentes de alimentación con un interruptor simple de dos contactos y otra es utilizar una fuente común con un interruptor doble. Para implementar la primera opción basta con utilizar dos transistores complementarios como muestra el siguiente esquema:

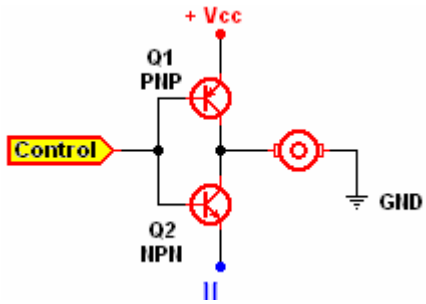


**Figura 2.15 - Motor de continua con fuente simétrica**

Para implementar la segunda opción es necesario implementar un puente H como se muestra a continuación:

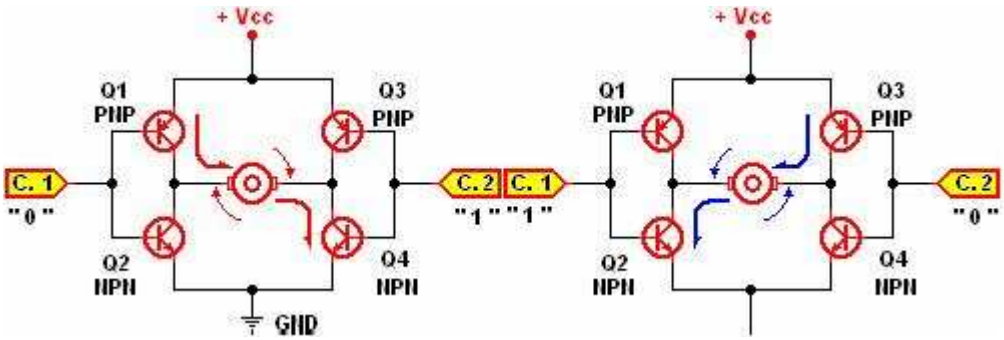


**Figura 2.16 - Motor de continua con puente H**

Si bien podría pensarse que es más sencilla la implementación con una fuente simétrica o con dos fuentes de alimentación, se decidió utilizar la implementación en la que se utiliza una fuente sencilla ya que existen circuitos integrados que implementan un puente H. En nuestro caso de decidió trabajar con el integrado L293D.

## 2.7.2 Motor PaP (Dirección)

Los motores PaP son motores cuyo giro se realiza a intervalos regulares en lugar de realizarse de continuo, como en el caso de los motores de continua. Estos motores giran en función de una secuencia de pulsos aplicados a sus devanados. El eje del motor gira un determinado ángulo por cada impulso de entrada, este ángulo se denomina **paso** y es uno de los parámetros fundamentales del motor. Una característica importante de este tipo de motores es que pueden quedar enclavados en una posición determinada.

Los motores PaP pueden ser Bipolares o Unipolares.

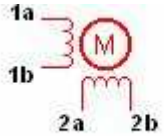


**Figura 2.17 - Motor PaP bipolar**

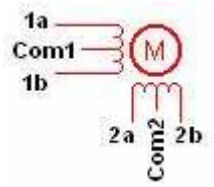


**Figura 2.18 - Motor PaP unipolar**

Los motores PaP bipolares utilizan dos bobinados independientes mientras que los motores PaP unipolares parecieran tener cuatro bobinados ya que tienen un Terminal central que es el común de cada par de bobinados. Por este motivo los motores PaP bipolares tienen cuatro cables y los motores PaP unipolares tienen cinco o seis cables dependiendo de si Com1 y Com2 están unidos o no.

Para controlar los motores PaP es necesario invertir las polaridades de los terminales de las bobinas en una determinada secuencia para lograr un giro en un sentido y en secuencia opuesta para que gire en el otro sentido. Con este propósito se utilizarán los dos puentes de un driver L293D.

Para controlar el motor se debe tener en cuenta la siguiente tabla:

| Número de Paso | RB4 | RB5 | RB6 | RB7 |
|---|---|---|---|---|
| **Paso 1** | + Vcc | GND | + Vcc | GND |
| **Paso 2** | + Vcc | GND | GND | + Vcc |
| **Paso 3** | GND | + Vcc | GND | + Vcc |
| **Paso 4** | GND | + Vcc | + Vcc | GND |

**Tabla 2-2 - Secuencia de pasos motor PaP**

## 2.7.3 Driver L293D

SGS-THOMSON Microelectronics (1996)

Como se mencionó anteriormente, la lógica de control del móvil será desarrollada a nivel de un PIC 16F877A. Por lo general la corriente de salida de un PIC es aproximadamente 25 mA, la cual es insuficiente para los motores por lo que resulta necesario amplificarla de algún modo.

Se investigaron los circuitos integrados existentes que implementaran la función de amplificación y que permitieran el control de los motores. Se concluyó que el integrado más comúnmente utilizado para aplicaciones similares a la de nuestro proyecto era el

L293. Se investigaron las variedades de L293 y se optó por utilizar el L293D, ya que éste incluye diodos de protección.

El driver L293D es un circuito integrado que implementa dos puentes H de transistores y está diseñado para aceptar niveles lógicos TTL o DTL estándar. Además, incluye diodos internos de protección para que en el momento en que se apaga el motor, el pico de tensión inversa que genera la inductancia del mismo no queme el circuito integrado.

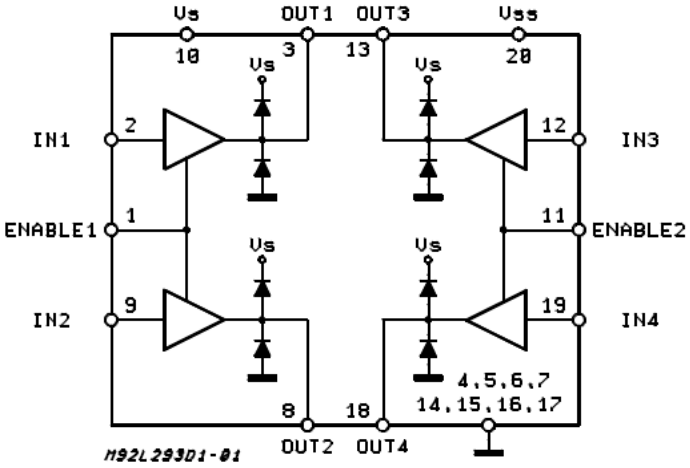


**Figura 2.19 - Esquema interno L293D**

Este integrado dispone de dos entradas: Enable 1 y Enable 2, las cuales permiten habilitar/deshabilitar los puentes 1 y 2 respectivamente, independientemente de otras señales de entrada.

El L293D dispone de dos voltajes de alimentación: Vs, que es el voltaje de alimentación para las etapas de salida de potencia (para los motores) y Vss, que es el voltaje de alimentación para la lógica. $V_S$ puede ser 36 V como máximo y $V_{SS}$ debe estar entre 4.5 V y 36 V.

Se recomienda conectar un capacitor no inductivo (por lo general de 100nF) entre $V_S$ y tierra y entre $V_{SS}$ y tierra, lo más cerca posible al integrado.

Por último, es recomendable que la tierra de la lógica y la tierra del motor se conecten separadas.

## *2.8 Manejo de sensores*

El móvil dispondrá de tres sensores: un sensor de sonido (para simular el sensor de vida) y dos sensores de obstáculos (para poder detectar los obstáculos que se presenten en su recorrido y así poder esquivarlos). En las siguientes secciones se describen estos sensores en más detalle.

### 2.8.1 Sensor de sonido

Como sensor de sonido se utilizó un circuito compuesto por un micrófono, un amplificador operacional, un detector de tono, resistencias, capacitores y un potenciómetro.

El micrófono capta los sonidos del ambiente y entrega una señal de voltaje. Mediante pruebas prácticas se analizó la señal que entrega el micrófono al reproducir a través de un auricular un tono de 1kHz. Se pudo observar una sinusoide aproximadamente centrada en 0 V de aproximadamente 35 mV pico a pico con frecuencia 1 kHz.

El menor voltaje de entrada que puede captar el detector de tono es de 20 mVrms. Por este motivo, es necesaria una pequeña amplificación para entregar la salida del micrófono al detector de tono. Además, a la señal de salida del micrófono se le superpone ruido eléctrico. Por estos motivos se implementó un filtro activo utilizando un amplificador operacional, el cual permite filtrar el ruido eléctrico y amplificar la señal.

La señal que se obtiene a la salida del filtro es entregada al detector de tono, el cual hará cambiar de estado un relé cuando detecte una señal de la frecuencia configurada. Esto hará que el PIC reciba un “1” lógico en la entrada correspondiente al sensor de sonido.

#### 2.8.1.1 Micrófono

Rossing, Thomas D. (1990)

Los micrófonos son transductores que detectan señales de sonido y producen una imagen eléctrica del sonido, es decir, producen un voltaje o una corriente que es proporcional a la señal de sonido.

Existen distintos tipos de micrófonos: dinámicos, de cinta, de condensador, de cristal y electret. En este proyecto se utiliza un micrófono electret por lo que se describirá este tipo en más detalle.

#### 2.8.1.1.1 Micrófono electret

Los micrófonos electret de condensador no deben compararse con los micrófonos de condensador estándar usados en estudios, los cuales tienen características de respuesta en frecuencia excelentes. Los micrófonos electret son micrófonos de condensador que usan un material electret para sus diafragmas que está permanentemente polarizado. De esta forma se evita la necesidad del voltaje de continua de polarización que es necesario en los micrófonos de condensador convencionales.

Los micrófonos electret de mejor calidad incorporan un preamplificador FET (transistor de efecto de campo) para emparejar su alta impedancia y amplificar la señal.

En la Figura 2.20 se muestra un esquema del micrófono electret.

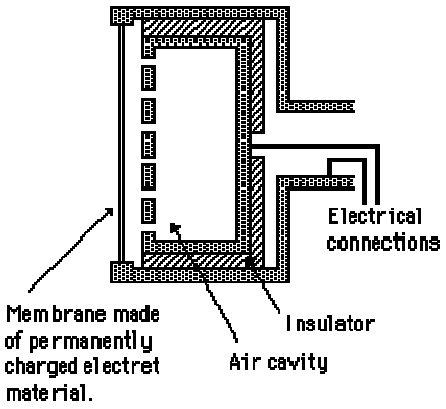


**Figura 2.20 - Esquema micrófono electret**

El material electret del diafragma puede tener un espesor menor a una milésima de pulgada. Aún así, está lo suficientemente polarizado para producir un cambio activo capacitivo en el voltaje entre la membrana y el plato posterior cuando ésta es movida por la presión de una onda de sonido.

La polarización se logra a través de una combinación de calor y alto voltaje durante la fabricación. El diafragma está respaldado por una película de metal evaporado.

#### 2.8.1.1.2 Alimentación del micrófono

Tomi Engdahl (1997-2000)

Los micrófonos electret necesitan energía para funcionar. Esta energía (típicamente entre 4 y 10 V) es alimentada al micrófono a través de una resistencia (típicamente entre 1 y 10 kΩ), llamada resistencia de polarización. Las cápsulas de los micrófonos electret por lo general consumen entre 0.5 y 2 mA de corriente al operar.

Existen micrófonos electret de dos patas y micrófonos electret de tres patas. Siempre una de ellas es la tierra. En el caso de los micrófonos de tres patas, una pata es para alimentarlo, otra es la pata de señal y la otra va conectada a tierra. En el caso de los micrófonos de dos patas, se utiliza la misma pata para la alimentación y para obtener la señal.

La señal de salida tiene una pequeña componente de continua proveniente de la corriente de polarización, por lo que suele agregarse un condensador adecuado en serie con la salida de audio para eliminar esta componente de continua.

En la Figura 2.21 se muestra un diagrama de cómo se alimenta un micrófono electret. A modo de ejemplo se muestra para un electret con tres patas.

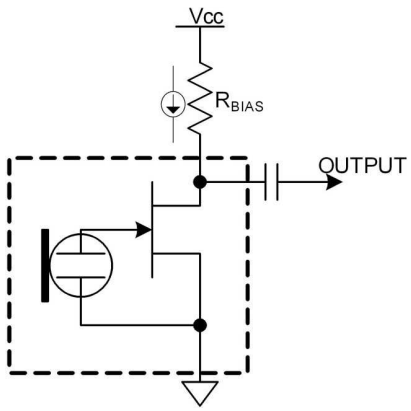


**Figura 2.21 - Alimentación de micrófono electret**

## 2.8.1.2 Amplificador Operacional

es.wikipedia.org

"Un amplificador operacional (comúnmente abreviado **A.O.** u **op-amp**), es un circuito electrónico (normalmente se presenta como circuito integrado) que tiene dos entradas y una salida. La salida es la diferencia de las dos entradas multiplicada por un factor (G) (ganancia): Vout = $G \cdot (V_+ - V_-)$. (…)Originalmente los **A.O.** se empleaban para operaciones matemáticas (suma, resta, multiplicación, división, integración, derivación, etc.) en calculadoras analógicas. De ahí su nombre."

Para representar un opamp se suele utilizar el símbolo que se muestra en la Figura 2.22.

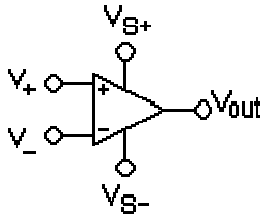


**Figura 2.22 - Símbolo opamp**

Como se puede observar en la figura, el opamp tiene como mínimo cinco terminales: entrada no inversora ($V_+$), entrada inversora ($V_-$), alimentación positiva ($V_{S+}$), alimentación negativa ($V_{S-}$) y salida ($V_{out}$).

### 2.8.1.2.1 Opamp ideal

Mancini, Ron (2002)

Al hablar de opamp ideal se asume que los parámetros del opamp son ideales. No existen en la práctica los opamps ideales, pero los opamps de hoy en día están tan cerca de lo ideal que el análisis para opamps ideales se aproxima a un análisis real. Para analizar los circuitos que incluyen opamps y entender su funcionamiento, se asume que se trabaja con opamps ideales.

Para los opamps ideales se asume que el voltaje de offset de entrada es cero y que la ganancia es constante. En los opamps reales existe un voltaje de offset de entrada y la ganancia es función de la frecuencia, por lo que va de valores muy altos en continua a valores pequeños a altas frecuencias.

También se asume que el flujo de corriente en las entradas del opamp es cero (es decir, que la impedancia de entrada del opamp es infinita), que la ganancia del opamp es infinita, que la diferencia de potencial entre las entradas es cero, que la impedancia de salida del opamp es cero.

En la Tabla 2-3 se resumen las asunciones que se realizan para opamps ideales.

| Nombre del parámetro | Símbolo del parámetro | Valor |
|---|---|---|
| Corriente de entrada | $I_{IN}$ | 0 |
| Voltaje de offset de entrada | $V_{OS}$ | 0 |
| Impedancia de entrada | $Z_{IN}$ | ∞ |
| Impedancia de salida | $Z_{OUT}$ | 0 |
| Ganancia | a | ∞ |

**Tabla 2-3 - Parámetros opamp ideal**

En la Figura 2.23 se muestra un esquema de un opamp con parámetros ideales.

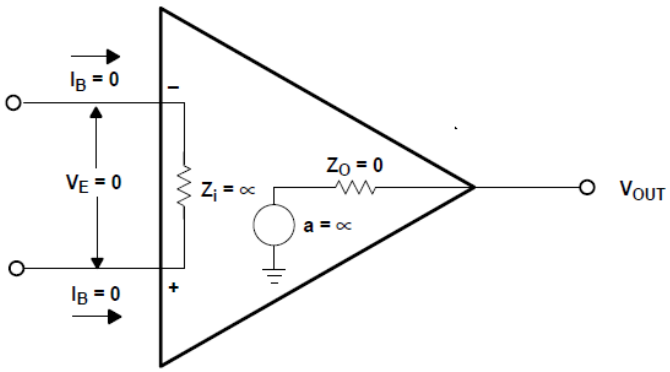


**Figura 2.23 - Opamp ideal**

En la práctica, los opamps basados en FETs tienen corrientes de entradas menores a 1 pA, por lo que no está muy lejos del caso ideal. Sin embargo, los opamps bipolares de alta velocidad tienen corrientes de entrada de decenas de µA.

Respecto de la ganancia, en el caso de los opamps reales, cuando el voltaje de salida es cercano al voltaje de la fuente de alimentación, el opamp satura, por lo que no puede lograrse una salida infinita.

### 2.8.1.2.2 Opamp TL070

Texas Instruments Incorporated (2001)

Inicialmente se utilizó como opamp el conocido LM 741. Luego de múltiples pruebas e intentos de lograr el correcto funcionamiento de la etapa de amplificación, se concluyó que el LM 741 no era adecuado para la aplicación del proyecto. Por este motivo se procedió a investigar sobre opamps más adecuados para aplicaciones de audio y se optó por el TL070.

El TL070 es un amplificador operacional de bajo ruido con entrada JFET. Tiene bajas corrientes de polarización de entrada, bajas corrientes de offset y una rápida tasa de respuesta. Esto logra baja distorsión de los armónicos y bajo ruido, por lo que el opamp TL070 resulta ideal para aplicaciones de pre-amplificación de audio y alta fidelidad.

### 2.8.1.2.3 Filtros activos

Mancini, Ron (2002)

Un filtro es un dispositivo que permite pasar señales eléctricas a ciertas frecuencias o ciertos rangos de frecuencia mientras evita el pasaje de otras.

En nuestro caso fue necesario agregar un filtro entre el micrófono y el detector de tono para eliminar el ruido eléctrico que se superponía a la señal de salida del micrófono.

A frecuencias altas (> 1 MHz), los filtros suelen consistir en componentes pasivos tales como inductores (L), resistencias (R) y capacitores (C), por lo que se llaman filtro LRC.

En el rango de frecuencias bajas (1 Hz – 1 MHz), sin embargo, el valor de los inductores sería muy alto y el inductor en sí sería muy voluminoso. En estos casos, los filtros activos se vuelven importantes.

Los filtros activos son circuitos que utilizan un opamp como el dispositivo activo en combinación con algunas resistencias y capacitores para proveer un desempeño similar al de los filtros LRC pero a bajas frecuencias.

En Figura 2.24 la se compara un filtro pasa bajo pasivo de segundo orden con un filtro pasa bajo activo de segundo orden.

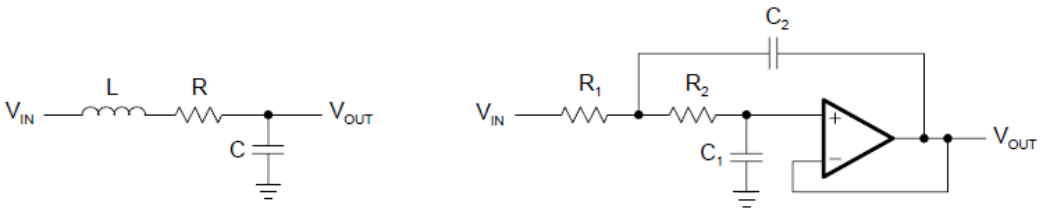


**Figura 2.24 - Filtro pasivo vs. filtro activo**

En este proyecto fue necesario implementar un filtro pasa banda centrado en 1 kHz, ya que el tono que se desea que sea detectado por el sensor de sonido tiene esta frecuencia. Por este motivo, en esta sección se describirán más en detalle los filtros activos pasa banda.

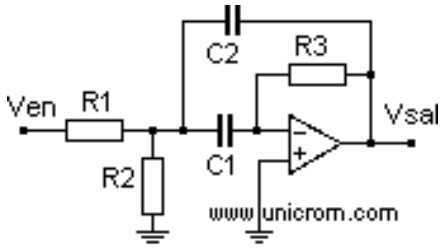


**Figura 2.25 - Filtro activo pasa banda**

El *filtro pasa banda* tiene una curva de respuesta en frecuencia como la que se muestra en la Figura 2.26. Dejará pasar todos los armónicos de la señal de entrada que se encuentren entre la frecuencia de corte inferior $f_1$ y la de corte superior $f_2$. Armónicos que se encuentren fuera de este rango de frecuencias serán atenuados y tendrán una amplitud menor al 70.7 % de su amplitud de entrada. La frecuencia central de este tipo de filtro se obtiene con la siguiente fórmula:

$$fo = 1 / [\, 2\pi C \times (R_3 R)^{1/2} \,]$$

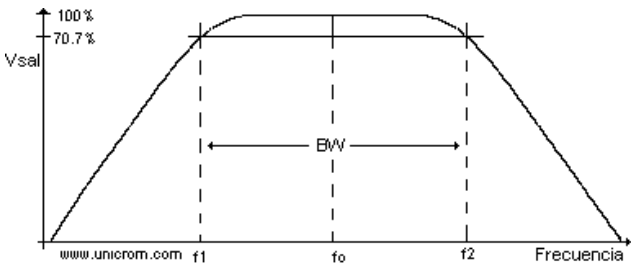


**Figura 2.26 - Respuesta en frecuencia de un filtro pasa banda**

Si se seleccionan los capacitores y resistores de modo que: **C1 = C2 = C** y **R1 = R2 = R**, el ancho de banda será:

$$\mathbf{BW = f_2 - f_1 = 1.41\ R\ /\ [\ CR_3\ (R_3R)^{1/2}\ ]}$$

Y el factor de calidad será:

$$\mathbf{Q} = \mathbf{fo\ /\ BW}$$

Las líneas discontinuas verticales sobre $f_1$ y $f_2$ y la línea horizontal del 70.7% representan la respuesta de un filtro pasa banda ideal.

Las frecuencias de corte ($f_1$ y $f_2$) son puntos en la curva de transferencia en que la salida ha caído 3 dB desde su valor máximo.

### 2.8.1.3 Detector de tono

National Semiconductor (1995)

Como detector de tono se utilizó el integrado LM567, cuyo diagrama de bloques se muestra a continuación:

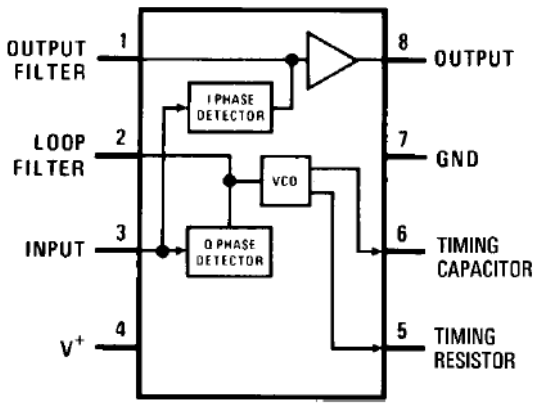


**Figura 2.27 - Diagrama de bloques LM567**

La frecuencia y el ancho de banda de detección se controlan con capacitores y resistencias externos. Para dimensionar el circuito se tuvieron en cuenta las siguientes ecuaciones:

$$f_O \approx \frac{1}{1.1R_1 C_1}$$

$$BW \approx 1070 \sqrt{\frac{V_I}{f_O C_2}} \text{ in \% of } f_O$$

$$V_I \leq 200mV_{RMS}$$

Where
$V_I$ = Input voltage ($V_{RMS}$)
$C_2$ = Low-pass filter capacitor (µF)

**Figura 2.28 - Ecuaciones LM567**

También se tuvieron en cuenta las siguientes recomendaciones mencionadas en la hoja de datos:

1. Se recomienda que $R_1$ esté entre 2 kΩ y 20 kΩ para mejor estabilidad de temperatura.

2. Para seleccionar el capacitor pasa-bajo $C_2$ referirse a la gráfica “Bandwith vs. Input signal amplitude”.

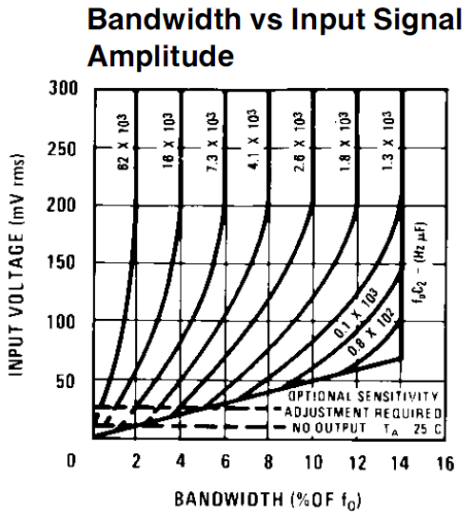


**Figura 2.29 - Ancho de banda vs amplitud de la señal de entrada**

3. Por lo general, el valor de $C_3$ no es crítico. $C_3$ establece el borde de la banda de un filtro pasa-bajo que atenúa las frecuencias fuera de la banda de detección para eliminar salidas espurias. Un valor típico para $C_3$ es $2C_2$.

4. Opcionalmente puede conectarse una resistencia entre los pines 1 (Output filter capacitor $C_3$) y 4 (Supply Voltage), el cual establece el umbral para el voltaje de entrada

más alto que no produce salida. Para asegurar el límite testeado de 10 $mV_{RMS}$ como mínimo, se utiliza un valor de 130 kΩ para $R_2$.

Cuando el integrado detecta una entrada cuya frecuencia está en la banda pasante y cuyo voltaje supera cierto umbral, se obtiene un "0" lógico en la salida.

### 2.8.1.4 Relé

es.wikipedia.org

Para que el PIC reciba un "1" lógico cuando el detector de tono entregue un "0" lógico se utilizó un relé.

"El relé o relevador, del inglés *relebeitor*, relevo, es un dispositivo electromecánico, que funciona como un interruptor controlado por un circuito eléctrico en el que, por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes."

En la Figura 2.30 se muestran los elementos que forman un relé.

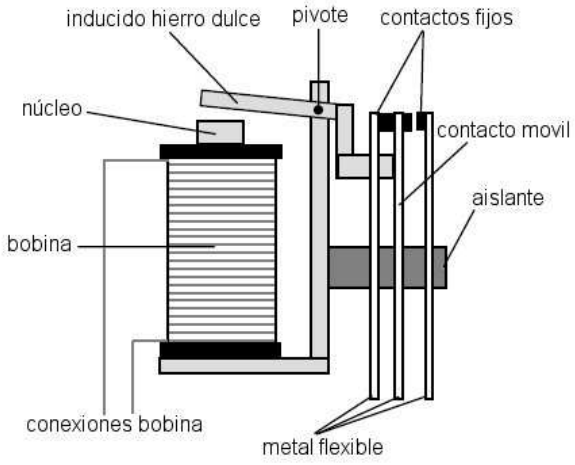


**Figura 2.30 - Partes del relé**

El relé tiene contactos normalmente abiertos (NA), es decir que conectan el circuito cuando el relé es activado y contactos normalmente cerrados (NC), es decir que, cuando el relé es activado, desconectan el circuito. El relé también tiene un contacto de conmutación, el cual controla dos circuitos: unos NA y uno NC con una terminal común.

En este proyecto se conectó el contacto NA a 5 V y el NC a tierra. Cuando el detector de tono entrega un “0” lógico en su salida, circula corriente por la bobina del relé. En este momento se conecta el contacto de conmutación con el contacto NA, es decir que se obtienen 5 V en el contacto de conmutación, el cual es conectado a una entrada del PIC.

## 2.8.2 Sensor infrarrojo

Luego de múltiples pruebas y modificaciones sobre el sensor de sonido, sin lograr el correcto funcionamiento del mismo, se decidió simular el sensor de vida con un sensor infrarrojo (IR) en lugar de simularlo con un sensor de sonido. Este cambio se explica en el capítulo Desarrollo del proyecto.

### 2.8.2.1 Luz infrarroja

La luz infrarroja (IR) es un tipo de luz no visible, cuya longitud de onda es 850-900 nm, por lo que en el espectro se encuentra entre la luz visible y las microondas. La luz IR contiene información que no contiene la luz visible, contiene información respecto de la temperatura de un cuerpo. Cualquier cuerpo cuya temperatura sea mayor que 0 Kelvin (K) emite radiación IR.

El nombre infrarrojo se debe a que en el espectro se encuentra por debajo del rojo, es decir que el espectro asociado a la luz IR termina justo antes de que comience el espectro visible, cuyo primer color es el rojo.

### 2.8.2.2 Emisor IR

Un emisor IR es un tipo especial de diodo emisor. Un diodo emisor es un diodo semiconductor que al ser atravesado por una corriente eléctrica emite radiaciones electromagnéticas en una estrecha banda de longitudes de onda, dependiendo del semiconductor con el que esté fabricado. En el caso del emisor IR, las radiaciones electromagnéticas que emite son en la banda de longitudes de onda del IR.

### 2.8.2.3 Transistores

enciclopedia.us.es (2009)

Para lograr que el emisor IR emita luz IR a una frecuencia determinada se utilizó un transistor PNP como oscilador y un transistor NPN para amplificar la señal.

"Un **transistor** (la contracción de *transfer resistor*, transferencia de resistencia) es un dispositivo semiconductor con tres terminales utilizado como amplificador e interruptor en el que una pequeña corriente o tensión aplicada a uno de los terminales controla o modula la corriente entre los otros dos terminales."

Un transistor tiene tres partes: una que emite electrones (el emisor), otra que los recibe o recolecta (el colector) y otra con la que se modula el paso de dichos electrones (la base). Una pequeña señal eléctrica aplicada entre la base y emisor modula la que circula entre emisor y receptor. La señal base-emisor puede ser muy pequeña en comparación con la emisor-receptor. La señal emisor-receptor es aproximadamente la misma que la base-emisor pero amplificada.

Existen distintos tipos de transistores: transistores de unión bipolar, transistores de unión unipolar, transistores de efecto de campo, transistores de punta de contacto, fototransistores. Para el emisor IR se utilizar dos transistores de unión bipolar por lo que se explicará este tipo de transistores.

#### 2.8.2.3.1 Transistores de unión bipolar

es.wikipedia.org (2009)

Los transistores de unión bipolar, también llamados transistores bipolares, están formados por dos uniones PN en un solo cristal semiconductor, separados por una región muy estrecha. Dentro de este tipo de transistores existen transistores PNP o NPN, dependiendo del tipo de cada una de las tres regiones (emisor, base y colector).

En la Figura 2.31 se muestran los símbolos correspondientes a estos transistores, la flecha indica la dirección del flujo de la corriente en cada caso.

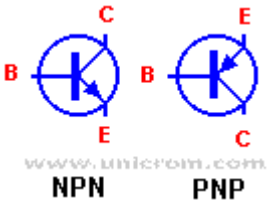


**Figura 2.31 - Transistores NPN y PNP**

En la Figura 2.32 se muestra la característica idealizada de un transistor bipolar, en la cual **Ic** representa la corriente en el colector, **Ib** representa la corriente en la base, **Vce** representa la diferencia de potencial entre colector y emisor y **Vbe** representa la diferencia de potencial entre base y emisor. **β** es la ganancia de corriente del transistor.

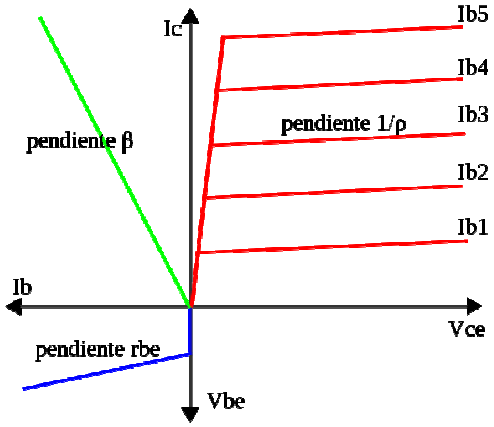


**Figura 2.32 - Característica del transistor bipolar**

### Transistor bipolar NPN

"NPN es uno de los dos tipos de transistores bipolares, en los cuales las letras "N" y "P" se refieren a los portadores de carga mayoritarios dentro de las diferentes regiones del transistor. La mayoría de los transistores bipolares usados hoy en día son NPN, debido a que la movilidad del electrón es mayor que la movilidad de los "huecos" en los semiconductores, permitiendo mayores corrientes y velocidades de operación.

Los transistores NPN consisten en una capa de material semiconductor dopado P (la "base") entre dos capas de material dopado N. Una pequeña corriente ingresando a la base en configuración emisor-común es amplificada en la salida del colector."

**Transistor bipolar PNP**

“El otro tipo de transistor de unión bipolar es el PNP con las letras "P" y "N" refiriéndose a las cargas mayoritarias dentro de las diferentes regiones del transistor. Pocos transistores usados hoy en día son PNP, debido a que el NPN brinda mucho mejor desempeño en la mayoría de las circunstancias.

Los transistores PNP consisten en una capa de material semiconductor dopado N entre dos capas de material dopado P. Los transistores PNP son comúnmente operados con el colector a masa y el emisor conectado al terminal positivo de la fuente de alimentación a través de una carga eléctrica externa. Una pequeña corriente circulando desde la base permite que una corriente mucho mayor circule desde el emisor hacia el colector.”

### 2.8.2.3.2 Regiones operativas del transistor bipolar

Según cómo esté polarizado el transistor, el mismo se encontrará en una de las siguientes regiones de operación:

1. **Región activa**. Es una región intermedia entre la región de corte y la región de saturación, en la cual la corriente de colector depende principalmente de la corriente de base, de β (ganancia de corriente) y de las resistencias que se encuentren conectadas en el colector y en el emisor. Es la región en la cual debe estar el transistor si se lo desea utilizar como amplificador.

2. **Región inversa**. El transistor está en esta región si se invierten las condiciones de polaridad del funcionamiento en modo activo. En esta región, el colector y el emisor intercambian roles. En modo inverso, β es mucho menor que en modo activo.

3. **Región de corte**. Un transistor está en corte cuando la corriente de colector y la corriente de emisor son cero. En esta región el voltaje entre el colector y el emisor es el voltaje de alimentación del circuito. Este caso normalmente se presenta cuando la corriente de base es nula.

4. **Región de saturación**. Un transistor está en saturación cuando la corriente de colector y la corriente de emisor son iguales a la corriente máxima, la cual

depende del voltaje de alimentación del circuito y de las resistencias conectadas en el colector o el emisor o en ambos. Este caso se presenta normalmente cuando la corriente de base es lo suficientemente grande como para inducir una corriente de colector β veces más grande.

#### 2.8.2.3.3 Transistor como interruptor

Como se explica en el capítulo de Desarrollo, al intentar comunicar el PIC con la placa BB se detectó el problema de que éstas manejan lógicas de distintos niveles. Para solucionar este problema se utilizó un transistor como interruptor. Esta configuración se explica a continuación.

Se conectó la fuente de 5 V al colector de un transistor NPN a través de una resistencia; se conectó el emisor del transistor a tierra y se conectó la salida de la placa BB a la base a través de otra resistencia. La señal en el colector se utilizó como señal de salida.

La Figura 2.33 muestra un esquema de la conexión utilizada.

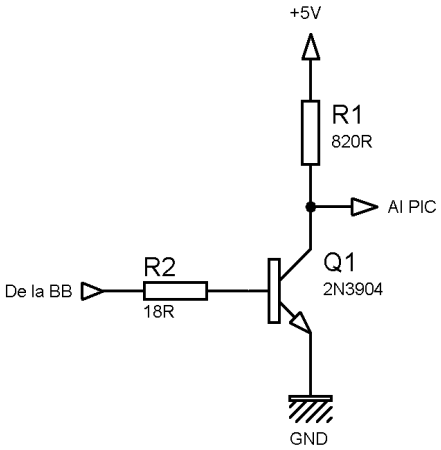


**Figura 2.33 - Transistor como interruptor**

En esta configuración, el transistor puede estar en uno de dos estados: apagado o encendido. Cuando la entrada es de 0 V, se dice que el interruptor está encendido. El voltaje $V_{CE}$ a través del transistor es casi cero y el transistor está en corte. Es decir que no circula corriente por el colector. Al no circular corriente por el colector, no hay caída de tensión en la resistencia R1 y, por lo tanto, la señal de salida es de 5 V.

Cuando la entrada es el nivel alto de la placa BB (1.8 V), como la resistencia colocada en la base del transistor (R2) es pequeña, la corriente de base será grande, por lo que inducirá en el colector una corriente β veces más grande. El transistor estará saturado, es decir que la corriente de colector será máxima por lo que la señal de salida será de 0 V.

#### 2.8.2.4 Receptor IR

Como receptor IR se utilizó un fototransistor IR. Se llama fototransistor a un transistor sensible a la luz, normalmente al IR. En este tipo de fototransistores, la base está reemplazada por un cristal fotosensible (por lo general, de silicio) que cuando recibe luz, produce una corriente y desbloquea el transistor. En el fototransistor la corriente circula solo en un sentido y el bloqueo del transistor depende de la luz; cuanta más luz hay, más corriente conduce.

### 2.8.3 Sensores de obstáculos

Como sensores de obstáculos se utilizan microswitches que cambian de estado cuando uno de los dos paragolpes del móvil choca contra algo. Debido a que probablemente el móvil choque y se aleje un poco del obstáculo, los interruptores entregarán un “1” lógico por un breve período de tiempo. Es decir que generarán un pulso de corta duración. Para retener este pulso y dar tiempo a que el programa del PIC lea esta información, se implementó un flip-flop RS asíncrono.

#### 2.8.3.1 Microswitches

Un microswitch es un interruptor eléctrico que puede cambiar de estado con una fuerza física muy pequeña, a través del uso de un mecanismo de punto de inflexión. Es decir que el dispositivo se encuentra en un estado de equilibrio estable y al ser presionado el botón pasa a otro estado de equilibrio, distinto del primero.

Internamente, al presionar el botón se dobla una tira de metal rígido, activando el interruptor. Al remover la presión sobre el botón, la tira de metal vuelve a su estado original.

Existen diversos tipos de microswitches. En la Figura 2.34 se muestra el esquema de un microswitch similar al que se utilizó en este proyecto.

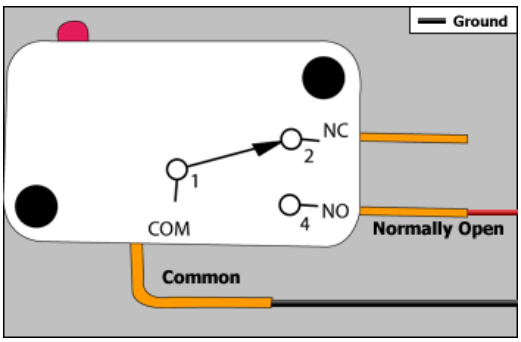


**Figura 2.34 - Esquema microswitch**

Como se puede observar en la figura, el microswitch dispone de un contacto NC, de un contacto NA y de un contacto común. Mientras el botón no se encuentra presionado, el común está conectado con el contacto NC. Mientras el botón se encuentre presionado, el común estará conectado al contacto NA.

En este proyecto se utilizó el común como salida y se conectó el contacto NC a tierra y el contacto NA a 5 V, por lo que al presionar el botón se generará un pulso de 5 V que será leído como un “1” lógico.

### 2.8.3.2 Flip-Flop RS asíncrono

Para retener los pulsos generados por los microswitches, se utilizaron flip-flops RS asíncronos.

Los flip-flops son los elementos básicos de memoria. Estando el flip-flop alimentado, puede mantener un estado binario indefinidamente hasta que reciba una señal de entrada que lo haga cambiar de estado.

Existen distintos tipos de flip-flops: RS, JK, T y D. Los últimos tres se pueden implementar a partir del primero. Además, cualquiera de estos puede ser de dos tipos: activado por nivel o maestro esclavo. El primero actúa de acuerdo a los niveles de amplitud que recibe "0" o "1", mientras que el segundo en realidad son dos flip-flops combinados de manera tal que uno "hace caso" al otro. Por otro lado, los flip-flops pueden ser síncronos o asíncronos. Los primeros tienen una entrada de reloj y responden a los niveles de entrada durante la ocurrencia del reloj, mientras que los otros no utilizan reloj.

Teniendo en cuenta que los flip-flops JK, T y D se construyen a partir del flip-flop RS y que en este proyecto se utilizaron flip-flops RS asíncronos, en esta sección se describirá más en detalle este tipo de flip-flop.

El flip-flop RS asíncrono tiene dos salidas, Q y Q', y dos entradas, S (set) y R (reset).

Un flip-flop RS asíncrono puede implementarse utilizando compuertas NOR o bien utilizando compuertas NAND. A continuación se muestra la tabla de verdad con las dos posibles implementaciones:

| R | S | Q (NOR) | Q' (NAND) |
|---|---|---|---|
| 0 | 0 | Q | N.D. |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | N.D. | Q |
| N.D. = Estado no determinado<br>Q = Estado anterior | | | |

**Tabla 2-4 - Tabla de verdad flip-flop RS**

Se optó por la implementación con compuertas NOR debido a que para esta aplicación se desea que si ambas entradas del flip-flop (Set y Reset) están en cero, el flip-flop mantenga su estado. En la Figura 2.35 se muestra un esquema de esta implementación.

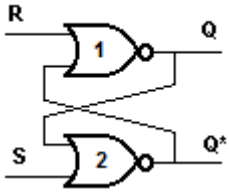


**Figura 2.35 - Flip-flop RS con compuertas NOR**

Para analizar el funcionamiento de un flip-flop RS implementado con compuertas NOR hay que tener en cuenta la tabla de verdad de estas compuertas, la cual se muestra en la Tabla 2-5.

| Entrada A | Entrada B | Salida |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**Tabla 2-5 - Tabla de verdad NOR**

Asumimos como punto de partida que la entrada S está en “1” y que la entrada R está en “0”. Como la compuerta 2 tiene una entrada en “1”, su salida, Q’, es “0”. Esto coloca ambas entradas de la compuerta 1 en “0”, por lo que su salida, Q, es “1”.

Si la entrada S cambia a “0”, las salidas permanecerán iguales ya que la salida Q permanece como “1”, dejando una entrada de la compuerta 2 en “1”. Esto causa que la salida Q’ permanezca en “0”, por lo que ambas entradas de la compuerta 1 estarán en “0” y la salida Q será “1”.

De forma similar, si la entrada R se pone en “1”, la salida Q cambia a “0” y la salida Q’ cambia a “1”. Cuando la entrada R cambia a “0”, las salidas no cambian. Cuando las dos entradas, R y S, se ponen en “1”, las dos salidas, Q y Q’, quedan en “0”. Esta situación viola el hecho de que las salidas Q y Q’ son complementarias entre sí. En operación normal esta situación debe evitarse.

## 2.9 Repositorio Subversion

es.wikipedia.org

"Un repositorio, depósito o archivo es un sitio centralizado donde se almacena y mantiene información digital, habitualmente bases de datos o archivos informáticos.(…)
Los depósitos están preparados para distribuirse habitualmente sirviéndose de una red informática como Internet o en un medio físico como un disco compacto. Y pueden ser de acceso público, o pueden estar protegidos y necesitar de una autentificación previa. Los depósitos más conocidos son los de carácter académico e institucional.

A diferencia de los ordenadores personales o de las PC de escritorio, los depósitos suelen contar con sistemas de Backup y mantenimiento preventivo y correctivo, lo que hace que nuestra información se pueda recuperar en el caso que nuestra máquina o pc quede inutilizable."

"Subversion es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como svn por ser ese el nombre de la herramienta de línea de comandos.

Una característica importante de Subversion es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único—si se ha hecho un cambio incorrecto a los datos, simplemente deshaga ese cambio."

Para almacenar la información del proyecto se utilizó el servicio Google Code, el cual proporciona almacenamiento Subversion para sus proyectos de software libre. Para acceder a esta información es necesario disponer de un software cliente.

Existen diversos clientes que proveen interfaces a Subversion. En este proyecto se utilizó como cliente el TortoiseSVN, el cual provee integración con el explorador de Windows, brindando una forma sencilla de mantener el repositorio actualizado.

# 3 Marco Metodológico del Proyecto Final de Carrera

## *3.1 Introducción a la metodología*

Las tareas asociadas a este proyecto se pueden dividir en tareas de gestión y tareas técnicas. Dentro de las tareas de gestión están incluidas las tareas de planificación, control de avance y redacción de documentos. Dentro de las tareas técnicas están incluidas las tareas de investigación teórica, implementación y testeo.

## *3.2 Gestión del proyecto*

### 3.2.1 Planificación

Para planificar el desarrollo del proyecto se creó un cronograma utilizando la aplicación Microsoft Project. En este cronograma se realizó un desglose de las tareas en sub-tareas y se asignó la responsabilidad de las mismas a los integrantes del equipo de trabajo. Además, la aplicación genera un diagrama de Gantt, el cual permite visualizar de forma clara el desarrollo del proyecto.

Para la creación del cronograma fue necesario estimar los tiempos que habría que dedicarle a cada tarea, teniendo como referencia el trabajo realizado en los proyectos de las asignaturas cursadas durante la carrera. Además, al decidir la duración de cada tarea se agregó un margen de seguridad teniendo en cuenta que podían surgir imprevistos.

Para la asignación de tareas se tuvo en cuenta la modularidad del proyecto, la cual permitió asignar las tareas de forma de que cada integrante pudiera avanzar con las tareas bajo su responsabilidad, en paralelo al avance del otro integrante. Es decir, que el avance de uno de los integrantes del equipo condicionaba lo mínimo posible el avance del otro integrante.

Es importante tener en cuenta que en algunos casos no fue posible lograr esta independencia, ya que algunas tareas requerían de la participación en simultáneo de ambos integrantes. Por ejemplo, aquellas tareas que requerían definir interfaces que afectan a ambas partes o bien aquellas tareas que requerían tomar decisiones de alcance global.

### 3.2.2 Control de avance

Mientras se cursaba la Asignatura Proyecto de Telemática se realizaron controles de avance semanales. En estos controles se verificaba que el proyecto se estuviera desarrollando conforme a lo previsto en el cronograma y se asentaban los avances en actas, las cuales eran firmadas tanto por los tutores del proyecto como por los integrantes del equipo.

En el Anexo II – Actas se pueden encontrar las actas generadas.

Para poder representar los avances de forma gráfica, se utilizó una funcionalidad de la aplicación Microsoft Project que permite establecer el porcentaje de completitud de cada tarea. Esto facilita la toma de decisiones respecto de la prioridad de las tareas a realizar.

### 3.2.3 Redacción de documentos

Se redactó un documento de Alcance y un documento denominado Plan de Proyecto.

#### 3.2.3.1 Documento de Alcance

En el documento de Alcance se realizó una introducción al proyecto, comentando sobre la oportunidad de negocio detectada y describiendo el producto que se pretendía desarrollar. Además, se especificaron las funcionalidades que serían incluidas y aquellas que no. También se especificaron los entregables del proyecto y se anexó el cronograma inicial en el que se podían observar las distintas fechas de entrega de los mismos, además de las fechas de realización de las distintas tareas.

#### 3.2.3.2 Plan de Proyecto

En el Plan de proyecto se incluyó el Acta del Proyecto, la cual está compuesta por los siguientes elementos:

- Nombre del proyecto
- Fecha de comienzo
- Áreas de conocimientos/procesos
- Áreas de aplicación (sector/actividad)

- Fecha de inicio del Proyecto
- Fecha tentativa de finalización del Proyecto

En este documento también se incluyeron los objetivos del proyecto, general y específicos; una descripción del producto; la necesidad del proyecto; el alcance (detallando las funcionalidades incluidas y las no incluidas); la justificación del impacto del proyecto; los entregables del proyecto, las restricciones del proyecto y se identificaron los grupos de interés (directos e indirectos) del proyecto.

En el

Anexo I – Plan del Proyecto Final de Carrera se adjunta el plan de proyecto.

## *3.3 Tareas técnicas*

### 3.3.1 Investigación

Como primer sub-tarea de la mayoría de las tareas previstas en el cronograma, se planificó la investigación sobre todos los temas teóricos relacionados con esa tarea.

En todos los casos se realizaron investigaciones de tipo secundario, buscando en internet, en manuales, tutoriales, foros y en distintos libros, creando la base teórica necesaria para poder proceder a la implementación.

Toda la información recolectada durante las etapas de investigación fue respaldada en un repositorio Subversion para luego ser usada en la redacción del marco teórico de este informe.

### 3.3.2 Implementación y testeo

Para los módulos asociados al PIC se dividió la implementación y el testeo en dos etapas: primero se implementó y testeó en una simulación (utilizando el programa Proteus) y luego se armaron los circuitos físicos y se testearon en el laboratorio. Esta última etapa, a su vez se dividió en dos: primero se armaron los circuitos sobre protoboards y luego de verificado su correcto funcionamiento, se soldaron sobre placas pertinax.

La simulación en la aplicación Proteus permitió depurar el programa del PIC y permitió corregir algunos detalles de los circuitos diseñados.

De todas formas, fue necesario seguir realizando modificaciones luego de armados los circuitos físicos. Esto se debe a que en la práctica existen diversos factores que afectan el funcionamiento de estos circuitos, tales como el ruido eléctrico, que no son tenidos en cuenta en la simulación.

# 4 Desarrollo del proyecto

En este capítulo se describen las tareas realizadas hasta el momento comentando cuáles de ellas ya se han completado, cuáles no, cuáles aún no han comenzado, si el porcentaje de cumplimiento está de acuerdo a lo planificado, etc.

## *4.1 Definición del tema*

La primer tarea de este proyecto fue la de definir el tema. Se partió de la base de que se pretendía aprovechar la placa de desarrollo BeagleBoard (BB) utilizada por otra alumna de la facultad en su proyecto final.

Además, era necesario utilizar algún dispositivo para implementar la interfaz entre la placa BB y los motores y sensores. Se decidió que este dispositivo fuera un microcontrolador PIC, debido a que ya se había trabajado con este tipo de microcontroladores en asignaturas anteriores de la carrera. Por otro lado, se planteó la idea de integrar al proyecto el uso de un sistema GPS.

Integrando todas estas ideas, el concepto del producto final de este proyecto empezó a tomar forma. Luego se buscó una aplicación que motivara al proyecto y que fuera adecuada a estos tiempos y a la situación presente.

De esta forma se llegó al tema del proyecto: desarrollo de un sistema automatizado de búsqueda de supervivientes de catástrofes.

## *4.2 Estudio de la placa*

Se leyeron los manuales y documentación disponibles para familiarizarse con la placa BB y su funcionamiento. Además, se buscaron proyectos de desarrollo basados en esta misma placa, analizando los informes u otros documentos de dichos proyectos.

## *4.3 Definición del Formato de Documentos*

Para que hubiera una coherencia entre todos los documentos, se creó una plantilla como template, a partir de la cual se generaron todos los documentos del proyecto. Esta plantilla establece el formato de la carátula, la información de historial y de referencias que debe incluirse, el formato de los títulos y del texto, etc. Este template se almacenó en el repositorio para que ambos integrantes del equipo tuvieran acceso a él.

## *4.4 Desarrollo del plan de proyecto*

Esta tarea ya fue explicada en el capítulo “Marco Metodológico del Proyecto Final de Carrera”.

## *4.5 Compilación e instalación de ARM EABI Ubuntu 9.04*

### 4.5.1 Fundamento de la elección del sistema operativo

En el caso del proyecto UMix se optó por trabajar con el sistema operativo Ubuntu por tres grandes motivos:

1. El primer motivo es que Ubuntu se basa en el desarrollo de la distribución Debian, que es, por su parte, ARM compliant. De esta forma, al trabajar con un sistema operativo abalado por el propio fabricante del procesador que tiene la placa en cuestión, se minimizan los riesgos de no funcionamiento.

2. Por otro lado, en el caso de Ubuntu, es más sencillo encontrar material de apoyo para esta distribución debido a la gran inserción que ha tenido en el mercado de las computadoras personales en los últimos años.

3. En último lugar, pero no menos importante, el equipo de trabajo del proyecto tiene más experiencia trabajando con dicha distribución que con cualquier otra. De esta forma se pueden aprovechar los tiempos que llevaría habituarse a otras distribuciones de Linux para realizar otras actividades más críticas.

### 4.5.2 Conexión Serial

Previo a la instalación de Ubuntu en la BB es necesario establecer una conexión serial con la placa BB. A continuación se detalla el procedimiento para establecer dicha conexión.

En primer lugar se deben conseguir al menos los cables y conectores necesarios. Es decir, un cable **Null-modem**, también conocido como cable **serial cruzado**, un "Ribbon cable" y un puerto en la computadora huésped que permita conexiones seriales (puede ser un **puerto serial PCI** o un **adaptador usb-serial**).

La Figura 4.1 muestra el interconexionado de los cables.

**Figura 4.1 - Conexión Serial**

El diagrama de pines de cada uno de los cables puede ser encontrado en el manual de referencia de la placa BB.

Para comprobar el funcionamiento de la conexión se deben ejecutar las siguientes operaciones:

1. Iniciar una sesión de Windows y ejecutar el programa **hyper terminal** para crear una nueva conexión. Debería aparecer una pantalla similar a la siguiente:

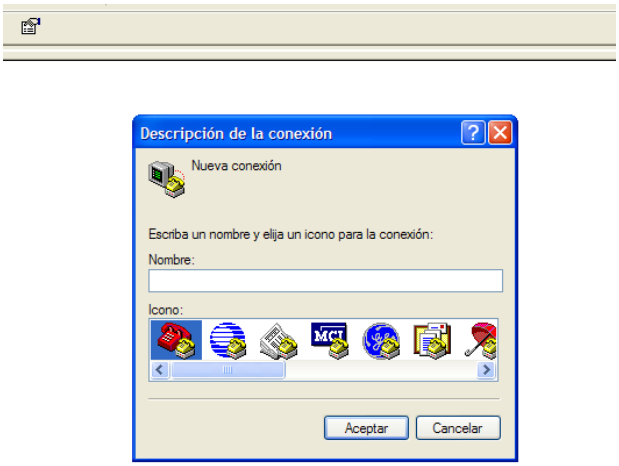


**Figura 4.2 - Nueva conexión HyperTerminal**

2. Escoger un nombre para la conexión. A continuación aparecerá una pantalla donde se podrá elegir el tipo de conexión a realizar. En el caso de este proyecto se seleccionó **com3** aunque esto depende de cada computadora en particular.

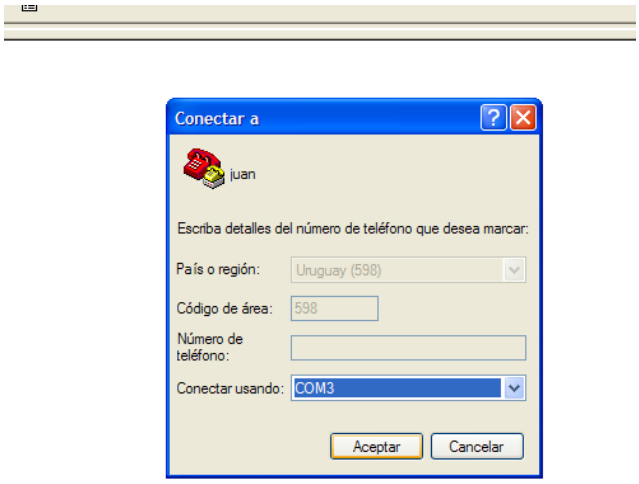


**Figura 4.3 - Establecimiento de la conexión serial**

3. Para terminar de configurar la conexión, la aplicación HyperTerminal presenta una ventana en donde se pueden cambiar determinados parámetros de la conexión. Para conectarse con la placa BB se debe seleccionar la siguiente configuración.

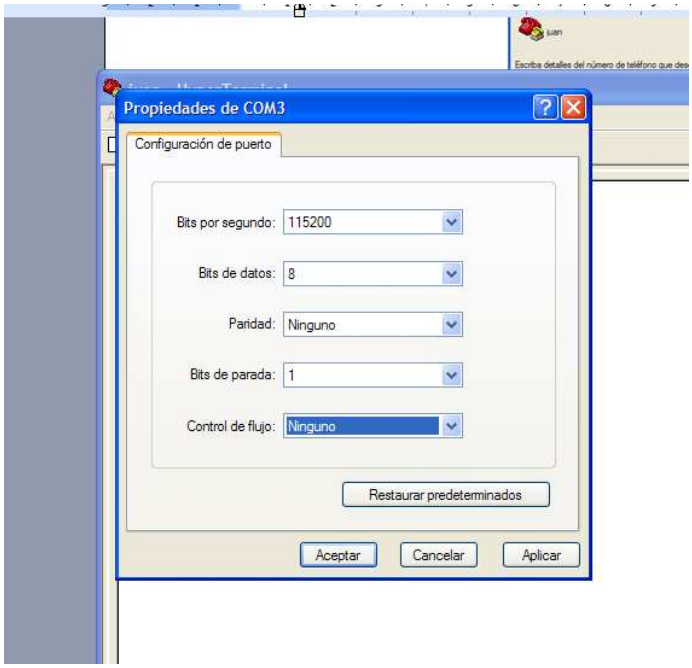


**Figura 4.4 - Propiedades de la conexión**

4. En caso de haber funcionado correctamente debería aparecer una serie de caracteres como los siguientes:

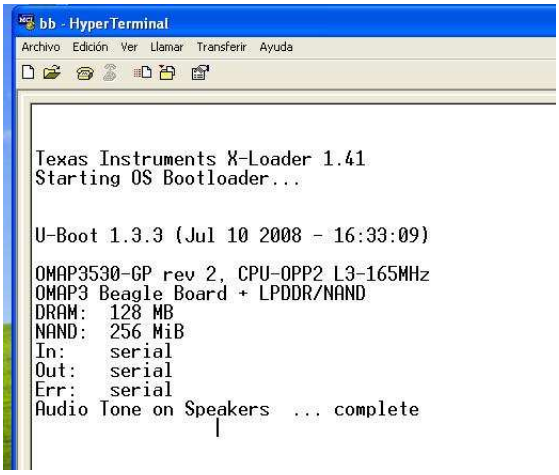


**Figura 4.5 - Conexión establecida**

Para más información se puede consultar el manual de usuario de la placa BB.

### 4.5.3 Actualización de uBoot

La BB posee en su memoria un gestor de arranque, llamado **uBoot**, que es conveniente actualizar antes de comenzar la instalación del sistema operativo. Las instrucciones sobre cómo realizar esta operación se encuentran claramente detalladas en el manual de usuario de la placa y por eso no serán tratadas en este documento.

### 4.5.4 Compilación del kernel (Método 1)

Actualmente existen dos formas para obtener una versión ejecutable del kernel de Ubuntu para la placa BB.

La primera forma, que es la que se describe en ésta sección, consiste en obtener el código pre-compilado para el hardware que utilizaremos. La segunda forma, que se describe en la siguiente sección, consiste en obtener el código fuente del kernel y luego compilarlo manualmente.

En nuestro caso, y debido a que no se tenía previsto hacer modificaciones en el kernel se optó por utilizar la primera forma.

#### 4.5.4.1 Adaptación de la PC huésped

Para comenzar la instalación de Ubuntu con un kernel pre-compilado es primero necesario adaptar determinada PC, que denominaremos huésped, en donde se adaptará el kernel seleccionado y se desarrollarán las aplicaciones que luego se ejecutarán en la placa BB.

- **Ubuntu Jaunty**

Debido a la gran disponibilidad de compiladores y el gran soporte que ofrece este sistema operativo se optó por utilizar el mismo para la PC huésped. Además, es conveniente, a la hora de desarrollar aplicaciones embebidas, que la PC huésped tenga la mayor similitud con la plataforma en donde residirá definitivamente la aplicación.

La instalación de este sistema operativo es intuitiva e incluso se puede realizar directamente desde el live-cd de la distribución. No se incluye en este manual una

descripción de la instalación de dicho sistema operativo. Para mayor información acerca de la instalación existen multitud de guías en Internet.

- **Qemu**

Una vez instalado el sistema operativo de la PC huésped es necesario instalar alguna herramienta que permita simular la plataforma de la placa BB, de forma de poder emular el comportamiento de cualquier aplicación en la PC huésped como si se estuviera directamente trabajando sobre la placa. Para ello es que existe una aplicación llamada Qemu.

La herramienta recién referida se encuentra dentro de las aplicaciones disponibles dentro del repositorio de Ubuntu y por lo tanto su instalación consiste simplemente en la ejecución de los siguientes comandos dentro de una terminal:

```
$sudo apt-get install qemu
```

Cabe aclarar que para instalar cualquier aplicación desde los repositorios es necesario disponer de una conexión a Internet.

- **Debootstrap**

Otra de las herramientas que facilitan la instalación del sistema operativo Ubuntu es la herramienta denominada **debootstrap**. La función de esta aplicación es simplificar todo el proceso de instalación de cualquier sistema operativo basado en Debian, sobre todo a la hora de manejar el particionamiento de las unidades de memoria de la plataforma objetivo.

Para instalar la última versión de la herramienta se deben ejecutar los siguientes comandos en una terminal:

```
$wget
http://ports.ubuntu.com/pool/main/d/debootstrap/debootstrap_1.0.13~jaunty1_all.deb

$sudo dpkg -i debootstrap_1.0.13~jaunty1_all.deb
```

- **Rootstock**

Finalmente existe un script que automatiza todo el proceso de compilación é instalación de ubuntu llamado rootstock. Cabe destacar que este script depende fuertemente de las dos aplicaciones antes instaladas. Su obtención re realiza mediante los siguientes comandos:

```
$wget http://launchpad.net/project-rootstock/trunk/0.1/+download/rootstock-0.1.3.tar.gz

$tar xf rootstock-0.1.3.tar.gz

$cd rootstock-0.1.3
```

- **Uboot-mkimage**

Para adaptar la imagen del kernel que se obtenga al cargador de booteo uboot es necesario realizar determinadas operaciones a fin de que quede correctamente instalada en la partición destino. Para esto existe un programa llamado **uboot-mkimage** que automatiza dicho proceso de adaptación. Dicha aplicación se encuentra dentro de los repositorios de ubuntu y por lo tanto su instalación se realiza de la siguiente forma:

```
$sudo apt-get install uboot-mkimage
```

### 4.5.4.2 Generación de imagenes y kernel

Una vez que se dispone de las herramientas mencionadas anteriormente en la PC de desarrollo es necesario utilizarlas para generar la imagen del kernel y el sistema de archivos que necesita la plataforma para funcionar adecuadamente. Para esto el comando en su forma más amplia sería el siguiente:

```
$sudo ./rootstock --fqdn <hostname> --login <rootuser> --password <rootuserpasswd> --
imagesize <qemu image size> --seed <packages> --dist <jaunty/karmic>
 --serial <ttySx>  --kernel-image <http>
```

En nuestro caso particular el comando resultó como se muestra a continuación:

```
$sudo ./rootstock --fqdn beagleboard --login ubuntu --password temppwd --imagesize 2G
--seed xfce4,gdm --dist jaunty
 --serial ttyS2 --kernel-image http://rcn-ee.net/deb/kernel/beagle/jaunty/v2.6.29-58cf2f1-
oer44.1/linux-image-2.6.29-oer44.1_1.0jaunty_armel.deb
```

De dicho comando cabe destacar que, en primer lugar, el usuario y la contraseña establecidos fueron “ubuntu” y “temppwd” respectivamente. En segundo lugar, la imagen del sistema de archivos debe ser de 2 gigas. Y por último, los paquetes agregados a la instalación común fueron los paquetes xfce4 y gdm aunque se podrían haber considerado otros como son los paquetes para los escritorios gráficos, adaptadores de red, etc. Para más información acerca de cada uno de los parámetros del comando se pueden consultar las guías disponibles en Internet.

Al final de la ejecución del comando, en caso de ser exitosa, se debería disponer de los archivos **armel-rootfs-<date>.tgz** y **vmlinuz-2.6.<version>**

### 4.5.5 Compilación del kernel (Método 2)

Si bien el procedimiento descripto en la sección anterior ofrece sus comodidades debido a la rapidez y facilidad con que se realiza, también tiene el inconveniente de que no es siempre aplicable y que oculta gran parte de las operaciones que se realizan impidiendo de esta forma aprender sobre el funcionamiento de Linux.

Es por esto que a continuación se describe un método más general de compilación e instalación.

En este punto se recomienda ampliamente la lectura del libro “**Linux Kernel in a nutshell**” referenciado en la bibliografía de este informe. Dicho libro describe con mayor profundidad y precisión todo el funcionamiento de Linux y su instalación.

### 4.5.5.1 Adaptación de la PC huésped

Antes de comenzar el proceso de compilación es necesario instalar determinadas herramientas:

- **Compilador**

Como ya se mencionó anteriormente, Linux es un kernel implementado casi en su totalidad en el lenguaje C, con algunas instrucciones directamente escritas en lenguaje ensamblador. Por lo tanto, para compilar el kernel es necesario disponer de un compilador de C que genere un código que pueda ser directamente interpretado por el procesador de la plataforma objetivo.

El compilador de C más conocido y utilizado actualmente es **gcc**, que es mantenido y desarrollado por la comunidad GNU. Usualmente gcc viene pre-instalado en todas las distribuciones de Linux, pero si se desea obtenerlo simplemente hay que dirigirse a la página Web del proyecto http://gcc.gnu.org, descargarlo y compilarlo.

- **Linker**

Debido a la complejidad del proceso de compilación de aplicaciones hechas en C, el compilador en si no es capaz de realizar esta tarea por sí solo. Es por esto que necesita de otro grupo de herramientas, denominadas **binutils**, para hacer el enlace (link) y armado de todo el código fuente.

Este grupo de herramientas viene empaquetado en un paquete que lleva el mismo nombre y puede ser descargado de http://www.gnu.org/software/binutils, aunque usualmente cada distribución ya dispone de una versión pre-compilada de este paquete.

- **Make**

**Make** es una herramienta que recorre el árbol de archivos de código fuente del kernel, decide cuáles de ellos es necesario compilar e invoca luego al compilador y a otras herramientas.

En general todas las distribuciones vienen con una versión de make ya incorporada, pero si se desea el código fuente, el mismo está disponible en la Web http://www.gnu.org./software/make y por lo tanto puede ser descargado y compilado.

Cabe destacar que las herramientas citadas son aquellas que son ABSOLUTAMENTE indispensables para la compilación. Dependiendo del sistema objetivo es posible que sea necesario incorporar otras herramientas y es por eso que se vuelve a recomendar la lectura del libro "**Linux kernel in a nutshell**" en donde se detallan con más precisión las distintas aplicaciones que se pueden incorporar al kernel.

### 4.5.5.2 Selección de la rama del kernel

Una vez completada la instalación de las herramientas es necesario escoger qué versión del kernel se va a instalar. Actualmente existen varias ramas en desarrollo, cada cual con sus características distintivas. La Figura 4.6 ilustra el árbol correspondiente a los núcleos disponibles y en desarrollo.

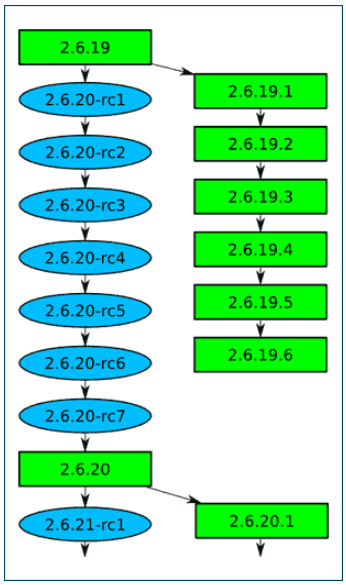


**Figura 4.6 - Núcleos disponibles**

### 4.5.5.3 Obtención del código fuente

El código fuente de todo el árbol de núcleos está disponible en la Web http://www.kernel.org. Para descargar el código de una rama en particular basta con hacer clic en la 'F' correspondiente a dicha rama.

Concluido el paso anterior, se debería tener un archivo comprimido que contiene todos los archivos de código fuente. Por lo cual, para terminar con la obtención del código se debe descomprimir dicho archivo a alguna carpeta independiente.

### 4.5.5.4 Crear y modificar una configuración

En esta etapa es donde cada desarrollador debe elegir los módulos y características que desea obtener del kernel, como son por ejemplo los drivers a ser incluidos en el mismo.

Toda la configuración del núcleo elegida se guarda en el archivo "**.config**" que se encuentra en la raíz de la estructura de directorios del kernel.

Para crear dicho archivo existen dos métodos:

- **Configuración desde cero**

Este método consiste en ir seleccionando una a una si se desea incluir determinada característica ó no. Para realizar esto es necesario pararse en el directorio raíz del código fuente y ejecutar:

```
$make config
```

- **Configuración por defecto**

Cada rama del kernel cuenta con una configuración del kernel por defecto que incluye las aplicaciones que el encargado de dicha rama considera idóneas para ejecutar dicho kernel. Por esto es que alcanza ejecutar el siguiente comando para generar un archivo de configuración con dichas opciones:

```
$make defconfig
```

Una vez generado un archivo de configuración es posible realizarle modificaciones. Para ello existen varias herramientas como son:

- **Menuconfig**

Esta herramienta se trata de una aplicación que funciona desde una consola. Para iniciarla es necesario ejecutar el siguiente comando:

```
$make menuconfig
```

- **Gconfig**

En este caso se trata de una herramienta que se ejecuta en entorno grafico basada en GTK+.

- **Xconfig**

Otra aplicación grafica pero basada en QT.

No se detallará aquí cómo se utiliza cada una de las herramientas, ni cuales módulos es conveniente instalar para la placa BB.

### 4.5.5.5 Compilación del kernel

Una vez establecido el archivo de configuración, lo único que resta hacer es compilarlo. Para eso se ejecuta el comando **make**, seguido, en nuestro caso, de una opción para establecer la arquitectura del procesador correspondiente a la placa BB.

```
$make arch=arm CROSS_COMPILE=<ruta a toolchain de arm>
```

En donde <ruta a toolchain de arm> indica la ruta hacia el directorio en donde se haya configurado el compilador de C  para arm.

Existen otras opciones útiles que se le pueden agregar al comando **make** pero que no se tratarán en este documento.

## 4.5.6 Particionado y armado de la SD

### 4.5.6.1 Particionado con gparted

La placa BB cuenta, como dispositivo de memoria, con un lector de memorias SD que es idóneo para la instalación del sistema operativo. En nuestro caso, se decidió utilizar una tarjeta SD de 2 GB por considerarla suficiente para cumplir con los requerimientos de memoria previstos y por ser de relativo bajo costo.

De cualquier forma, independientemente del tamaño de la tarjeta de memoria, es necesario particionar dicha tarjeta para que se pueda utilizar para la instalación del sistema operativo.

Para esto se utilizó la herramienta **gparted**, cuyas últimas versiones funcionan como un sistema operativo independiente contenido dentro de un live-cd. Dicho software está licenciado bajo la licencia GPL y, por lo tanto, puede ser descargado y utilizado libremente.

Una vez que se dispone del CD con la imagen de **gparted** grabada, se debe reiniciar la PC host e iniciar desde dicho CD.

Lo importante de la tabla de particiones que se debe generar con gparted es lo siguiente:

- Que exista una partición primaria del tipo **fat32 booteable** al principio de la memoria de la tarjeta de aproximadamente 50 MB.

- Que el resto de la unidad de memoria sea llenado con una partición **primaria del tipo ext3/ext2**.

A continuación se explica el proceso de creación de las particiones con los requerimientos recién referidos dentro de gparted.

1. Seleccionar la partición fat32 existente.

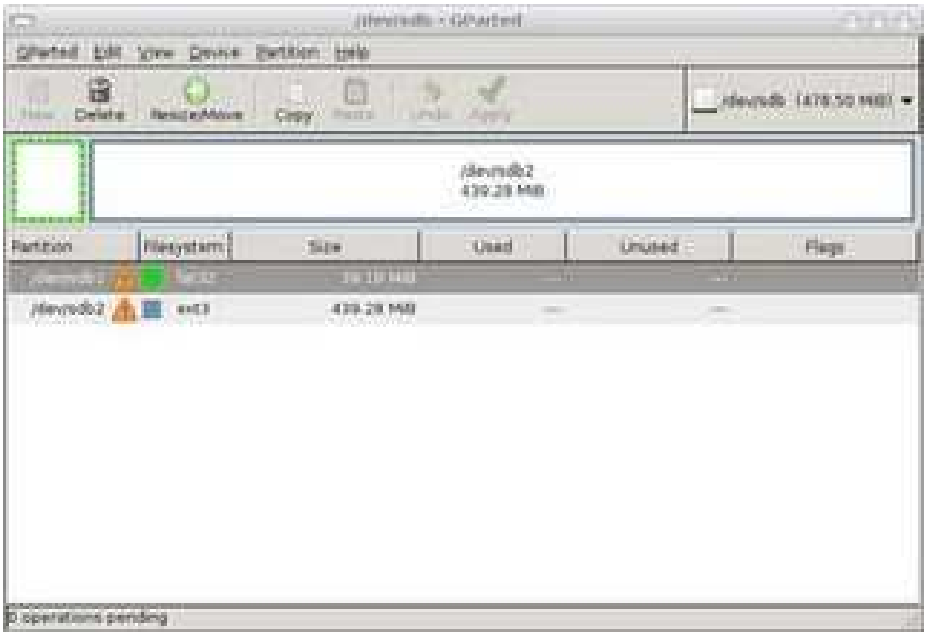


**Figura 4.7 – Partición fat32 existente**

2. Haciendo clic con el botón derecho del mouse, seleccionar la opción **Delete**, para eliminar la partición fat32 existente.

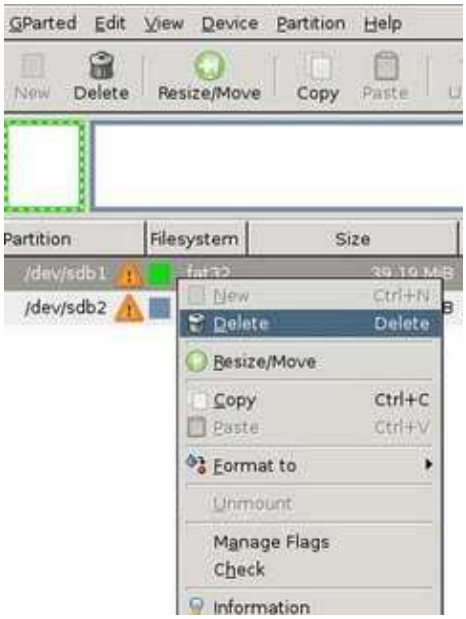


**Figura 4.8 – Eliminar partición fat32 existente**

3. Crear una nueva partición primaria de tipo fat32.

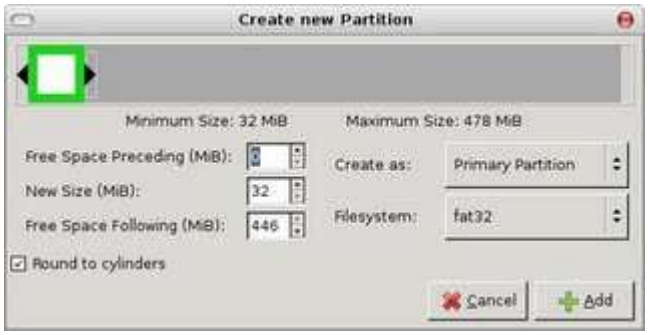


**Figura 4.9 – Creación de nueva partición fat32**

4. Crear una nueva partición primaria de tipo ext3.

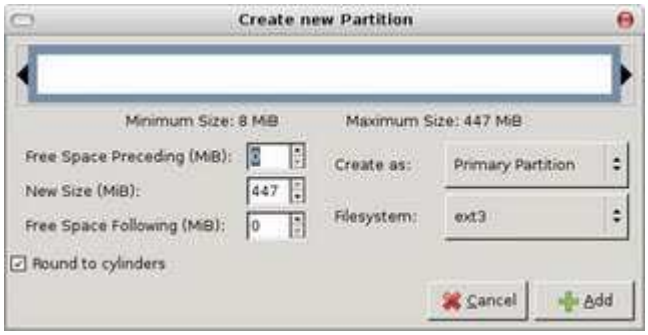


**Figura 4.10 - Creación de nueva partición ext3**

5. Hacer clic derecho sobre la nueva partición fat32 y seleccionar la opción **Manage Flags**.

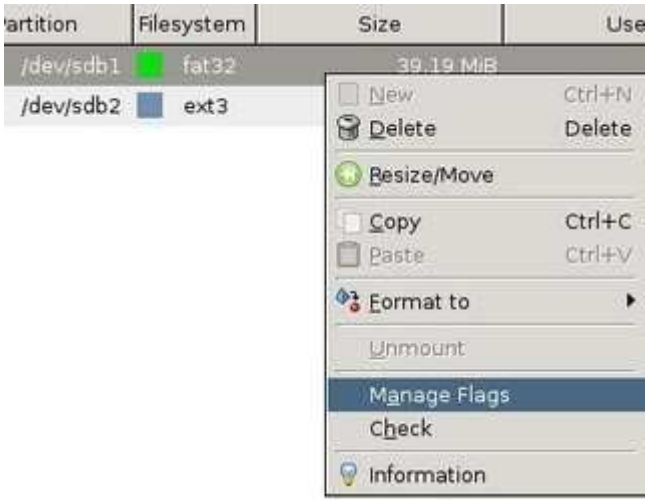


**Figura 4.11 – Opción Manage flags**

6. Chequear la casilla correspondiente a la opción **boot** para que la partición fat32 creada sea booteable.


**Figura 4.12 – Partición booteable**

### 4.5.6.2 Instalación de uboot/uImage en la partición de la SD

Terminado el particionado de la tarjeta de memoria hay que instalar la imagen del kernel dentro de la partición fat32 creada. Para esto hay que montar dicha partición en algún directorio (en nuestro caso /media/disk) y luego ejecutar los comandos que se muestran a continuación.

Dentro de la carpeta donde se crearon las imágenes del kernel:

```
$mkimage -A arm -O linux -T kernel -C none -a 0x80008000 -e 0x80008000 -n "Linux" -d
./vmlinuz-*  ./uImage
```

Luego:

```
$cd /media/disk
$sudo cp ./<directorio de las imagenes>/uImage uImage
```

### 4.5.6.3 Instalación de la partición ext2/ext3 de la SD

En la partición ext2/ext3 que se generó en el paso anterior, es necesario copiar el sistema de archivos que será montado como '/' cuando se inicie el sistema operativo en la placa BB. Dichos archivos se encuentran comprimidos dentro del archivo **armel-rootfs-<date>.tgz** que se obtuvo en la sección *Generación de imágenes y kernel.* Por lo tanto, es necesario descomprimir dicho archivo y copiar su contenido manteniendo el sistema de permisos. Para esto hay que montar la partición ext2/ext3 en algún directorio (en nuestro ejemplo se utilizó /media/disk) y ejecutar el siguiente comando:

```
$sudo tar xfp armel-rootfs-[date].tgz -C /media/disk
```

Terminado el copiado de archivos a la SD, es conveniente modificar el archivo de las interfaces de red que posee la placa, para luego poder establecer conexiones entre la placa BB y una PC. Para esto debemos montar nuevamente la partición en caso de que haya sido desmontada y ejecutar el siguiente comando (nuevamente se utilizó el directorio /media/disk como punto de montaje):

```
$sudo gedit /media/disk/etc/network/interfaces
```

Este comando abrirá un editor de texto en donde se deben ingresar las siguientes líneas:

```
auto eth0
iface eth0 inet static
address 192.168.1.254
netmask 255.255.255.0
```

## 4.5.7 Configuración de U-boot

Para culminar la instalación de Ubuntu en la placa BB, debemos configurar U-boot para que pueda descargar adecuadamente la imagen del kernel a la memoria de la placa.

En este paso es necesario disponer de la conexión serial que se explicó en la sección *Conexión Serial* de este informe. Se debe encender la placa BB con la memoria SD

insertada en el slot correspondiente e interrumpir la ejecución de U-boot presionando cualquier tecla.

Una vez interrumpida la ejecución de U-boot, OMAP ofrecerá una terminal en donde se deben ingresar los siguientes comandos:

```
setenv bootcmd 'mmcinit; fatload mmc 0:1 0x80300000 uImage; bootm 0x80300000'

setenv bootargs 'console=ttyS2,115200n8 console=tty0 root=/dev/mmcblk0p2 rootwait rootfstype=ext3 ro omapfb.mode=dvi:1280x720MR-16@60'

saveenv

boot
```

Estos comandos pueden variar según de qué revisión de la placa se disponga. Para más información se puede consultar el manual de referencia de la placa BB referenciado en la Bibliografía de este informe.

De ser exitosa la instalación, debe aparecer en la pantalla toda la secuencia de mensajes de información del booteo de Linux.

## *4.6 Instalación de aplicaciones y actualización del kernel*

### 4.6.1 Conexión de red cableada

El primer paso, luego de haber terminado la instalación del sistema operativo, es configurar una red cableada para disponer de una conexión de datos alternativa a la conexión serial. A partir de esta conexión podremos luego controlar la placa BB desde cualquier PC independientemente de que se disponga o no de una conexión serial.
Además facilitará toda la transferencia de archivos que sea necesaria para la instalación de las aplicaciones que más adelante se mencionarán.

En la sección *Instalación de la partición ext2/ext3 de la SD* se explicaron los pasos de configuración para disponer de una red Ethernet cableada dentro de la placa BB. Resta ahora configurar adecuadamente la PC host y armar la interconexión física de los equipos.

A simple vista se puede constatar que la placa BB no cuenta con ningún puerto que permita directamente la entrada de un conector RJ-45, que es el más utilizado en las redes Ethernet cableadas. Sin embargo, la placa BB sí cuenta con una entrada para dispositivos USB que se adapta perfectamente a las necesidades de las conexiones Ethernet.

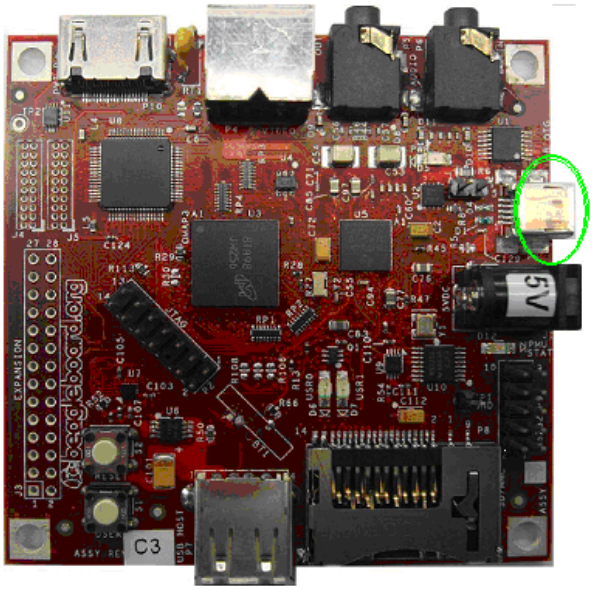


**Figura 4.13 - Puerto USB**

En primera instancia se podría pensar en conectar directamente un adaptador USB-Ethernet al puerto USB de la placa BB. Lamentablemente, la placa no cuenta con la energía suficiente como para alimentar un dispositivo USB. Es por esto que, para poder establecer una conexión de red, es indispensable contar con un HUB USB que disponga de un puerto maestro. De esta forma, se conecta el adaptador USB-Ethernet a cualquiera de las entradas del HUB y se conecta la placa BB al puerto maestro del HUB como muestra la Figura 4.14.

**Figura 4.14 - Conexión de red**

Es necesario aclarar que no todos los conectores USB que se pueden introducir en el puerto USB de la placa sirven para que ésta funcione en modo maestro, que es como debe funcionar para establecer una conexión de red. Por más información se puede consultar el manual de la placa BB referenciado en la Bibliografía de este documento.

Para terminar de armar la plataforma física de la red solo resta conectar el adaptador USB-Ethernet con la PC host mediante un cable UTP.

La configuración de la PC host requiere únicamente que se le asigne una dirección IP que esté dentro del rango de direcciones IP de la red que se configuró para la placa, y que se le asigne la misma máscara.

Finalmente se debe encender la placa BB con el HUB USB y la red conectada (la versión de Ubuntu instalada en la placa no reconoce el adaptador USB a menos que éste esté conectado en la etapa de booteo).

Para comprobar que se ha configurado todo correctamente, se puede ejecutar un ping desde la PC host hacia la placa BB y verificar que responde a dicho comando.

### 4.6.2 Conexión cableada a Internet

En el momento en que se culmina la instalación de una red cableada entre una PC y la placa surge la tentación de conectar la placa BB directamente a Internet. Dicha conexión abriría la posibilidad de descargar e instalar los paquetes que sean necesarios en las secciones siguientes, directamente desde los servidores de repositorios de Ubuntu, sin necesidad de ningún intermediario.

A continuación se detallan los pasos para establecer una conexión a Internet, en una red donde existe un router como intermediario entre la red e Internet mediante un servicio ADSL.

En base a la plataforma física de la red armada en la sección anterior, como puede verse, lo único que hay que alterar es que, en vez de conectar el adaptador USB-Ethernet a una PC hay que conectarlo al router.

En cuanto a la configuración establecida para las interfaces de red de la placa BB, resta únicamente configurar el GATEWAY de la red y los servidores DNS (esto siempre y cuando la dirección IP y la máscara de la red establecidas anteriormente sean compatibles con las del router). Para esto hay que iniciar nuevamente una conexión serial para acceder a una terminal en la placa.

Cuando se disponga de la terminal, hay que comenzar estableciendo el gateway de la red con el siguiente comando:

```
$sudo route add default gw <ip del router>
```

De esta forma se le indica al sistema operativo de la placa que cualquier paquete de red que desee enviar a una dirección IP fuera de su área lo debe enviar al router.

Los servidores DNS son necesarios para que el sistema operativo de la placa BB pueda asociar el nombre de los servidores de Internet con su dirección IP. En Ubuntu, la configuración de los servidores DNS se guarda en el archivo /etc/resolv.conf, lo que causa que la configuración de los mismos se resuma al siguiente comando que debe ser ejecutado como root:

```
#echo nameserver <ip del dns> > /etc/resolv.conf
```

Para obtener la dirección IP del servidor DNS, cada usuario deberá consultar con su ISP (Internet Service Provider) particular.

De aquí en más se asumirá que la placa cuenta con una conexión directa a Internet para descargar los paquetes directamente de los servidores de repositorios.

En caso de que no se disponga de esta conexión, se deberán descargar los paquetes de instalación (*.deb) manualmente, guardarlos en la memoria de la placa (mediante un servidor SSH o copiándolos en la partición ext3 de la tarjeta SD) y luego instalarlos ejecutando el siguiente comando:

```
$sudo dpkg -i nombredelpaquete.deb
```

### 4.6.3 Servidor SSH

Una aplicación que simplifica enormemente la ejecución de comandos y la manipulación y la transferencia de archivos en la placa BB, es un servidor SSH. Mediante este servidor podremos disponer de una terminal dentro de la placa y transmitir archivos desde y hacia la placa BB desde cualquier PC que tenga una conexión de red y un cliente SSH. De esta forma, se evitan las limitaciones asociadas con contar únicamente con una conexión serial.

Para instalarlo, simplemente hay que establecer una conexión serial de la forma que se explicó anteriormente e iniciar sesión en la placa BB utilizando el usuario y la contraseña establecidos.

Una vez que se dispone de una terminal a través de la conexión serial, hay que ejecutar los siguientes comandos:

```
$sudo apt-get install openssh-server openssh-client
```

Desde la PC host se puede comprobar el correcto funcionamiento del servidor ejecutando los siguientes comandos a fin de obtener una terminal dentro del sistema operativo de la placa:

```
$ssh <usuario>@<ip de la placa>
```

Si todo funcionó correctamente, se solicitará el ingreso de la contraseña del usuario y, una vez comprobada la identidad del mismo, se le otorgará una terminal para ejecutar comandos de la misma forma que si los estuviera escribiendo con un teclado conectado directamente a la placa.

A partir de ahora se asumirá que se dispone de un servidor SSH instalado y funcionando en la placa BB.

### 4.6.4 Actualización del kernel

Antes de comenzar a trabajar con el sistema operativo, es conveniente actualizar el kernel a la última versión disponible, principalmente para disponer de los últimos drivers para los distintos componentes.

En Ubuntu existe un script que automatiza todo el proceso de obtener el último kernel compilado para ARM, generar las imágenes y actualizar la configuración de U-boot.

El primer paso a llevar a cabo es descargar este script en la PC host. Esto se realiza siguiendo el link http://www.rcn-ee.net/deb/kernel/ubuntu-update-kernel.sh

Una vez que se haya descargado el script, hay que pasarlo a la placa BB. Para esto se utilizará el servidor SSH instalado en la sección anterior. En Ubuntu, el comando para iniciar una transmisión mediante SSH es el siguiente:

```
scp <path de local del script> <usuario>@<ip de la BB>:<path remoto para el script>
```

Por último hay que abrir una terminal en la placa BB y ejecutar el comando como root. Se preguntará si se desea actualizar el link simbólico de vmlinuz, lo cual es necesario aceptar.

El script informará el resultado de la operación. Para que los cambios sean aplicados, se debe reiniciar el sistema operativo de la placa.

### 4.6.5 Servidor Apache

En el alcance del proyecto se especificó la necesidad de disponer de una interfaz web para administrar y manipular el móvil. Esto requiere que se instale alguna aplicación que permita alojar dicha interfaz en la placa BB.

En este proyecto se optó por implementar una interfaz en base al lenguaje PHP por los siguientes motivos:

- Es relativamente fácil de utilizar en comparación con los otros lenguajes.
- Tiene una gran capacidad para interactuar con el sistema operativo.
- No requiere demasiados recursos. En particular el consumo de energía adicional que requiere su utilización no es significativo.

Teniendo en cuenta esta restricción sobre el lenguaje de programación a utilizar, se decidió utilizar el servidor Apache con su módulo para PHP para sustentar la interfaz.

Dicho servidor se encuentra en los repositorios conjuntamente con su conector para PHP. Es por esto que, para instalarlo, basta con ejecutar el siguiente comando en una terminal de la placa BB:

```
sudo apt-get install apache2 php5 libapache2-mod-php5 php5-gd php5-mysql
```

### 4.6.6 DNS Dinámico

Las características del proyecto UMix hacen que sea deseable conocer en todo momento la dirección IP que le ha sido asignada. De esta forma es posible acceder a la interfaz Web para administrar y manipular el móvil.

Pensando en no limitar el proyecto a un servicio de IP fija, se optó por incorporar al proyecto una aplicación que permitiera aprovechar los servicios que ofrecen los proveedores de DNS dinámicos.

En este proyecto se decidió utilizar los servicios de no-ip por ser una de las empresas líderes proveyendo este tipo de servicios. No se hará en este informe un seguimiento sobre cómo registrar una cuenta y un dominio en no-ip, simplemente se detallarán los pasos a seguir para instalar el cliente y configurarlo en Ubuntu.

El cliente de no-ip también se encuentra dentro de los repositorios de Ubuntu, bajo el nombre de noip2. Por lo tanto, su instalación consiste en ejecutar el siguiente comando en una terminal de la placa:

```
$sudo apt-get install noip2
```

Aparecerá una pantalla en la cual se piden los datos que se configuraron en la cuenta de no-ip. Finalizado esto, la aplicación hará todo el trabajo de actualización de la dirección IP automáticamente.

## *4.7 Diseño y construcción del móvil*

En esta sección se describe la tarea de diseño y construcción del móvil, incluyendo también los circuitos anexos necesarios para controlar los motores.

Se decidió que el móvil utilizara para su desplazamiento dos motores: un motor de continua para la tracción (ruedas traseras) y un motor paso a paso (PaP) para controlar la dirección (ruedas delanteras).

Para mantener bajos los gastos del proyecto se decidió reciclar todo lo que fuera posible. Con este propósito se utiliza el motor de continua de la bandeja de una compactera y el motor PaP bipolar que controla la posición de la cabeza lectora de un disco duro. Además, se reciclaron el chasis y las ruedas de autos de juguete.

Para controlar el motor de continua se utiliza uno de los puentes del driver L293D. Se conectaron 3 pines del puerto B del PIC: RB0, RB1 y RB2 a los pines Enable1, In1 e In2 del L293D respectivamente. Los pines Out1 y Out2 del L293D se conectaron a los terminales del motor de continua.

De esta forma, cuando RB0 está en 1, el puente 1 del L293D está habilitado. En caso contrario, dicho puente queda deshabilitado por lo que el motor queda libre. Estando el puente 1 del L293D habilitado: si RB1 = 1 y RB2 = 0 el motor gira en un sentido (hacia adelante), si RB1 = 0 y RB2 = 1 el motor gira en sentido contrario (hacia atrás) y si RB1 = RB2 el motor es detenido.

Para identificar los terminales del motor PaP se utilizó un tester: entre dos de los cables del motor se midió la misma resistencia que entre los otros dos y al medir en forma cruzada no se detectó continuidad. Es decir que el primer par de cables está conectado a los bornes de una de las bobinas y el otro par de cables está conectado a los bornes de la otra bobina.

Como se mencionó en el capítulo Marco Teórico del Proyecto Final de Carrera, para controlar los motores PaP es necesario invertir las polaridades de los terminales de las bobinas en una determinada secuencia para lograr un giro en un sentido y en secuencia opuesta para que el motor gire en el otro sentido. Con este propósito se utilizaron los dos puentes de un driver L293D.

Se conectó el pin RB3 del PIC a los pines Enable1 y Enable2 del L293D. Con el bit RB3 se habilitan/deshabilitan los puentes del L293D. Se conectó el pin RB4 del PIC al pin In1 del L293D, el pin RB5 del PIC al pin In2 del L293D, el pin RB6 del PIC al pin In3 del L293D y el pin RB7 del PIC al pin In4 del L293D. Se conectaron los pines Out1 y Out2 a los bornes de una de las bobinas y los pines Out3 y Out4 a los bornes de la otra bobina.

La cantidad de pasos que es necesario que el motor realice para que el móvil doble hacia la derecha o hacia la izquierda se determinó de forma práctica. Del mismo modo, se determinó a través de pruebas la duración necesaria de los pulsos de control de entrada al L293D, teniendo en cuenta las especificaciones de este integrado.

En la Figura 4.15 se muestra un esquema con las conexiones comentadas anteriormente.

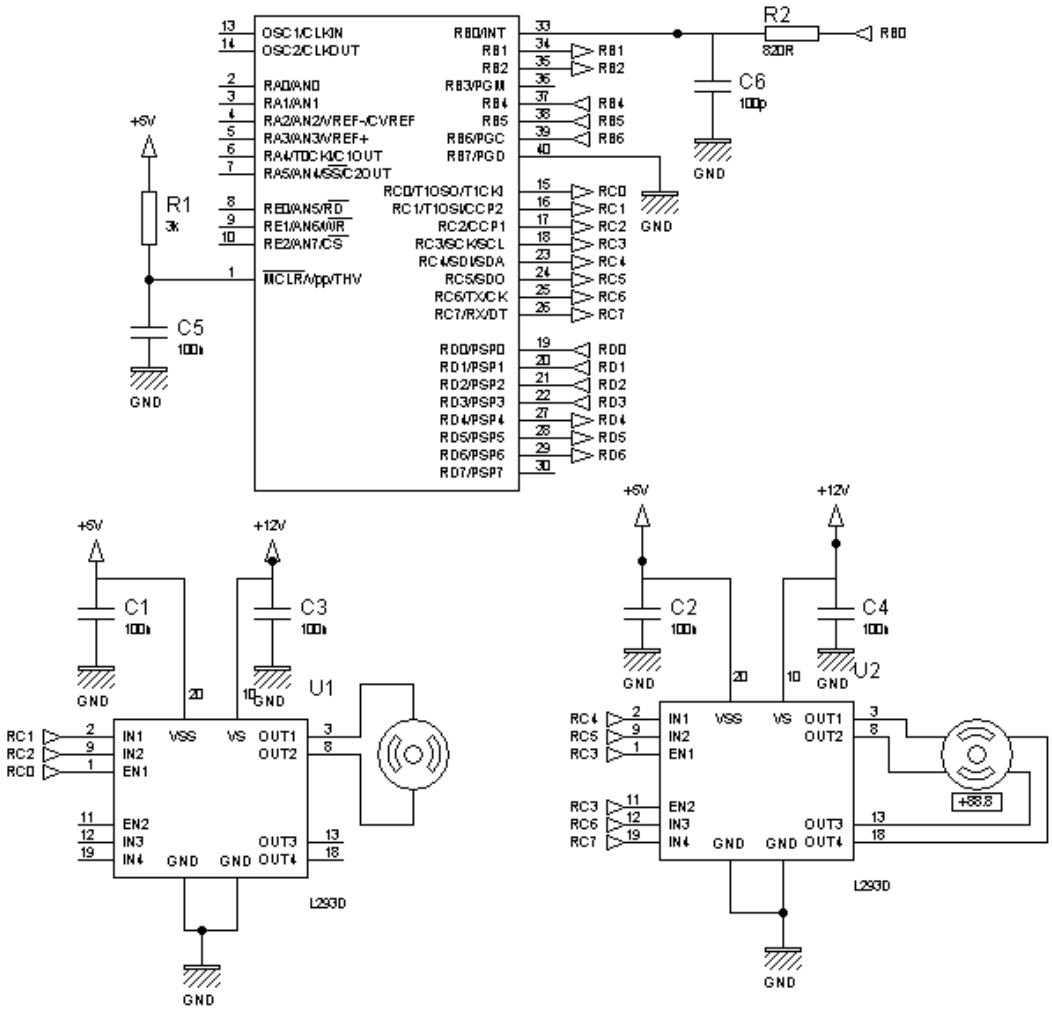


**Figura 4.15 - Esquema PIC y drivers**

## *4.8 Implementación del Módulo de Movimiento*

### 4.8.1 Estudio del módulo PWM

En un principio se consideró que era necesario utilizar el módulo PWM del PIC para el control de los motores. Luego de estudiar el funcionamiento de dicho módulo y sus aplicaciones se llegó a la conclusión de que la principal aplicación del módulo PWM en el control de motores es el control de velocidad de motores de continua. En nuestro caso no era necesario este control.

Por otro lado, se detectó la oportunidad de utilizar el módulo PWM para generar las señales de control que actuarían como entradas para el driver del motor PaP. Sin embargo, debido al desplazamiento de medio pulso requerido para estas señales de control, se consideró más adecuado generar los trenes de pulsos necesarios utilizando un bucle que escriba ceros y unos en cuatro bits de uno de los puertos del PIC. Para que estos bits se mantuvieran en el nivel correspondiente el tiempo que fuera necesario, se incluyeron llamadas a una rutina de espera entre cada par de instrucciones de escritura en el puerto.

En resumen, aunque se consideró al planificar el proyecto que sería necesario utilizar el módulo PWM del PIC para controlar los motores, llegado el momento se decidió que éste no sería utilizado.

### 4.8.2 Desarrollo del código

Para desarrollar el código del módulo de movimiento para el PIC se utilizó el lenguaje assembler. Fue fundamental para el desarrollo de este código la experiencia obtenida en asignaturas de la carrera tales como Sistemas Digitales y Laboratorio de Telemática, en las cuales se realizaron proyectos utilizando el microcontrolador PIC.

Además, se utilizaron como libros de cabecera el libro de referencia de la familia PICmicro de rango medio y la hoja de datos del PIC, teniéndose en cuenta todas las buenas prácticas y recomendaciones de diseño que en dichos documentos se mencionan.

El main del programa lo que hace es ejecutar una rutina de inicialización, en la cual pone en cero todas las variables, configura los bits de los puertos como entradas o como salidas según corresponda, habilita y configura las interrupciones que se van a utilizar y por último pone en cero todos los puertos.

Luego ejecuta la instrucción sleep, es decir que pasa al modo SLEEP. Al ser interrumpido, el PIC se despierta del modo SLEEP, deshabilita las interrupciones y chequea cuál fue la fuente de interrupción. Según cuál haya sido la fuente de interrupción, setea determinado flag en una variable prevista para este propósito. Luego vuelve a habilitar las interrupciones y regresa al flujo de ejecución normal del programa.

Una vez en el flujo de ejecución normal, chequea cuál de los flags fue seteado y llama a la subrutina adecuada (SubrutinaAdelante, SubrutinaAtras, SubrutinaDerecha, SubrutinaIzquierda, SubrutinaObstDer, SubrutinaObstIzq o SubrutinaSonido). Las sub rutinas asociadas a directivas ejecutan las instrucciones necesarias para mover los motores. Las rutinas asociadas a detección de obstáculos o sonido ejecutan las instrucciones necesarias para notificar a la BB de estos eventos.

El código desarrollado se encuentra adjunto en el Anexo III – Código del PIC.

### 4.8.3 Testeo del módulo de movimiento

Como primer paso se verificó el funcionamiento del motor de continua conectándolo directamente a una fuente y se identificaron las polaridades de sus cables. Por otro lado, Se identificaron las bobinas del motor paso a paso midiendo la impedancia entre sus cables.

Luego se verificó el funcionamiento del driver L293D conectado al motor de continua. Se verificó que al poner un “1” lógico en las entradas En1 e In1 y un “0” lógico en la entrada In2 del L293D, el vehículo se movió hacia adelante. También se verificó que al poner un “1” lógico en las entradas En1 e In2 y un “0” lógico en la entrada In1 del L293D, el vehículo se movió hacia atrás.

Para depurar el código del PIC se utilizaron la herramienta de debug incluida en el entorno de desarrollo MPLAB y la herramienta de debug que ofrece el simulador Proteus.

Ambas herramientas permiten ejecutar el programa paso a paso, viendo el estado de los distintos registros del PIC. Esto facilita la detección de comportamientos distintos al esperado y permite corregir errores.

Luego de depurado el código se grabó en el PIC y se verificó su funcionamiento simulando las entradas con pulsadores y observando las salidas con el osciloscopio.

Una vez verificado el funcionamiento del driver L293 conectado al motor de continua y depurado el código del PIC, se realizaron las conexiones entre dicho driver y el PIC y se generó una interrupción simulando una directiva “Adelante” enviada por la BB y el vehículo se movió hacia adelante.

Durante esta prueba se experimentaron comportamientos extraños por partes del PIC. Luego de investigar las posibles causas y realizar diversas pruebas, se llegó a la conclusión que la causa era el ruido eléctrico que ingresaba al PIC por las entradas flotantes (aquellas que no estaban conectadas a nada). Se solucionaron estos problemas conectando a tierra aquellas entradas que no se utilizarían.

También se experimentaron problemas con las rutinas de espera. Se pudo observar que la ejecución nunca salía de estas rutinas. Se investigó el problema y se descubrió que éste radicaba en un error al configurar el oscilador utilizado por el PIC. Se configuraba XT en lugar de HS. Se solucionó el problema corrigiendo los bits correspondientes en el registro de configuración.

Luego se verificó que el PIC reaccionara correctamente al ser interrumpido por un sensor de obstáculos. Es decir, que al recibir una interrupción en uno de los pines RB4 o RB5, pone un “1” lógico durante un breve período en uno de los bits RD4 o RD5 según corresponda. También se verificó que al tener seteado uno de los bits RD0 o RD1 (simulando una directiva “Adelante” o “Atrás” de la BB) y recibir una interrupción en el pin INT (RB0), el motor de continua se mueve hacia adelante o hacia atrás respectivamente, lo cual representa el correcto funcionamiento de esta sección.

Durante estas pruebas se experimentó un problema: se lograba el correcto funcionamiento del motor de continua controlado por el PIC, solo si los sensores de obstáculos no estaban conectados a los flip-flops. Se analizaron las diferencias entre la situación en la que se lograba el correcto funcionamiento y la situación en la que no y se

concluyó que el contacto NC de los microswitches no debía estar conectado directamente a tierra, sino que debía conectarse a través de una resistencia pull-down.

Al testear el funcionamiento de la lógica de control del motor PaP, no se lograron los resultados esperados por lo que fue necesario ajustar la duración de los pulsos enviados por el PIC al driver L293D.

## *4.9 Implementación Módulo Comunicación PIC - Placa Desarrollo*

La información que era necesario transmitir entre el PIC y la placa BB eran bits independientes: para directivas de movimiento (Adelante, Atrás, Derecha, Izquierda) desde la placa hacia el PIC y para avisos de condiciones positivas en los sensores (de obstáculos y de sonido) desde el PIC hacia la placa.

### 4.9.1 Investigación de los protocolos del puerto de expansión

Como se mencionó en el capítulo Marco Teórico, el puerto de expansión de la placa BB puede manejar los protocolos: General Port Input Output (GPIO), I2C y SPI. Luego de investigar estos protocolos, se decidió utilizar para la comunicación con el PIC el protocolo GPIO (General Purpose Input/Output) por considerarse el más adecuado para esta aplicación.

### 4.9.2 Desarrollo protocolo de comunicación

#### 4.9.2.1 A nivel del PIC

Se utilizan los puertos B y D para esta función. Se utiliza la interrupción externa del bit RB0 para advertir al PIC que se le enviará una directiva y los bits RD0:RD3 para enviar las directivas en sí. Para recibir las condiciones positivas de los sensores se utilizan los bits RB4:RB6 y para enviarle a la placa notificaciones de estas condiciones positivas se utilizan los bits RD4:RD6.

A modo de resumen la  muestra la función de cada bit utilizado para la comunicación con la placa BB.

| Pin del PIC | Función |
|---|---|
| RB0 | Entrada. Genera interrupción externa para advertir que se va a transmitir una directiva de movimiento |
| RD0 | Entrada. Directiva “Adelante” |
| RD1 | Entrada. Directiva “Atrás” |
| RD2 | Entrada. Directiva “Derecha” |
| RD3 | Entrada. Directiva “Izquierda” |
| RB4 | Entrada. Sensor de obstáculo Derecho |

| | |
|---|---|
| RB5 | Entrada. Sensor de obstáculo Izquierdo |
| RB6 | Entrada. Sensor infrarrojo |
| RD4 | Salida. Notificación sensor de obstáculo Derecho |
| RD5 | Salida. Notificación sensor de obstáculo Izquierdo |
| RD6 | Salida. Notificación sensor infrarrojo |

**Tabla 4-1 - Resumen de funciones de pines del PIC**

### 4.9.2.2 A nivel de la placa BB

Para esta función se utiliza el protocolo GPIO en el puerto de expansión de la placa BB.

Para definir los pines del puerto de expansión que se utilizarían para la comunicación con el PIC se tuvo en cuenta la Tabla 4-2 presentada en el manual de referencia de la placa BB.

| Signal | Description | I/O | Pin |
|---|---|---|---|
| General Purpose I/O Pins | | | |
| GPIO_130 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 21 |
| GPIO_131 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 19 |
| GPIO_132 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 17 |
| GPIO_133 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 15 |
| GPIO_134 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 13 |
| GPIO_135 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 11 |
| GPIO_136 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 9 |
| GPIO_137 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 7 |
| GPIO_138 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 5 |
| GPIO_139 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 3 |
| GPIO_140 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 4 |
| GPIO_141 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 10 |
| GPIO_142 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 6 |
| GPIO_143 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 8 |
| GPIO_156 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 20 |
| GPIO_158 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 12 |
| GPIO_159 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 18 |
| GPIO_161 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 16 |
| GPIO_162 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 14 |
| GPIO_168 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 24 |
| GPIO_183 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 23 |
| GPIO_144 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 4 |
| GPIO_146 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 6 |
| GPIO_145 | General Purpose Input/Output pin. Can be used as an interrupt pin. | I/O | 10 |

**Tabla 4-2 - Pines GPIO**

Se utilizaron como salidas los pines 17, 18, 19, 20 y 21 correspondientes a las señales GPIO_132, GPIO_159, GPIO_131, GPIO_156 y GPIO_130 respectivamente. El pin 17 se utiliza para interrumpir al PIC, el pin 18 se utiliza para enviar la directiva “Adelante”, el pin

19 se utiliza para enviar la directiva “Atrás”, el pin 20 se utiliza para enviar la directiva “Derecha” y el pin 21 se utiliza para enviar la directiva “Izquierda”.

Como entradas se utilizaron los pines 14, 15 y 16 correspondientes a las señales GPIO_162, GPIO_133 y GPIO_161 respectivamente. El pin 14 se utiliza para recibir la notificación del sensor de obstáculos derecho, el pin 15 se utiliza para recibir la notificación del sensor de obstáculos izquierdo y el pin 16 se utiliza para recibir la notificación del sensor infrarrojo.

La Tabla 4-3 resume la utilización de los pines del puerto de expansión de la placa BB.

| Pin de la BB | Señal | Función |
|---|---|---|
| 14 | GPIO_162 | Entrada. Notificación de sensor de obstáculos derecho. |
| 15 | GPIO_133 | Entrada. Notificación de sensor de obstáculos izquierdo. |
| 16 | GPIO_161 | Entrada. Notificación de sensor infrarrojo. |
| 17 | GPIO_132 | Salida. Interrupción al PIC. |
| 18 | GPIO_159 | Salida. Directiva “Adelante” |
| 19 | GPIO_131 | Salida. Directiva “Atrás” |
| 20 | GPIO_156 | Salida. Directiva “Derecha” |
| 21 | GPIO_130 | Salida. Directiva “Izquierda” |

**Tabla 4-3 - Resumen de funciones de pines de la BB**

### 4.9.3 Integración con otros módulos

Cuando el PIC es interrumpido por un nivel alto en el pin INT (RB0), se despierta del modo SLEEP y la ejecución salta a la rutina de atención de interrupción, en la cual se verifica cuál de los bits RD0:RD3 está en estado alto. Dependiendo de cuál sea este bit, se setea un flag en la variable TipoInt y se regresa de la interrupción. Una vez en la ejecución normal, se llamará a la subrutina correspondiente (SubrutinaAdelante, SubrutinaAtras, SubrutinaDerecha, SubrutinaIzquierda), la cual hará moverse los motores.

Las condiciones positivas en los sensores también generan interrupciones, pero éstas son del tipo RB Port Change. Al detectarse un obstáculo o un sonido se genera una interrupción del tipo RB Port Change, la cual despierta al PIC del modo SLEEP. De forma análoga a la comentada en el párrafo anterior, se examina cuál fue el bit que generó la interrupción y se setea el flag correspondiente en la variable TipoInt. Luego se regresa de la interrupción y se llama a la subrutina correspondiente que se encargará de enviar un

"1" lógico a la placa BB en el bit que corresponda (RD4:RD6). A continuación se ejecuta una rutina de espera para dar tiempo a la placa a leer este dato y luego se pone el bit en cero.

### 4.9.4 Testeo protocolo comunicación

#### 4.9.4.1 A nivel del PIC

Para testear el protocolo de comunicación a nivel del PIC, en principio se simularon las entradas provenientes de la placa BB con pulsadores y las salidas se conectaron a LEDs. De esta forma fue posible generar interrupciones sin que la placa BB estuviera conectada al PIC y ver si el programa del PIC reaccionaba de forma adecuada.

#### 4.9.4.2 A nivel de la placa BB

A nivel de la placa BB, se testeó el protocolo de comunicación colocando LEDs en los pines configurados como salidas y encendiéndolos o apagándolos.

Para testear las entradas se hicieron pruebas directamente conectando el PIC con la placa BB y recibiendo niveles altos o bajos enviados por el PIC.

#### 4.9.4.3 PIC y placa BB

Una vez testeado el protocolo de comunicación desde ambos extremos (PIC y placa BB), se realizó la conexión entre el puerto de expansión de la placa BB y los puertos B y D del PIC y se testeó la comunicación.

Surgieron complicaciones debido a que la placa BB trabaja con una lógica de 1.8 V (es decir que entrega 1.8 V como un "1" lógico), mientras que el PIC trabaja con una lógica de 5 V, con un umbral de 2 V. Por este motivo, al enviar la placa BB un "1" lógico, no es detectado por el PIC.

Se investigaron otros proyectos que integraban la placa BB con PICs intentando buscar la solución a este problema. Sin embargo, el equipo se encontró con la dificultad de que los circuitos integrados utilizados en otros proyectos no estaban disponibles en el mercado local. Por este motivo fue necesario buscar otra solución.

Se decidió utilizar transistores como interruptores, cuyo funcionamiento se explica en el capítulo Marco Teórico.

La Figura 4.16 muestra un esquema de la conexión utilizada para los transistores.

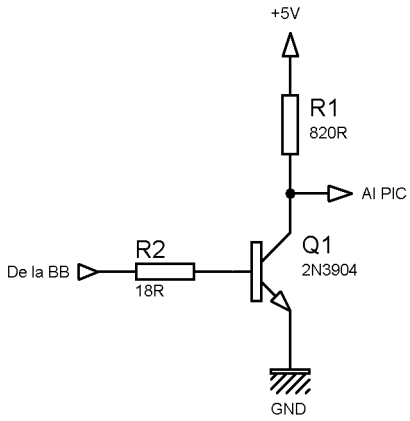


**Figura 4.16 - Transistor como interruptor**

En esta configuración, cuando se recibe un “1” lógico de la placa BB (1.8 V), el transistor se satura, por lo que la corriente en el colector es máxima. Es decir que a la salida que va al PIC se obtienen 0 V.

Cuando se recibe un “0” lógico de la placa BB (0 V), el transistor está en corte. La corriente en la base es cero, por lo que la corriente en el colector también es cero. Por este motivo, no hay caída de voltaje en la resistencia R1, por lo que la salida que va al PIC entrega 5 V.

Como se puede observar, esta configuración es inversora: cuando se obtiene un “1” lógico de la placa BB, se envía un “0” lógico al PIC y viceversa. Por lo tanto, fue necesario agregar una compuerta NOT entre la salida del transistor y el PIC para no invertir la lógica.

## *4.10 Implementación Módulo de Detección de Sobrevivientes*

### 4.10.1 Estudio de integración de simulación de sensor de vida

Se decidió que, para mantener bajos los costos del proyecto y para no perder el foco en el objetivo del proyecto, el módulo de detección de vida sería simulado por un sensor de sonido.

La idea es que el sensor de sonido entregue al PIC un “1” lógico al detectar un tono de una frecuencia determinada. Este “1” lógico entregado por el sensor de sonido genera un interrupción del tipo RB Port Change haciendo que el PIC despierte del modo SLEEP.

Para lograr este objetivo se diseñó un sensor de sonido compuesto por un micrófono, un amplificador operacional, un detector de tono, resistencias, capacitores y un potenciómetro.

El micrófono capta los sonidos del ambiente y entrega una señal de voltaje. Mediante pruebas prácticas se analizó la señal que entrega el micrófono al reproducir a través de un auricular un tono de 1 kHz. Se pudo observar una sinusoide aproximadamente centrada en 0 V de aproximadamente 35 mV pico a pico con frecuencia 1 kHz.

El menor voltaje de entrada que puede captar el detector de tono es de 20 mV rms. Por este motivo, es necesaria una pequeña amplificación para entregar la salida del micrófono al detector de tono. Con este propósito se implementó una etapa de amplificación utilizando un amplificador operacional.

La señal que se obtiene a la salida del amplificador es entregada al detector de tono, el cual activará un interruptor controlado por voltaje cuando detecte una señal de la frecuencia configurada. Esto hará que el PIC reciba un “1” lógico en la entrada correspondiente al sensor de sonido (RB6).

Inicialmente se utilizó como opamp el conocido LM741. Luego de múltiples pruebas e intentos de lograr el correcto funcionamiento de la etapa de amplificación, se concluyó que el LM741 no era adecuado para la aplicación del proyecto. Por este motivo se procedió a investigar sobre opamps más adecuados a aplicaciones de audio y se optó por el TL070.

Como detector de tono se utilizará el integrado LM567, el cual al detectar una entrada cuya frecuencia está en la banda pasante y cuyo voltaje supera cierto umbral, entrega un "0" lógico en su salida.

Teniendo en cuenta las recomendaciones que aparecen en las hojas de datos de los componentes a utilizar, se diseñó el circuito del sensor de sonido que se muestra en la Figura 4.17.

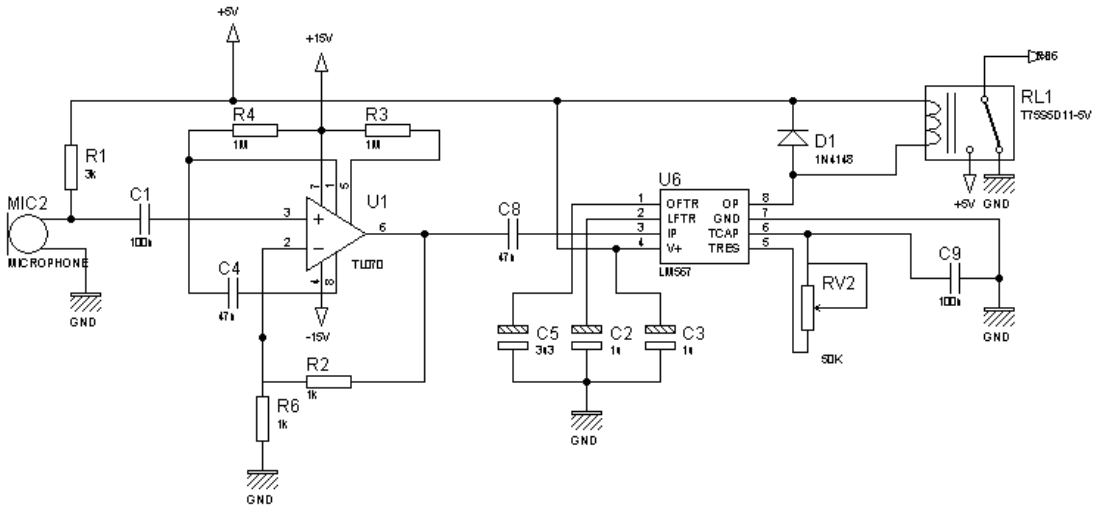


**Figura 4.17 - Circuito sensor de sonido**

## 4.10.2 Desarrollo de código de sensor de vida

El código del PIC asociado al sensor de vida, cumple con la función de enviar una notificación a la placa BB cuando sea detectado un sonido cuya frecuencia sea igual a la pre-establecida.

Al generarse una interrupción debido a un cambio en el pin RB6, el PIC se despierta del modo SLEEP y ejecuta la rutina de atención de interrupción. Al verificar que fue el sensor de sonido el que generó la interrupción, pone en "1" el bit 6 de la variable TipoInt y regresa de la interrupción.

Una vez de regreso a la ejecución normal, llama a la subrutina correspondiente (SubrutinaSonido), la cual pone en "1" el pin RD6 para notificar a la placa BB de la

detección del tono. Luego, ejecuta una rutina de espera para dar tiempo a la placa BB a que lea la notificación y por último, pone en “0” el pin RB6 y vuelve a ejecutar la rutina Main.

### 4.10.3 Testeo sensor de simulación de vida

Inicialmente se armó el circuito diseñado y se testeó su funcionamiento, pero no se lograron los resultados esperados. Por este motivo se decidió testear el circuito por partes.

En principio se armó la etapa del micrófono y se observó la señal de salida al reproducir a través de un auricular un tono de 1 kHz. Se pudo observar una sinusoide aproximadamente centrada en 0 V, de 35 mV pico a pico aproximadamente, con frecuencia 1 kHz.

Conociendo la forma de la onda, se realizaron pruebas en la etapa de detección generando la señal de entrada con un generador de onda. Observando con el osciloscopio la señal en el pin 5 del LM567 se ajustó la frecuencia central a 1 kHz. Luego se verificó que al ingresar por el pin 3 del LM567 una sinusoide de 1 kHz, el integrado entregó un nivel bajo en su salida (pin 8).

Una vez implementada y testeada la etapa de amplificación, se procedió a integrar esta etapa con la etapa de detección de tono. Se utilizó como entrada al opamp una sinusoide de frecuencia 1 kHz generada por el generador de onda. Luego de amplificar esta sinusoide se la utilizó como entrada al LM567 y se verificó el correcto funcionamiento de la etapa de detección de tono. Es decir, mientras la sinusoide generada por el generador de onda sea de 1 kHz de frecuencia, el LM567 proporciona un nivel bajo a su salida (verificado utilizando un LED). Si la frecuencia de la sinusoide es modificada fuera del ancho de banda configurado para el detector de tono, el LED se apaga.

Luego de verificar el correcto funcionamiento de las tres etapas por separado (micrófono, amplificador, detector de tono) se procedió a integrar las tres etapas. No se logró el correcto funcionamiento de las tres etapas juntas, ya que al conectar la salida del micrófono al opamp se pudo observar que la señal se contamina con ruido, por lo que no es detectada por el LM567.

Para solucionar este problema se soldó la etapa del micrófono a una placa universal de pertinax. Se pretendía que el ruido fuera menor pero no se logró este resultado. Como segunda solución se decidió implementar un filtro activo para filtrar la salida del micrófono.

Utilizando la aplicación FilterPro de Texas Instrument se diseñó un filtro activo pasa banda de primer orden, centrado en 1 kHz, factor de calidad 10 y ganancia 2, obteniéndose el circuito que se muestra en la Figura 4.18.

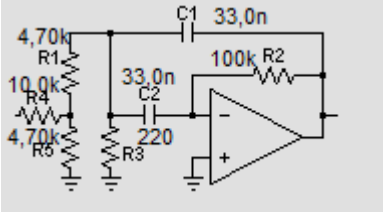


**Figura 4.18 - Filtro activo de primer orden**

En la Figura 4.19 se muestra la respuesta en frecuencia (amplitud en verde; fase en rojo) del filtro diseñado.

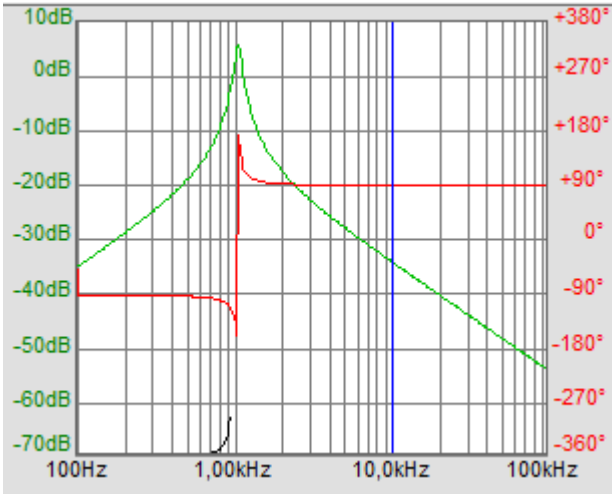


**Figura 4.19 - Respuesta en frecuencia filtro activo**

Luego de implementar el filtro anterior e integrarlo al circuito del sensor de sonido se realizaron pruebas de funcionamiento, no obteniéndose los resultados deseados. Por este motivo se decidió simular el sensor de vida con un sensor infrarrojo (IR) en lugar de simularlo con un sensor de sonido.

### 4.10.4 Sensor infrarrojo

Luego de múltiples pruebas y modificaciones sobre el sensor de sonido sin lograr el funcionamiento esperado, se evaluó la posibilidad de simular el sensor de vida de otra forma. Se tuvo en cuenta que el ambiente en el que habría de trabajar el sensor de sonido era muy ruidoso (debido a la presencia de los motores).

Se decidió que lo más adecuado a la aplicación era simular el sensor de vida con un sensor infrarrojo (IR). Éste activa una salida al detectar una luz IR de una frecuencia determinada.

Para lograr dicho objetivo, solo es necesario modificar en el sensor de sonido que se había diseñado, el micrófono y los valores de algunas resistencias y condensadores. El micrófono fue sustituido por un receptor IR. El condensador a la salida del receptor IR se cambió por un condensador de 1 µF. Se decidió volver a utilizar como amplificador operacional el LM741 debido a que el TL070 se quemó durante el testeo del sensor. Se utilizó una resistencia de 100 kΩ en el divisor de la entrada inversora del opamp, debido a que es necesario atenuar la señal entregada por el fototransistor.

Una vez implementado el sensor, se observó la señal a la salida del fototransistor y se concluyó que la etapa de amplificación no era necesaria, por lo que se eliminó la misma del circuito.

Se hicieron pruebas con el relé a la salida del LM567 y no se logró el funcionamiento deseado. El LM567 entrega un nivel alto en todo momento, excepto cuando detecta una señal de la frecuencia configurada. La salida del LM567 debe interrumpir al PIC por lo que es necesario invertir la lógica y además bajar el voltaje de 9 V a 5 V. Para lograr esto se utilizó una compuerta NOT, la cual permite que su entrada sea de 9 V.

En la Figura 4.20 se muestra un esquema del sensor IR diseñado.

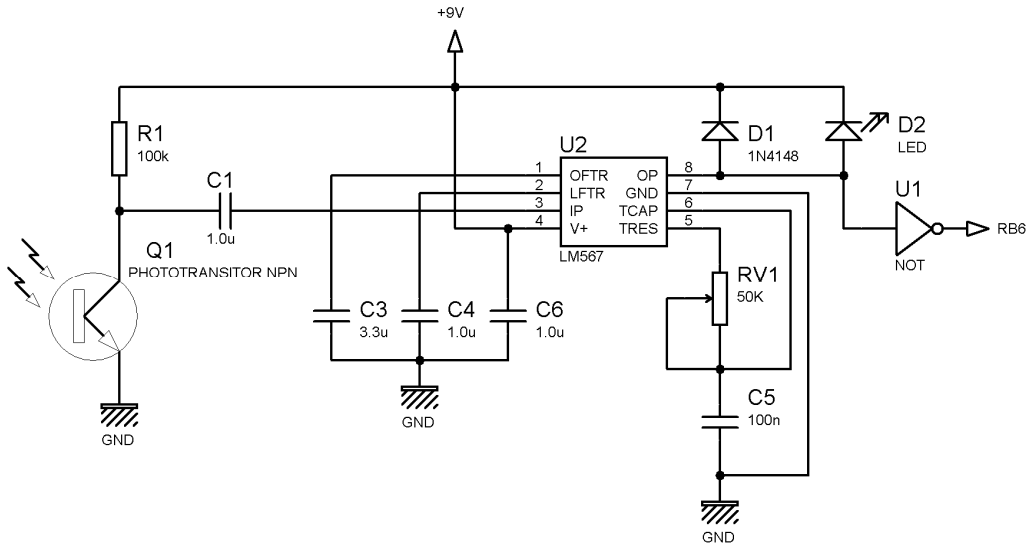


**Figura 4.20 - Sensor IR**

El funcionamiento de este sensor es análogo al funcionamiento del sensor de sonido que fue explicado en las secciones anteriores.

Además, se diseñó un emisor IR, el cual emite una luz IR a una frecuencia determinada. Para lograr que el LED IR emita a una frecuencia determinada es necesario aplicar una onda cuadrada en sus bornes. El transistor PNP (2N2907) funciona como oscilador al conmutar entre su región de corte y su región de saturación. El transistor NPN (2N3904) se utiliza para amplificar la señal.

En la Figura 4.21 se muestra el esquema del emisor IR.

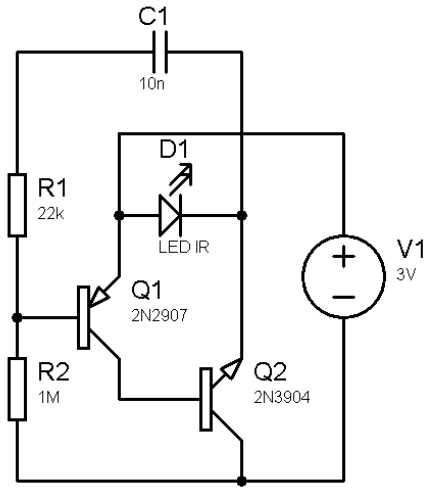


**Figura 4.21 - Esquema emisor IR**

## *4.11 Adaptación e integración de Módulo de Detección de obstáculos*

### 4.11.1 Estudio de integración de sensores de detección de obstáculos

Para implementar los sensores de obstáculos se utilizan microswitches reciclados de un mouse de computadora, cuyo funcionamiento fue explicado en el capítulo de Marco Teórico.

El vehículo fue diseñado con dos paragolpes para permitir distinguir entre un choque del lado derecho y un choque del lado izquierdo. Estos paragolpes, al ser empujados, mueven unas cintas metálicas que accionan los microswitches. Estos últimos, al ser accionados generan un “1” lógico que interrumpe al PIC y lo hace despertarse del modo SLEEP.

En la rutina de atención de interrupción, al detectar que la interrupción fue debida a un choque, el PIC notifica a la placa BB enviándole en “1” lógico en el pin correspondiente (RD4 si el choque fue del lado derecho, RD5 si el choque fue del lado izquierdo).

## *4.12 Conexión inalámbrica a Internet*

La funcionalidad del proyecto requiere que el móvil pueda comunicarse mediante algún protocolo de red inalámbrico.

En principio, casi cualquier protocolo puede ser utilizado por la placa, sin embargo, por la gran disponibilidad de las redes 3G se decidió hacer uso de esta tecnología.

El proyecto UMix hace uso de esta tecnología a través de un modem 3G USB. Por otro lado, es necesario utilizar alguna aplicación que realice la conexión.

En principio se pensó utilizar wvdial ya que es la aplicación más difundida para los propósitos mencionados, sin embargo, al momento de su utilización se encontraron problemas debido a que determinadas llamadas de la API de Debian no fueron implementadas para la versión para ARM.

Es por esto que finalmente se optó por utilizar **pppdial**. Los comandos para su instalación son los siguientes:

```
$sudo apt-get install pppdial
```

La configuración de la aplicación es un tanto engorrosa, pero puede resumirse en los siguientes pasos:

1. Crear el archivo **/etc/chatscripts/claro**, con el siguiente contenido

```
#!/bin/sh
/usr/sbin/chat -v "" ATZ "" AT+CGDCONT=1,\"IP\",\"internet.ctimovil.com.uy\" ""
"ATD*99#" CONNECT ""
```

Luego, aplicarle permisos de ejecución como root con el siguiente comando:

```
#chmod +x /etc/chatscripts/claro
```

2. Crear el archivo /**etc/ppp/peers/claro**, con el siguiente contenido:

```
connect                                          /etc/chatscripts/claro
/dev/ttyUSB0
360000
noipdefault
crtscts
modem
noccp
defaultroute
novj
noauth
ipcp-accept-remote
ipcp-accept-local
debug
idle 60
```

El parámetro **/dev/ttyUSB0** puede variar.

Luego se le ajustan los permisos con los siguientes comandos, ejecutados como root:

```
# chmod 0640 /etc/ppp/peers/claro
# chown root:dip /etc/ppp/peers/claro
```

3. Por último, es posible conectarse o desconectarse con los siguientes comandos:

```
$pon claro
$poff claro
```

4. En caso de que la conexión no funcione correctamente, es posible que haya que configurar adecuadamente las rutas y DNS. Para ello se debe consultar al proveedor de Internet y ejecutar los siguientes comandos:

```
Para cambiar el gateway
$sudo route add default gw <IP del Gateway>

Para cambiar el DNS
#echo nameserver <ip del DNS> > /etc/resolv.conf
```

## *4.13 Armado primer prototipo*

Al planificar el proyecto se pretendía que para la fecha de comenzar el armado del primer prototipo, los distintos módulos se encontraran funcionando correctamente. Por este motivo, se planificó esta tarea con la idea de que solo consistiera en integrar estos módulos. Sin embargo, llegada la fecha del armado del primer prototipo, no todos los módulos funcionaban como era esperado, por lo que fue necesario seguirles dedicando tiempo.

Durante esta tarea surgieron complicaciones al intentar comunicar la placa BB con el PIC. Estas complicaciones y la solución a la que se llegó se explican en la sección Testeo protocolo comunicación.

# 5 Conclusiones

Hasta el momento se han cumplido con los objetivos fijados para la entrega del primer prototipo. Sin embargo, algunas tareas importantes para el proyecto fueron pospuestas para poder dedicar más tiempo a la documentación y a los testeos necesarios para esta entrega. El equipo asume la responsabilidad de dicha postergación y prevé que será necesario reajustar los tiempos de la planificación.

Uno de los grandes motivos de los atrasos fue el desarrollo de una aplicación C que consultara una base de datos SQL. El problema que se encontró fue que, en primer lugar, la cadena de herramientas (toolchain) utilizada no poseía las bibliotecas necesarias para cumplir esta función y además, el código generado causaba fallos de segmentación al ser ejecutado en la placa BB. La solución a esto fue instalar directamente el compilador dentro de la placa junto con las bibliotecas y compilarlo directamente sobre la placa.

Otro de los problemas encontrados se relacionó con la aplicación que establece la comunicación inalámbrica. En este caso en particular, algunas de las primitivas del kernel que utiliza dicha aplicación no estaban implementadas para la versión aplicable al procesador ARM. Es por esto que se optó por utilizar una aplicación menos amigable pero que resolvía este problema.

También se experimentaron demoras debido a la dificultad en conseguir un cable serial adecuado para poder establecer una comunicación primaria entre una PC y la placa BB. Por otro lado, la conexión cableada de red fue demorada debido a que, en principio, no se tenía conocimiento de la necesidad de utilizar un hub USB como intermediario entre el adaptador USB/Ethernet y el puerto USB-OTG de la placa BB. Además, también se demoró en la adquisición de dicho hub debido a los estrictos requerimientos.

Además, durante las tareas de testeo de circuitos se experimentaron dificultades debido a que en la práctica los circuitos no se comportan exactamente igual a como lo hacen en la simulación. Si bien todos los circuitos habían sido diseñados y simulados en la aplicación Proteus, y se encontraban funcionando correctamente en la simulación, al momento de armar los circuitos en el laboratorio, estos no funcionaban como era esperado. Este

problema se debió principalmente a que en la simulación no se consideran factores externos, tales como la presencia de ruido eléctrico o falsos contactos en los protoboards.

Es importante tener en cuenta que al momento de la planificación del proyecto no se consideró la gran demanda de tiempo requerida para la elaboración de documentos. Este fue otro motivo por el cual fue necesario postergar tareas.

Por último, se experimentaron dificultades al intentar comunicar la placa BB con el PIC debido a que los mismos manejan lógicas con distintos niveles de voltaje. Si bien esta dificultad fue experimentada por otros equipos en otros proyectos, la solución encontrada utiliza circuitos integrados disponibles en el mercado extranjero pero no en el local.

# 6 Recomendaciones

Por cuestiones de tiempo y para poder mantener bajos los costos del proyecto, se decidió simular el sensor de vida con otro sensor. Sin embargo, podría resultar interesante incorporar a futuros proyectos que utilicen al proyecto UMix como base, la implementación de un sensor que permita detectar supervivientes.

También resultaría interesante la implementación de una plataforma física de movimiento más flexible que la utilizada en este proyecto, la cual permitiera recorrer superficies más irregulares.

Se recomienda ampliamente el uso del libro “Linux kernel in a nutshell” de Greg Kroah-Hartman para comprender mejor el funcionamiento del kernel de Linux y así poder adaptar mejor sus componentes a la placa BB. Esto permitiría optimizar el rendimiento de la placa BB y el aprovechamiento de su memoria.

# 7 Bibliografía


- ATE-Universidad de Oviedo. *Adaptación de Salidas: Control de Motores* disponible en http://www2.ate.uniovi.es/fernando/.../Control%20de%20motores.pdf, 3 de diciembre de 2009
- beagleboard.org. *BeagleBoard System Reference Manual Rev C3*. 2009 disponible en http://beagleboard.org/static/BBSRM_latest.pdf, 7 de diciembre de 2009
- en.wikipedia.org. *General Purpose Input/Output*. 2009 disponible en http://en.wikipedia.org/wiki/General_Purpose_Input/Output, 10 de diciembre de 2009
- enciclopedia.us.es. *Transistor*. 2009 disponible en http://enciclopedia.us.es/index.php/Transistor, 14 de diciembre de 2009
- es.wikipedia.org. *Amplificador operacional* disponible en http://es.wikipedia.org/wiki/Amplificador_operacional, 4 de diciembre de 2009
- es.wikipedia.org. *Motor de corriente continua* disponible en http://es.wikipedia.org/wiki/Motor_de_corriente_continua, 4 de diciembre de 2009
- es.wikipedia.org. *Relé* disponible en http://es.wikipedia.org/wiki/Rel%C3%A9, 4 de diciembre de 2009
- es.wikipedia.org. *Repositorio* disponible en http://es.wikipedia.org/wiki/Repositorio, 5 de diciembre de 2009
- es.wikipedia.org. *Subversion* disponible en http://es.wikipedia.org/wiki/Subversion, 5 de diciembre de 2009
- es.wikipedia.org. *Transistor de unión bipolar*. 2009 disponible en http://es.wikipedia.org/wiki/Transistor_de_uni%C3%B3n_bipolar, 14 de diciembre de 2009
- Greg Kroah-Hartman. *Linux kernel in a nutshell*. 1st edition. Editorial O'Reilly. 2006. 182p
- Hewlett-Packard Company et al. *Universal Serial Bus 3.0*. 2008 disponible en http://www.gaw.ru/pdf/interface/usb/USB%203%200_english.pdf, 7 de diciembre de 2009
- Mancini, Ron. *Opamps for everyone*. 2002 disponible en focus.ti.com.cn/cn/lit/an/slod006b/slod006b.pdf, 4 de diciembre de 2009
- Martin, Daniel C., *Motorización*. 2001-2008 disponible en http://www.x-robotics.com/motorizacion.htm, 4 de diciembre de 2009
- Microchip Technology Inc. *MPASM™ Assembler, MPLINK™ Object Linker, MPLIB™ Object Librarian User's Guide.* 2005 disponible en http:// ww1.microchip.com/downloads/en/DeviceDoc/33014J.pdf, 4 de diciembre de 2009
- Microchip Technology Inc. *MPLAB® IDE User.s Guide.* 2006 disponible en http://ww1.microchip.com/downloads/en/DeviceDoc/51519B.pdf, 4 de diciembre de 2009
- Microchip Technology Inc. *PICmicro™ Mid-Range MCU Family Reference Manual. 1997 disponible en* http://ww1.microchip.com/downloads/en/devicedoc/33023a.pdf, 4 de diciembre de 2009
- National Semiconductor Corporation. LM567/LM567C Tone Decoder Datasheet. 1995 disponible en www.national.com/ds/LM/LM567.pdf, 4 de diciembre de 2009
- Powering microphones, *Copyright Tomi Engdahl 1997-2000* disponible en http://www.epanorama.net/circuits/microphone_powering.html, 3 de diciembre de 2009.
- Rossing, Thomas D. *The Science of Sound.* 2nd Ed. Addison-Wesley. 1990
- SGS-THOMSON Microelectronics. *Datasheet L293D*. 1996 disponible en http://www.datasheetcatalog.org/datasheet/stmicroelectronics/1330.pdf, 3 de diciembre de 2009
- Texas Instruments Incorporated. *TL070 JFET-INPUT OPERATIONAL AMPLIFIER Datasheet.* 2001 disponible en http://pdf1.alldatasheet.com/datasheet-pdf/view/28771/TI/TL070.html, 4 de diciembre de 2009
- Texas Instruments. *OMAP3530/25 Applications Processor*. 2009 disponible en http://focus.ti.com/lit/ds/symlink/omap3530.pdf, 7 de diciembre de 2009

# 8 Anexo I – Plan del Proyecto Final de Carrera

## *8.1 Acta del Proyecto*

### 8.1.1 Nombre del Proyecto

Implementación de un sistema automatizado de búsqueda de supervivientes de catástrofes.

### 8.1.2 Fecha de comienzo

31 de Agosto de 2009

### 8.1.3 Áreas de conocimientos / procesos

Las áreas de conocimientos que están involucradas en este proyecto son:

| Área General | Área específica | Descripción |
|---|---|---|
| **Desarrollo de plataformas** | Adaptación del sistema operativo Linux | El proyecto requiere la adaptación del sistema operativo linux a determinada plataforma de desarrollo |
| | | |
| **Electrónica** | Desarrollo de fuentes de alimentación de Corriente continua | Se utilizarán fuentes de alimentación de corriente continua que deben ser desarrolladas por el equipo de proyecto |
| | Drivers de motores | Los motores que se utilizan en el proyecto se manejaran mediante un hardware que será diseñado específicamente para el proyecto |
| | Interfaces de comunicación | Todas las interfaces de comunicación requieren una interfaz de hardware que será estudiada (y desarrollada en caso de ser necesario). |
| | | |
| **Programación** | Assembler para el manejo de sensores y motores | Todo el manejo de sensores y motores será realizado mediante software desarrollado en assembler |
| | Interfaz Web | El proyecto requiere de una interfaz web que permita la administración del móvil |
| | Base de datos | Como requerimiento del proyecto se especifico la posibilidad de almacenar resultados en una base de datos. Es por esto que el proyecto requiere de la implementación de algún modelo que permita respaldar la información necesaria. |
| | Interfaces de comunicación | El proyecto depende en gran medida de la comunicación de los distintos módulos. Es por esto que es necesario desarrollar e implementar mediante software distintos protocolos de comunicación. |

| | | |
|---|---|---|
| **Protocolos de comunicación inalámbrica** | Utilización de protocolos de comunicación inalámbrica | El proyecto es dependiente de la capacidad de comunicarse inalambricamente con otros terminales. Es por esto que este proyecto involucra el estudio y uso de las redes inalámbricas existentes. |

### 8.1.4 Áreas de aplicación (sector / actividad)

- Fuerzas armadas, ejércitos
- Entes Gubernamentales que participen en la búsqueda de supervivientes
- ONG que participen en la búsqueda de supervivientes
- Investigadores particulares de robótica
- Desarrolladores de linux

### 8.1.5 Fecha de inicio del Proyecto

12 de Agosto de 2009

### 8.1.6 Fecha tentativa de finalización del Proyecto

15 de Marzo de 2010

## *8.2 Objetivos del proyecto*

### 8.2.1 Objetivo General

Contribuir con el desarrollo de tecnologías que faciliten la tarea de búsqueda de supervivientes en áreas de catástrofes. Para ello se utilizarán componentes de fácil adquisición y estandarizados y se recurrirá a código open source, de forma de permitir futuros desarrollos sobre la base de este proyecto.

### 8.2.2 Objetivos Específicos

Implementar funcionalidades básicas de búsqueda de personas, es decir realizar un móvil que se desplace de forma autónoma en un área determinada e informe sobre los lugares específicos donde se encontraron posibles supervivientes. Para esto se utilizará un sistema operativo Linux sobre una placa BeagleBoard, y un microcontrolador PIC 16F877A.

Desarrollar un módulo, dentro del móvil, que permita la comunicación inalámbrica con una computadora personal utilizando protocolos de red estandarizados. Esto permitirá la fácil administración y manipulación del móvil.

Incluir un módulo de posicionamiento que haga uso de un dispositivo GPS estándar para determinar la ubicación del móvil en todo momento y en particular cuando se detecta un superviviente.

Diseñar el sistema de movimiento de forma tal que la plataforma física de movimiento sea fácilmente intercambiable.

## 8.3 Descripción del producto

UMix pretende ser un sistema automatizado para la búsqueda de supervivientes de catástrofes.

Se utilizarán diversos dispositivos de ubicación, monitoreo y comunicación ya estandarizados para cumplir con su objetivo. Fundamentalmente hará uso de un sistema de GPS para contar con la ubicación del producto en todo momento y delimitar el área de búsqueda, en coordinación con sensores que le permitirán esquivar obstáculos y hallar los supervivientes y un sistema de comunicación inalámbrica estándar que permitirá notificar a un dispositivo cliente la ubicación de las victimas.

Todo el sistema será implementado en base a software, interfaces y plataformas libres, lo cual permitirá la fácil adaptación de este proyecto a otros sistemas.

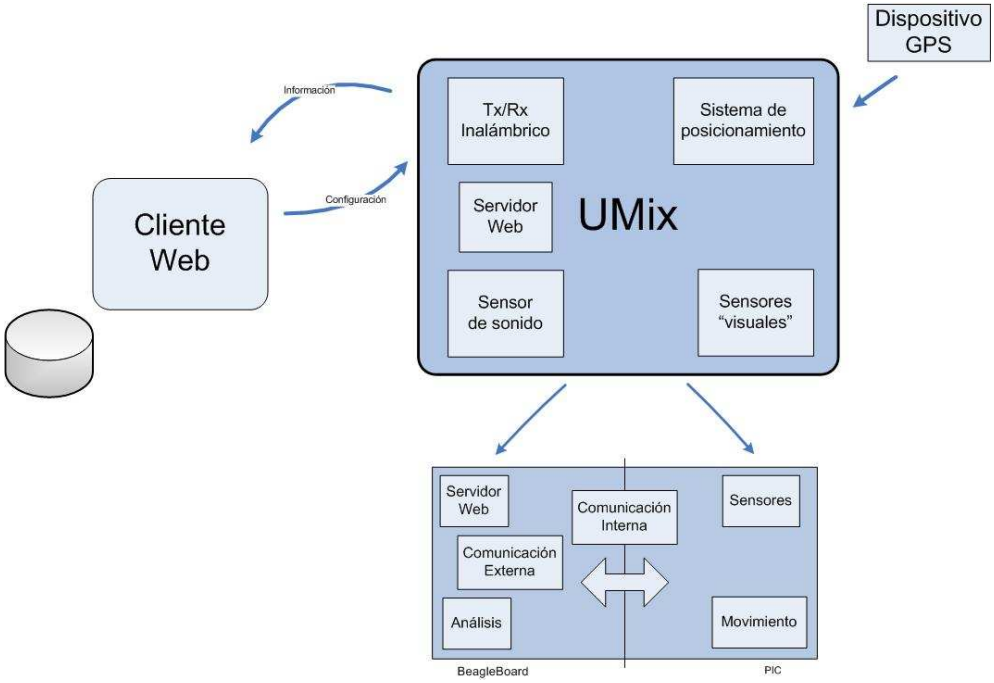


**Figura 8.1 - Esquema modular**

## 8.4 Necesidad del Proyecto

Hoy en día en el mundo suceden catástrofes muy frecuentemente. Esto genera enormes cantidades de heridos cuyo rescate suele ser una tarea ardua y compleja. Por este motivo es que cualquier proyecto que facilite dichas tareas es atractivo. Nuestro proyecto en particular pretende automatizar la tarea de búsqueda de supervivientes de dichas catástrofes.

## *8.5 Alcance*

### 8.5.1 Incluido

Las funcionalidades incluidas previstas para este proyecto abarcan:

**AP-1.** Sistema de posicionamiento: Este sistema hará uso de un dispositivo de GPS disponible comercialmente para establecer la posición del móvil en todo momento y permitir al usuario delimitar la zona de búsqueda.
**AP-2.** Sistema de sensores de objetos: Se utilizaran sensores cuya función será detectar posibles obstáculos para que el móvil pueda esquivarlos.
**AP-3.** Sensor de sonido: Para simplificar la implementación del prototipo se descartó utilizar sensores de signos vitales. En su lugar se utilizará un sensor de sonido.
**AP-4.** Sistema de comunicación: El sistema ofrecerá una interfaz web para su uso y administración, la cual podrá ser accedida por los usuarios mediante un receptor/transmisor 3G.
**AP-5.** Sistema de respaldo: Se ofrecerá la posibilidad de guardar diversa información en una base de datos remota.
**AP-6.** Circuitos anexos para integrar los sistemas anteriores

### 8.5.2 No incluido

El prototipo del sistema NO incluirá la implementación de los siguientes ítems:

**AP-7.** Placa multipropósito
**AP-8.** Plataforma física de movimiento
**AP-9.** Kernel de Linux
**AP-10.** Dispositivo GPS
**AP-11.** Adaptador 3G

## *8.6 Justificación del Impacto (aporte y resultados esperados)*

UMix es un proyecto que no pretende competir con sistemas automatizados de búsqueda que utilicen alta tecnología, si no que su objetivo es brindar una solución económica, sencilla y modular que pueda adaptarse a distintas realidades. Además pretende la base para futuros proyectos de investigación y desarrollo que haya en la rama.

Si bien muchos de los componentes que se utilizarán en la implementación de UMix ya existen, no existe ningún proyecto hasta el momento que las vincule y las integre en una solución de estas características.

Se espera que el valor agregado por la integración de todas las tecnologías y las herramientas desarrolladas permitan que el proyecto UMix tenga un gran impacto y aceptación en el mercado.

## *8.7 Entregables*

**EP-1.** **25 de agosto de 2009** Documento de definición del tema
**EP-2.** **9 de setiembre de 2009** Plan de Proyecto
**EP-3.** **16 de diciembre de 2009** Primer prototipo
**EP-4.** **16 de diciembre de 2009** Primer borrador del Informe (60%)
**EP-5.** **15 de febrero de 2010** Segundo borrador del Informe (100%)
**EP-6.** **25 de febrero de 2010** Informe Final
**EP-7.** **30 de marzo de 2010** Afiche
**EP-8.** **30 de abril de 2010** Paper para memoria de trabajos de difusión científica y técnica de la UM

## *8.8 Restricciones*

Consideramos que los siguientes puntos son restricciones para este proyecto:

**RP-1.** No contar con el hardware adecuado para la implementación del sistema, es decir no disponer de una placa BeagleBoard y un PIC 16F877A en correcto funcionamiento.
**RP-2.** No disponer del software necesario para el proyecto, es decir no disponer de un kernel de Linux, drivers ó herramientas que funcionen adecuadamente.
**RP-3.** No disponer de documentación sobre el hardware, software y protocolos utilizados.

## *8.9 Identificación de grupos de interés (directos e indirectos)*

Los siguientes grupos fueron identificados como interesados en el proyecto:

**GIP-1.** Grupos sociales que habiten en zonas de accidentes o catástrofes naturales frecuentes.
**GIP-2.** Gobiernos que prioricen la ayuda humanitaria
**GIP-3.** Organizaciones vinculadas al rescate de personas.
**GIP-4.** Empresas dedicadas a la venta de este tipo de tecnología
**GIP-5.** Desarrolladores de sistemas automatizados
**GIP-6.** Desarrolladores del kernel de Linux

# 9 Anexo II – Actas

## *9.1 Acta N°1*

Montevideo, 2 de setiembre de 2009

En la clase del día Miércoles 26 de agosto de 2009 fue planteada por el equipo la opción de acotar el alcance del proyecto cambiando la tarea “Implementación del módulo de detección de obstáculos” por la tarea “Adaptación e integración del módulo de detección de obstáculos”.

Es decir que en lugar de desarrollar el módulo de detección de obstáculos desde cero se obtendrá código existente en Internet y se procederá a hacer las modificaciones necesarias para adaptarlo e integrarlo al sistema del proyecto.

Este cambio fue aprobado por los tutores del proyecto, Marcelo Abreu y Thomas Hobbins.

En el día de la fecha se deja constancia del cambio mencionado mediante la presente acta, la cual será firmada por los tutores mencionados y por los integrantes del equipo (Juan Brenes y Sofía Silva).

## *9.2 Acta N°2*

Montevideo, 16 de Setiembre de 2009

Durante la semana comprendida entre la clase del día Miércoles 9 de setiembre de 2009 y la clase del día de la fecha el equipo decidió construir la plataforma de movimiento en lugar de utilizar un carro comprado.

La mencionada plataforma será construida reutilizando el chasis de un auto de juguete, el motor paso a paso (para la dirección) de una disquetera, un motor de continua (para la tracción) de algún otro juguete y algunos otros implementos.

Este cambio fue aprobado por los tutores del proyecto, Ing. Marcelo Abreu e Ing. Thomas Hobbins.

En el día de la fecha se deja constancia del cambio mencionado mediante la presente acta, la cual será firmada por los tutores mencionados y por los integrantes del equipo (Juan Brenes y Sofía Silva).

## *9.3 Acta N°3*

Montevideo, 14 de Octubre de 2009

En la clase del día Miércoles 7 de octubre de 2009 el equipo planteó un cambio en el orden de las tareas que había sido establecido en el cronograma inicial y manifestó la necesidad de agregar una subtarea de la tarea “Adaptación e integración del módulo de detección de obstáculos” debido a que ésta no había sido tenida en cuenta.

Este cambio fue aprobado por los tutores del proyecto, Ing. Marcelo Abreu e Ing. Thomas Hobbins, por lo que se realizó el control de cambio con la justificación correspondiente.

En el día de la fecha se deja constancia del cambio mencionado mediante la presente acta, la cual será firmada por los tutores mencionados y por los integrantes del equipo (Juan Brenes y Sofía Silva).

## *9.4 Acta N°4*

Montevideo, 21 de Octubre de 2009

Durante la semana comprendida entre la clase del día Miércoles 7 de octubre de 2009 y la clase del día Miércoles 14 de octubre de 2009 se tomaron decisiones respecto al sensor de sonido y respecto a los sensores de obstáculos. Además, se avanzó con la implementación a nivel del PIC de rutinas de atención de las interrupciones generadas por los sensores y por la placa BeagleBoard.

Se decidió implementar el sensor de sonido utilizando un micrófono, un amplificador operacional y un detector de tono. El micrófono captará los sonidos del ambiente y entregará una señal de voltaje. El amplificador operacional permitirá filtrar y amplificar la señal para luego entregársela al detector de tono. La salida del detector de tono activará un interruptor controlado por voltaje, lo cual hará que el PIC reciba un “1” lógico en la entrada correspondiente al sensor de sonido (RB6).

Se decidió utilizar flip-flops RS asíncronos para retener el pulso generado por los sensores al detectar un obstáculo. También se decidió implementar dichos flip-flops utilizando compuertas NOR.

Se implementaron las rutinas de atención de interrupciones a nivel del PIC, las cuales permiten distinguir entre los distintos tipos de interrupciones: generada por la placa BeagleBoard para enviar una directiva, generada por alguno de los sensores de obstáculos (derecho o izquierdo) al detectar un obstáculo y generada por el sensor de sonido al detectar un sonido en una frecuencia predeterminada.

Estos avances fueron verificados por los tutores de proyecto, Ing. Marcelo Abreu e Ing. Thomas Hobbins.

En el día de la fecha se deja constancia de los avances mencionado mediante la presente acta, la cual será firmada por el tutor mencionado y por los integrantes del equipo (Juan Brenes y Sofía Silva).

## *9.5 Acta N°5*

Montevideo, 21 de Octubre de 2009

Durante la semana comprendida entre la clase del día Miércoles 14 de octubre de 2009 y la clase del día de la fecha se realizó un debug del programa principal del PIC en el modelo simulado con el programa Proteus. El resultado de este debug fue que todas las rutinas implementadas hasta el momento funcionaron como se esperaba.

En esta semana también se investigó sobre los circuitos integrados 308 y 567 a utilizarse para el sensor de sonido y se documentó lo investigado.

Estos avances fueron verificados por los tutores de proyecto, Ing. Marcelo Abreu e Ing. Thomas Hobbins.

En el día de la fecha se deja constancia de los avances mencionado mediante la presente acta, la cual será firmada por el tutor mencionado y por los integrantes del equipo (Juan Brenes y Sofía Silva).

## *9.6 Acta N°6*

Montevideo, 11 de Noviembre de 2009

Durante la semana comprendida entre la clase del día Miércoles 4 de noviembre de 2009 y la clase del día de la fecha se avanzó con la implementación de circuitos.

Por un lado, se implementaron los flip-flops para los sensores de obstáculos utilizando la compuerta NOR CMOS 4001 y se verificó que funcionaran correctamente.

Por otro lado, se implementó el circuito para el sensor de sonido. Dicho circuito está compuesto por tres etapas: micrófono, amplificador operacional y detector de tono. Se logró el correcto funcionamiento de las tres etapas por separado. Luego, se integró la etapa de amplificación con la etapa de detección de tono y se logró el correcto funcionamiento de esta etapa. Sin embargo, no se logró el correcto funcionamiento de las tres etapas juntas, ya que al conectar la salida del micrófono al amplificador operacional se pudo observar que la señal se contamina con ruido, por lo que el harmónico de 1 kHz no es detectado por el detector de tono.

Estos avances fueron verificados por los tutores de proyecto, Ing. Marcelo Abreu e Ing. Thomas Hobbins.

En el día de la fecha se deja constancia de los avances mencionado mediante la presente acta, la cual será firmada por el tutor mencionado y por los integrantes del equipo (Juan Brenes y Sofía Silva).

# 10 Anexo III – Código del PIC

```
;PROGRAMA PARA CONTROLAR EL MOVIMIENTO DEL CARRO
;*****************************************************
;*  Microchip Technology Incorporated
;*  PIC 16F877A
;*****************************************************

; Include file, change directory if needed
      include "p16f877a.inc"
      list p=16f877a
      ERRORLEVEL 1,-302


      ORG 0x00
      goto  Main

      ORG 0x04
      goto  AtencionInterrupcion


      ORG 0x19
Aux
      ORG 0x20
CONT
      ORG 0x21
PASOS
      ORG 0x22
TEMP              ;Variable temporal
      ORG 0x23
TipoInt           ;Variable auxiliar para grabar el tipo de interrupción


      ORG 0x25
Main
      call  Inicializacion
      sleep
      nop
      call  banco0
      clrf  INTCON                      ;Deshabilito interrupciones
      btfsc TipoInt,0               ;Me fijo si me enviaron Adelante
      goto  SubrutinaAdelante ;Si, entonces ejecuto SubrutinaAdelante
      btfsc TipoInt,1         ;No, entonces me fijo si me enviaron Atrás
      goto  SubrutinaAtras          ;Si, entonces ejecuto SubrutinaAtras
      btfsc TipoInt,2   ;No, entonces me fijo si me enviaron Derecha
      goto  SubrutinaDerecha  ;Si, entonces ejecuto SubrutinaDerecha
      btfsc TipoInt,3   ;No, entonces me fijo si me enviaron Izquierda
      goto  SubrutinaIzquierda ;Si, entonces ejecuto SubrutinaIzquierda
      btfsc TipoInt,4   ;No, entonces me fijo si se detectó un obstáculo
                        ;en la derecha
      goto  SubrutinaObstDer  ;Si, entonces ejecuto SubrutinaObstDer
      btfsc TipoInt,5   ;No, entonces me fijo si se detectó un obstáculo
                        ;en la izquierda
      goto  SubrutinaObstIzq  ;Si, entonces ejecuto SubrutinaObstIzq
      btfsc TipoInt,6   ;No, entonces me fijo si se detectó un sonido
      goto  SubrutinaSonido         ;Si, entonces ejecuto SubrutinaSonido
      goto  Main                    ;No, entonces vuelvo a Main
```

```
Inicializacion
      call  banco1
      movlw b'11111111'
      movwf TRISA       ;Configuramos el puerto A como entrada (Por
                        ;ahora no lo vamos a usar)

      movlw b'01110001' ;Configuramos los bits RB0 y RB4:RB6 como
                        ;entradas y el resto como salidas,
      movwf TRISB       ;vamos a utilizar las entradas para la
                        ;interrupción para recibir directivas (RB0)
                        ;y para los sensores (RB4:RB6)
                        ;y las salidas RB1 y RB2 para volver a
                        ;poner en cero los flip-flops asociados a
                        ;los sensores de obstáculos.

      movlw b'00000000'
      movwf TRISC       ;Configuramos el puerto C como salida, lo
                        ;utilizaremos para manejar los motores.

      movlw b'00001111'
      movwf TRISD       ;Configuramos los bits RD0:RD3 como entrada
                        ;(directivas de movimiento) y el resto como
                        ;salidas (los bits RD4:RD6 se utilizan
                        ;para reportarle a la placa condiciones positivas
                        ;en los sensores).

      call  banco0      ;Nos posicionamos en el banco 0
      bsf   PORTD,7     ;DEBUG
      clrf  INTCON      ;Deshabilitamos todas las interrupciones
      clrf  PORTD       ;Borra el puerto D
      clrf  PORTC       ;Borra el puerto C
      bsf   PORTB,1     ;Restea flip-flop asociado al sensor derecho
      bsf   PORTB,2     ;Restea flip-flop asociado al sensor izquierdo
      clrf  PORTB       ;Borra el puerto B
      clrf  PORTA       ;Borra el puerto A

      clrf  CONT        ;Pongo en cero el contador
      clrf  TEMP        ;Pongo en cero la variable temporal
      movlw b'00110010'
      movwf Aux         ;Seteo Aux en 50 para que el tiempo de espera
                        ;para notificar
                        ;a la BB sea de 2 seg aprox (HAY QUE AJUSTARLO)
      clrf  TipoInt     ;Pongo en cero la variable para el tipo de
                        ;interrupción
      movlw b'00001111'
      movwf PASOS       ;Seteo en 15 la cantidad de pasos (Es necesario
                        ;ajustarlo)
      call  banco1
      movlw 0x06
      movwf ADCON1      ;set puerto analógico como digital
      bsf   OPTION_REG,6 ;Configuramos flanco ascendente para la
                        ;interrupción en RB0.
      call  banco0
      clrf  PORTB       ;Limpio el puerto B
      bcf   INTCON,RBIE ;Deshabilitamos la interrupción al cambiar
                        ;RB4:RB7
      movf  PORTB,W     ;Leemos el puerto B
```

```
      movlw b'10011000' ;Habilitamos interrupciones globales,
                        ;interrupción externa de RB0 e interrupción RB
                        ;port change
      movwf INTCON      ;y limpiamos los flags

      return

AtencionInterrupcion
      call  banco0
      bcf   INTCON,GIE  ;Deshabilito interrupciones globales
      btfsc INTCON,RBIF ;La interrupción fue por un cambio en RB4:RB7?
      goto  AtencionSensores  ;Si, entonces voy a la rutina de atención
                              ;correspondiente
      btfsc INTCON,INTF ;No, entonces me fijo si la interrupción fue
                        ;porque la placa BB va a enviar una directiva.
      goto  RecibirDirectiva  ;Si, entonces veo qué directiva me envió la
                              ;placa
      bcf   INTCON,GIE  ;No, entonces vuelvo a habilitar interrupciones
                        ;globales
      retfie      ;y regreso a donde estaba antes de la interrupción

AtencionSensores
      call  banco0
      bcf   INTCON,RBIE ;Deshabilito interrupción RB port change
      bcf   INTCON,RBIF ;Pongo en cero el flag correspondiente a la
                        ;interrupción RB port change
      movf  PORTB,W     ;Vuelvo a leer el puerto B
      movwf TEMP        ;Pongo lo leído en la variable temporal
      btfsc PORTB,4     ;Me fijo si fue el Sensor de obstáculos Derecho
      goto  Tipo4       ;Si, entonces voy a Tipo4
      btfsc PORTB,5     ;No, entonces me fijo si fue el Sensor de
                        ;obstáculos Izquierdo
      goto  Tipo5       ;Si, entonces voy a Tipo5
      btfsc PORTB,6     ;No, entonces me fijo si fue el Sensor de Sonido
      goto  Tipo6       ;Si, entonces voy a Tipo6

      bsf   INTCON,RBIE ;No, entonces vuelvo a habilitar la interrupción
                        ;RB port change,
      bsf   INTCON,GIE  ;vuelvo a habilitar las interrupciones globales
      retfie      ;y regreso a donde estaba antes de la interrupción


Tipo4
      bsf   TipoInt,4   ;Seteo el flag 4 de la variable TipoInt
      bsf   INTCON,RBIE ;Vuelvo a habilitar la interrupicón RB port
                        ;change
      bsf   INTCON,GIE  ;Vuelvo a habilitar interrupciones globales
      retfie      ;y regreso a donde estaba antes de la interrupción

Tipo5
      bsf   TipoInt,5   ;Seteo el flag 5 de la variable TipoInt
      bsf   INTCON,RBIE ;Vuelvo a habilitar la interrupicón RB port
                        ;change
      bsf   INTCON,GIE  ;Vuelvo a habilitar interrupciones globales
      retfie      ;y regreso a donde estaba antes de la interrupción

Tipo6
      bsf   TipoInt,6   ;Seteo el flag 6 de la variable TipoInt
      bsf   INTCON,RBIE ;Vuelvo a habilitar la interrupicón RB port
                        ;change
```

```
      bsf   INTCON,GIE  ;Vuelvo a habilitar interrupciones globales
      retfie      ;y regreso a donde estaba antes de la interrupción


RecibirDirectiva
      call  banco0
      bcf   INTCON,INTE ;Deshabilito interrupción externa de RB0
      movf  PORTB,W     ;Leo el puerto B
      movwf TEMP        ;Pongo lo leído en la variable temporal
      bcf   INTCON,INTF ;Pongo en cero el flag correspondiente a la
                        ;interrupción externa de RB0
      btfsc PORTD,0     ;Testeo si me enviaron Adelante
      goto  Tipo0       ;Si, entonces voy a Tipo0
      btfsc PORTD,1     ;No, entonces testeo si me enviaron Atrás
      goto  Tipo1       ;Si, entonces voy a Tipo1
      btfsc PORTD,2     ;No, entonces testeo si me enviaron Derecha
      goto  Tipo2       ;Si, entonces voy a Tipo2
      btfsc PORTD,3     ;No, entonces testeo si me enviaron Izquierda
      goto  Tipo3       ;Si, entonces voy a Tipo3
      bsf   INTCON,INTE ;No, entonces vuelvo a habilitar la interrupción
                        ;externa de RB0,
      bcf   INTCON,GIE  ;vuelvo a habilitar interrupciones globales
      retfie      ;y regreso a donde estaba antes de la interrupción

Tipo0
      bsf TipoInt,0     ;Seteo el flag 0 de la variable TipoInt
      bsf INTCON,INTE ;Vuelvo a habilitar la interrupicón externa de RB0
      bsf INTCON,GIE    ;Vuelvo a habilitar interrupciones globales
      retfie      ;y regreso a donde estaba antes de la interrupción

Tipo1
      bsf TipoInt,1     ;Seteo el flag 1 de la variable TipoInt
      bsf INTCON,INTE ;Vuelvo a habilitar la interrupicón externa de RB0
      bsf INTCON,GIE    ;Vuelvo a habilitar interrupciones globales
      retfie      ;y regreso a donde estaba antes de la interrupción

Tipo2
      bsf TipoInt,2     ;Seteo el flag 2 de la variable TipoInt
      bsf INTCON,INTE ;Vuelvo a habilitar la interrupicón externa de RB0
      bsf INTCON,GIE    ;Vuelvo a habilitar interrupciones globales
      retfie      ;y regreso a donde estaba antes de la interrupción

Tipo3
      bsf TipoInt,3     ;Seteo el flag 3 de la variable TipoInt
      bsf INTCON,INTE ;Vuelvo a habilitar la interrupicón externa de RB0
      bsf INTCON,GIE    ;Vuelvo a habilitar interrupciones globales
      retfie      ;y regreso a donde estaba antes de la interrupción

;*********************************************************************
;Control del motor de Continua (Tracción)
;*********************************************************************

Adelante
      call  banco0
      bsf   PORTC,0     ;Habilitamos canales del L293D (Tomamos el
                        ;control del motor)
      bsf   PORTC,1           ;Seteamos el bit RC1
      bcf   PORTC,2           ;Ponemos en cero el bit RC2
      return
```

```
SubrutinaAdelante
      call  banco0
      bcf   TipoInt,0   ;Limpio el flag de TipoInt
      call  Adelante
      call  Espera
      call  Detener
      call  LiberarMotorTraccion
      goto  Main

Atras
      call  banco0
      bsf   PORTC,0 ;Habilitamos canales (Tomamos el control del motor)
      bcf   PORTC,1           ;Ponemos en cero el bit RC1
      bsf   PORTC,2           ;Seteamos el bit RC2
      return

SubrutinaAtras
      call  banco0
      bcf   TipoInt,1   ;Limpio el flag de TipoInt
      call  Atras
      call  Espera
      call  Detener
      call  LiberarMotorTraccion
      goto  Main

Detener
      call  banco0
      bcf   PORTC,1           ;Ponemos en cero el bit RC1
      bcf   PORTC,2           ;Ponemos en cero el bit RC2
      return

LiberarMotorTraccion
      call  banco0
      bcf   PORTC,0           ;Deshabilitamos canales del L293B
                              ;(Liberamos el motor)
      return

;*********************************************************************
;Control del motor Paso a Paso (Dirección)

Derecha
      bsf   PORTC,3           ;Tomamos el control del motor
      call  Paso1
      call  EsperaPaP
      call  Paso2
      call  EsperaPaP
      call  Paso3
      call  EsperaPaP
      call  Paso4
      call  EsperaPaP
      incf  CONT,1                  ;Incrementamos el contador
      movf  PASOS,0  ;Escribo la cantidad de pasos en W para que la
                     ;secuencia 1-2-3-4 se ejecute PASOS veces
      subwf CONT,0   ;Le resto W a CONT y lo almaceno en W
      btfss STATUS,2    ;Me fijo si la operacion dio 0
      goto  Derecha  ;Si no dio 0, CONT aún no llegó a PASOS entonces
                     ;repite
      bcf   PORTC,3  ;Si dio 0, CONT llegó a PASOS entonces libero el
                     ;motor,
      clrf  CONT  ;pongo en 0 el contador,
```

```
      call  LiberarMotorPaP   ;libero el motor
      return

SubrutinaDerecha
      call  banco0
      bcf   TipoInt,2   ;Limpio el flag de TipoInt
      call  Adelante
      call  Derecha
      call  Detener
      call  LiberarMotorTraccion
      goto  Main

Izquierda
      bsf   PORTC,3               ;Tomamos el control del motor
      call  Paso4
      call  EsperaPaP
      call  Paso3
      call  EsperaPaP
      call  Paso2
      call  EsperaPaP
      call  Paso1
      call  EsperaPaP
      incf  CONT,1                ;Incrementamos el contador
      movf  PASOS,0 ;Escribo la cantidad de pasos en W para que la
                    ;secuencia 4-3-2-1 se ejecute PASOS veces
      subwf CONT,0  ;Le resto W a CONT y lo almaceno en W
      btfss STATUS,2    ;Me fijo si la operacion dio 0
      goto  Izquierda   ;Si no dio 0, CONT aún no llegó a PASOS entonces
                        ;repite
      bcf   PORTC,3     ;Si dio 0, CONT llegó a PASOS entonces libero el
                        ;motor,
      clrf  CONT        ;pongo en 0 el contador,
      call  LiberarMotorPaP   ;libero el motor
      return

SubrutinaIzquierda
      call  banco0
      bcf   TipoInt,3         ;Limpio el flag de TipoInt
      call  Adelante
      call  Izquierda
      call  Detener
      call  LiberarMotorTraccion
      goto  Main

LiberarMotorPaP
      call  banco0
      bcf   PORTC,3     ;Deshabilitamos canales del L293B (Liberamos el
                        ;motor)
      return

Paso1
      bsf   PORTC,4
      bcf   PORTC,5
      bsf   PORTC,6
      bcf   PORTC,7
      return

Paso2
      bsf   PORTC,4
      bcf   PORTC,5
```

```
      bcf   PORTC,6
      bsf   PORTC,7
      return

Paso3
      bcf   PORTC,4
      bsf   PORTC,5
      bcf   PORTC,6
      bsf   PORTC,7
      return

Paso4
      bcf   PORTC,4
      bsf   PORTC,5
      bsf   PORTC,6
      bcf   PORTC,7
      return

;Rutinas de espera

Espera
      call  banco1
      movlw 0xFF
      movwf PR2 ;Escribimos 255 en PR2 para que TMR2 incremente hasta 255
      call  banco0
      clrf  TMR2  ;Pongo en 0 el registro del timer 2
      movlw b'01111011' ;Configuramos prescaler y postscaler 1:16
      movwf T2CON
      bsf   T2CON,2           ;Encendemos el Timer 2
      call  Test              ;Testeo si terminó la espera
      decf  Aux,1             ;Decremento Aux
      btfss STATUS,2          ;Me fijo si Aux llegó a 0
      goto  Espera            ;Si no llegò a 0 vuelvo a esperar
      return                  ;Si llegó a 0 regreso

EsperaPaP
      call  banco1
      movlw b'11111111' ;El período típico de conmutación del L298 son
                        ;40us
      movwf PR2         ;Escribimos 255 en PR2 para que TMR2 incremente
                        ;hasta 255
      call  banco0
      clrf  TMR2        ;Pongo en 0 el registro del timer 2
      movlw b'01111011' ;Configuramos prescaler y postscaler 1:16
      movwf T2CON
      bsf   T2CON,2     ;Encendemos el Timer 2
      call  Test        ;Testeo si terminó la espera
      return

Test
      btfss PIR1,1            ;Terminó la espera?
      goto  Test              ;Si no terminó sigo esperando
      bcf   PIR1,1            ;Si terminó, limpio el flag
      bcf   T2CON,2           ;Apago el timer 2
      return                  ;y regreso

SubrutinaObstDer
      call  banco0
      bcf   TipoInt,4         ;Limpio el flag de TipoInt
      bsf   PORTB,1           ;Envío Reset al flip-flop
```

```
      bsf   PORTD,4           ;Notifico a la placa que se detectó un
                              ;obstáculo del lado derecho
      call  EsperaNotific     ;Ejecuto rutina de espera para que la placa
                              ;lea el dato
      bcf   PORTB,1           ;Apago el bit de reset
      bcf   PORTD,4           ;Limpio el bit RD4
      goto  Main

SubrutinaObstIzq
      call  banco0
      bcf   TipoInt,5         ;Limpio el flag de TipoInt
      bsf   PORTB,2           ;Envío Reset al flip-flop
      bsf   PORTD,5           ;Notifico a la placa que se detectó un
                              ;obstáculo del lado izquierdo
      call  EsperaNotific     ;Ejecuto rutina de espera para que la placa
                              ;lea el dato
      bcf   PORTB,2           ;Apago el bit de reset
      bcf   PORTD,5           ;Limpio el bit RD5
      goto  Main

SubrutinaSonido
      call  banco0
      bcf   TipoInt,6         ;Limpio el flag de TipoInt
      bsf   PORTD,6     ;Notifico a la placa que se detectó un sonido
      call  EsperaNotific     ;Ejecuto rutina de espera para que la placa
                              ;lea el dato
      bcf   PORTD,6           ;Limpio el bit RD6
      goto  Main

EsperaNotific
      call  banco1
      movlw 0xFF
      movwf PR2 ;Escribimos 255 en PR2 para que TMR2 incremente hasta 255
      call  banco0
      clrf  TMR2        ;Pongo en 0 el registro del timer 2
      movlw b'01111011' ;Configuramos prescaler y postscaler 1:16
      movwf T2CON
      bsf   T2CON,2               ;Encendemos el Timer 2
      call  Test              ;Testeo si terminó la espera
      decf  Aux,1             ;Decremento la variable Aux
      btfss STATUS,2          ;Me fijo si llegó a cero
      goto  EsperaNotific     ;Si no llegó a 0 vuelvo a esperar
      return                            ;Si llegò a 0 regreso

banco0
      bcf   STATUS,RP1
      bcf   STATUS,RP0
      return

banco1
      bcf   STATUS,RP1
      bsf   STATUS,RP0
      return
end
```

# 11 Anexo IV - Acrónimos y abreviaturas

A/D = Conversor Analógico-Digital
ALU = Arithmetic Logic Unit
BB = BeagleBoard
CPU = Center Processing Unit
CUS = Conductor Universal en Serie
FET = Field Effect Transistor
GIE = Global Interruption Enable
GPR = General Purpose Register
GPS = Global Positioning System
IR = Infrarrojo
IRP = Indirect Register Pointer
ISP = Internet Service Provider
MDDR = Mobile Double Data Rate
NA = Normalmente Abierto
NC = Normalmente Cerrado
ONG = Organización No Gubernamental
PaP = Paso a Paso
PC = Program Counter
PCON = Power Control Register
POP = Package On Package
PSP = Parallel Slave Port
PWM = Pulse Width Modulation
RISC = Reduced Instruction Set Computer
SFR = Special Function Register
SSP = Synchronous Serial Port
TMR2IF = Timer2 Interrupt Flag
USART = Universal Synchronous Asynchronous Receiver Transmitter
USB = Universal Serial Bus
USB-OTG = USB On-The-Go
WDT = Watch Dog Timer