Trabajo de fin de carrera

Ingeniería técnica en informática de sistemas

Plataforma para el desarrollo de aplicaciones autónomas GPS y sistema para el registro de trayectorias

Jordi Bartolomé Ramos

9 de abril del 2005

Abstract

Este trabajo ha consistido en la realización de una plataforma hardware para el desarrollo de dispositivos GPS autónomos. Luego, esta plataforma se ha utilizado para crear un dispositivo capaz de registrar la trayectoria de un móvil en una memoria Flash. Se ha intentando, dentro de las posibilidades técnicas, minimizar el tamaño, precio y consumo del conjunto. También se ha procurado facilitar la interconexión del sistema con un ordenador personal con el fin de facilitar su utilización y el intercambio de datos entre ambos.

Resumen

El objetivo inicial de este proyecto era desarrollar un dispositivo autónomo, basado en la tecnología GPS, capaz de registrar en una memoria la trayectoria de un móvil, pero debido a que en el mercado ya existen implementaciones que ofrecen este servicio se ha decidido que sería más interesante crear una plataforma polivalente que permitiera desarrollar cualquier aplicación básica GPS , y luego aprovechar esta para desarrollar un montaje que satisfaciera las intenciones iniciales del proyecto. Así, el proyecto, primero ha consistido en crear la plataforma de propósito general, programable en placa, para el desarrollo de aplicaciones GPS. Luego, para demostrar las posibilidades de la placa de desarrollo, se ha diseñado el sistema para el registro de trayectorias de móviles que en un principio se tenía planeado diseñar.

Por tal de hacer de este un sistema autónomo y portátil se ha intentado minimizar el tamaño y consumo del conjunto. Otro punto importante, al tratarse de un sistema de desarrollo, ha sido ofrecer sencillez y el máximo número de recursos y facilidades al mínimo precio posible al potencial usuario. Todos estos factores han influido decisivamente en el diseño final de la placa y en la elección de los componentes.

Índice

Jordi Bartolomé Ramos

is10003

9 de abril del 2005

Capítulo 1
Introducción

1.1.Introducción

Hace poco más de 20 años conocer algo tan crucial como la ubicación geográfica de un móvil era una tarea lenta y relativamente compleja, sólo al alcance de personas con determinados conocimientos técnicos. La tecnología GPS ha cambiado radicalmente esta situación, y hoy en día esta tarea se ha convertido en un sencillo proceso automatizado que ha abierto un amplio abanico de posibilidades y aplicaciones en diferentes ámbitos.

El objetivo principal de este proyecto es aportar otro grano de arena al inmenso mundo de las aplicaciones GPS y ofrecer una solución sencilla y económica para aquellos que deseen disponer de una plataforma de desarrollo de aplicaciones GPS elementales. Se entiende por plataforma de desarrollo a un conjunto de elementos hardware y software que sirven como componentes y herramientas para facilitar el desarrollo de sistemas de complejidad superior. En el mercado ya existen completas plataformas de desarrollo de aplicaciones GPS, cada una con sus puntos fuertes: se ha intentado que esta sea lo más económica y sencilla de utilizar posible.

El objetivo secundario, que inicialmente era el motivo del proyecto, ha sido crear el sistema para el registro de la trayectoria de móviles para demostrar las posibilidades de la plataforma diseñada.

Se pretende también hacer de este un proyecto abierto, de ahí su nombre GPS Abierto. Por ello todo el código, esquemáticos e información recopilada durante el desarrollo se va a hacer libremente accesible a través de Internet con tal de servir a otros usuarios interesados en desarrollar aplicaciones similares o relacionadas.

1.2.Objetivos y especificaciones

Se creará una sencilla plataforma hardware sobre la que poder desarrollar aplicaciones autónomas basadas en GPS. La plataforma se compondrá de una parte hardware y de otra software. La parte hardware incluirá los componentes utilizados para la programación "in situ" del microcontrolador más los componentes necesarios para el procesado y recepción de la información enviada por el módulo GPS. La parte software, agrupará las aplicaciones necesarias para programar la parte hardware, más las librerías de soporte que se ejecutaran sobre esta. Resumiendo, las especificaciones iniciales del conjunto han sido:

-Crear un dispositivo hardware autónomo, minimizando el consumo, tamaño, y coste del conjunto.

-Ofrecer la posibilidad de programación directa "on board", sin necesidad de tener que extraer ni insertar ningún componente durante el proceso de programación. Esto obliga a crear el software de telecarga adecuado.

-El hardware ha de incorporar todos los componentes necesarios para la recepción y proceso de la información de posición GPS.

-Ofrecer las librerías necesarias para simplificar al usuario el acceso a la información GPS en el proceso de desarrollo.

-Simplificar al usuario , dentro de lo posible, la utilización de la plataforma.

De todos los requisitos, quizás el más importante, ha sido la correcta elección de los componentes hardware, ya que esto ha condicionado el tamaño, consumo, coste, modo de programación y complejidad final de la placa.

En cuanto a la segunda parte del proyecto, es decir, la correspondiente al registrador de trayectorias de móviles, se ha de intentar que cumplir las siguientes especificaciones:

-Funcionar en base a la plataforma de desarrollo GPS creada.

-Contar con un dispositivo de memoria eficiente sobre el que almacenar la información de posición.
-Disponer de una interfaz de entrada/salida sencilla que facilite la interacción con el usuario.
-Poder conectarse a un ordenador personal para transferir los datos registrados.

1.3.Estructura del documento

Llegados hasta este punto, ya se han visto cuales son los objetivos generales del proyecto, y es en los siguientes capítulos donde se detallan todos los aspectos relacionados con este.

En el siguiente capítulo, el segundo, se concretan los objetivos iniciales del proyecto, para a continuación realizar un diseño a muy alto nivel del conjunto, diferenciando los bloques funcionales que lo han de componer y cuales han de ser sus tareas.

El siguiente paso es diseñar adecuadamente cada uno de estos bloques funcionales o módulos que han de componer el proyecto. El diseño de cada una de las partes, casi siempre aparece precedido de una pequeña introducción teórica necesaria para comprender las particularidades de cada una de estas.

Así el tercer capítulo está dedicado al módulo receptor GPS, en este se explican los conceptos básicos de la tecnología GPS, y se describe el funcionamiento del módulo escogido. El capítulo cuarto justifica y describe el proceso de elección del microcontrolador y muestra también las características del modelo escogido. En el capítulo quinto se describe el circuito de alimentación utilizado para suministrar energía a todos los módulos del sistema. El siguiente capítulo, el sexto, muestra las alternativas inicialmente planteadas para la interconexión de los dispositivos, describiendo con detalle el bus escogido para la interconexión del microcontrolador con el módulo GPS y el ordenador. El capítulo séptimo describe el software utilizado para la programación de la placa de desarrollo, detallando el proceso ejecutado por este para telecargar el binario de una aplicación sobre el microcontrolador.

Hasta este punto, el documento se centra en los componentes elementales de la placa de desarrollo, su interconexión y las herramientas software disponibles para su utilización. En los siguientes capítulos se detallan los aspectos más importantes de la placa registradora de trayectorias el fin de la cual es demostrar la viabilidad del uso de la placa de desarrollo para la realización de una aplicación GPS.

Así el capítulo octavo describe la interfaz de usuario de la placa registradora de trayectorias que ha de servir al usuario para controlar el sistema. A continuación, en el capítulo noveno, se explica el proceso seguido para la elección del dispositivo de memoria donde se almacena la información de posición, entrando también en detalles sobre su funcionamiento. En el décimo capítulo se describen las distintas librerías firmware utilizadas para la realización del regitrador de trayectorias. En el capítulo décimo se muestra el funcionamiento y los detalles del software utilizado para la transferencia de la información almacenada en la placa registradora de trayectorias hacia el ordenador personal. El siguiente capítulo describe el firmware de la placa registradora de trayectorias, mostrando sus peculiaridades, flujo de ejecución ... Finalmente, en el capítulo décimo tercero, se exponen las conclusiones de este proyecto acompañadas de reflexiones sobre posibles mejoras de este. Este capítulo también muestra el coste económico del proyecto.

Capítulo 2
Arquitectura y diagrama de bloques del sistema

2.1.Introducción

En todo proyecto es fundamental un buen diseño, siendo este un requisito clave para el éxito del resultado final. El proceso de diseño se ha llevado a cabo partiendo de las especificaciones iniciales globales, subdividiendo el conjunto en diagramas de bloque lo más generales posibles que se han ido detallando a medida que ha ido avanzando el proceso de diseño. A continuación se describe la evolución del proceso de diseño.

2.2.Arquitectura del sistema

Se ha querido hacer un diseño lo más abierto posible utilizando componentes intercambiables por otros con diferentes prestaciones sin tener que modificar la estructura del resto de la placa, por ello se ha tendido a la utilización de estándares. Esto ofrece mayor capacidad de expansión y crecimiento de la placa. El primer paso ha sido la distinción de los diferentes elementos de hardware y software necesarios :

-Placa de desarrollo: esta ha de componerse de:

-Módulo encargado de la recepción de la información GPS.

-Unidad de proceso de la información de posición.

-Bus de interconexión con un ordenador personal.

-Bus para la conexión de placas de expansión con periféricos.

-Circuito de alimentación de los distintos componentes.

-Componentes adicionales.

-Software de programación

-Librerías firmware de soporte

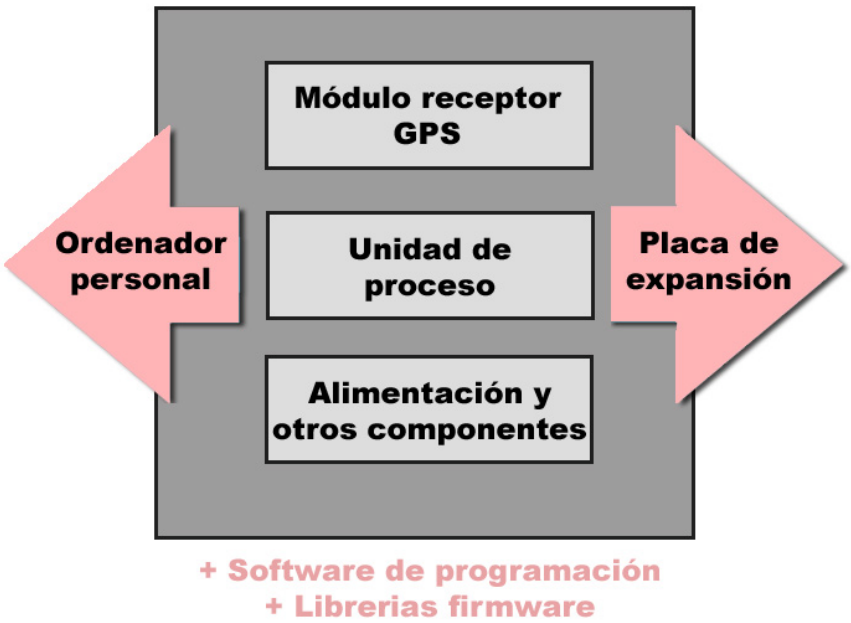


Fig 2.2.1 Esquema general de la placa de desarrollo

-Placa del registrador de trayectorias: esta ha de constar de:

-Interfaz de usuario: pulsadores y visualizador

-Memoria donde almacenar la información de posición.

-Software de transferencia de datos

-Firmware de la aplicación

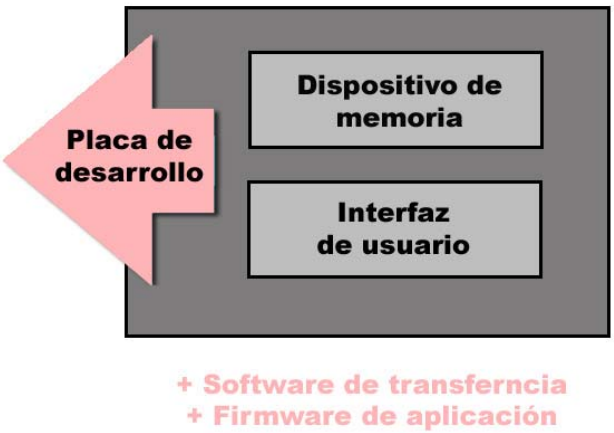


Fig 2.2.2 Esquema general de la placa de expansión.

2.3.Placa de desarrollo:

2.3.1.Módulo encargado de la recepción de la información GPS:

Este es el módulo encargado de captar la información de los satélites de la red NAVSTAR GPS y de calcular a partir de esta la posición geográfica. Es importante que el módulo seleccionado sea compatible con el estándar NMEA (National Marine Electronic Association) utilizado por la mayoría de los dispositivos GPS. Al tratarse de un estándar hardware (conectores, niveles eléctricos ...) y software (formato, protocolo ....) la compatibilidad con este permite que el módulo de recepción se pueda reemplazar por modelos de otras marcas sin necesidad de modificar en nada el diseño del sistema. Es decir, se ha decidido que el módulo a utilizar sea compatible con este estándar para hacer el sistema lo más abierto posible a otros receptores. Este protocolo, que se basa una transmisión serie convencional, es el utilizado para el envío de los mensajes del modulo receptor a la unidad de proceso.

Se podría utilizar con este fin un típico receptor GPS portátil con salida NMEA, pero esto incrementaría notablemente el tamaño, consumo y precio final del conjunto. Por ello la opción más interesante es la utilización de un módulo receptor autónomo compacto. En el mercado existen numerosos módulos autónomos de recepción GPS con interfaz NMEA, de todos ellos, y teniendo en cuenta los requisitos anteriormente comentados (precio, tamaño, consumo ...) se ha escogido el módulo Lassen SQ del fabricante Trimble. En el capitulo 3 se detallan sus prestaciones y protocolo de funcionamiento.

2.3.2.Unidad de proceso de la información de posición:

Este es el bloque encargado de procesar la información del receptor GPS y de ejecutar el código de usuario para el tratamiento de esta. Algunas alternativas para su implementación serian utilizar una FPGA (Field Programable Gate Array) , un CPLD (Complex Programable Logic Device) o un microcontrolador. Las FPGAs y CPLDs quedan bastante alejadas de los objetivos de este proyecto, ya que estos son dispositivos enfocados a otro contexto, como el del desarrollo de sistemas lógicos o de proceso especifico, además estas requieren conocimientos de lenguajes muy especializados para

su uso. De las tres opciones, la del microcontrolador es la más interesante ya que estos ofrecen de sobras las prestaciones necesarias para el proyecto. Y además de que existen infinidad de modelos con variados periféricos, son dispositivos programables mediante lenguajes de programación de propósito general muy extendidos como C o C++.

En la elección del microcontrolador se ha valorado que este dispusiera de suficientes prestaciones hardware (memoria de programa, memoria de datos, puertos de E/S, UART, SPI ...), y sobre todo de un entorno de programación gratuito. También se ha considerado importante la disponibilidad de modelos compatibles con prestaciones superiores por tal de ampliar las prestaciones del sistema sin modificar el diseño en caso de que fuera necesario. Debido a las limitaciones técnicas en el proceso de montaje el microcontrolador escogido también debe estar disponible en encapsulado DIP.

Tras contrastar diferentes opciones y considerando los requisitos y limitaciones técnicas, las opciones más interesantes han sido las correspondientes a las familias de 8 bits PIC 16F de Microchip o AVR-Atmega de Atmel. Tras evaluar diferentes parámetros se ha decidido recurrir a la segunda. En el capítulo 4 se justifica su elección y se describen sus prestaciones.

Hay que comentar, que durante el desarrollo de este proyecto, Microchip anunció la inminente salida al mercado de su serie PIC 18F con bastantes mejoras respecto a su predecesora, la serie PIC 16F. Si esta hubiese aparecido en el mercado unos meses antes, se hubiesen podido estudiar mejor sus prestaciones, y probablemente la elección entre PIC o AVR-ATmega hubiese sido más difícil. Quizás se hubiese optado por escoger una PIC18 en lugar de un AVR-ATmega.

2.3.3.Bus de interconexión con un ordenador personal:

Es necesario que el dispositivo disponga de un bus de interconexión con un ordenador personal que permita programarlo directamente desde este sin tener que extraer ningún componente. Esto disminuirá el tiempo de turn-around (tiempo que pasa desde que se modifica el código hasta que se comprueban los resultados) mejorando, facilitando y acelerando el desarrollo de las aplicaciones al usuario final. Este bus también se ha de

poder utilizar para la trasmisión de datos desde la placa al ordenador personal o viceversa en caso de que sea necesario.

Existen infinidad de posibilidades para la comunicación entre microcontroladores: I2C, SPI, CAN, puerto serie UART , puerto paralelo ..., pero los ordenadores personales solo ofrecen tres posibles soluciones nativas para este fin: el puerto serie COM, el puerto paralelo LPT , o el bus USB, así que la placa deberá permitir la conexión a alguno de estos tres buses. La elección de la familia de microcontroladores AVR-ATmega ha condicionado la elección del tipo de bus, ya que estos no disponen de una implementación USB interna (habría que añadir otro integrado más al diseño aumentando así la complejidad de este). Así finalmente se ha decidido dotar a la placa de un puerto paralelo y un puerto serie: mediante el puerto paralelo se puede programar la placa o enviar y recibir datos al ordenador, mientras que el puerto serie solo se utilizará para enviar datos al ordenador y recibir la información del módulo GPS. De hecho, mediante unos jumpers el puerto serie también se puede configurar para recibir datos. De todas formas, como la familia de microcontroladores AVR-Atmega crece rápidamente, no hay que descartar que en un futuro exista algún miembro de esta con posibilidades USB. En el capítulo 5 se describen con detalle los modos de comunicación de la placa.

2.3.4.Bus para la conexión de placas de expansión con periféricos:

Además la placa de desarrollo debe disponer de algún conector que dé acceso a los puertos del microcontrolador para poder conectar a este la placa de expansión con los periféricos diseñados por el usuario. Se ha decidido utilizar un conector típico de bus para cinta plana de 40 pins DSUB-40 ya que este satisface de sobra los requisitos de tamaño, precio y sencillez de montaje.

2.3.5.Circuito de alimentación de los distintos componentes:

El sistema debe disponer de un sistema de alimentación en la propia placa para evitar tener que hacer uso siempre de una fuente externa y poderle así conectar directamente una batería en caso de que fuese necesario. Este circuito ha de ajustarse a las

necesidades de alimentación del microcontrolador y del módulo de recepción GPS suministrando siempre la tensión adecuada. Ha de ser capaz también de alimentar la placa de expansión. Lo ideal sería optimizar el rendimiento de este circuito por tal de alargar al máximo la duración de las baterías en el caso de que el sistema hiciera uso de ellas. Finalmente se ha decidido utilizar un circuito de alimentación típico basado en el regulador lineal LM317 que aunque no es la solución óptima funcionará correctamente. En el capítulo 6 se justifica su elección y también se describe su estructura.

2.3.6.Componentes adicionales:

Son el resto de componentes necesarios para el correcto funcionamiento del módulo receptor GPS, microcontrolador y demás componentes de la placa. Las limitaciones técnicas de montaje han condicionado el uso de componentes con encapsulado DIP-DIL convencional. Lo ideal, por tal de disminuir el tamaño de la palca, hubiese sido utilizar componentes y encapsulados del tipo SMD, pero la falta de herramientas adecuadas para su manipulación lo ha impedido.

2.3.7.Software de programación:

Es necesario disponer de una aplicación que se ejecute en el ordenador personal y que permita programar la placa de desarrollo: esta ha de tomar el archivo .hex con el código máquina generado por el compilador y transferirlo a la memoria de programa del microcontrolador para que este pueda ejecutarlo. Para el desarrollo de esta se ha decidido utilizar el entorno de desarrollo Visual Studio .NET, más en concreto Visual Basic .NET con las correspondientes librerías de acceso a los puertos de Entrada/Salida (INPOUT32.DLL , OUTPORT.DLL) que han permitido el acceso al puerto paralelo. El entorno .NET permite desarrollar rápidamente completas aplicaciones con los componentes estándar de Windows (formularios con botones, cajas de texto ...) . Dadas las características de la plataforma .NET, hubiese sido lo mismo utilizar el lenguaje Visual C# .NET, de hecho la elección de Visual Basic .NET ha sido debida más a gustos personales que a otra cosa. Se puede decir que las prestaciones de los dos lenguajes son prácticamente las mismas y solo difieren en la sintaxis. En el capítulo 7

se describe el proceso de telecarga seguido por la aplicación desde que lee el archivo .hex hasta que lo transfiere a la placa de desarrollo.

2.3.8.Librerías firmware de soporte:

Las librerías firmware de soporte tienen como objetivo ofrecer al usuario una interfaz sencilla para facilitarle el acceso a la información suministrada por el GPS evitándole tener que generar todo el código para la recepción y gestión de datos del GPS. Así, mediante acceso a los diferentes campos de una estructura de datos de esta librería, el usuario podrá conocer cualquiera de los datos suministrado por el GPS. Esta librería se enlaza en el compilador C como las demás librerías C o C++, y si el usuario lo desease podría prescindir de ellas simplemente no incluyéndolas en el proyecto.

2.4.Placa del registrador de trayectorias:

2.4.1.Interfaz de usuario: visualizador y pulsadores:

El registrador de trayectorias construido sobre la placa de desarrollo debe disponer de un sistema de visualización alfa-numérico a través del cual mostrar al usuario las diferentes opciones del menú y los valores de las diferentes variables suministradas por el módulo receptor GPS. En consonancia con el resto de componentes este debe también ser económico, pequeño y consumir poca corriente y pocos recursos del microcontrolador. La solución ideal es la utilización de un LCD 2x20 compatible con el estándar HD44780: estos consumen poco (sin retroiluminación), se controlan mediante un protocolo sencillo, y ocupan solo 4+3 o 8+3 bits de entrada /salida.

Debe disponer también de un conjunto de pulsadores por tal de poder manipular los menús y seleccionar las diferentes opciones de este. Si fueran necesarias muchas teclas la opción óptima sería utilizar un teclado de matriz por tal de ocupar el mínimo numero de puertos, pero como solo se necesitan tres teclas (dos para avanzar o retroceder en los menús y otra para seleccionar las opciones) con tres pulsadores bastará.

2.4.2.Memoria donde almacenar la información de posición:

Toda la información suministrada por el módulo GPS se ha de ir almacenando en un dispositivo de memoria no volátil suficientemente grande para almacenar el mayor número de muestras posible. Siguiendo la filosofía del proyecto, se ha considerado que lo ideal es utilizar una memoria Flash extraíble, dejando así abierta la posibilidad de instalar una memoria de mayor o menor tamaño en función de las necesidades del usuario. Si se utilizase una memoria Flash integrada en la placa existiría el inconveniente de que en caso de que se deseara disponer de mayor capacidad se tendría que desoldar el integrado o cambiar el diseño. Por tanto la opción a considerar es la utilización de una tarjeta de memoria extraíble.

En el mercado existen diferentes modelos de tarjetas de memoria extraíbles, prácticamente todas de tecnología Flash: Compact Flash, Memory Stick, Smart Card, Secure Disk, MultiMediaCard y sus versiones compactas (RS-MMC, Mini-SD). De todas ellas se ha decidido utilizar la opción de las MultiMediaCard, ya que estas se acceden mediante un protocolo serie que ocupa muy pocos puertos de entrada/salida , son muy pequeñas, económicas, están muy extendidas y existen diferentes modelos con distintas capacidades. Otro punto a favor de estas ha sido la disponibilidad de documentación sobre su funcionamiento, cosa que no sucede con algunas de las otras tarjetas. En el capítulo 9 se explica con detalle el modo de utilización y funcionamiento de estas tarjetas.

2.4.3.Software de transferencia de datos:

Esta aplicación tiene como objetivo permitir la transferencia de los datos almacenados en la tarjeta de memoria MultiMediaCard hacia el ordenador personal para luego poderlos tratar o visualizar. Al igual que la aplicación de programación de la placa de desarrollo, está se ha desarrollado en la plataforma Visual Studio .NET, pero en este caso, en lugar de utilizarVisual Basic .NET, se ha utilizado Visual C# .NET. El motivo de usar Visual C# .NET ha sido simplemente el de practicar con el nuevo lenguaje de Microsoft, ya que Visual Basic .NET hubiese servido igualmente para este fin.

Como la transferencia de la información placa – ordenador personal se hace a través del puerto serie ha sido necesario buscar un control para poder acceder al puerto serie desde la aplicación. Se ha utilizado el control NetComm de Richard Grier. A pesar de que para tener una mayor velocidad de transferencia, inicialmente se consideró utilizar la opción de transferir la información a través del puerto paralelo, finalmente se desechó debido a problemas de lentitud en las librerías utilizadas para el acceso al puerto paralelo desde Visual Studio .NET.

A parte de permitir la transferencia de la información de la placa al ordenador personal, esta aplicación también permite exportar los datos obtenidos a formatos compatibles con Excel, Microsoft MapPoint, Ozi Explorer u otros, lo que ofrece grandes posibilidades a la hora de “explotar” la información registrada.

Capítulo 3
El módulo receptor GPS

3.1.Introduccion

Tal como se comenta en el capítulo anterior, se ha decidido utilizar un módulo autónomo compacto para la recepción de la información GPS. En el mercado se encuentran disponibles gran variedad de modelos, y después de evaluar parámetros como el precio, tamaño, consumo y disponibilidad se ha decidido utilizar el modelo Lassen SQ de la casa Trimble. Antes de mostrar sus especificaciones y por tal de comprender de que se habla, es conveniente conocer algunos entresijos de la tecnología GPS, las personas con estos conocimientos pueden obviar el siguiente apartado.

3.2.Tecnología GPS

El sistema de posicionamiento GPS (Global Positioning System) fue diseñado por el Departamento de Defensa de los Estados Unidos (D.O.D.) con la intención de sustituir al sistema de posicionamiento TRANSIT usado desde 1965 por la marina de ese país. El primer satélite GPS se lanzó en el año 1978 pero no fue hasta mediados de los ochenta cuando el sistema comenzó a estar parcialmente operativo. El 8 de diciembre de 1993 la red de 24 satélites estuvo completa. Desde entonces no han cesado de surgir nuevas aplicaciones de esta tecnología en campos como la navegación, cartografía, geodesia, topografía, sistemas de información geográfica, mercado de recreo...

La tecnología GPS es un sistema relativamente complejo basado en tres grandes bloques:

-La red NAVSTAR de satélites encargados de trasmitir la información de posición.

-Las estaciones terrestres encargadas de controlar los satélites y actualizar su información.
-Los receptores que recogen la información trasmitida por los satélites y la computan para obtener la información de posición.

Existe el malentendido de que el receptor GPS de alguna forma envía datos a los satélites para conocer su posición, esto no es cierto, tal como indica su nombre el receptor solo recibe información.

3.2.1.Funcionamiento de los receptores

Los receptores GPS usan una adaptación de la técnica que se ha utilizado durante siglos por los navegantes. Esta es la de calcular la posición actual en base a un conjunto de posiciones conocidas, de forma que un observador puede deducir su posición, conociendo las distancias a que se encuentra de diferentes puntos. Conociendo la distancia respecto a un primer punto, mediante un compás , el observador puede trazar una circunferencia alrededor de este en el mapa. El observador se encontrará en cualquier punto de esta circunferencia (Fig 3.2.2.1). Conociendo la distancia respecto a un segundo punto y repitiendo el proceso anterior el observador acotará 2 puntos de esta circunferencia (Fig 3.2.2.2).

Repitiendo el proceso sobre un tercer punto este podrá saber sobre cual de los dos puntos se encuentra (Fig 3.2.2.3). Esto es la teoría, pero generalmente las medidas de distancia no eran nada precisas, esto hace que las circunferencias en lugar de intersectar en un punto delimiten una superficie triangular dentro de la que se encontrará el observador, si este triangulo es suficientemente pequeño las medidas se dan por buenas.

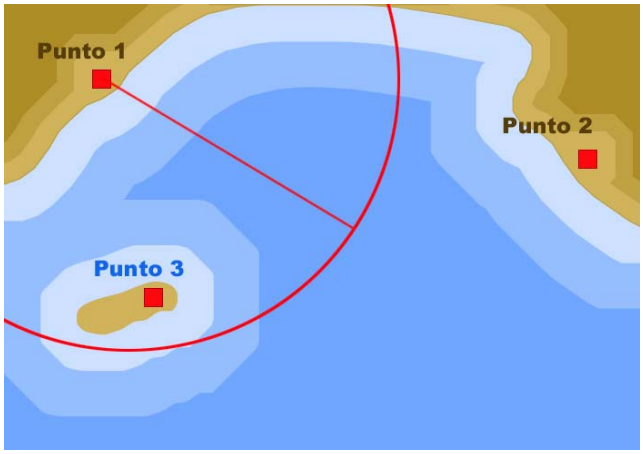


Fig 3.2.1.1 El observador puede estar en cualquier punto de la circunferencia

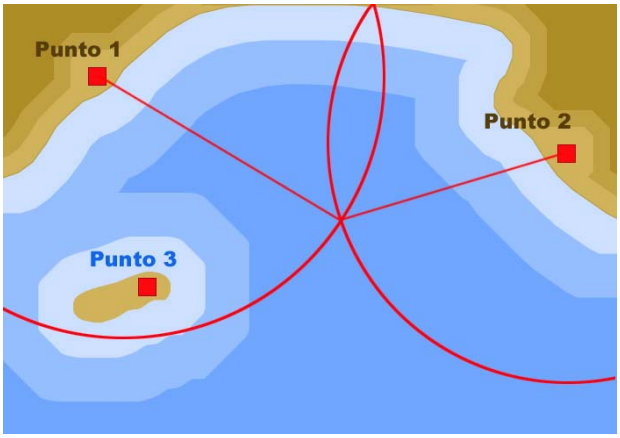


Fig 3.2.1.2 El observador puede estar en cualquiera de los dos puntos de la intersección.

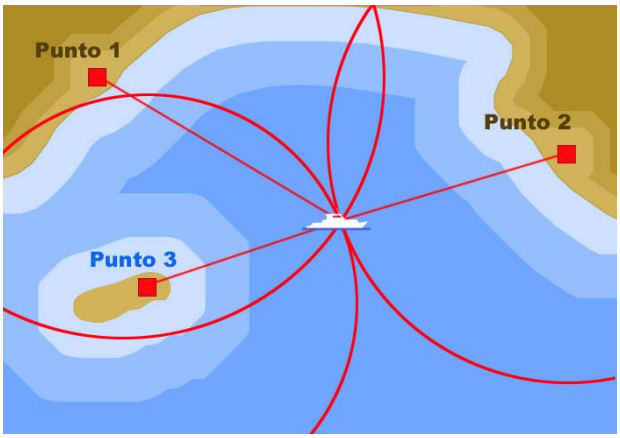


Fig 3.2.1.3 Tras trazar la tercera circunferencia se deduce la posición exacta del observador.

Los receptores GPS utilizan un método parecido a este para conocer su posición, pero en lugar de tomar como referencia ubicaciones geográficas toman como referencia los satélites de la red NAVSTAR. Simplificando mucho, el método utilizado para calcular

las distancias respecto a estos satélites es el siguiente: sabiendo que la señal se trasmite a la velocidad de la luz, el receptor GPS calcula la distancia de este respecto al satélite en base al tiempo que transcurre desde que la señal sale del satélite hasta que llega al receptor. De hecho cada satélite incorpora hasta 4 relojes atómicos de alta precisión (su error es de 1 segundo cada 70.000 años) para dar medidas de tiempo lo más exactas posibles. Por razones obvias los receptores no pueden incorporar este tipo de relojes y es por eso que sus relojes no son tan precisos, lo que causa errores en el cálculo de la posición. Estos errores se pueden minimizar haciendo uso de la información de más satélites de la red.

Un inconveniente respecto al método tradicional, es que, a diferencia de las ubicaciones geográficas utilizadas en este, los satélites no son estáticos, sino que se desplazan. Por ello, en la señal utilizada para calcular la distancia, los satélites también envían información sobre su posición. Así al final, los receptores disponen de la posición exacta de cada satélite y también de la distancia a que se encuentran de estos: estos dos parámetros permiten definir una esfera para cada satélite, con centro en la posición de este y con radio la distancia entre el satélite y el receptor. Los satélites emiten dos portadoras L1 = 1575.42 MHz para la información de navegación y L2 = 1227.6 MHz para la medición el retardo ionosférico. La información de navegación transmitida por cada satélite incluye el almanaque de todo el sistema GPS, su propia efemérides y su propia corrección de reloj. El almanaque contiene información sobre de todos los satélites de la constelación, datos de la ionosfera y otros mensajes especiales del sistema. El almanaque del GPS se actualiza semanalmente y suele ser valido para meses. La efemérides contiene información detallada de la orbita de cada satélite. La efemérides cambia cada hora pero es válida durante 4 horas. El almanaque y efemérides son necesarios para que el GPS comience a funcionar adecuadamente.

En lugar de utilizar circunferencias como en el método tradicional, los sistemas GPS utilizan esferas por tal de determinar la posición del receptor sobre el geoide terrestre. Con 4 satélites es suficiente para determinar la posición: conociendo la posición y distancia del primer satélite se obtiene la primera esfera, pudiendo estar el receptor en cualquier punto de la superficie de esta (Fig 3.2.2.4 **)**. Con la información del segundo satélite se obtiene una segunda esfera que se solapa parcialmente con la primera, esto

acota la posición del receptor a la circunferencia resultante de la intersección de estas dos esferas (Fig 3.2.2.5 **)**. El tercer satélite proporciona una tercera esfera que intersecta con la circunferencia anterior, dando lugar a dos puntos como posible ubicación del receptor (Fig 3.2.2.6 **)**. Finalmente, si los dos puntos obtenidos son congruentes ,es necesario utilizar un cuarto satélite o hacer uso de información como la velocidad para saber cual de los dos corresponde a la verdadera ubicación del receptor. Las figuras 3.2.2.4 , 3.2.2.5 y 3.2.2.6 muestran gráficamente este proceso.

Fig 3.2.1.4 Mediante la posición y distancia al primer satélite se obtiene la primera esfera.

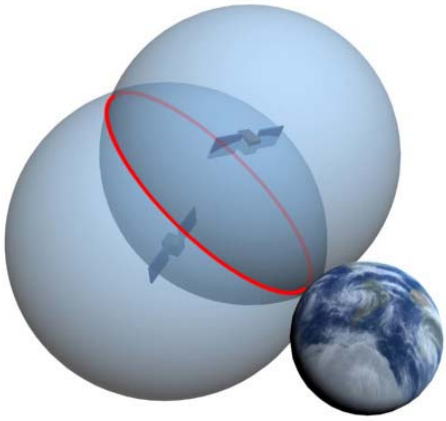


Fig 3.2.1.5 La intersección entre la segunda esfera y la primera da lugar a una circunferencia pudiendo estar el receptor en cualquier punto de esta.

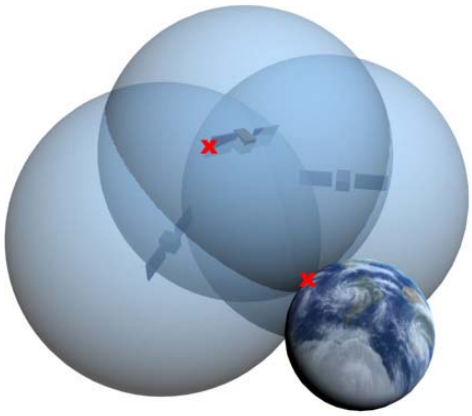


Fig 3.2.1.6 La intersección de la tercera esfera con la circunferencia anterior da dos puntos como posible ubicación del receptor.

Otro parámetro interesante de los receptores GPS es el número de canales: cuantos más canales tienen más satélites pueden monitorizar, pudiendo seleccionar así aquellos que permiten obtener mejores resultados en el cálculo de la posición.

3.2.2.La red NAVSTAR y las estaciones terrestres

Visto el proceso realizado por el receptor para la obtención de la posición se deduce que para que todo el sistema GPS funcione correctamente es necesario disponer de un grupo grande de satélites para ofrecer suficiente cobertura, y de algunas estaciones terrestres para el control de estos.

La red de satélites (SVs Space Vehicles), conocida como la constelación NAVSTAR, está formada por un grupo de 21 satélites (más 3 de reserva) situados a 20.000 Kms de la Tierra. El total de satélites puede crecer hasta 32. Estos están distribuidos en 6 orbitas, cada una de ellas inclinada 55º respecto el ecuador y 60º respecto su predecesora. Ninguna de estas pasa directamente por los polos pero es allí donde se pueden captar más satélites simultáneamente. El objetivo de esta distribución es ofrecer siempre como mínimo 4 satélites visibles, en realidad siempre hay más de 4 satélites visibles, en ocasiones hay hasta 12 . Cada satélite tarda 11 horas y 58 minutos en dar una vuelta completa a su órbita (tienen un pequeño offset de 2+2 minutos diarios).

Estos satélites cuentan con un depósito de fuel y servo-sistemas para poder desplazarlos en caso de que se necesite corregir su trayectoria. Esta tarea se realiza desde un conjunto de estaciones terrestres ubicadas en posiciones geográficas muy precisas y con suficientes recursos (antenas, relojes atómicos, hardware de control...) para monitorizar en todo momento el estado y trayectoria de los satélites. Comprueban si hay fallos en la trayectoria de este corrigiéndola si es necesario, comprueban y ajustan los relojes de los satélites.

3.3.El módulo GPS Lassen SQ

Tal como se comenta en el capítulo anterior, tras estudiar diferentes modelos finalmente se ha decidido utilizar el módulo Lassen SQ de Trimble. El módulo Lassen SQ es un receptor encapsulado GPS muy compacto diseñado para aplicaciones móviles, de ahí sus reducidas dimensiones y bajo consumo. Este proporciona toda la información relativa a la posición a través de un puerto serie que puede trabajar con dos protocolos distintos: el protocolo TSIP (Trimble Standar Interface Protocol) y el estándar NMEA 0183 (National Marine Electronics Association). El protocolo nativo del módulo es el protocolo bidireccional TSIP de Trimble, y a parte de permitir realizar todas las operaciones típicas relacionadas con la información GPS, también es el protocolo utilizado para la configuración de este. En este protocolo los datos se transmiten en binario. El protocolo NMEA es un estándar establecido por la industria naval, y es parcialmente soportado por el módulo Lassen SQ ya que este sólo implementa un subconjunto de todos los mensajes MNEA. En este protocolo los datos se transmiten en ASCII por lo que se pueden capturar mediante cualquier aplicación de comunicación serie como el Hyperterminal de Windows. Así el protocolo TSIP permite exprimir más las funcionalidades del módulo Lassen SQ, mientras que el protocolo NMEA 0183 ofrece más sencillez y compatibilidad al módulo con otros dispositivos. En resumen, las principales características de este son:

-Especificaciones físicas y eléctricas: este viene ensamblado en una funda metálica cuadrada de 26x26mm y consume solo 100mW a 3.3V, 133mW con la antena.

-Resolución: en cuanto a la resolución horizontal de las medidas, el 50% de estas tienen un error inferior a los 6m, y el 90% inferior a los 9m. Respecto a la resolución vertical

(altura), el 50% de las medidas tienen un error inferior a los 11 metros, y el 90% inferior a los 18m. El error en la adquisición de velocidad es de 0.06 m/s.

-Límites de operación: este puede operar a una altura máxima de 18000m y a una velocidad de hasta 512 m/s

-Protocolos: este puede trabajar con el protocolo NMEA 0183 v3 (mensajes GGA, VTG, GLL, ZDA, GSA, GSV y RMC) y el protocolo TSIP del propio fabricante.

-Antenas: es compatible con antenas activas de 3.3VDC, de tipo compacto para facilitar su integración, o de tipo externo para facilitar su ubicación en zonas de mala cobertura.

-Conectores: la conexión entre el módulo y el microcontrolador se hace a través de un conector 2x4 (ASP 69533-01). Este conector incluye la alimentación principal, la alimentación de backup y el puerto serie. La antena se conecta al módulo a través de un conector coaxial de bajo perfil (H.FL-R-SMT 10, 50 Ohm).

Fig 3.3.1 Módulo receptor GPS Lassen SQ (26x26x6mm)

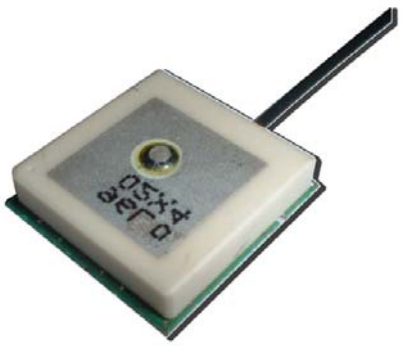


Fig 3.3.2 Antena compacta (19x19x7mm)

Fig 3.3.3 Antena externa (50x40x10mm)

3.4.Protocolo NMEA

El estándar NMEA 0183 implementado en el módulo Lassen SQ permite la comunicación serie unidireccional del receptor GPS (talker) hacia uno o varios dispositivos (listeners). La comunicación série ha de cumplir las siguientes especificaciones:

Baud Rate	4800
Bits de Datos	8
Paridad	ninguna
Bits de parada	1
Tensiones	0

Aunque según NMEA 0183 el baud rate ha de ser de 4800 y sin paridad, el módulo Lassen SQ puede trabajar también con configuraciones distintas, aunque por razones de compatibilidad con otros dispositivos es aconsejable ceñirse a las del estándar. También hay que tener en cuenta que la configuración del puerto serie de este es única, es decir que si se configura un baud rate de 4800 para la salida de datos, entonces la entrada también será a 4800, por lo que no es posible configurar velocidades de entrada y salida distintas. Otro punto importante es que aunque el módulo esté trabajando en modo de salida NMEA este puede seguir recibiendo comandos TSIP, eso sí, al mismo baud rate a que este configurada la salida NMEA.

El protocolo NMEA comprende gran variedad de datos relacionados con la navegación, los cuales se estructuran en mensajes: cada uno de estos mensajes está formado por diferentes campos con un tipo de información concreta. El módulo Lassen SQ sólo

implementa un subconjunto de los mensajes del estándar NMEA, pero estos son más que suficiente para cubrir las principales funcionalidades GPS. La estructura general de estos mensajes es la siguiente:

$IDMSG,D1,D2,D3,D4, ... , DN*CS [CR] [LF]

$	Byte que indica el inicio del mensaje.
ID	2 bytes que indican la fuente de la información (GP indica que la fuente es un GPS).
MSG	3 bytes que identifican el tipo de mensaje que viene a continuación.
D1..DN	Son los diferentes campos con los datos, cada uno de los cuales tiene un número distinto de bytes en función del tipo de mensaje y campo de que se trate.
*	Byte que delimita los bytes de checksum.
CS	2 bytes con el valor hexadecimal del checksum.
[CR][LF]	Bytes de salto de carro [CR] y de [LF] alimentación de línea indicando el fin del mensaje.

3.4.1.Mensajes NMEA soportados por Lassen SQ

Los mensajes NMEA soportados por el módulo Lassen SQ son los siguientes:

GGA: GPS Fix Data

GLL : Geografic position – Lattitude/Longitude

GSA: GPS DOP and Active Satellites

GSV: GPS Satellites in view

RMC: Recomended Mínimum Specific GPS / Transit data

VTG: Track Made Good and Ground Speed

ZDA: Time & Date

3.4.2.Configuración

Para inicializar el módulo Lassen SQ en modo NMEA hay que enviar a este el comando TSIP de configuración de protocolo por el puerto serie. Para ello, antes hay que asegurarse que las configuraciones de los puertos serie del equipo y del módulo coinciden ya que de lo contrario no se establecerá comunicación entre los dos.

A no ser que se haya reconfigurado, el módulo viene preparado de fábrica para trabajar

con las siguientes especificaciones (Default Factory Settings) :

Entrada.	Salida:
Protocolo:	TSIP
Baud:	9600
Paridad:	Odd
Data Bits:	8
Stop Bits:	1

Tras verificar que se esta trabajando con la misma configuración que la del puerto serie ya se puede inicializar el módulo para trabajar en modo MNEA mediante los correspondientes comandos TSIP. Todos los comandos TSIP comienzan con 0x10, a este le siguen el byte de comando y los bytes con los parámetros de los comandos, finalizan con los bytes 0x10 y 0x03 los cuales indican el fin del comando. Una secuencia típica de inicialización completa del modulo Lassen SQ para trabajar en mod NMEA sería:

-Enviar el comando “Clear Battery Backup, then Reset: Factory Reset” (0x10 0x1E 0x46 0x10 0x03). Esto resetea completamente el módulo y lo deja tal como salió de fábrica (Default Factory Settings). Si esto ha modificado los parámetros anteriores del puerto serie del módulo también se han de reconfigurar los del equipo por tal de mantenerlos igual configurados. Es importante recordar que este comando también resetea el almanaque y las efemérides almacenadas en este, por lo que el modulo tardará unos 15 minutos en descargárselas y volver a dar datos GPS validos.

-Tras inicializar el modulo, este se encuentra en modo TSIP input y TSIP output: para configurarlo en modo estándar NMEA output hay que enviar el comando “Protocol configuration: Default, 4800baud, 4800baud, 8, No Parity, 1 stop bit, TSIP, MNEA” (0x10 0xBC 0xFF 0x06 0x06 0x03 0x00 0x00 0x00 0x02 0x04 0x00 0x10 0x03) . Tras recibir este comando el modulo pasara a trabajar con NMEA como protocolo de salida, y si hay satélites visibles y dispone del almanaque y efemérides comenzará a enviar tramas NMEA con información GPS.

-Al inicializarlo en modo NMEA output por defecto este solo emite tres tramas NMEA (GGA, VTG, y GSV) a una cadencia de un segundo. Mediante comandos TSIP se puede indicar al módulo que tramas NMEA se desea que genere y la cadencia de estas. Algunos ejemplos son:

MNEA message configuration: 1sec , GGA
0x10 0x7A 0x00 0x01 0x00 0x00 0x00 0x01 0x10 0x03
MNEA message configuration: 1sec , GLL
0x10 0x7A 0x00 0x01 0x00 0x00 0x00 0x02 0x10 0x03
MNEA message configuration: 1sec , VTG
0x10 0x7A 0x00 0x01 0x00 0x00 0x00 0x04 0x10 0x03
MNEA message configuration: 1sec , GSV
0x10 0x7A 0x00 0x01 0x00 0x00 0x00 0x08 0x10 0x03
MNEA message configuration: 1sec , GSA
0x10 0x7A 0x00 0x01 0x00 0x00 0x00 0x10 0x10 0x03
MNEA message configuration: 1sec , ZDA
0x10 0x7A 0x00 0x01 0x00 0x00 0x00 0x20 0x10 0x03
MNEA message configuration: 1sec, GGA , VTG , ZDA
0x10 0x7A 0x00 0x01 0x00 0x00 0x00 0x25 0x10 0x03

Capítulo 4
La unidad de proceso de la información GPS: El microcontrolador

4.1.Introducción

Tal como se comenta en el capítulo 1 se ha decidido utilizar un microcontrolador como unidad de proceso para la ejecución del firmware de usuario y del procesado de la información GPS, puesto que las FGPAs y CPLDs son dispositivos muy potentes pero orientados a otro contexto como es el diseño de hardware específico. El perfil de un microcontrolador encaja perfectamente en las necesidades de este proyecto: se necesita un dispositivo compacto con capacidad de proceso, económico, y sobre todo que se pueda programar en un lenguaje de programación de alto nivel, como C o C++.

4.2.Los parámetros básicos del microcontrolador

La comprobación y valoración de las especificaciones de los microcontroladores es una tarea bastante subjetiva, ya que no es posible dar unas reglas rápidas y exactas que permitan compararlos entre si de forma decisiva. En la mayoría de los casos, el punto de partida son los requerimientos de la aplicación a la cual están destinados.

Los siguientes parámetros no son una guía estricta para la comparación de microcontroladores, pero sí han ofrecido un punto de partida importante en la valoración durante el proceso de elección de este componente:

-Velocidad: la velocidad de un microcontrolador no depende sólo de la máxima frecuencia de reloj de la CPU y el generador de reloj (cristal de cuarzo). También hay que tener en cuenta el número de ciclos de reloj que el microcontrolador utiliza para ejecutar una instrucción, así como el leguaje de programación empleado (el lenguaje ensamblador puede ser varias veces mas rápido que un lenguaje de alto nivel).

-Memoria de programa: el programa que tiene que ejecutar el microcontrolador se almacena en una memoria no volátil. Una memoria interna del tipo EPROM OTP solamente puede grabarse una vez, por lo que , normalmente, durante la fase de desarrollo del programa se utilizan microcontroladores mas caros que disponen de memoria Flash interna. La memoria Flash puede grabarse en segundos y, con la misma sencillez, también puede borrarse. Esta tarea se realiza mediante un programador o una programación en el propio circuito impreso (ISP). Hoy en día, el tamaño de una memoria de programa interna convencional está en el rango de 0 a 1024 Kbytes. De memoria Flash (quizás incluso más).

Los microcontroladores que disponen de una pequeña ventana para el borrado de la memoria EPROM, utilizando luz ultravioleta, están ya obsoletos. Del mismo modo, actualmente sólo se emplean las memorias EPROM externas con ventanas en el caso de que estemos trabajando con programas muy largos. La memoria RAM Flash ya ha incrementado su uso en lugar del tipo de memoria ERPOM.

-Memoria EEPROM: cuando las variables del programa tienen que almacenarse, incluso si el microcontrolador está desconectado completamente, la memoria EEPROM no volátil, interna o externa, es la que se usa más frecuentemente. Contrariamente a lo que la mayoría de la gente cree, el número de operaciones de escritura que soporta una memoria EEPROM no es infinito. En general, las memorias EEPROM externas se conectan al microcontrolador a través de un bus de dos hilos. Estas memorias se utilizan normalmente cuando ciertos datos específicos de la aplicación (como variables de calibración), tienen que ser leídos al inicio del programa.

-Memoria RAM: la memoria RAM se emplea para almacenar variables durante la ejecución del programa . La memoria RAM interna de los microcontroladores está

normalmente limitada a 4Kbytes y los requerimientos actuales normalmente son bastante inferiores. También es posible utilizar una memoria RAM externa.

-Entradas/Salidas: el número del líneas de E/S digitales que podemos necesitar en nuestra aplicación especifica es otro parámetro importante en el proceso de decisión. Sin embargo, si los recursos internos del microcontrolador son un poco limitados, las líneas de E/S también pueden usarse para conectar otros circuitos periféricos en caso de que nuestro microcontrolador no disponga de los necesarios. En algunos casos, se requieren puertos completos en modo paralelo y estos son los momentos en los que hay que recurrir a procesadores un poco más completos, como sucede en este proyecto.

-Temporizadores/Contadores: si un programa tiene periodos de medida o contadores de eventos, quiere decir que el microcontrolador elegido debe tener temporizadores y/o contadores internos. Afortunadamente, la mayoría de los modelos actuales contienen un mínimo de 3 temporizadores y contadores, cada uno de ellos de 8 o 16 bits, que están bajo el control estricto de los registros internos. Los temporizadores y contadores también son necesarios para generar una señal de reloj independiente (PWM, UART) en caso de que la necesitemos.

El temporizador de vigilancia (watchdog) es un caso especial. Este dispositivo se configura a un intervalo determinado por el programa que se está ejecutando, generando una señal de reset si este no es actualizado por el programa.

-Interrupciones externas: no sólo los temporizadores y los contadores generan interrupciones. Cuando un evento externo tiene que detener la ejecución del programa principal y forzar la ejecución de la correspondiente rutina de servicio de interrupción , es necesario disponer de una o varias líneas de interrupción externas. Hoy en día la mayoría de microcontroladores disponen de una de estas líneas.

-Interfaces: siempre será muy útil disponer internamente en los microcontroladores que elijamos de algunos interfaces estándares definidos en la industria, como pueden ser el I2C, I2S, SPI, CAN, USB, LIN y algunos para la tradicional pantalla LCD. Es verdad que algunos de estos interfaces pueden ser emulados en el propio programa, pero si lo

intentamos nos daremos cuenta que ello requiere una gran cantidad de tiempo, a al vez que nos obliga a profundizar en los conocimientos de programación de ensamblador.

-Circuitos analógicos: la mayoría de los microcontroladores suelen proporcionar prestaciones como las de interfaces integrados para el mundo analógico. Estos interfaces incluyen conversores analógico/digitales (con resoluciones diferentes y multiplexores analógicos enfrente de los mismos), comparadores analógicos e incluso amplificadores operacionales (con salida a un terminal).

-Modos de funcionamiento: también es interesante comprobar si un microcontrolador puede trabajar con un circuito alimentado a través de batería. Así, un cierto número de pequeños circuitos internos del microcontrolador pueden conectarse en su modo de "reposo" para ahorrar energía.

-Facilidad de programación: cuando un controlador dispone de una interfaz ISP, quiere decir que puede programarse en el propio circuito que lo aloja. Si no es así, necesitaremos disponer de un programador más o menos complejo. Un controlador que no disponga de bus ISP y que esté soldado al circuito, no podrá programarse de nuevo sin tener que estar montando y desmontando continuamente.

-Prestaciones especiales: existen microcontroladores que solamente son adecuados para una aplicación específica, por ejemplo, la de control de motores, DSP y contadores con una sección de entrada de RF. En caso de tener que trabajar con este tipo elementos, estos dispositivos especiales pueden ser la mejor opción.

-Precio, disponibilidad y encapsulado: el precio de un microcontrolador no es demasiado importante en el caso de que se quiera desarrollar un único proyecto o una pequeña serie de este. Sin embargo, siempre nos puede perjudicar bastante si, finalmente, hemos encontrado el modelo ideal para nuestra aplicación y descubrimos que el circuito integrado solamente está disponible en cantidades de 10.000 y que se entrega directamente desde Corea. El encapsulado del microcontrolador también es un factor importante, ya que en la realización manual de prototipos, puede ser muy difícil,

por no decir imposible, soldar integrados con encapsulado TQFPK de por ejemplo 200 terminales.

4.3.La familia elegida

En el proceso de selección de la familia del microcontrolador, se han comparado bastantes modelos teniendo en cuenta los parámetros descritos en el apartado anterior . Se han considerado también otros hechos, como que el micro tuviera un entorno de programación C y C++ gratuito, encapsulado DIP... Otro de los requisitos al seleccionar el microcontrolador ha sido no condicionar la escalabilidad de la placa, es decir, se ha intentado que la elección de un modelo de microcontrolador no limite la ampliación de esta, de forma que si en un momento dado se desease incrementar las prestaciones de esta, hubiese suficiente con cambiar el microcontrolador sin tener que modificar ni el firmware ni la placa.

Siguiendo estas condiciones y después de evaluar diferentes familias de microcontroladores se ha considerado que las opciones más interesantes serian la utilización de algún microcontrolador de las familias de 8 bits PIC de Microchip o AVR-Atmega de Atmel. Los modelos de estas familias no son ni mucho menos los más potentes del mercado, pero si en cambio de los que ofrecen una mejor relación rendimiento-precio, y más variedad de modelos, existiendo un gran número de ellos con diferentes y a veces sorprendentes prestaciones (existen modelos con emisor de radio incorporado). Además ambos disponen de gran cantidad de recursos y documentación en la red, lo que sin duda ha de facilitar y acelerar cualquier trabajo con estos. También se encuentran en formato DIP, y son fáciles de obtener a través de distribuidores o tiendas de electrónica.

Tras analizar las dos familias de microcontroladores se ha decidido que para este proyecto la más adecuada de las dos es la AVR-Atmega. Las razones son las siguientes:

-Compilador C y C++ gratuito: existe una versión gratuita del famoso compilador C C++ gcc adaptado para el código máquina del AVR-Atmega. Este viene en un paquete denominado WinAVR que a parte de incluir un completo entorno gráfico para la

programación incluye también diversas herramientas y documentación. Para PIC no se ha encontrado un entorno de desarrollo C o C++ gratuito, únicamente se ha encontrado el completo compilador HiTech C que aunque dispone de una versión de prueba gratuita, por ahora es de pago. El uso de un compilador de pago incrementaría el precio del producto final lo que va en contra de uno de los principios básicos del proyecto: hacerlo todo lo más económico posible.

-Mayor rendimiento: en cuanto al rendimiento, los dos son microcontroladores RISC de 8 bits: la familia PIC tiene un juego de 35 instrucciones, mientras que la familia AVR-Atmega tiene un juego de 118 instrucciones. Se puede deducir que en general las instrucciones de las PIC son mas sencillas que las del AVR-Atmega , lo que no sería un inconveniente si estas se ejecutaran más rápido que las del AVR, pero esto no es así. En una PIC cada instrucción requiere 4 ciclos de reloj para ejecutarse, mientras que en un AVR requiere 1 único ciclo de reloj lo que se resume en que, en 1 ciclo de reloj, un microcontrolador AVR puede hacer una operación más compleja que la que es capaz de realizar una PIC en 4. A pesar de que las PIC pueden trabajar con cristales del mayor frecuencia que los AVR (hasta 20Mhz) el balance sigue siendo positivo para los AVR.

-Consumo: comparando las hojas técnicas de modelos de las dos familias se puede ver que en estado activo y a la misma frecuencia de reloj prácticamente consumen lo mismo: a 4Mhz el modelo de PIC alimentado con 5.5V consume 4.5ma como máximo, mientras que el microcontrolador AVR-Atmega alimentado a 3.3 consume 4ma. Pero desde el punto de vista del “throughput” en función del consumo, el microcontrolador de Atmel sigue saliendo mejor parado. Si las PIC, a pesar de tener un rendimiento menor consumieran menos que los AVR, serian una opción a tener en cuenta si únicamente se desase minimizar le consumo, pero tal como se muestra en el apartado anterior a igualdad de frecuencia el rendimiento de los AVR-Atmega es mayor, y como se explica ahora, a la misma frecuencia el consumo de los dos es prácticamente el mismo, por lo que la relación rendimiento /consumo sigue siendo favorable para el micro de Atmel..

-Arquitectura más simple: desde el punto de vista de la arquitectura, la de la familia AVR-Atmega, al ser más moderna, es más flexible (dispone de más acumuladores) lo

que la hace más fácil de programar en ensamblador o de optimizar por el compilador. A pesar de que cuando se trabaja en C o C++ esto no supone un problema, las PIC tienen algunos aspectos que las hacen engorrosas de programar. Un ejemplo es la estructuración en bancos de los registros lo que obliga a llamar continuamente a la instrucción de conmutación de banco para poder acceder a determinados registros.

Por otro lado, las PIC también tienen algunas ventajas como por ejemplo la mayor diversidad de modelos y periféricos, o la gran facilidad con que su fabricante, Microchip, ofrece muestras para el desarrollo de prototipos. No obstante, estas ventajas no se han considerado suficientes para utilizar un microcontrolador PIC en lugar de un AVR-Atmega.

Hay que comentar también que la elaboración de este proyecto ha coincidido con el inicio de la implantación en el mercado de la nueva familia de microcontroladores de Microchip PIC 18F que se aproximan, e incluso superan, en prestaciones a la familia AVR-Atmega de Atmel. De todas formas se ha decidido no utilizar estos microcontroladores por lo que conlleva su reciente aparición: poca disponibilidad y falta de documentación y material en Internet.

4.4.El modelo elegido

Después de examinar los diferentes modelos de la familia AVR-Atmega se ha decidido utilizar el AVR-ATmega8515, uno de los más sencillos de la familia, pero que cumple con las especificaciones necesarias para este proyecto. Con el fin de minimizar el consumo, el cristal será de 4 Mhz en lugar de 8Mhz, y la versión utilizada la L, es decir el AVR-Atmega8515L que trabaja a 3.3V en lugar de a 5V. Hubiese sido más interesante utilizar algún modelo DIP más potente (p.ej Atmega32) pero debido a que el distribuidor (Anatronic) se niega a servir directamente a particulares se ha utilizado el AVR-ATmega8515 disponible en muchas tiendas de electrónica. Además, el uso de este microcontrolador no supone ninguna limitación ya que se en caso de necesidad se podría sustituir por cualquier otro modelo superior de la familia sin apenas modificaciones en el diseño de la placa ni en del firmware.

El microcontrolador AVR-ATmega8515 dispone de un procesador RISC de 8 bits capaz de ejecutar un juego de 118 instrucciones todas ellas de un único ciclo de instrucción, siendo un ciclo de instrucción igual a un ciclo de reloj. Dispone de una memoria de programa de 4096 palabras (16 bits), es decir, que puede almacenar hasta 4096 instrucciones. Dispone también de un banco de 32 registros de propósito general, de 64 registros de entrada / salida para la gestión de periféricos. En cuanto a la memoria RAM disponde de 512 bytes, pero puede hacer uso de una RAM externa. Tiene también una memoria EEPROM para almacenar datos de forma permanente. La gestión de las interrupciones se realiza mediante vectores de interrupción.

En cuanto a los periféricos, cuenta con un temporizador de 8 bits con preesacaler independiente, otro temporizador de 16 bits con preesacaler independiente (PWM) y un watch-dog timer con oscilador propio. Tiene también un comparador analógico, una UART, una interfaz SPI, y 4 puertos convencionales de entrada / salida. En el Anexo B se descibren con más detalle las características y funcionamiento de este.

Por falta de herramientas adecuadas para el desarrollo de este proyecto se ha tenido que recurrir al formato DIP (fig 4.3.1), pero este microcontrolador también esta disponible en otros formatos como Flat Pack, o PLCC.

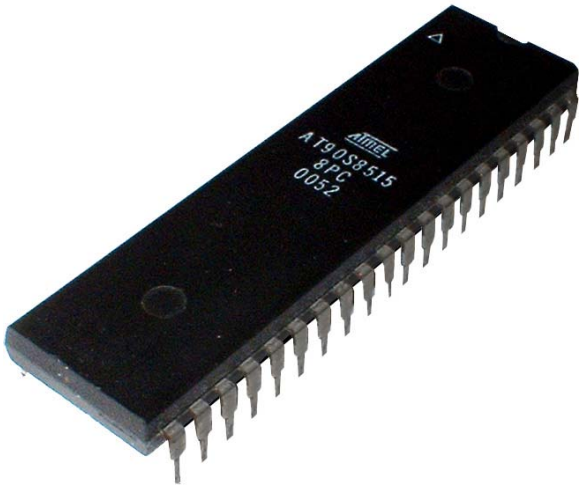


Fig 4.3.1 Imagen del microcontrolador AVR-ATmega8515 en encapsulado DIP

Capítulo 5
Bus de interconexión con el ordenador personal

5.1.Introducción

La placa de desarrollo ha de disponer de algún bus para la comunicación con el ordenador personal por tal de poder intercambiar información con este o recibir el código máquina con que se ha de programar. Tal como se justificó en el capítulo 1, las opciones disponibles para este fin son dos: el puerto serie o el puerto paralelo. La opción del puerto USB queda descartada debido a que por el momento ningún modelo de la familia de microcontroladores seleccionada dispone de soporte para USB (quizás en el futuro surjan modelos con soporte para este).

Se ha decidido aprovechar las dos opciones disponibles y permitir el uso tanto del puerto paralelo como del puerto serie para la comunicación placa-ordenador personal. El puerto paralelo se puede utilizar para la transmisión de datos ordenador-placa o placa-ordenador y también para la programación de esta. El puerto serie solo se puede utilizar par la transmisión placa-ordenador, ya que el canal de entrada está ocupado por la línea de datos del módulo receptor GPS, es decir que la línea TX de la UART del microcontrolador se puede utilizar para enviar datos al ordenador personal, mientras que la línea RX se encuentra ocupada en la recepción de datos del módulo GPS. De todas formas la placa de desarrollo dispone de unos jumpers que permiten cambiar esta

configuración. A continuación se describe con más detalle la implementación de cada uno de los buses.

5.2.Comunicación por el puerto paralelo

El fin de este bus es permitir la programación de la placa y la comunicación de esta con el ordenador. Debido a la falta de hardware específico para este fin en el microcontrolador, la implementación de este bus se ha hecho partiendo desde cero, utilizando puertos de entrada /salida convencionales y un protocolo de diseño propio. En cuanto al hardware, se han utilizando los 8 bits de un mismo puerto de entrada / salida como bits de datos (Data [0..7]), más 2 bits de otro puerto de entrada / salida como bits de control (Send y Clock). El protocolo hace uso de estos puertos de entrada / salida, coordinando la comunicación y gestionando el acceso a las líneas del bus por parte de los diferentes dispositivos, permitiendo así el uso de los puertos de entrada / salida por otros periféricos. En el apartado “5.2.2-Protocolo paralelo” se describe con más detalle este protocolo.

5.2.1.Conexiones

A parte del intercambio de datos, el bus de conexión también ha de permitir la programación del microcontrolador sin tener que conectarlo ni desconectarlo de ninguna parte. Se ha procurado que el conexionado del puerto paralelo coincida con los pins de programación serie ISP del microcontrolador, por tal de permitir también la programación del firmware en el microcontrolador a mediante el puerto paralelo. En el proceso de comunicación mediante el protocolo diseñado, los dispositivos se conectan tal como muestra el esquema:

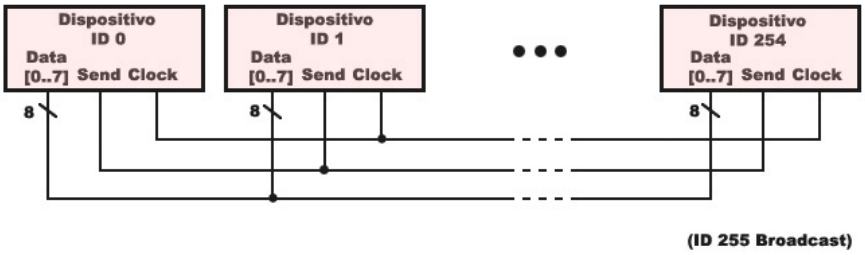
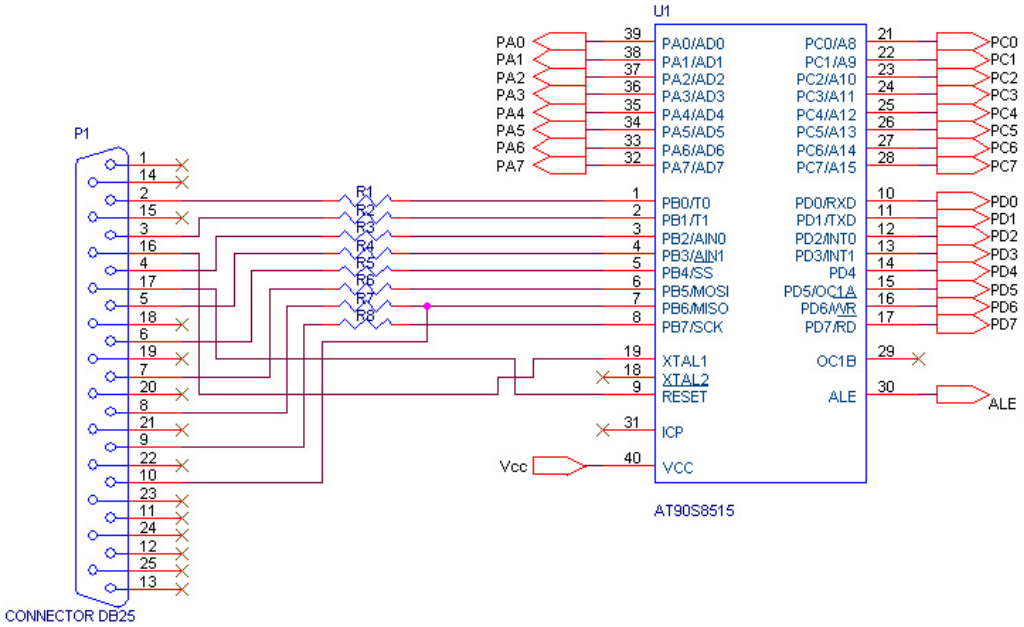


Fig 5.2.1.1 Esquema general de la conexión de dispositivos al bus.
El ordenador personal es otro dispositivo más del bus.

La línea send debe disponer de una resistencia de pull-down para fijar el valor a 0 en el momento en que ningún dispositivo este usando el bus, ya que en ese momento todos los dispositivos tienen el pin correspondiente a la señal Send como pin de entrada y por tanto esta línea tendría un valor indeterminado. Mediante esta resistencia, en el momento que ningún dispositivo esté forzando la línea a 1, la señal Send valdrá 0. Es decir que su valor por defecto es 0.

En el caso del proyecto, solo hay 2 dispositivos conectados al bus: el microcontrolador y el ordenador personal. La Fig 5.2.1.2 muestra con detalle las conexiones eléctricas entre los dos dispositivos.



R1-R6=0.330K

Fig 5.2.1.2 Esquema general de conexión entre el microcontrolador y el ordenador. En este faltan el cristal y el pulsador de reset. Mirar el esquema completo de la placa para más detalles.

!Reset	Select	Pin 17 DB25
Xtal	Initialize	Pin 16 DB25
Sck	Data7	Pin 9 DB25
Mosi	Data5	Pin 7 DB25
Miso	Data6	Pin 8 DB25
Miso	Ack	Pin 10 DB25

Fig 5.2.1.3 Tabla de correspondencia entre la señales del microcontrolador utilizadas para la progrmación del firmware por el puerto paralelo

Tal como se puede ver en la figura 5.2.1.2 las conexiones entre el puerto paralelo y el microcontrolador se han hecho de forma que las 8 líneas de datos del puerto paralelo (Data 0..7) coincidan directamente con los 8 bits de entrada/salida del puerto B del microcontrolador (Port B 0..7). Esta correspondencia facilita mucho el intercambio de los bytes entre el ordenador y el microcontrolador, de forma que para obtener un byte solo hay que leer el puerto. Si la correspondencia fuera desordenada, primero habría que leer el puerto o los puertos y luego reordenar los bits hasta tener el byte original. Esto, a parte de complicar un poco la transmisión de datos, también la relentizaría ya que se requiere más tiempo para leer y ordenar cada byte que para simplemente leerlo.

Véase también, que entre el puerto B del microcontrolador y los pins de datos del conector del puerto paralelo se han colocado 8 resistencias. El fin de estas es evitar posibles colisiones: cuando el puerto B solo tiene conectado el conector del puerto paralelo la tensión en los pins de este es la tensión fijada por las líneas del puerto paralelo. Estas resistencias hacen que si, hay otro elemento conectado a estos pins fijando una tensión distinta, no se produzca una colisión, es decir, las resistencias "separan" las dos tensiones. En resumen, que si estas no estuviesen, en el momento que hubiera algún dispositivo conectado al puerto o este actuara como salida, habría una colisión ya que en los mismos pins del puerto se estarían fijando tensiones distintas simultaneamente.

5.2.2.Protocolo paralelo

El protocolo propuesto, es un protocolo de comunicación paralelo, síncrono y half-dúplex. Paralelo porque en cada ciclo se transmiten de forma simultanea 8 bits. Síncrono porque hace uso de una señal de reloj común para la sincronización entre emisor y receptor. Half-Dúplex porque permite la comunicación bidireccional entre diferentes dispositivos pero nunca de forma simultanea. Hay que dejar claro que este protocolo está diseñado para intercambiar datos entre dos o más dispositivos a través de un bus paralelo y no tiene nada que ver con el protocolo utilizado para la programación del microcontrolador que es totalmente distinto.

5.2.2.1.Señales

Este consta de las siguientes señales:

Data[0..7]: son las 8 líneas de datos a través de las que se transmiten de forma simultanea los diferentes bits de un mismo byte.

Send: es la línea que indica que se está transmitiendo información por el bus y por tanto sirve a los demás dispositivos para saber si pueden emitir o se han de esperar a que termine el que lo esta haciendo.

Clock: esta señal marca la cadencia a la que el emisor va transmitiendo la información, y permite al receptor saber cuando los datos que hay en las líneas Data[0..7] son válidos. Así, los bytes enviados por el emisor son capturados por el receptor en el flanco de subida de la señal Clock.

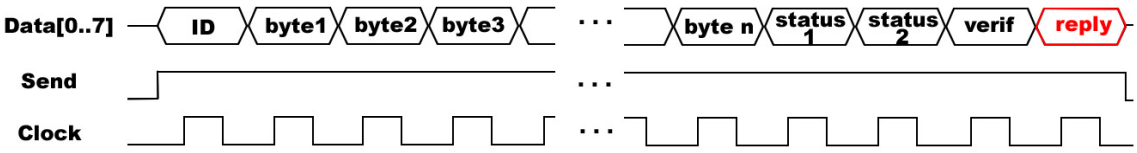


Fig 5.2.2.1.1 Secuencia de envío de los diferentes bytes de un paquete.

5.2.2.2.Estructura de un paquete y secuencia de envío

La comunicación se basa en el envío de la información mediante diferentes paquetes. Cada uno de estos paquetes está formado por 1 byte de ID, los n bytes con los datos, 2 bytes de estado y por 1 byte de verificación de errores. Tras recibir un paquete el receptor responde al emisor con 1 byte de reply. La secuencia exacta es la siguiente:

-El dispositivo que desea emitir, consulta el estado de la línea Send:

-Si esta activada (‘1’), significa que hay otro dispositivo emitiendo y aborta el proceso de emisión. Reintentándolo tras un intervalo de tiempo aleatorio

-Si esta desactivada (‘0’), significa que el bus está libre y el dispositivo puede iniciar la comunicación. Luego activa (‘1’) la línea Send tomando así el control del bus.

-Tras tomar el bus, el emisor pone en las líneas de datos el identificador (ID) del dispositivo a quien va destinado el paquete. Todos los dispositivos conectados al bus capturan el ID pero solo aquel cuyo ID coincide con el enviado por el emisor continua con el proceso de recepción, los demás abortan. 0xFF es el ID de broadcast, y en este

caso el proceso de recepción lo realizan todos los dispositivos conectados al bus simultáneamente, es decir: todos escuchan.

-A continuación el emisor transmite los n bytes de datos con la información a enviar. El receptor los va capturando al ritmo de la señal Clock.

-Tras enviar los n bytes con la información, se envían los 2 bytes de Status, mediante los que el emisor informa al receptor sobre el estado de la comunicación.

-El último byte enviado por el emisor del paquete, es el de verificación, y permite la detección de errores en la transmisión de todos los bytes enviados (bytes de ID y Status incluidos). Este byte puede ser la XOR o la suma acumulada de todos los bytes transmitidos. Al recibir este byte el receptor lo compara con el byte de verificación que él mismo ha calculado a partir de los bytes que ha recibido, y si coinciden es que con mucha probabilidad la transmisión ha ido bien.

-Para finalizar la transmisión del paquete, el receptor envía el byte de respuesta Reply al emisor indicándole de este modo como ha ido la transmisión. A través de este byte el receptor puede indicar al emisor que le envíe el siguiente paquete o que le reenvíe otra vez el último paquete porque ha habido algún error. En el caso de las emisiones de broadcast no se utiliza byte de Reply, por lo que en el último ciclo de reloj nadie envía nada.

5.3.Comunicación por el puerto serie

El puerto serie del microcontrolador (UART) , está configurado y conectado adecuadamente para poder recibir los datos de posición enviados por el módulo GPS, y para poder emitir datos hacia el ordenador personal. Es decir que en la configuración estándar de la placa, a través de la UART solo se pueden enviar datos al ordenador y recibir del GPS. Esto es debido a que en esta configuración la línea de recepción Rx está conectada directamente a la línea Tx del módulo GPS, mientras que la de transmisión Tx está conectada a la Rx del puerto serie del ordenador. De todas formas esta configuración se puede modificar ya que la placa dispone de un conjunto de jumpers que permite configurar las líneas del puerto serie de diferentes formas adaptándolas a nuestras necesidades (fig 5.3.1).

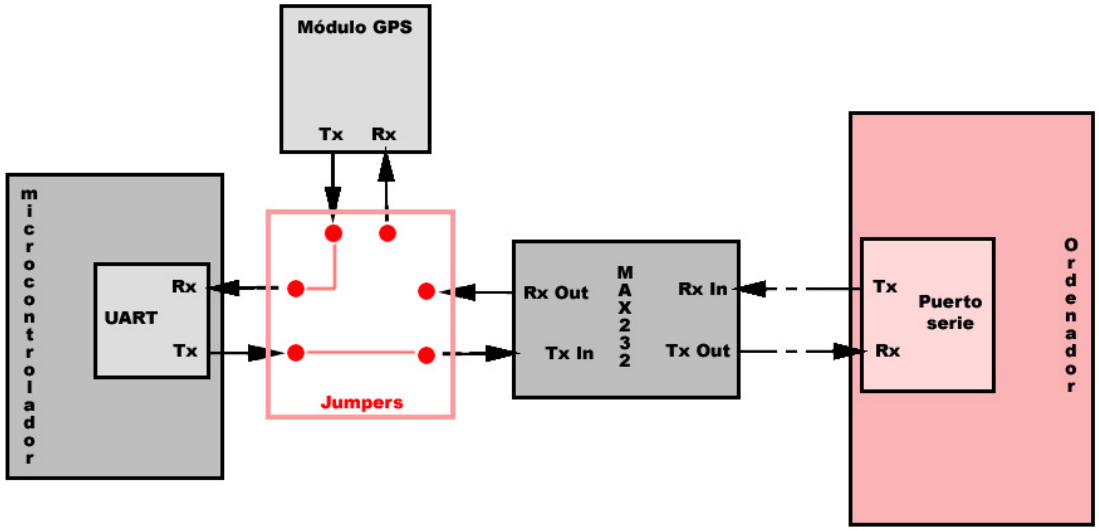
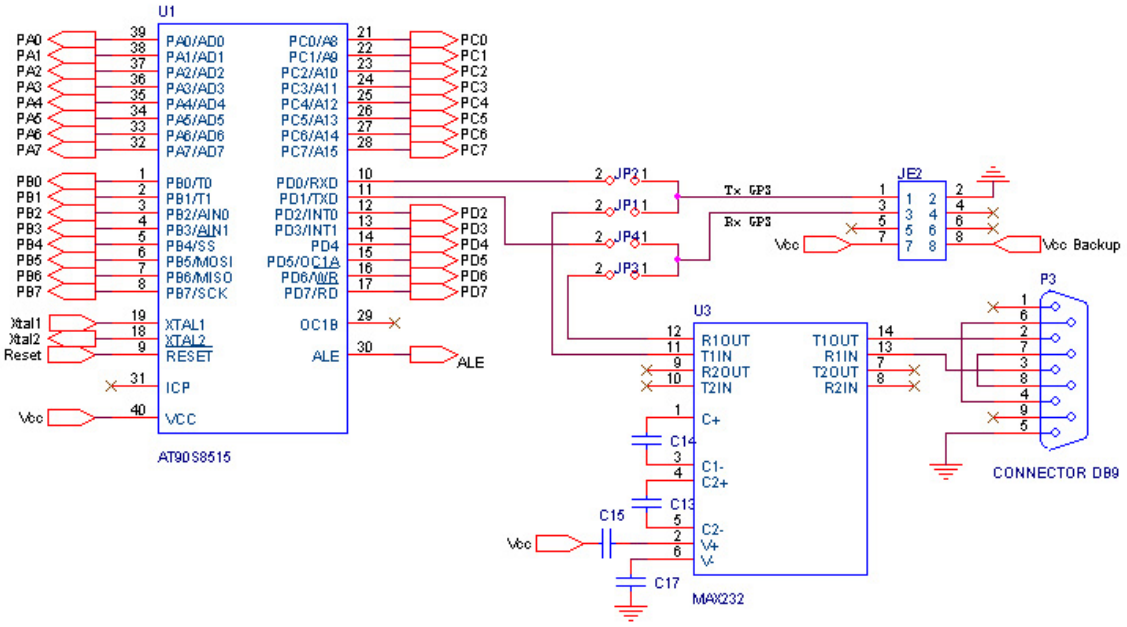


Fig 5.3.1 Esquema general de la conexión serie entre los diferentes dispositivos de la placa y el ordenador personal. Los jumpers se encuentran conectados en la configuración estándar.

En lo referente a la recepción de la información de posición GPS a través del puerto serie la librería firmware GPS (capítulo 9) se encarga de todo, es decir que no hay que configurar nada. Esta librería no incorpora rutinas para el envío de información al ordenador personal, pero esta tarea es relativamente sencilla y se reduce a escribir en el registro de transmisión de datos de la UART, mirando siempre antes el estado de los flags de control de esta. En el Anexo B se describe con detalle el funcionamiento y proceso de configuración de la UART. Hay que recordar que la configuración de la UART se realiza tanto para la recepción como para la transmisión, de forma que no se puede recibir a una velocidad y transmitir a otra distinta simultáneamente. Esto implica que mientras se están recibiendo datos del GPS si se desea transmitir hay que hacerlo según las especificaciones del estándar NMEA (4800 baud, bit de imparidad, 8 bits de datos...) de lo contrario no se podrán leer los datos del GPS. Si se desea transmitir con la UART configurada de forma distinta a la del estándar NMEA, habrá que reconfigurar la UART lo que interrumpirá la recepción de datos del GPS.

Tal como se muestra en la figura 5.3.1 la conexión serie entre el ordenador y la placa de desarrollo no se realiza directamente, sino que entre los dos se sitúa un adaptador de niveles MAX232 para convertir los +12V –12V del estándar RS232 del puerto serie del ordenador a los +5V 0V de la UART del microcontrolador. Si este no se utilizase la UART no funcionaría correctamente y probablemente, a pesar de tener protecciones,

acabaría estropeándose. La figura 5.3.2 muestra con más detalle las conexiones eléctricas entre el ordenador, el microcontrolador y el módulo receptor GPS.



C14=0.1uF, C13=0.1uF, C15=0.1uF

Fig 5.3.2 Esquema con las conexiones eléctricas entre el ordenador, el microcontrolador y el módulo receptor GPS.

Capítulo 6
Circuito de alimentación

6.1.Introducción

Todos los componentes (módulo GPS, microcontrolador, MAX232...) se alimentan a partir de una tensión de 3.3V, por lo que es necesario incorporar, entre el conector de alimentación y el resto de componentes, un circuito capaz de mantener esta tensión estable ante variaciones en el consumo o variaciones en el suministro. Este circuito ha de ser lo más compacto y eficiente posible y disponer además de protecciones para evitar que problemas en el generador dañen el resto de la placa.

6.2.Parámetros básicos y tipos de reguladores

El centro de cualquier circuito de alimentación es el regulador y es el componente encargado de mantener la tensión constante frente a las variaciones de consumo de la carga. Para seleccionar el tipo de regulador se han considerado dos parámetros básicos:

-Rendimiento: resumiendo, el rendimiento indica la cantidad de energía de la entrada del regulador que es convertida en energía a la salida. Por tanto interesa que el rendimiento del regulador sea grande por tal de aprovechar al máximo la energía de la entrada, ya que por ejemplo en el caso de funcionar con batería, un bajo rendimiento podría implicar que el sistema estuviese activo durante menos tiempo del que podría si dispusiera de un buen circuito regulador. Así un regulador con un buen rendimiento minimizará el consumo.

-Complejidad: en función del tipo de regulador el número de componentes utilizados en la implementación aumenta, lo que también implica un incremento de coste, complejidad, y de la superficie ocupada en la placa. Como se pretende que la placa sea un sistema autónomo y compacto este es un parámetro a minimizar.

Existen diferentes tipos de reguladores, pero los dos más utilizados son los siguientes:
-Reguladores conmutados: resumiendo mucho, este tipo de reguladores trabaja modulando el ancho de pulso de una señal que excita un transistor (transistor de paso), poniéndolo en conducción o en corte. Este transistor actúa a modo de “interruptor”, conectando y desconectando la carga de la alimentación a gran velocidad, en función de las necesidades de ésta. Así, cuando la carga necesita más energía aumenta el ancho del pulso y el transistor permanece conduciendo más tiempo. Y cuando necesita menos energía disminuye, de forma que permanece cortado (sin conducir) más tiempo. El regulador sabe si la carga necesita más o menos energía comparando la tensión que cae en ésta con una tensión de referencia: si la tensión que cae en la carga es menor que la tensión de referencia significa que la carga necesita más corriente, mientras que cuando la tensión que cae en la carga es mayor que la tensión de referencia es porque la carga no necesita tanta corriente. Es de este proceso de conmutación del que reciben su nombre. Las principales virtudes de estos son su gran eficacia y poca disipación de calor pero como contrapartida son bastante complejos y emiten bastante ruido electromagnético.

-Reguladores lineales: este tipo de reguladores también se basan en un transistor de paso, pero en lugar de utilizarlo en conmutación (corte-conducción) lo hacen en su zona de funcionamiento lineal. En esta zona el transistor actúa como un “grifo” de corriente y deja pasar más o menos intensidad en función de las necesidades de la carga. Así, cuando ésta requiere más energía deja pasar más corriente, y cuando requiere menos reduce el paso de corriente. Uno de los inconvenientes de estos es que al trabajar en la zona lineal tienen menor margen de trabajo, a parte de que en el transistor se pierde bastante potencia en forma de calor y sólo suelen ser viables cuando no se requiere mucha corriente y la variabilidad en el consumo no es muy grande. Es decir, que el rendimiento de estos es bajo en comparación con el de los reguladores conmutados.

6.3.El circuito de alimentación

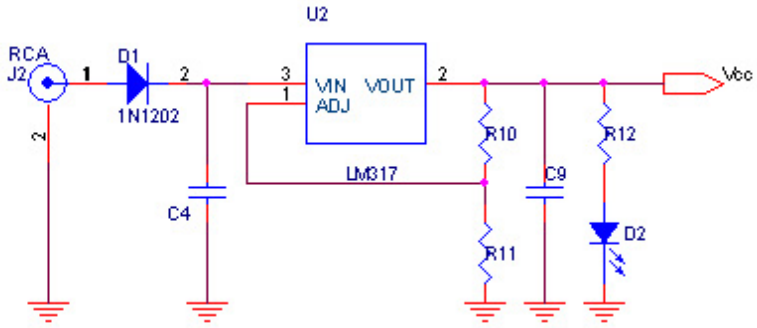
Finalmente las alternativas consideradas han sido: utilizar un regulador conmutado de tipo step-up para conseguir los 3.3V a partir de una tensión inferior, o un regulador conmutado step-down o un regulador lineal convencional para obtener los 3.3V a partir de una tensión superior.

Tal como se ha visto en el apartado anterior, de cara a aplicaciones autónomas la opción más interesante hubiese sido el uso de un regulador conmutado ya que el rendimiento de estos es mayor, pero debido a que su complejidad queda fuera del abasto de este proyecto se ha descartado su uso. No obstante la elección de este tipo de reguladores también hubiese supuesto el siguiente problema: la tensión de la alimentación de entrada hubiese tenido que estar siempre dentro de un rango relativamente limitado y esto condicionaría el uso de diferentes tipos de baterías o fuentes de alimentación.

Así finalmente se ha optado por utilizar un regulador lineal LM317 configurado de forma que permita obtener una tensión de 3.3 V a partir de una tensión superior. Para que este regule bien es necesario que la tensión del generador sea aproximadamente 2V superior a la tensión deseada, es decir de unos 5V. No obstante al contrario que en el caso del regulador conmutado, la tensión de entrada puede variar en un rango relativamente amplio desde los 5V a los 10V o más, pudiendo utilizar diferentes tipos de generador (alimentadores de red, pilas convencionales, baterías ...)

Este regulador se presenta como un único componente y solo precisa de un condensador en la entrada (C4) para compensar los efectos inductivos, otro condensador a la salida (C9) para mejorar la respuesta transitoria, y de un divisor de tensión a la salida (R10 y R11) para ajustar la tensión de referencia del regulador. Además tiene protección contra cortocircuitos internos que limitan la corriente máxima que pasa por el circuito y otra protección contra inconvenientes térmicos. Es decir, que excepto el rendimiento, que no es excesivamente bueno, todo lo demás son ventajas.

A la entrada del regulador se ha situado un diodo (D1) para proteger el circuito contra posibles inversiones de polaridad, y a la salida un LED con su resistencia (D2 y R12)) para ver cuando hay alimentación. La figura 6.3.1 muestra el esquema:



C4=100uF , R10=0.25K , R11=0.41K ,C9=100uF ,R12=1K

Fig 6.3.1 Esquema del circuito de alimentación.

Capítulo 7
Software de programación

7.1.Introducción

Este software se ejecuta en el ordenador personal y su fin es cargar en el microcontrolador el código generado por el usuario mediante el compilador. Resumiendo, esta aplicación lee byte a byte el contenido del archivo .hex generado por el compilador y lo carga en una estructura de datos del software de programación. Luego, cuando el usuario lo decide, transfiere cada byte almacenado en la estructura, a la memoria de programa del microcontrolador. Tras esto, el microcontrolador ya se encuentra programado y listo para ejecutar la aplicación.

Tal como se explica en el capítulo 2, el lenguaje utilizado para el desarrollo de este software es Visual Basic .NET , básicamente porque ofrece un entorno sencillo y eficiente para crear aplicaciones visuales en Windows. A parte se ha tenido que buscar una DLL (librería de enlace dinámico) para poder acceder al puerto paralelo del ordenador, ya que como se ha explicado en los apartados anteriores la programación se hace a través de este puerto. También hay que recordar que para que la aplicación funcione bien en cualquier ordenador, antes es necesario haber instalado la plataforma .NET Framework de Microsoft la cual contiene todos los recursos necesarios para la ejecución de cualquier aplicación hecha en .NET.

Para instalar y poder utilizar la aplicación en un ordenador personal con Windows XP hay que realizar los siguientes pasos:

- Instalar el .NET Framework 1.1 de Microsoft. Este se puede obtener a partir del disco Windows Component Update de Visual Studio, o descargarse de la web oficial de Microsoft.
- Si se desea ejecutar esta aplicación desde el disco duro, solo hay que copiar en este la carpeta TelecargaAVR, la cual contiene el binario de la aplicación y las DLLs (OUTPORT.DLL e INPORT32.DLL) necesarias para el acceso al puerto paralelo. Otra opción es copiar las DLLs en la carpeta "$WINDOWS\system32\", y el binario en la carpeta de la aplicación. Siempre es útil crear un acceso directo al binario para agilizar el acceso al programa mientras se esta desarrollando.
- Instalar la aplicación UserPort para liberar el puerto paralelo del control de Windows y permitir así el acceso a este a las DLLs de la aplicación. El primer paso del proceso de instalación consiste en extraer el contenido del fichero UserPort.ZIP, luego en copiar el fichero "UserPort.sys" en la carpeta "$WINDOWS\system32\drivers\" y en ejecutar a continuación la aplicación "UserPort.exe" y presionar la opción "Start".
-Tras realizar estos pasos la aplicación de telecarga ya debería estar lista para poder utilizarse sin problemas.

7.2.Descripción de la aplicación

La aplicación esta constituida por una interfaz o formulario y por varios módulos donde se definen las clases de los objetos utilizados en esta. Estos módulos son: "BajoNivel" que contiene todas las rutinas necesarias para acceder al puerto paralelo y hacer conversiones de tipo, "Comunicación" que contiene las rutinas encargadas de transferir la información siguiendo el protocolo adecuado, y "Memoria" que describe los objetos necesarios para almacenar el código que se va a telecargar y los algoritmos para leer el contenido de los archivos .hex .

Así, al arrancar el programa de telecarga aparece una interfaz consistente en 2 cajas de texto con una serie de botones en los laterales. La primera caja de texto contiene los bytes del código de programa a cargar en la Flash interna, y la segunda los bytes de datos a cargar en la EEPROM interna. Toda la información está presentada en

hexadecimal. Los botones de los laterales permiten realizar diferentes operaciones con estos datos: abrir un archivo .hex para luego poder telecargarlo, realizar una programación completa de todos los bytes de la memoria, realizar una programación rápida de todos los bytes omitiendo aquellos iguales a 0xFF, cargar en la estructura de datos el código almacenado en el microcontrolador, realizar una verificación completa de todos los bytes almacenados, realizar una verificación rápida de los bytes omitiendo aquellos distintos de 0xFF, o resetear e inicializar a 0xFF toda la memoria. Además dispone de una barra de selección horizontal que permite configurar el retardo de la señales de programación para evitar así posibles problemas debidos a las diferentes velocidades de ejecución de la aplicación en distintos ordenadores .

Es importante recordar, que nada más arrancar el ordenador este suele mantener la línea del puerto paralelo que gobierna la señal de ¡Restet a 0V, por lo que se esta se encuentra haciendo un reset permanente. El reset no cesará hasta que se abra la aplicación de telecarga (la cual activa la línea de ¡Restet) o hasta que se desconecte la placa del ordenador personal.

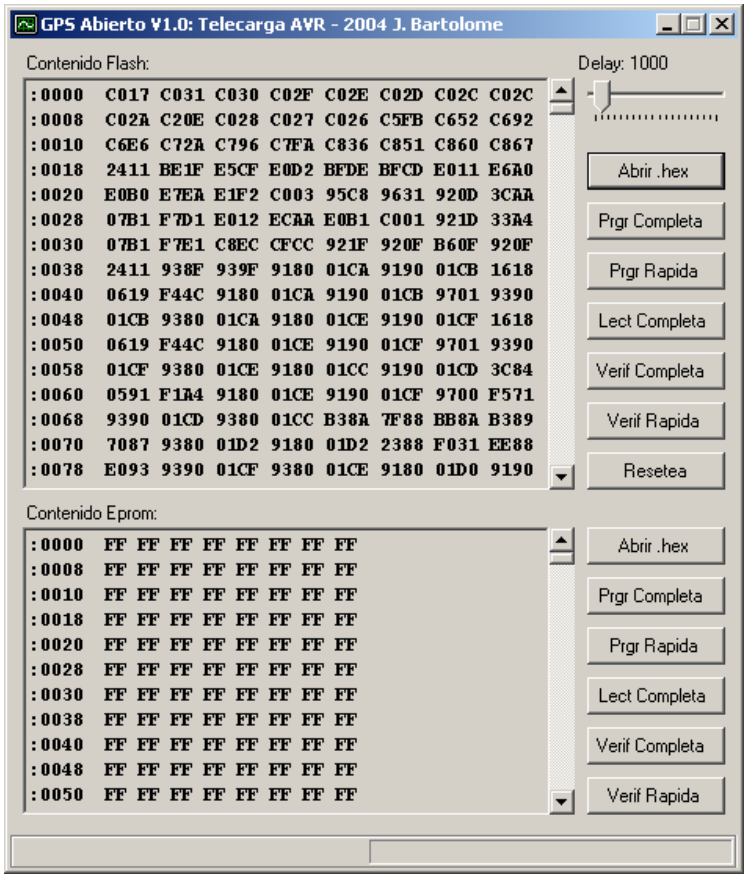


Fig 7.2.1 Interfaz de la aplicación de telecarga del microcontrolador

La secuencia típica para la programación del microcontrolador es la siguiente:

- Resetear la memoria de programa del microcontrolador mediante el botón “Resetea” . Esto inicializa toda la memoria del microcontrolador, poniendo cada una de las casillas de la memoria de programa a 0xFF, es decir lo deja virgen para poder ser programado. Es indispensable hacer un Reset antes de cualquier programación del microcontrolador.
- Abrir el archivo con el código máquina en formato .hex generado por el compilador mediante el botón “Abrir .hex”. Esto almacena en la estructura de datos de la aplicación los bytes del programa que se han de telecargar en la memoria Flash del microcontrolador. Tras cargar este archivo en la estructura, en la caja de texto “Contenido Flash:” se puede ver la representación hexadecimal de este.
-Programar en la memoria de programa del microcontrolador el contenido almacenado en la estructura de datos de la aplicación, es decir hacer la telecarga. Esto se puede hacer mediante dos opciones “Prgr completa” o “Prg rápida”. La primera corresponde a una programación completa y lo que hace es programar todas y cada una de las casillas de la memoria del microcontrolador del principio hasta el final. La segunda corresponde a una programación rápida y lo que hace es programar solo aquellas posiciones de memoria cuyo contenido ha de ser diferente de 0xFF ya que las que han de ser 0xFF no hace falta programarlas porque al hacer el reset estas ya quedan a 0xFF. Este último método acelera muchísimo el proceso de telecarga, sobretodo cuando no se ocupa toda la memoria de programa.
- Finalmente, tras haber telecargado el código en el microcontrolador toca verificar que realmente este se ha telecargado bien y que no ha habido ningún error en el proceso. Para ello hay que dar al botón “Verif completa” que lo que hace es leer cada una de las posiciones de la memoria de programa del microcontrolador y compararla con el contenido descrito para aquella misma posición en el archivo .hex cargado en la memoria del ordenador. Si el contenido de la misma posición en la memoria de programa del microcontrolador difiere del cargado en la estructura de datos de la aplicación es que se ha producido un error. Cuando esto sucede la aplicación informa de la posición en que se ha producido el error, cuando esto sucede se ha de volver a programar el micro de nuevo.

Todo este proceso aquí descrito es para la memoria de programa. El microcontrolador dispone de otra pequeña memoria de datos que también puede ser programada siguiendo los mismos pasos pero utilizando los botones que se encuentran junto a la caja de texto de la memoria de datos.

7.3.Lectura del archivo .hex

Una fase importante en el desarrollo de la aplicación de telecarga ha sido descubrir la estructura de los archivos .hex en los que el compilador guarda el código máquina generado a partir del código fuente del usuario. Estos archivos indican los datos que hay que transferir al microcontrolador y la posición de la memoria en que hay que situarlos.

El actual formato .hex fue ideado por Intel para la descripción del código objeto de sus microprocesadores de 8, 16 y 32 bits. Es un formato flexible porque toda la información se representa en ASCII lo que ofrece muchas prestaciones: es fácil de representar mediante medios no binarios como hojas impresas o tarjetas perforadas, se puede editar y visualizar desde numerosas aplicaciones y plataformas no especializadas. Estos archivos se pueden ver y modificar prácticamente con cualquier editor de texto. Esta flexibilidad lo convierte en un formato muy utilizado para la descripción del contenido de otros tipos de dispositivos como memorias, o microcontroladores.

Los siguientes apartados pretenden explicar la estructura de los ficheros .hex utilizados en los dispositivos de 8 bits, los cuales son un poco mas sencillos que los .hex utilizados en los dispositivos de 16 y 32 bits. Por tanto sólo se describen los registros de datos y los registros de fin de fichero, pues estos son los únicos utilizados en los archivos .hex de 8 bits.

7.3.1.Formato de un archivo .hex

Un archivo .HEX se estructura en líneas, cada una de las cuales representa un registro. Cada registro se compone de un conjunto de campos con diferente información en función del tipo de registro que se trate. Existen diferentes tipos de registros, los cuales se pueden combinar indistintamente dentro de un mismo fichero:

- Registros de datos ..... (formato .HEX de 8, 16 o 32 bits)
- Registros de fin de fichero ..... (formato .HEX de 8, 16 o 32 bits)
- Registro de dirección extendida de segmento. (formato .HEX de 16 o 32 bits)
- Registro de dirección de inicio de segmento... (formato .HEX de 16 o 32 bits)
- Registro de dirección lineal extendida..... (formato .HEX de 32 bits)
- Registro de inicio de dirección lineal ..... (formato .HEX de 32 bits)

7.3.2.Formato general de los registros

La estructura general de un registro es:

1-ASCII		RECORD MARK ':'
2-ASCII	1-byte	RECLEN
4-ASCII	2-byte	LOAD OFFSET
2-ASCII	1-byte	RECTYP
2n-ASCII	n-byte	INFO or DATA
2 ASCII	1-byte	CHECKSUM

RECORD MARK Marca de registro

Indica el inicio de registro, es decir que comienza un nuevo registro.

RECLEN Longitud del registro 1-byte:

Especifica el numero de bytes de información (INFO OR DATA) que hay tras el campo RECTYP, es decir el número de bytes que hay entre RECTYP y CHECKSUM, estos no incluidos. No hace falta recordar que un byte se representa mediante dos digitos hexadecimales, por lo que se usaran dos caracteres ASCII (0..F) por byte.

LOAD OFFSET Dirección de memoria 2-bytes:

Aunque aparece en todos los tipos de registros, este campo solo se usa realmente en los registros de datos. Indica la dirección de memoria inicial donde se han de guardar los datos contenidos en este. Los n bytes contenidos en este registro se guardaran en las direcciones sucesivas a la dirección indicada por LOAD OFFSET, por lo que el byte que ocupa la posición n dentro del campo de datos del registro se guardara en la posición LOAD OFFSET + n de la memoria.

RECTYP Tipo de registro 1-byte:

Indica el tipo de información contenida en el registro, es decir el tipo de registro que se esta leiendo, lo que permite anticipar que tipo de campos se han de leer de este, es decir

que campos se van a encontrar. Puede tomar diferentes valores:

00 - Registros de datos

01 - Registros de fin de fichero

02 - Registro de dirección extendida de segmento

03 - Registro de dirección de inicio de segmento

04 - Registro de dirección lineal extendida

05 - Registro de inicio de dirección lineal

INFO or DATA Información o datos n-bytes :

Contienen diferente información que se interpreta de una u otra forma en función del tipo de registro.

CHECKSUM Checksum de comprobación 1-byte:

Es el byte de comprobación de integridad del resto de campos del registro. Este campo es la suma complementada de todos los bytes del registro desde RECLEN hasta el ultimo byte del campo INFO OR DATA, ambos bytes incluidos. Si el fichero esta en buen estado, la suma de estos bytes y el byte de CHECSKUM debería dar 00h, debido a que son valores complementarios. El RECORD MARK no interviene en la suma.

Ej: Dado el registro (recordar que los valores están en hexadecimal) ": 02 00 00 00 07 C0 37 " la comprobación del checksum se haría de la siguiente manera: primero se suman todos los bytes desde RECLEN hasta el ultimo campo INFO or DATA : 02+00+00+00+07+CO = C9, a continuación el valor obtenido se suma al valor de checksum: C9+37 = 1) 00. La suma es 0 por tanto los dos valores se complementan, lo que indica que no hay ningún error en el registro.

7.3.3.Formato de los registros de datos

Estos registros describen una porción de memoria: contienen los datos y la dirección donde se han de ubicar estos.

1-ASCII		RECORD MARK ':'
2-ASCII	1-byte	RECLEN
4-ASCII	2-bytes	LOAD OFFSET
2-ASCII	1-byte	RECTYP '00'
2-n ASCII	n-bytes	DATA
2-ASCII	1-byte	CHECKSUM

En estos registros el campo RECLEN puede valer hasta FFh, lo que significa que puede contener hasta 255 bytes de datos.

El campo LOAD OFFSET contiene la dirección donde se guarda el primer byte de la secuencia de datos contenida en el registro de datos, así la dirección de los diferentes bytes de la secuencia se puede obtener sumando su posición a LOAD OFFSET. A la hora de interpretar la dirección hay que tener en cuenta si se trabaja con archivos .HEX de 8 bits, de 16 bits o de 32 bits ya que en los formatos de 16 o 32 bits LOAD OFFSET sólo da parte de la dirección, la otra parte se obtiene a partir de otros tipos de registros.

7.3.4.Formato del registro de fin de fichero

Este registro indica el fin del fichero .hex . Los campos de este siempre toman los mismos valores:

1-ASCII		RECORD MARK ‘:’
2-ASCII	1-byte	RECLEN ‘00’
4-ASCII	2-bytes	LOAD OFFSET ‘0000’
2-ASCII	1-byte	RECTYP ‘01’
2-ASCII	1-byte	CHECKSUM ‘FF’

7.3.5.Resto de registros

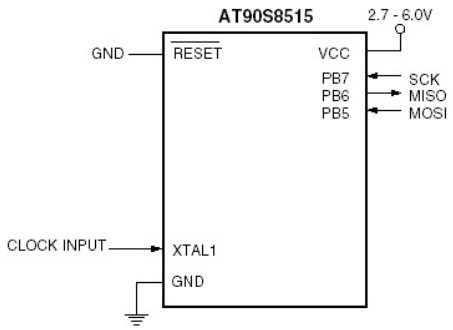
En el documento “Hexadecimal file object specification” de Intel se puede encontrar la descripción detallada de la estructura y funcionalidad del resto de registros necesarios sólo si se desea trabajar con sistemas de 16 o 32 bits.

7.4.Programación serie del Avr8515

Otro punto importante en el desarrollo de la aplicación de telecarga ha sido implementar adecuadamente el protocolo de programación del microcontrolador, el cual se puede programar mediante un protocolo serie o mediante un protocolo paralelo: los siguientes apartados describen la programación mediante el protocolo serie. En este proyecto se utiliza el protocolo de programación serie del microcontrolador, implementado mediante el puerto paralelo del ordenador personal el cual ataca la interfaz para periféricos serie (SPI) del microcontrolador.

El proceso consiste en enviar y grabar mediante el protocolo adecuado el contenido que se desea tengan las memorias Flash y EEPROM del microcontrolador y que se encuentra en el ordenador personal. La memoria Flash es la memoria de programa (contiene el código) mientras que la memoria EEPROM es la memoria de datos. El contenido de la memoria Flash no se puede modificar en tiempo de ejecución, mientras que el de la EEPROM sí.

Todo el proceso de programación se realiza a través de la interfaz serie SPI (Serial Peripheral Interface) del microcontrolador. Este se realiza a través del puerto paralelo del PC, conectando algunos de los pins de datos de este a las lineas correspondientes de la SPI del microcontrolador. Los pins implicados son SCK (PB7), MISO (PB6), MOSI (PB5), RESET y XTAL1. El pin XTAL1 no es obligatorio controlarlo desde el programador, y si se quiere ahorrar trabajo se puede conectar directamente a un cristal siguiendo el esquema convencional. La única restricción que se ha de cumplir es que el periodo de este debe ser como mínimo la cuarta parte del empleado en la señal SCK.



Para escribir se envían los diferentes bytes, bit a bit al pin MOSI del microcontrolador: este considera un bit válido en el flanco de subida del pin SCK. En cambio para leer un byte, se captura bit a bit, del pin MISO del microcontrolador: los bits son validos en el flanco de bajada del pin SCK.

7.4.1.Envío y recepción de un byte a través de la SPI

Antes de todo, hay que comprender como se realiza el intercambio de datos a través del la SPI. Básicamente lo que se hace es enviar en serie cada uno de los bits del byte a enviar, empezando por el bit de mas peso y acabando con el de menos peso. Para ello

hay que colocar el bit a enviar en MOSI y una vez este esta listo, en dar un pulso a SCK para que el microcontrolador lo capture. El microcontrolador captura el valor al subir SCK, mientras que su respuesta se considera valida al bajar SCK. Un ejemplo del algoritmo a seguir es:

```
procedimiento enviar_recibir(byte_enviado tipo byte, ref byte_recibido tipo byte)
   para peso=7 hasta 0 hacer
         ClearSck                                  ‘se baja la señal SCK
         GeneraClocks (4)
         SacarBitMOSI(byte_enviado,peso)‘se pone el bit corresopondiente en MOSI
         GeneraClocks (4)
         SetSck                                    ‘se sube la señal SCK
         GeneraClocks (4)
         EntrarBitMISO(byte_recivido,peso)         ‘se lee el bit recibido del micro
  fin_para
fin procedimiento
```

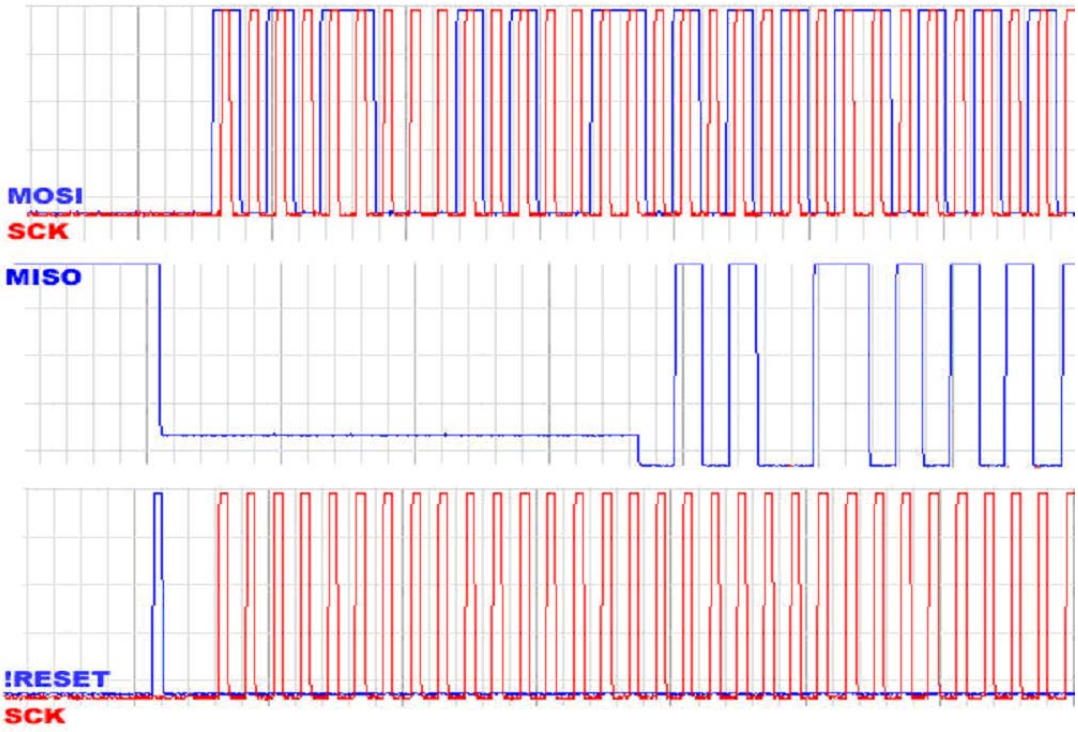


Fig 7.4.1.1 ejemplo de activación seguida de envío y recepción : se puede observar el proceso de activación y parte del envío de la secuencia de inicialización del modo de programación.

Para entrar en el modo de programación del microcontrolador hay que seguir la secuencia:

-Activación de la programación.

-Envío de la instrucción de Inicialización del Modo de Programación.

-Envío de la instrucción de la operación que se quiera realizar:

- Instrucción de Borrado del Chip
- Instrucción de Escritura en la Memoria de Programa (Flash)
- Instrucción de Lectura de Memoria de Programa (Flash)
- Instrucción de Escritura en la Memoria EEPROM
- Instrucción de Lectura de la Memoria EEPROM

7.4.2.Activación de la programación

El primer paso es mantener !RESET y SCK a 0V en el momento que se da alimentación al microcontrolador. Si no existe un oscilador conectado habrá que dar un pulso de clock a XTAL1. Si por alguna razón no se puede mantener SCK y RESET a 0V en el momento del encendido, habrá que dar un pulso a !RESET de al menos dos ciclos de XTAL1 de duración. A continuación hay que esperar unos 20 ms. Els siguiente paso es el envío de la Instrucción de Inicialización del Modo de Programación.

Un algoritmo válido para implementar la activación de la programación seria el siguiente:

```
procedimiento activa_programacion()
        ClearReset                  ‘se deja !RESET a 0
        ClearSck                    ‘se deja SCK a 0
        ActivarVcc                  ‘se activa la alimentación
        GeneraClocks(1)             ‘se aplica un pulso de clock a la senyal Xtal1
        SetReset                    ‘se da un pulso a !RESET
        GeneraClocks(4)
        ClearReset
        Espera 20 msg               ‘se espera 20 msg
fin procedimiento
```

La función GeneraClocks no es necesaria si se utiliza un cristal externo, en este caso en lugar de las llamadas a esta función habrá que situar una pausa de la duración conveniente por tal de prolongar la duración de las señales.

7.4.3.Instrucción de inicialización del modo de programación

Una vez activada la programación hay que enviar al microcontrolador la secuencia correspondiente a la Instrucción de Inicialización del Modo de Programación: “10101100 , 01010011 , XXXXXXXX , XXXXXXXX” (X=cualquier valor). Si existe

sincronización con el microcontrolador este aceptará la secuencia y devolverá el eco del segundo byte e irá respondiendo a medida que le vamos enviando el resto de bytes. Es decir que si todo va bien durante la transmisión del tercer byte este nos enviara un 01010011 (53h). Si el microcontrolador no responde es que algo no ha salido bien, y deberemos intentar enviar la secuencia de nuevo. En el datasheet se especifica que si después de 32 intentos no se recibe respuesta es que no hay microcontrolador o esta mal conectado, pero lo mas seguro es que si en un par o dos de intentos no recibimos respuesta el microcontrolador no responda en los siguientes 30 intentos restantes, así que con probarlo 2 o 3 veces ya es suficiente.

Para poder ejecutar cualquier otra instrucción, antes hay que haber seguido la secuencia de activación, y haber enviado la Instrucción de Inicialización del Modo de Programación correctamente, sino no se podrá hacer nada.

7.4.4.Instrucción de borrado de la memoria

Esta instrucción borra el contenido de la memoria Flash, memoria EPROM y de los Lock Bit, mientras que no modifica los Fuse Bits. Es aconsejable ejecutar esta instrucción antes de programar el microcontrolador. La secuencia a enviar es: “10101100 , 100XXXXX , XXXXXXXX , XXXXXXXX” (X=cualquier valor). No hace falta decir que antes de poder enviar esta secuencia hay que haber activado el modo de programación y haber enviado la Instrucción de Inicialización del Modo de Programación correctamente.

7.4.5.Instrucción de escritura en la memoria de programa (Flash)

El microcontrolador AVR8515 dispone de una memoria de programa de 8k organizada en 4096 posiciones de 16bits, cada una de las cuales se divide en una parte alta y en una parte baja, amabas de 8 bits. El contenido de esta memoria se modifica mediante la Instrucción de Escritura en la Memoria de Programa: se hacen dos accesos para cada dirección. En el primer acceso se actualiza la parte alta de la dirección de memoria, y en el segundo la parte baja, o viceversa. La secuencia es: “0100H000 , XXXXAAAA , BBBBBBBB , IIIIIIII” (H=bit que indica si se modifica la parte alta H=1, o la parte baja

H=0 , X=cualquier valor, AAAA = bits altos 11..8 de la dirección, BBBBBBBB = bits bajos 7..0 de la dirección, IIIIIIII = byte a guardar en la memoria). Por tanto hay que enviar dos secuencias de 4 bytes cada una: una para actualizar la parte alta, y la otra para actualizar la parte baja.

La actualización de un byte en memoria no es inmediata, por lo que después de escribir un valor en la memoria de programa hay que esperar un pequeño intervalo de tiempo antes de escribir el siguiente valor, este intervalo se especifica en el datasheet como Twd_prog. Una alternativa a la espera es hacer lecturas succesivas sobre la dirección escrita, de forma que si este todavía no ha terminado de escribir el valor, retornara el valor 7Fh, mientras que si la escritura ha terminado retornara el valor escrito, lo que indica que el microcontrolador ya esta listo para escribir el siguiente valor. Lógicamente el método de polling no es útil cuando se guarda el valor 7Fh en memoria, y en este caso solo queda la opción de esperar Twd_prog.

Como con todas las demás instrucciones, es obvio que antes de poder enviar la secuencia de Escritura en la Memoria de Programa hay que haber Activado el programador y haber enviado la Instrucción de Inicialización del Modo de Programación correctamente.

7.4.6.Instrucción de lectura de memoria de programa (Flash)

Como la memoria de programa se estructura en 4096 posiciones de 16 bits, y la instrucción de Lectura de Memoria de Programa solo permite leer un byte, hay que hacer 2 accesos a la misma dirección de memoria por tal de leer su contenido completo: en un primer acceso se lee la parte alta, y en un segundo acceso se lee la parte baja, o viceversa. La secuencia de la instrucción de Lectura de Memoria de Programa es la siguiente: “0010H000 , XXXXAAAA , BBBBBBBB , oooooooo” (H=bit que indica si se consulta el byte de la parte alta H=1, o el de la parte baja H=0 , X=cualquier valor, AAAA = bits altos 11..8 de la direccion, BBBBBBBB = bits bajos 7..0 de la direccion, oooooo byte en el que el micrcontrolador envia el valor). Asi, cuando se envia el ultimo byte de la secuencia, el microcontrolador responde a traves de MISO con el byte almacenado en la parte alta o baja de la posicion consultada.

Al contrario que en la escritura, al leer succesivos valores de memoria, no es necesario hacer ninguna pausa.

Como sucede con todas las demas instrucciones, antes de poder enviar la secuencia de Lectura en la Memoria de Programa hay que haber Activado el programador y haber enviado la Instruccion de Inicializacion del Modo de Programacion correctamente.

7.4.7.Instrucción de escritura en la memoria de datos (EEPROM)

El microcontrolador AVR8515 dispone de una memoria EEPROM de 512 bytes organizada en 512 posiciones de 8bits, para datos. El contenido de esta memoria se programa mediante la Instrucción de Escritura en la Memoria EEPROM. La secuencia correspondiente a esta instrucción es: “11000000 , XXXXXXXA , BBBBBBBB , IIIIIIII” (A = bit alto 8 de la dirección, BBBBBBBB = bits bajos 7..0 de la dirección, IIIIIIII = byte a guardar en la memoria). Cada dirección de la memoria EPROM se programa mediante un único acceso a memoria, es decir de una sola vez, al contrario de lo que sucede con la memoria de programa.

Al igual que con la memoria Flash, la actualización de un byte en esta memoria no es immediata, por lo que después de escribir un valor hay que esperar un pequeño intervalo de tiempo antes de escrbir el siguiente, la duración de este se especifica en el datasheet como Twd_prog. También se pueden utilizar técnicas de polling, pero lo más aconsejable es esperar siempre el intervalo de tiempo Twd_prog.

Como con todas las demás instrucciones, no hace falta decir que antes de poder enviar la secuencia de Escritura en la Memoria de Programa hay que haber Activado el programador y haber enviado la Instrucción de Inicialización del Modo de Programación correctamente.

7.4.8.Instrucción de lectura de la memoria EEPROM

La lectura del contenido de una posición de la memoria EEPROM se hace en único acceso a la memoria mediante la instrucción de Lectura de Memoria EEPROM. La secuencia de la instrucción de Lectura de Memoria EEPROM es la siguiente: “10100000 , XXXXXXXA , BBBBBBBB , oooooooo” (A = bit alto 8 de la dirección, BBBBBBBB = bits bajos 7..0 de la dirección, oooooooo byte en el que el micrcontrolador envía el valor). Así, cuando se envía el último byte de la secuencia, el microcontrolador responde a través de MISO con el byte almacenado en la posición consultada. Al contrario que en la escritura, al leer succesivos valores de memoria, no es necesario hacer ninguna pausa.

Como sucede con todas las demás instrucciones, antes de poder enviar la secuencia de Lectura en la Memoria EEPROM hay que haber activado el programador y haber enviado la Instrucción de Inicialización del Modo de Programación correctamente.

Capítulo 8
Interfaz de usuario: visualizador y pulsadores

8.1.Introducción

Tal como se comenta en el Capítulo 1, se ha diseñado también una placa de registro de trayectorias que hace uso de la plataforma de desarrollo GPS hasta aquí descrita.. La placa registradora de trayectorias sirve a su vez para demostrar las amplias posibilidades de esta plataforma Esta placa de registro ha de disponer de una interfaz de usuario para que este pueda controlar el módulo y visualizar la información suministrada. La interfaz de entrada ha de permitir al usuario desplazarse por los menús y seleccionar las distintas opciones. La interfaz de salida ha de poder mostrar al usuario la información de posición, de hora, los menús ... Como se puede ver, sólo se va a tener que mostrar información en forma de texto, con lo que cualquier medio de visualización alfanumérico será suficiente.

8.2.Visualizador alfanumérico

El módulo registrador GPS construido sobre la placa de desarrollo ha de poder mostrar toda la información a través de algún tipo de medio de visualización de tipo alfanumérico o gráfico. En un principio, las alternativas planteadas han sido el uso de visualizadores 16 segmentos, de un visualizador LCD gráfico, o de un visualizador LCD alfanumérico. La primera de las opciones se ha descartado casi de inmediato debido a las limitaciones que esta conlleva, ya que se necesitaría un mínimo de unos 20

visualizadores más toda la electrónica para su control. Obviamente esto complicaría muchísimo la placa y aumentaría considerablemente su tamaño a parte de su precio final.

La opción correspondiente a la utilización de un LCD gráfico es una opción interesante debido a que esta permitiría combinar el texto con gráficos haciendo la interfaz más agradable, pero tiene como inconveniente que estos visualizadores son un poco complejos de programar, a parte de que por lo general tienen un precio relativamente elevado para las pretensiones de este proyecto.

Finalmente se ha considerado como mejor opción la utilización de un visualizador LCD alfanumérico. Se ha escogido un modelo compatible con el estándar HD44780 debido a que estos consumen poco (sin retroiluminación), se controlan mediante un protocolo muy sencillo, ocupan solo 4+3 o 8+3 bits de entrada /salida del micrcontrolador, y son muy fáciles de encontrar en las tiendas de electrónica.

En el Anexo C se describe con detalle el modo de funcionamiento de este tipo de LCDs.

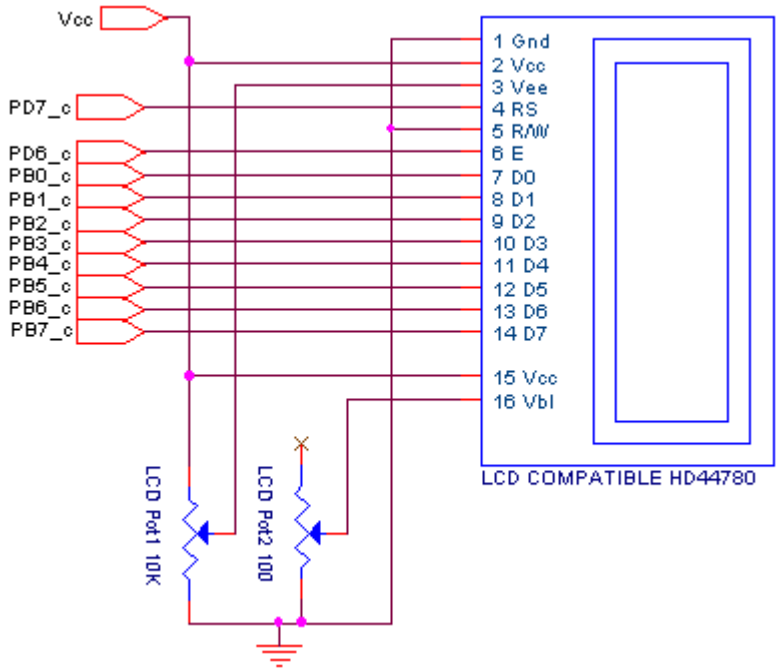


Fig 8.2.1 Esquema y conexionado del display LCD en modo 8 bits

8.3.Pulsadores mediante los que manipular los menús de selección

La elección de la interfaz de entrada no ha supuesto ningún problema ya que tres pulsadores han sido suficientes para cubrir las necesidades del proyecto. Inicialmente se pensó en utilizar un teclado de matriz, pero se ha visto que un sistema de menús inteligentemente diseñado, hace innecesario el uso de más de 3 teclas, lo que simplifica el conexionado, el tamaño y complejidad de la interfaz de entrada. El pulsador A permite avanzar a través de las opciones del menú, el B permite retroceder, y el C permite seleccionar las distintas opciones de este. Como sólo son tres pulsadores, no es necesario realizar ningún tipo de rastreo al estilo del utilizado en los teclados de matriz, así que los tres pulsadores se conectan directamente a tres bits de entrada del puerto A, con sus respectivas resistencias de pull-down: el pulsador A a PA0, el pulsador B a PA1, y el C a PA2 tal como muestra el esquema:

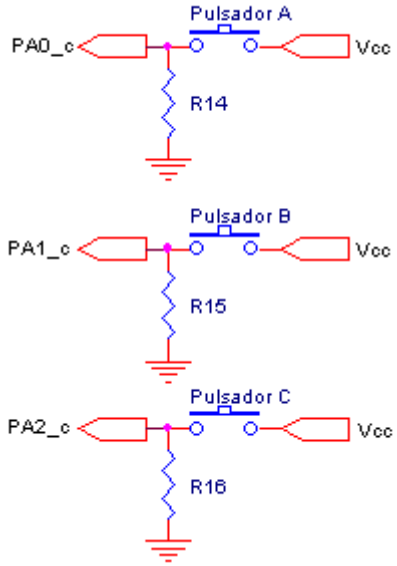


Fig 8.3.1 Esquema y conexionado de los pulsadores

R14=R15=R16=1K

Capítulo 9
Dispositivo de memoria

9.1.Introducción

El registrador de trayectorias ha de ser capaz de almacenar la trayectoria (secuencia de posiciones GPS) que ha seguido un móvil. Por tanto es necesario disponer de algún tipo de memoria no volátil que permita almacenar la información GPS que va suministrando la plataforma de desarrollo.

En consonancia con la filosofía seguida en todo el proyecto, el dispositivo elegido ha de ajustarse al máximo a los criterios de disponibilidad, escalabilidad, coste ...

9.2.Tipos de memorias no volátiles

Es prioritario el uso de un dispositivo de memoria no volátil debido a que el uso de uno volátil obligaría a mantener encendido el sistema después de haber tomado los datos, lo que iría en detrimento del consumo. Por tanto es evidente que la opción más inteligente es la correspondiente a la utilización de una memoria no volátil. En el mercado existen distintos tipos de memorias no volátiles:

-ROM (Read Only Memory) : son memorias cuyo contenido se fija durante el proceso de manufactura, y no puede ser modificado por el microcontrolador, únicamente puede ser leído. Evidentemente estas no se adaptan, ni muchísimo menos, a los requerimientos del proyecto. Este tipo de memoria únicamente se utiliza en producciones industriales, en los que se necesita realizar una tirada numerosa de integrados preprogramados.

-PROM (Programable Read Only Memory) : vienen a ser ROMs que pueden ser programadas una única vez por el usuario mediante el programador adecuado, tras ser programadas, sobre estas solo se pueden hacer operaciones de lectura, de forma que si se produce algún error durante la programación esta restará inservible. Al igual que en el caso anterior, este tipo de memoria no es nada útil para el propósito de este proyecto, ya que se debería desechar un integrado tras registrar cada trayectoria, algo totalmente absurdo.

-EPROM (Erasable Programable Read Only Memory) : estas son memorias no volátiles que pueden ser borradas lo que permite reutilizarlas tras cada operación de programación. La operación de borrado elimina toda la información por completo, es decir que no se puede eliminar información puntual. Otro gran inconveniente de estas es que para borrar una EPROM es necesario exponerlas a una fuente de luz ultravioleta durante unos 15 minutos, lo que convierte a esta en una solución poco práctica para el proyecto, ya que implica tener que desmontarla de la placa y esperar 15 minutos para borrarla cada vez que se quisiera inicializar, a parte de que obliga al usuario a disponer de una lámpara de UV. Otra desventaja es que para escribir en ella es necesario utilizar tensiones de 12 o 24V lo que complicaría más el diseño del hardware.

-EEPROM (Electrically Erasable Programable Read Only Memory) : son memorias con las mismas prestaciones que las EPROM pero con la ventaja de que la operación de borrado se puede realizar electrónicamente, es decir que no hace falta exponer estas a una fuente de luz ultravioleta, lo que evita tener que extraerlas de la placa. Además, como se puede borrar eléctricamente, es posible borrar o escribir palabras individuales desde el propio microcontrolador sin necesitar ningún programador específico. Para escribir no es necesario borrar antes. Las memorias EEPROM se suele utilizar cuando las operaciones predominantes son las de lectura, ya que en estas el tiempo de escritura es bastante mayor que el de lectura.

-Flash: este tipo de memoria, al igual que las EEPROM también se puede borrar eléctricamente, pero una de las principales diferencias es que la tecnología de tipo Flash, permite una mayor densidad de integración lo que se traduce en capacidades de almacenamiento mayores, a parte de que el coste de estas también es inferior al delas

EEPROMs. Otra ventaja de las memorias Flash respecto a las EEPROM es su mayor velocidad de acceso. El único punto en contra de las Flash es que generalmente las operaciones de borrado no se pueden hacer sobre palabras puntuales, sino que se suelen hacer en bloques.

La mayor capacidad, y menor coste han convertido a las memorias Flash en el tipo de memoria no volátil más utilizado en dispositivos móviles. Estas se vende en forma de integrados, existiendo infinidad de modelos con diferentes prestaciones orientados al mercado industrial, y también en forma de tarjetas externas orientadas al mercado de consumo (cámaras, agendas electrónicas, teléfonos móviles ...). Existen diferentes tipos y modelos de tarjetas Flash, las cuales tienen prestaciones similares en cuanto a la capacidades tamaño y precio. Actualmente los tipos más extendidos son: Compact Flash, SD-MultiMediaCard, Smart Media , Memory Stick y XD .

Fig 9.2.1 Imágenes de los distintos tipos de tarjetas de memoria Flash disponibles en el mercado. De izquierda a derecha: MemorySitck (Sony), XD (Olympus Optical y Fuji Photo Film), SD/MultiMediaCard , SmartMedia (Olympus Optical y Fujy Foto Film) , Compact Flash.

Como se puede ver, las únicas opciones que podrían resultar interesantes para el almacenamiento de datos en este proyecto son las correspondientes a las memorias de tipo EEPROM o de tipo Flash, ya que son los únicos que permiten el borrado y escritura de forma sencilla desde el propio microcontrolador.

Finalmente se ha elegido una memoria de tipo Flash debido a que son las que ofrecen mayor capacidad, y se pueden adquirir en forma de tarjetas extraíbles, lo que permite modificar la capacidad de nuestro dispositivo cambiando únicamente la tarjeta sin tener

que desoldar ningún integrado. Además estas son bastante baratas, fáciles de encontrar (incluso en supermercados!) y tienen un tamaño reducido. En concreto se ha escogido el tipo SD-MultiMediaCard ya que en la actualidad es el tipo de tarjeta con mayor proyección, y porque dispone de un protocolo de acceso serie que reduce el número de pins necesarios para accederla, simplificando a su vez el hardware necesario para su control.

9.3.Las tarjetas MultiMediaCard

Las tarjetas MultiMediaCard son dispositivos de almacenamiento Flash muy versátiles debido a su gran capacidad, y reducido tamaño y precio. Esto las convierte en una opción muy interesante a usar en dispositivos móviles como teléfonos, agendas electrónicas o videojuegos portátiles.

Fig 9.3.1 Tarjeta MultiMediaCard con sus pins numerados.

Desde el punto de vista técnico, estas pueden trabajar mediante dos protocolos serie distintos: el protocolo MultiMediaCard propiamente dicho, y el protocolo SPI. El primero de los protocolos es el más potente ya que permite más operaciones que el segundo, pero por otro lado, el segundo es más fácil de implementar si se dispone de una interfaz SPI y es suficiente para la mayoría de aplicaciones. De hecho el protocolo SPI se puede considerar como una versión reducida del protocolo MultiMediaCard. Los siguientes apartados muestran como acceder las tarjetas MultiMediaCard mediante el protocolo SPI.

9.3.1.Conexiones y señales

El protocolo de acceso SPI se puede implementar mediante cualquier controlador (ordenador personal, microcontrolador ...) que disponga de un canal SPI hardware o emulado por software. Cuando se opera en este modo los pins de la tarjeta se denominan del siguiente modo:

# Pin:	Nombre:	Tipo:	Descripción SPI:
1	CS	Entrada	Chip Select (activo a 0)
2	DataIn	Entrada	Línea de datos y comandos hacia la tarjeta.
3	VSS1	Alimentación	Masa
4	VDD	Alimentación	Alimentación
5	CLK	Entrada	Clock
6	VSS2	Alimentación	Masa
7	DataOut	Salida	Línea de datos y status de la tarjeta al microcontrolador.

Líneas de la interfaz SPI

CS permite al controlador seleccionar la tarjeta sobre la cual quiere operar, así cuando CS vale 0 la tarjeta se encuentra seleccionada y lista para operar. Este pin puede ser controlado por cualquier pin de salida del controlador.

DataIn es la entrada de datos serie a la tarjeta y debe estar conectada a la salida MOSI de la interfaz SPI del controlador.

DataOut es la salida de datos serie de la tarjeta y debe estar conectada al pin MISO de la interfaz SPI del controlador.

CLCK es la señal de reloj generada por el controlador y es la que marca el ritmo de transferencia de la información serie entre ambos, así los datos se capturan o transmiten por la tarjeta al ritmo marcado por esta señal.

VDD es el pin de alimentación y **VSS1** y **VSS2** son los pins de masa.

Tanto la tensión de alimentación como las señales deben encontrarse en el rango de los 2,6 y 3,7V.

9.3.2.Funcionamiento general del protocolo

Básicamente el protocolo SPI consiste en el intercambio de información entre el controlador (master) y la tarjeta (slave). Este intercambio se lleva a cabo mediante el envío de comandos por parte del controlador y de respuestas por parte de la tarjeta. Así,

en la lectura, el controlador envía el comando de petición de lectura a la tarjeta y esta le envía la respuesta de confirmación seguida del bloque de datos con la información contenida a partir de la dirección solicitada. En la escritura el proceso es parecido, el controlador indica a la tarjeta mediante el comando de escritura que quiere escribir información en una determinada dirección, esta le responde indicando que esta lista y a continuación el controlador envía el bloque de datos a escribir. Las operaciones que no requieren intercambio de datos funcionan de igual forma pero sin usar bloques de datos.

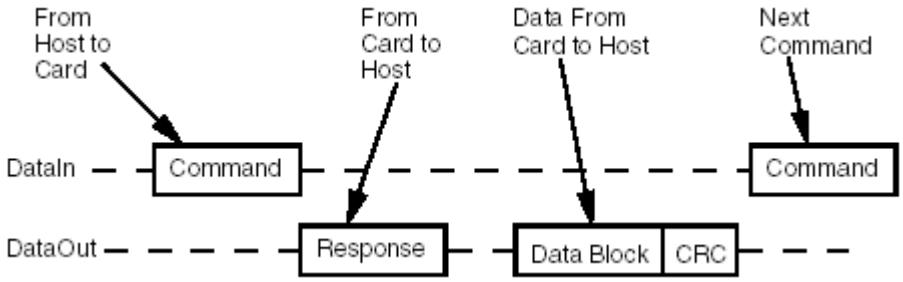


Fig 9.3.2.1 Ejemplo de comunicación. Lectura de un bloque.

9.3.3.Comandos

Los comandos son bloques de bytes con un formato fijo:

Byte 1			Bytes 2 - 5	Byte 6	
7	6	5 0	31 0	7	0
0	1	Command	Command Argument	CRC	1

La tarjeta identifica los comandos porque el primer byte de estos (byte 1) siempre comienza por 01, el resto de bits del primer byte contiene el numero de comando codificado en binario natural. Así el primer byte correspondiente al comando 0 (CMD0) seria: 01000000, o el primer byte correspondiente al comando 39 (CMD39) sería: 01100111. Los siguientes 4 bytes (bytes 2 –5) contienen los argumentos del comando. En los comandos que no requieren argumentos estos valen 0. El último byte (byte 6) es el byte de CRC para la verificación de errores y en realidad en el protocolo SPI no se utiliza, a no ser que en el registro de configuración se especifique que se desea utilizar CRC.

Toda la comunicación entre la tarjeta y el controlador se realiza según el orden del esquema, de izquierda a derecha, es decir que primero se transmite el bit de mas peso (bit 7) del byte 1, y por último el bit de menos peso (bit 0) del byte 6, es decir es una transferencia More Significant Bit First (MSB First). Estos son algunos de los principales comandos:

Comando:	**Argumentos:**	**Respuesta:**	**Descripción:**
CMD0	No	R1	Resetea la tarjeta
CMD1	No	R1	Inicializa la tarjeta
CMD9	No	R1	Pide a la tarjeta su información CSD
CMD10	No	R1	Pide a la tarjeta su identificación CID
CMD13	No	R2	Consulta el estado de la tarjeta
CMD16	[31..0] Longitud del bloque.	R1	Establece la longitud (en bytes) del bloque para los datos en las operaciones de lectura y escritura.
CMD17	[31..0] Dirección de datos.	R1	Lee un bloque del tamaño indicado por el comando 16.
CMD24	[31..0] Dirección de datos	R1 R1 R1	Escribe un bloque del tamaño indicado por el comando 16.

Principales comandos del protocolo de acceso mediante SPI

9.3.4.Respuestas

Las respuestas de la tarjeta son bloques formados por 1 o 2 bytes, dependiendo del tipo de respuesta que se trate. El tipo de respuesta es función del comando, es decir que cada comando tiene asociado un tipo de respuesta.

Bit:	
0	Idle State
1	Erase Reset
2	Illegal Command
3	Com CRC Error
4	Erase_Seq_Error
5	Address Error
6	Parameter Error
7	0

Respuestas R1 y RB

Bit	
0	0
1	WP Erase Skip
2	Error
3	CC Error
4	Card ECC Failed
5	WP Violation
6	Erase Param
7	Out of Range
0	In Idle State
1	Erase Reset
2	Illegal Command
3	Com CRC error
4	Erase_Seq_Error
5	Address Error
6	Parameter Error
7	0

Respuesta R2

9.3.5.Bloques de datos

Los bloques de datos comienzan siempre con el byte 0xFE, a este le siguen los bytes de datos y por último los 2 bytes de CRC. El número de bytes de datos depende de la longitud de bloque definida mediante el comando 16, y este puede ir de 1 hasta 512 bytes (por defecto 512). Por tanto sumando a los bytes de datos el byte de incio y los dos bytes de CRC la longitud total del bloque de datos puede variar entre 4 y 512 bytes. Como por defecto en el protocolo de acceso SPI no se consideran los bytes de CRC, estos pueden tomar cualquier valor.

9.3.6.Reset de la tarjeta

Al arrancar, la tarjeta se encuentra por defecto en modo MultiMediaCard. Para que entre en modo SPI, hay que enviarle el comando 0 mientras se mantiene activa la señal ¡CS (¡CS=0), pero antes de todo, para poder iniciar la comunicación por el bus hay que

enviar como mínimo 74 ciclos de clock a la tarjeta. Así para hacer el reset de la tarjeta y prepararla para trabajar en modo SPI hay que seguir la siguiente secuencia:
-Dar como mínimo 74 ciclos de clock, es decir enviar unos 10 bytes a través de la SPI.
-Activar la señal ¡CS (¡CS=0).
-Enviar el comando 0 con el CRC bien calculado, ya que todavía no estamos en modo SPI, por lo que sí se considera el CRC. De hecho la secuencia del comando 0 siempre es la misma: 0x40,0x00,0x00,0x00,0x00,0x95
-Esperar el byte de respuesta que ha de ser 00000001 (tarjeta en modo idle).

9.3.7.Activar la inicialización de la tarjeta

Una vez reseteada y en modo SPI, hay que activar la inicialización de la tarjeta, para ello hay que enviar el comando 1. Simplemente hay que seguir la secuencia:
-Activar la señal ¡CS (¡CS=0).
-Enviar el comando 1 0x41,0x00,0x00,0x00,0x00, 0xXX. Como la tarjeta ya esta en modo SPI el CRC puede tomar cualquier valor.
-Esperar el byte de respuesta que ha de valer 00000000 (tarjeta lista).

9.3.8.Escritura de un bloque en la tarjeta

Una vez inicializada la tarjeta, para escribir un bloque en esta, hay que enviar el comando 24 con la dirección de inicio a partir de la cual se desean guardar los datos. Si todo va bien la tarjeta enviará tres respuestas R1 repetidas informando al controlador que ya puede enviar el bloque de datos, que ha de tener una longitud de 512 bytes (en la escritura solo se permiten 512 bytes) más el byte de inicio de bloque de datos y los dos bytes de CRC. La secuencia a seguir es:
-Activar la señal ¡CS (¡CS=0).
-Enviar el comando 24 0x58, 0xXX,0xXX,0xXX,0xXX,0xYY. Los 4 bytes XX corresponden a la dirección a partir de la cual se quieren guardar los datos. 0xYY corresponde al byte de CRC y como la tarjeta esta en modo SPI pueden tomar cualquier valor ya que no se consideran.
-Si todo va bien la tarjeta responde con el byte de respuesta R1 tres veces consecutivas.
-Enviar a la tarjeta el bloque de datos que consiste en:

- 1 byte de inicio de bloque de datos 0xFE
- 512 bytes con los datos a guardar.
- 2 bytes de CRC

-Mientras la tarjeta esta ocupada guardando el valor, irá enviando bytes indicando que está ocupada, y cuando finalice la escritura enviará un byte de confirmación.

9.3.9.Lectura de un bloque de la tarjeta

Para leer un bloque de la tarjeta hay que enviar a esta el comando 17 con la dirección de inicio de lectura en los bytes de argumento. La dirección puede tomar cualquier valor comprendido dentro del rango de direcciones válidas de la tarjeta pero todo el bloque leído debe estar dentro de un mismo sector físico. A continuación la tarjeta envía un byte de respuesta R1 seguido del bloque de datos, que comienza por 0xFE, continua con los bytes de datos y finaliza con los 2 bytes de CRC que no se usan. El número de bytes de datos depende del tamaño de bloque que se haya programado mediante el comando 16, y en la lectura puede ir de 1 a 512. La secuencia a seguir es la siguiente:

-Activar la señal ¡CS (¡CS=0).

-Enviar el comando 17 0x51,0xXX,0xXX,0xXX,0xXX,0xYY. Los 4 bytes XX corresponden a la dirección a partir de la cual se quieren leer los datos. 0xYY corresponde al byte de CRC y como la tarjeta esta en modo SPI puede tomar cualquier valor ya que no se considera.

-Si todo va bien, la tarjeta responde con un byte de respuesta R1, seguido del bloque de datos con la información solicitada y que el controlador tendrá que ir capturando. Esta tiene la misma estructura que los bloques de datos utilizados en la escritura:

-1 byte de inicio de bloque de datos 0xFE

-n bytes con los datos a guardar.

-2 bytes de CRC.

-Si se produce un error durante la comunicación, la tarjeta no transmitirá ningún dato y en lugar de estos enviará un byte indicador de error.

Algunos de los bits de control de la tarjeta se podrían modificar: por ejemplo el pin 5 del puerto D se utiliza para activar y desactivar la tarjeta (CS), pero este se podría cambiar por otro en el caso de que fuese necesario.

9.4.Organización de la información en la tarjeta

Cuando la placa está en modo grabación, esta guarda la información GPS en la tarjeta MultiMediaCard cada cierto tiempo. Este tiempo viene marcado en segundos, por la variable REGISTRO_tiempo_muestreo del firmware la cual se puede modificar en tiempo de compilación. Así, cada vez que pasan REGISTRO_tiempo_muestreo segundos se guarda en la tarjeta MultiMediaCard el estado actual de la estructura con la información GPS en ASCII. Lo que se hace es serializar la estructura convirtiéndola en una tira de bytes que luego se envía a la tarjeta a través de la SPI. La tarjeta MultiMediaCard no puede ser escrita en direcciones aleatorias, sino que solo puede ser escrita en bloques de 512 bytes alineados en direcciones múltiples de 512 bytes. Es decir que estas tarjetas se estructuran en bloques de 512 bytes, y para escribir en una dirección primero se ha de acceder a la dirección inicial de su bloque que lógicamente ha de ser multiple de 512, y allí escribir la tira de 512 bytes, o lo que es lo mismo todo el bloque. Es decir, o se escribe todo un bloque, o no se puede escribir nada. Esto entra en conflicto con el tamaño de la estructura de datos GPS del firmware que sólo ocupa 128 bytes, lo que obliga a dejar los 384 últimos bytes del bloque vacíos con lo que ello implica: desaprovechamiento de la capacidad .

Se podrían usar distintas técnicas para aprovechar el espacio libre que queda al final de cada bloque. Una forma de hacerlo sería no escribir nada en la memoria Flash hasta que no se tuvieran 512 bytes de información. Otra opción sería leer todo el bloque, modificar sólo los bytes sobre los que se quisiera escribir, y luego volver a escribir todos los bytes del bloque a la tarjeta. El problema a la hora de implementar estas dos opciones es la limitación de memoria, es decir que no se dispone de suficiente RAM para crear arrays o buffers de 512 bytes.

Así, cada cierto tiempo se escribe un bloque de la tarjeta MultiMediaCard con la información GPS de ese instante. Los bloques se van escribiendo secuencialmente, uno detrás de otro, a medida que va transcurriendo el tiempo. Para saber donde se encuentra el ultimo bloque grabado, cada vez que se escribe un bloque, el bloque siguiente se llena con una tira de caracteres ‘*’.

Así en el momento que se lea la tarjeta se irán leiendo bloques, hasta encontrar un bloque donde todos los caracteres sean ‘*’. Es decir que el bloque con todos los caracteres a ‘*’ es el bloque marcador de fin de grabación.

09/09/2004/22:35:43/00:00:00:00/000.0/42°51.8606’N/002°07.6940’E/00170/07********** ... ***
09/09/2004/22:35:45/00:00:00:00/000.0/42°51.8606’N/002°07.6942’E/00170/07********** ... ***
09/09/2004/22:35:47/00:00:00:00/000.0/42°51.8607’N/002°07.6942’E/00170/07********** ... ***
09/09/2004/22:35:49/00:00:00:00/000.0/42°51.8607’N/002°07.6943’E/00171/07********** ... ***
...
09/09/2004/23:07:52/00:00:00:00/000.0/42°51.7387’N/002°07.1839’E/00201/06********** ... ***
09/09/2004/23:07:54/00:00:00:00/000.0/42°51.7387’N/002°07.1839’E/00201/06********** ... ***
09/09/2004/23:07:56/00:00:00:00/000.0/42°51.7388’N/002°07.1840’E/00201/06********** ... ***
09/09/2004/23:07:58/00:00:00:00/000.0/42°51.7388’N/002°07.1840’E/00201/06********** ... ***
09/09/2004/23:08:00/00:00:00:00/000.0/42°51.7388’N/002°07.1841’E/00201/06********** ... ***
********** ... ***

Fig 9.4.1 Estructura típica de la información guardada en la tarjeta MultiMediaCard, cada línea representa un bloque de 512 bytes. Fijarse en que sólo se aprovechan los primeros 128 bytes y el resto de cada bloque se rellena con el carácter ‘*’. Fijarse también en que la ultima fila escrita (último bloque) se rellena toda con ‘*’.

Capítulo 10
Librerías firmware de soporte

10.1.Introducción

Uno de los objetivos del proyecto es ofrecer sencillez al usuario, por eso es importante que este disponga de herramientas que le simplifiquen el trabajo durante el desarrollo. Una de estas herramientas son las librerías firmware las cuales se pueden incluir opcionalmente en el proyecto. Todas ellas han sido desarrolladas en C con el compilador gcc para AVR-ATmega incluido en el completo entorno de desarrollo gratuito WinAVR. Estas librerias se pueden usar en cualquier proyecto realizado en C o C++, y permiten al usuario realizar tareas relativamente complejas de forma transparente acelerando a su vez el tiempo de desarrollo. Para utilizarlas por tanto es necesario disponer del entorno WinAVR, el cual se pude descargar de forma totalmente gratuita de Internet (ver Bibliografía). El propio paquete incluye toda la documentación necesaria para su instalación, de todas formas en el Apéndice D se presentan alternativas para algunos de los pasos de instalación que no quedan muy claros en la documentación.

Los siguientes apartados muestran las distintas librerías de soporte implementadas para este proyecto.

10.2.Libreria firmware de comunicación por el puerto paralelo: LPTCOM

A pesar de que al final esta no se ha utilizado por problemas de velodidad de las DLL de acceso al puerto paralelo, se ha desarrollado una librería firmware con una serie de rutinas cuyo fin es el de facilitar al usuario la tarea de comunicación a través del puerto paralelo. Este sólo tiene que incluir la librería LPTCOM, formada por los archivos

lptcom.c lptcom.h, en su proyecto y llamar a sus funciones o procedimientos cuando los necesite:

```
void LPTCOM_set_clock();
void LPTCOM_clear_clock();
void LPTCOM_set_send();
void LPTCOM_clear_send();
char LPTCOM_envia_buffer(char byte_ID,char byte_status1, char
byte_status2);
void LPTCOM_asigna_caracter_buffer(char caracter,int i);
```

Obviamente el uso de esta librería es opcional, y el usuario puede omitirla o adaptarla en caso de que no se adecuase a sus necesidades, teniendo siempre en cuenta que modificar los puertos implicaría también modificar el conexionado entre el microcontrolador y el conector DB25 que permite la conexión con el ordenador personal.

10.3.Librería firmware de gestión del módulo GPS: GPS

Se ha creado otra librería para facilitar al usuario las tareas de recepción de la información de posición suministrada por el GPS. Esta librería se llama GPS y evita al usuario tener que gestionar la recepción y tratamiento de las tramas NMEA enviadas por el módulo a través de la UART. Así para conocer la información GPS el usuario solo tiene que limitarse a consultar los distintos campos de la estructura de datos, la librería GPS se encarga del resto.

En caso de que, por la razón que sea, el usuario no desee utilizar la librería GPS, con no incluirla en el proyecto hay suficiente, luego sería él quien debería encargarse de implementar todas las rutinas para la recepción de información GPS procedente del módulo GPS a través de la UART. Otra opción, sería editar el código para adaptar esta a sus necesidades, teniendo en cuenta que modificar los puertos implicaría también modificar el conexionado entre le modulo GPS y el microcontrolador, sabiendo también que existen limitaciones en cuanto a las posibilidades de conexión entre estos, ya que en el caso del AVR-Atmega8515, este solo dispone de una única UART por lo que no existen muchas alternativas.

10.3.1.Funcionamiento interno de la librería GPS

La librería incluye diferentes funciones y subrutinas, pero la pieza clave de esta es la RSI de la UART, que es la encargada de recibir continuamente los bytes enviados por el módulo y colocarlos en el byte adecuado del campo adecuado de la estructura de datos. La RSI analiza las distintas cabeceras de los mensajes NMEA y a partir de estas sabe que tipos de datos están llegando. Tras identificar de que tipo de datos se trata lo coloca en el campo adecuado de la estructura de datos. Además esta RSI también se encarga de verificar el estado del módulo GPS actualizando también el flag que indica al usuario si la información almacenada en la estructura es válida o no.

La librería GPS también incluye la rutina de inicialización de la UART para trabajar con datos NMEA, la rutina de inicialización del módulo GPS para trabajar en modo NMEA, las rutinas de gestión del crono (que hacen uso de la temporización GPS) , y la rutina que permite conocer al usuario la validez de la información almacenada en la estructura.

10.3.2.Rutinas y estructuras de la librería firmware

Para utilizar las funciones o procedimientos, el usuario solo tiene que incluir la librería en su proyecto y llamar a sus funciones o procedimientos cuando los necesite. La librería se agrupa en los ficheros gps.c gps.h y consta de las siguientes funciones y procedimientos:

```
char paridad(char byte_recibido);
void GPS_resetea_crono();
void GPS_activa_crono();
void GPS_desactiva_crono();
char GPS_crono_activo();
char GPS_estado();
void GPS_consulta_datos(t_datos_GPS * * datos);
void GPS_gestiona_crono();
void GPS_UART_envia_cadena_con_imparidad(char * cadena,uint8_t
num_caracteres);
void GPS_UART_envia_cadena_sin_paridad(char * cadena,uint8_t
num_caracteres);
void GPS_UART_inicializa();
void GPS_UART_inicializa_modulo_NMEA();
```

Estas rutinas almacenan los datos recibidos del receptor GPS en la estructura que se muestra a continuación desde la cual el usuario puede conocer la información GPS más

importante. Pero para poder acceder a esta, el usuario antes ha de llamar a la función GPS_consulta_datos por tal de obtener un puntero o referencia a ella, ya que desde el main no se puede acceder directamente a las estructuras de datos de las librerías.

```
typedef struct{
	struct {
		char dia[2];
		char mes[2];
		char anio[4];
	} fecha;
	struct {
		char hora[2];
		char minutos[2];
		char segundos[2];
	} hora;
	struct {
		char activo
		char dias[2];
		char horas[2];
		char minutos[2];
		char segundos[2];
		char segundo_anterior
	} crono;
	char velocidad[5];
	struct{
		char grados[2];
		char segundos[7];
		char hemisferio;
	} latitud;
	struct{
		char grados[3];
		char segundos[7];
		char direccion;
	} longitud;
	char altitud[5];
	char num_satelites[2];

}t_datos_GPS;
```

Por defecto, la información de esta estructura se actualiza cada segundo, pero esto depende de la frecuencia con la que el receptor GPS suministre las tramas NMEA. Por defecto la frecuencia de estas tramas es de 1 segundo, pero esta se puede modificar en el proceso de configuración del modulo GPS, por lo que si se esta se modifica, también se estará modificando la frecuencia de actualización de la información en la estructura de datos GPS del microcontrolador.

10.4.Librería firmware de control del visualizador: LCD

El control del visualizador alfanumérico se realiza mediante la librería LCD, que se agrupa en los ficheros lcd.h y lcd.c. Esta contiene todas las rutinas necesarias para

controlar un visualizador basado en el integrado HD4780 de Hitachi configurado en el modo de 8 bits. Para que esta funcione correctamente las 8 lineas de datos del LCD deben estar conectadas a PORTB 0-7, la señal RS al bit 7 de PORTD, y la señal de enable al bit 6 de PORTD. Si se desea cambiar el conexionado del visualizador al microcontrolador, también habrá que modificar el código de la librería con los nuevos puertos utilizados.

Las rutinas de esta librería permiten realizar la mayor parte de las operaciones posibles sobre este tipo de visualizadores, operaciones como la inicialización, borrado de pantalla, escritura de texto ... Las rutinas disponibles son:

```
char LCD_consulta_buffer(int posicion); void
LCD_asigna_caracter_buffer(char caracter,int posicion);
void LCD_envia_comando();
void LCD_envia_dato();
void LCD_borra();
void LCD_borra_buffer();
void LCD_escribe_buffer_linea(char num_linea);
void LCD_asigna_cadena_buffer(char * cadena_origen);
void LCD_inicializa();
void LCD_escribe(char * cadena, char num_linea);
```

Por si se desease ampliar la librería, o modificar algunas de sus funciones, se ha incluido el anexo Anexo C donde se explica con detalle el funcionamiento exacto de este tipo de visualizadores.

10.5.Librería firmware de acceso a la tarjeta MultimediaCard: MMC

También se ha desarrollado una librería firmware que permite el acceso a la tarjeta MMC de forma sencilla a través de la SPI. De hecho esta librería solo se ha de incluir cuando se desee almacenar información en la tarjeta MMC, como es en el caso de la placa registradora de trayectorias. Incluye las rutinas básicas para leer y escribir un bloque de la tarjeta. Estas son:

```
uint8_t MMC_envia_byte(char valor);
void MMC_envia_comando( uint8_t      byte_0,uint8_t      byte_1,uint8_t
byte_2,uint8_t byte_3,uint8_t byte_4,uint8_t byte_5 );
void MMC_activa();
void MMC_desactiva();
uint8_t MMC_espera_respuesta ();
uint8_t MMC_espera_data_token( );
char MMC_prepara_tarjeta();
```

Tal como se comenta, el uso de esta librería es opcional, y el usuario puede omitirla o adaptarla en caso de que no se adecuase a sus necesidades, teniendo siempre en cuenta a que pins de que puertos están conectados los diferentes contactos de la tarjeta, ya que modificar los puertos implicaría modificar las conexiones. A parte, los pins escogidos para las señales DataIn y DataOut se deben corresponder con los pins MOSI y MISO de una interfaz SPI, y el AVR-Atmega 8515 sólo dispone de una única interfaz SPI, por lo que las alternativas son nulas. Es decir que aunque se quisiera, poca cosa se puede cambiar en lo que refiere al conexionado entre la tarjeta MMC y el microcontrolador, quizás únicamente las señal de activación y desactivación CS.

Capítulo 11
Software de transferencia de datos

11.1.Introducción

Es necesario disponer, en el ordenador personal, de una aplicación que permita transferir toda la información de la trayectoria almacenada en la memoria MMC del registrador de trayectorias al ordenador donde luego poderla utilizar en diferentes aplicaciones. Es decir que la aplicación de transferencia de datos ha de ser capaz de gestionar la comunicación entre la placa y el ordenador a través del puerto serie, y también de exportar la información recibida de la placa a otros formatos de archivo comprensibles por otras aplicaciones para poder así sacar mayor provecho a la información registrada.

A diferencia del software de programación de la placa de desarrollo que está programado en Visual Basic .NET, el software de transferencia de datos está desarrollado en Visual C# .NET también para Windows. De hecho este lenguaje ofrece unas prestaciones muy similares a las ofrecidas por Visual Basic, facilidad y agilidad de desarrollo combinado con un rendimiento bastante bueno, no comparable al de Visual C++ pero más que suficiente para el tipo de aplicación que se ha querido desarrollar. La elección de este, en lugar de Visual Basic, ha sido básicamente una decisión personal con el fin de profundizar y aprender más sobre este lenguaje, que como se puede intuir por su nombre, viene a ser una combinación de la facilidad y vistosidad de Visual Basic con la potencia del lenguaje C++.

Al igual que sucede con el software de programación, para que la aplicación funcione bien en cualquier ordenador personal, es necesario haber instalado antes la plataforma

.NET Framework de Microsoft la cual contiene todos los recursos necesarios para la ejecución de cualquier aplicación hecha en .NET. Debido a que Visual Studio.NET 2003 no incluye un control específico para la comunicación a través del puerto serie, es necesario instalar y registrar en el sistema el control NETComm de Richard Grier (consultar bibliografía). Una vez instalado el .NET Framework y el control NETComm, ya se puede ejecutar la aplicación directamente desde su carpeta. Las versiones posteriores de Visual Sutido.NET si incluyen un control para la comunicación por el puerto serie, así que si se desea se puede modificar el código fuente de la aplicación para hacer uso de este.

Los pasos exactos para instalar y poder utilizar la aplicación en un ordenador personal con Windows XP son los siguientes:

-Instalar el .NET Framework 1.1 (o superior) de Microsoft. Este se puede obtener a partir del disco Windows Component Update de Visual Studio, o descargarse de la web oficial de Microsoft.
-Extraer el contenido del fichero “NETComm.zip” y ejecutar la aplicación de instalación “Setup.exe”
-Reiniciar el ordenador tras la instalación del control NETComm.
-Tras realizar estos pasos la aplicación de transferencia de información entre la placa registrador y el ordenador debería estar lista para poder ejecutarse y utilizarse sin problemas.

11.2.Descripción de la aplicación

A nivel de diseño interno, la aplicación consta de una interfaz o formulario y de varios módulos donde se definen las clases de los objetos utilizados en esta.

Estos módulos son: “LowLevel” que contiene todas las rutinas necesarias para realizar las conversiones de tipos de datos y “Serial Port” que contiene las rutinas encargadas de configurar el puerto serie y gestionar la comunicación entre la UART del microcontrolador y el puerto serie del ordenador.

Al ejecutar la aplicación aparece el formulario principal con las opciones referentes a la configuración del puerto serie y dos botones que permiten descargar la información de la placa al ordenador, y exportar los datos descargados a formatos reconocibles por otras aplicaciones como Map Point, Ozi Explorer, Excel ...

Fig 11.2.1 Formulario principal de la aplicación de transferencia y exportación de datos.

Los parámetros del puerto serie se encuentran ya configurados para trabajar con el firmware creado para la placa, pero se pueden ajustar para no tener que modificar la aplicación en el caso de que se modificase la velocidad de transferencia en el firmware. Es importante recordar que la versión utilizada de Visual Studio .NET, la 2003, no dispone de un control del estilo de MSComm para la comunicación serie como el incluido en otras versiones de Visual Studio , y por ello se ha tenido que buscar por Internet algún sustituto de este (las versiones más modernas sí lo incluyen). En concreto se ha utilizado el control NETComm de Richard Grier el cual, tal como se comenta anteriormente, hay que instalar y registrar en el sistema antes de utilizar la aplicación, ya que en caso contrario esta causará una excepción.

El botón “Descargar” permite realizar la transferencia de datos de la placa al ordenador personal. Antes de iniciar la transferencia se muestra un dialogo donde se pregunta el nombre del archivo donde se desea almacenar la información de la tarjeta. Tras entrar el nombre de este en el formulario se muestra el mensaje “Preparado para recibir” indicando que el ordenador se encuentra esperando datos, en este momento ya se puede iniciar la transferencia desde la placa. Al finalizar la transferencia, si todo ha ido bien, se mostrará el mensaje “Los datos se han descargado con éxito”, y se habrán guardado en el archivo *.GPA anteriormente indicado.

Tal como su nombre indica, el botón “Exportar” permite convertir los datos almacenados en un archivo de descargas .GPA a otros formatos importables desde otras aplicaciones. Básicamente se pueden generar 3 tipos de ficheros:

Fig 11.2.2 Formulario principal de la aplicación de transferencia y exportación de datos.

-Ficheros de tabla: estos son ficheros de texto con la información GPS presentada en forma tabla. Mediante “Delimitador” se puede especificar que carácter delimitador de campo se quiere utilizar (‘,’ , ‘\’ , ‘;’) para delimitar las distintas columnas. Mediante “Campos” se puede seleccionar que información se desea almacenar en el fichero y cual no. Este tipo de ficheros se puede abrir prácticamente desde cualquier hoja de cálculo como Excel, o desde aplicaciones de cartografía digital del estilo de Map Point.

A continuación se muestra un ejemplo completo de este tipo de ficheros:

Id; Dia; Mes; Año; Horas; Minutos; Segundos;C dias;C horas;C minutos;C segundos; Velocidad; Latitud; Longitud ; Altitud ; Num satélites;
0;9;9;2004;22;35;27;0;0;0;0;0;41.5293316666667ºN;2.07819833333333ºN;163;6;
1;9;9;2004;22;35;29;0;0;0;0;0;41.5293333333333ºN;2.07819833333333ºN;163;6;

2;9;9;2004;22;35;31;0;0;0;0;0;41.5293333333333ºN;2.07820333333333ºN;163;7;
...
961;9;9;2004;23;7;58;0;0;0;0;0;41.5456466666667ºN;2.0864ºN;201;6;
962;9;9;2004;23;8;0;0;0;0;0;0;41.5456466666667ºN;2.08640166666667ºN;201;6;

No hace falta explicar el significado de cada campo, ya que tal como se aprecia en este ejemplo la primera línea contiene la descripción de la información que se almacena en los distintos campos de cada línea (descripción de la columna). Los campos que comienzan con C se corresponden al cronómetro.

-Ficheros de waypoint: son ficheros con información sobre localizaciones geográficas puntuales, es decir son ficheros que contienen un conjunto de puntos del mapa. De esta forma toda la información de localización almacenada en el archivo descargado de la placa se convierte en una secuencia de puntos que pueden ser visualizados desde cualquier aplicación que pueda leer este tipo de ficheros, como es el caso de OziExplorer.

Estos son archivos de texto. A continuación se muestra un ejemplo de este tipo de ficheros, y el significado de cada uno de sus campos:

Datum,WGS 84
WP,D,1,42.7293317,3.0781983,09/09/2004,22:35:27,,A,N,-9999
WP,D,2,42.7293333,3.0781983,09/09/2004,22:35:29,,A,N,-9999
WP,D,3,42.5293333,3.0782033,09/09/2004,22:35:31,,A,N,-9999
...
WP,D,962,42.5456467,3.0864,09/09/2004,23:07:58,,A,N,-9999
WP,D,963,42.5456467,3.0864017,09/09/2004,23:08:00,,A,N,-9999

Datum, WGS 84: cabecera con el Datum de proyección utilizado.
WP: la información se corresponde con un way point.
D: los datos de posición están expresados en grados (Degrees).
1: identificador del way point.
42.7293317 :la latitud en grados.
3.0781983: la longitud en grados.
09/09/2004: la fecha.
22:35:27.000: la hora.
,,A,N,-9999: ¿se desconoce la función de estos campos?

-Ficheros de track: son ficheros con información de recorridos o trayectorias, y son muy parecidos a los ficheros de waypoint, solo que en este caso los puntos se unen formando una línea. Este tipo de ficheros suele ser el utilizado para almacenar información referente a rutas o caminos y también pueden ser visualizados desde cualquier aplicación del estilo de OziExplorer.

También son archivos de texto. A continuación se muestra un ejemplo de este tipo de ficheros, junto con el significado de cada uno de sus campos:

Datum,WGS 84
TP,D,42.7293317,3.0781983,09/09/2004,22:35:27.000,0,163
TP,D,42.7293333,3.0781983,09/09/2004,22:35:29.000,0,163,
TP,D,42.7293333,3.0782033,09/09/2004,22:35:31.000,0,163,
...
TP,D,42.7456467,3.0864,09/09/2004,23:07:58.000,0,201,
TP,D,42.7456467,3.0864017,09/09/2004,23:08:00.000,0,201,
Datum,WGS 84: cabecera con el Datum de proyección utilizado.

Datum, WGS 84: cabecera con el Datum de proyección utilizado.
TP: la información se corresponde con un track point
D: los datos de posición están expresados en grados (Degrees).
42.7293317 :la latitud en grados.
3.0781983: la longitud en grados.
09/09/2004: la fecha.
22:35:27.000: la hora.
0: ¿se desconoce la función de este campo?
163: la altitud en metros

Capítulo 12
Fírmware de la placa de registro

12.1.Introducción

El firmware del registrador de trayectorias hace uso de las principales prestaciones de la placa de desarrollo con el fin de demostrar las posibilidades de esta. En concreto, este ha de recoger toda la información de posición suministrada por el módulo GPS, registrarla en la tarjeta de memoria MultiMediaCard, y enivarla más tarde a un ordenador personal a través del puerto serie. También ha de implementar una interfaz de usuario lo más simple y eficaz posible para facilitar su utilización.

Este se ha programado en C, en el entorno de desarrollo gratuito WinAVR

12.2.Estructura

Antes de comenzar a desarrollar el firmware de la placa registradora de trayectorias, se han identificado las tareas a más bajo nivel que este ha de realizar:

- Comunicación con el módulo receptor GPS
- Control de la tarjeta MultiMediaCard
- Interfaz de usuario
- Transferencia de datos.

El siguiente paso ha sido decidir como se han de implementar estas tareas, y la respuesta ha sido fácil: mediante las librerías firmware previamente diseñadas. De la comunicación con el módulo receptor GPS se encarga la librería GPS, del control de la tarjeta MultiMediaCard la librería MMC y de la interfaz de usuario la librería LCD. La interfaz de usuario también hace uso sencillas rutinas añadidas a la librería Tiempo para

leer los tres pulsadores. Inicialmente se tenía pensado utilizar la librería LPTCOM para la transferencia de datos entre el ordenador personal y la placa pero debido a problemas de lentitud de la DLL utilizada en el acceso al puerto paralelo, se ha decidido utilizar el canal serie, ya que a efectos prácticos ofrece una velocidad de transferencia mayor que la obtenida mediante la DLL a través el puerto paralelo.

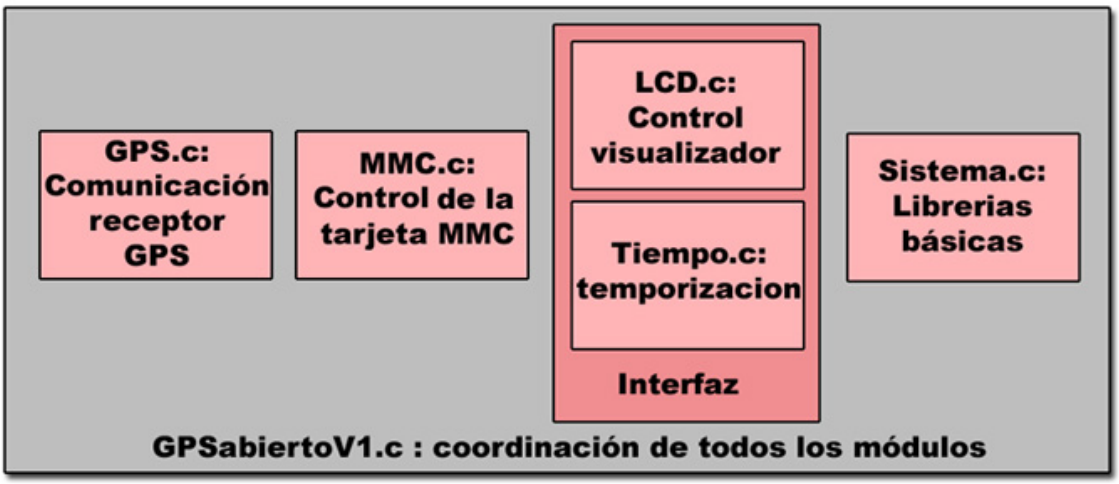


Fig 12.2.1 Diagrama con los diferentes módulos con las tareas que estos realizan.

Así las librerías o módulos utilizados son los siguientes (fig 12.2.1) :

-GPS (GPS.h GPS.c): se encarga de gestionar la comunicación entre el microcontrolador y el módulo GPS, y también de tratar adecuadamente las diferentes tramas de información enviadas por este último. Por tanto es el encargado de realizar todo el tratamiento de las tramas NMEA y de obtener de estas la información de posición GPS. Tras ver la ineficacia de la DLL de acceso al puerto paralelo, se ha considerado que la opción más eficiente para la transferencia de datos entre la placa y el ordenador es el puerto serie, y este está bajo el control de este módulo, se han tenido que añadir aquí las rutinas de transferencia de datos de la placa al ordenador personal.

-MMC (MMC.h MMC.c) : implementa todas las funciones básicas de acceso a la tarjeta MultiMedia Card por tal de almacenar en esta la información de la trayectoria, es decir que contiene el conjunto de rutinas necesarias para guardar datos en la MMC. Así las funciones de MMC permiten guardar en memoria no volátil la información enviada por el modulo GPS.

-LCD (LCD.h LCD.c) : contiene las rutinas necesarias para poder controlar de forma sencilla el LCD HD44780 utilizado para mostrar las diferentes opciones del menú y el estado de las variables GPS. Las funciones de este módulo permiten al firmware mostrar la información de posición, hora , velocidad ... por el LCD mediante parámetros muy sencillos sin necesidad de controlar los detalles de bajo nivel.

-Tiempo (Tiempo.h Tiempo.c) : es la librería donde se encuentran todas las funciones básicas asociadas a temporizaciones del Timer. En concreto, contiene funciones para el rastreo de los pulsadores evitando los rebotes, funciones para el control de frecuencia de actualización del display, y funciones de temporización para la realización de esperas activas. Hay que recordar que la hora ofrecida al usuario no se gestiona mediante ningún Timer, sino que es la hora GMT+0 suministrada por el propio módulo GPS.

-GPSAbierto (GPSabiertov1.c): es el módulo principal del firmware desde el que se coordina la comunicación entre los demás módulos. Contiene el main de la aplicación y también algunas rutinas encargadas de dar formato a los datos que se van a almacenar en la MMC. El main de la aplicación contiene las rutinas de inicialización de los distintos periféricos seguidas de un bucle infinito desde el que se va llamando repetidamente a los diferentes módulos por tal de que estos vayan realizando sus tareas.

12.3.Flujo de ejecución

El esquema 12.3.1 muestra como determinadas tareas del firmware se realizan por polling mientras que otras se realizan por interrupciones. Las tareas menos críticas como la captura de las pulsaciones, actualización del visualizador o actualización del cronómetro se realizan por polling en el bucle principal del firmware. De hecho el cronómetro no hace uso del Timer sino de la información de hora suministrada por el módulo GPS. Es por eso que la actualización de este se realiza mediante polling, ya que se va comprobando repetidamente sobre la hora GPS cuantos segundos han pasado tras la última consulta.

La consulta de los pulsadores y refresco del visualizador también se realiza por polling. En el caso de los pulsadores se hace uso de una variable que indica el tiempo que hay

que dejar pasar entre consulta y consulta de estos, así por polling se mira continuamente si ha pasado ya este tiempo, de forma que si ha pasado, se lee el estado estos. Algo parecido sucede con el visualizador. Este también hace uso de una variable que indica el tiempo que tiene que haber entre actualización y actualización del visualizador, algo así como la frecuencia de refresco de este. Continuamente se va consultando si ha pasado el tiempo suficiente tras la última actualización y si es así se actualiza el contenido del display.

Al igual que la gestión del Timer, la recepción de las tramas del módulo GPS se realiza mediante interrupciones por tal de garantizar que no se pierde ninguna trama en ningún momento. Así cada vez que se recibe un carácter por la UART se comprueba a que tipo de información corresponde y se coloca en la estructura adecuada para que esté disponible para las demás rutinas del firmware que la necesiten.

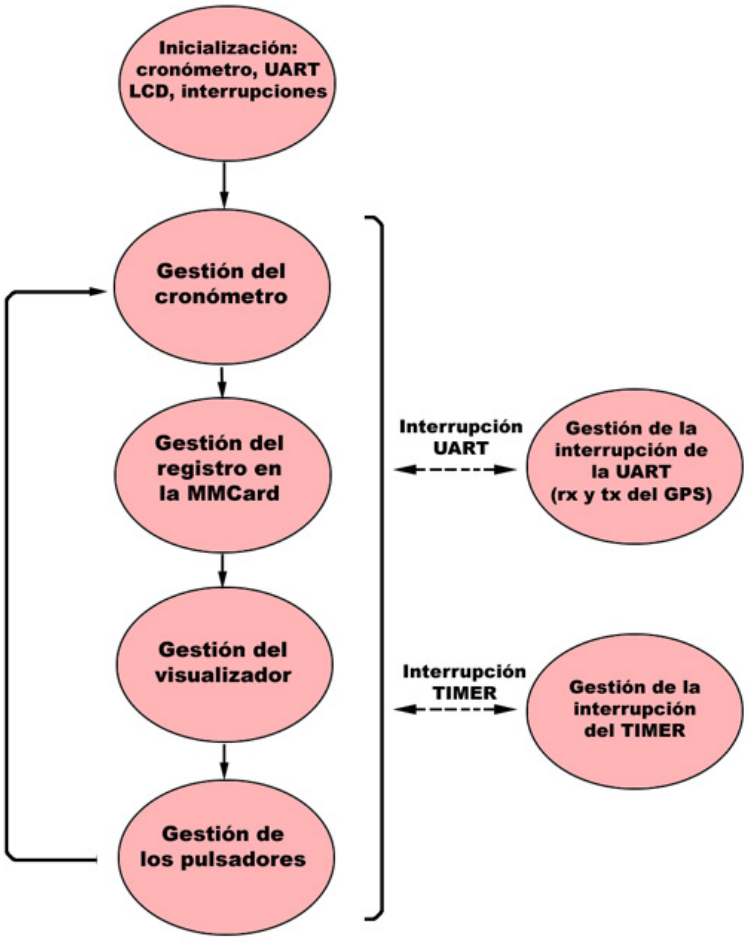


Fig 12.3.1 Esquema general del flujo de ejecución del firmware

12.4.Descripción de la interfaz de usuario

Tal como se comenta en los apartados anteriores, el principal objetivo de la interfaz de usuario es hacer lo más eficiente y sencilla posible la utilización de la placa registradora de trayectorias al usuario. Esta hace uso del visualizador LCD alfanumérico y de tres pulsadores. De los tres pulsadores, el de la izquierda es para retroceder de opción en el menú, el de la derecha es para avanzar y el del medio es para activar o desactivar una opción. En la línea superior del visualizador se muestra en todo momento la opción del menú en que se encuentra el usuario y en la inferior la información asociada a esta opción, de forma que si por ejemplo el usuario se encuentra en la opción del cronómetro en la primera línea del visualizador aparece “CRONOMETRO:” y en la segunda “00:00:00:00”.

La siguiente tabla muestra las distintas funciones que la placa de registro de trayectorias ofrece al usuario:

#	Contenido del LCD:	Descripción de la opción:
1	FECHA: 22-10-2004	Muestra la fecha actual
2	HORA: 17:22:45	Muestra la hora GMT
3	CRONOMETRO: 00:00:01:12	Muestra el estado del cronómetro y permite activarlo o desactivarlo
4	VELOCIDAD: 060.2 Km/h	Muestra la velocidad
5	LONGITUD: 002º4,72020’-E	Muestra la longitud
7	LATITUD: 41º32.747274’-N	Muestra la latitud
7	ALTITUD: 0120 m	Muestra la altitud
8	SATELITES: 09 activos	Muestra la cobertura de satélites del receptor GPS
9	REGISTRAR: Iniciar registro	Permite iniciar o detener el registro de datos en la MMC
10	ENVIAR AL PC: Iniciar envio	Permite enviar al ordenador los datos registrados
11	INI. LASSEN SQ: Inicializar	Ejecuta la secuencia de inicialización del módulo GPS

Fig 12.4.1 Funciones de la placa de registro de trayectorias

Capítulo 13
Evaluación de los resultados

13.1.Introducción

Es hora de mostrar las conclusiones extraídas del desarrollo de este proyecto como también de mostrar algunas ideas de futuro que permitirían mejorar o ampliar algunas de sus prestaciones. También hay que hacer un balance del coste del desarrollo de este proyecto y de la viablidad de su producción en serie.

Lo primero a destacar es que finalmente se ha conseguido construir una senzilla pero completa plataforma de desarrollo de aplicaciones GPS formada por la placa de desarrollo, el software de telecarga, las librerías firmware de soporte, y el entorno de programación público WinAVR. Prescindiendo del modulo GPS y de las librerias firmware, el sistema también se puede utilizar como plataforma de desarrollo de otras aplicaciones basadas en microcontroladores AVR-Atmega ya que incluye el hardware de programación, software de telecarga y entorno de programación necesarios para ello.

Haciendo uso de la plataforma de desarrollo, y para demostrar su funcionalidad, se ha creado lo que en un principio iba a ser el objetivo principal del proyecto: la placa registradora de trayectorias GPS. Los resultados han sido muy positivos, y salvo algunos problemas con la tarjeta MultMediaCard y con el compliador C de WinAVR, la realización de esta ha sido un tarea prácticamente inmediata gracias a las facilidades y recursos ofrecidos por la plataforma creada. Por tanto queda demostrada la viabilidad y utilidad de la plataforma de desarrollo, habiéndose alcanzado los objetivos planteados al inicio del proyecto.

13.2.Conclusiones y líneas de futuro

Que mejor manera de evaluar los objetivos alcanzados, que comparando los resultados obtenidos con los objetivos iniciales:

13.2.1.Placa de desarrollo

"...Crear un dispositivo hardware autónomo, minimizando el consumo, tamaño, y coste del conjunto...."

Queda claro que se ha obtenido un dispositivo hardware completamente autónomo, de dimensiones relativamente pequeñas, económico y de consumo relativamente bajo. En cuanto al tamaño se ha de añadir que el sistema podría haber sido más pequeño si se hubiese dispuesto de las herramientas necesarias para trabajar con encapsulados SMD y circuitos impresos. Además, el escaso número de componentes utilizados en el diseño y su reducido tamaño demuestra que con los medios adecuados se podría obtener una palca muy pequeña. Respecto al tamaño, sería interesante sustituir los conectores DB-25 y DB-9 por otros más pequeños puesto que estos son relativamente grandes y ocupan bastante superficie.

En lo referente al consumo, se ha intentado utilizar componentes de 3.3V para minimizar la potencia consumida por el conjunto, no obstante el sistema flaquea un poco en el punto clave del circuito de alimentación: el regulador. Se ha utilizado un regulador lineal, los cuales son muy económicos, compactos, estables y sencillos en su uso, pero su rendimiento no es muy bueno en comparación con el de los reguladores conmutados. De todas formas se puede afirmar que a pesar que el regulador utilizado no tiene un rendimiento muy bueno, es una opción sencilla, robusta y que cumple bien con su finalidad. El consumo del sistema es de 100 ma con antena GPS incorporada.

En lo referente al coste económico, se ha intentado utilizar los componentes que mejor satisfacieran la relación prestaciones/precio y dentro de las posibilidades de un usuario convencional se ha conseguido, no obstante es evidente que pudiendo trabajar en otro contexto como el de la industria, seguro que el coste de producción del sistema sería muy inferior al de la elaboración de este proyecto.

“ ... Ofrecer la posibilidad de programación directa “on board”, sin necesidad de tener que extraer ni insertar ningún componente durante el proceso de programación. Esto ha obligado a crear el software de telecarga adecuado ...”

Este objetivo se ha alcanzado plenamente, y la plataforma se puede programar rápidamente con un simple “click” de ratón. Esto permite comprobar el correcto funcionamiento del programa de forma inmediata desde el mismo ordenador en que se está programando la aplicación, sin perder tiempo montando y desmontando chips. A parte de las opciones de programación se han añadido otras funcionalidades en placa como por ejemplo las de verificación de programación ...

De cara a un futuro sería interesante ofrecer la posibilidad de realizar el depurado de la aplicación sobre el propio chip del microcontrolador, aunque para ello probablemente se tendría que cambiar el modo de programación del microcontrolador y utilizar algún tipo de bootloader.

Otro punto interesante sería poder utilizar algún mecanismo de programación más rápido puesto que cuando el código a descargar es grande el proceso de programación puede llegar a durar 2 minutos, lo que hace el proceso de pruebas a veces sea un poco pesado.

“...El hardware ha de incorporar todos los componentes necesarios para la recepción y proceso de la información de posición GPS...”

Tal como se demuestra en el apartado dedicado al módulo receptor GPS, el módulo seleccionado dota a la plataforma de desarrollo de todas las funciones básicas de un sistema GPS por tal de permitir al usuario elaborar casi cualquier aplicación de posicionamiento global.

“...Ofrecer las librerías necesarias para simplificar al usuario el acceso a la información GPS en el proceso de desarrollo...”

Se han desarrollado distintas librerías firmware con el propósito de simplificar al usuario la creación de aplicaciones sobre la plataforma de desarrollo, y tal como se ha podido ver en la realización de la placa registradora de trayectorias estas cumplen perfectamente su cometido, facilitando y acelerando enormemente la creación de ese proyecto. Estas librerías funcionan correctamente por lo que se puede dar este objetivo por cumplido. No obstante presentan algunas limitaciones ya que trabajan con la representación ASCII de los valores, y no con su valor real, por lo que es necesario utilizar algún tipo de rutina del estilo ASCII_to_FLOAT para convertir estas cadenas a números y poder realizar operaciones con ellas. Estas rutinas están disponibles para la librería pero no se han incluido debido a que incrementan enormemente el binario de la aplicación reduciendo mucho el espacio de memoria de programa para el resto de la aplicación. De todas formas si se dispusiera de un modelo de microcontrolador superior de la misma familia AVR-ATmega estas rutinas se deberían poder incluir y utilizar sin problemas.

"... Simplificar al usuario , dentro de lo posible, la utilización de la plataforma..."
Es evidente que el sistema creado está dirigido a desarrolladores y a usuarios con conocimientos avanzados para los que no tendría que suponer ningún problema la utilización de la plataforma de desarrollo. De todas formas, se han simplificado diferentes aspectos por tal de facilitar su uso. Respecto a la interfaz se ha intentado hacer esta lo más sencilla posible. De hecho, para programar la placa no hay que cambiar de posición ningún interruptor, basta con conectarla al ordenador a través del puerto paralelo, arrancar la aplicación de telecarga y hacer "click" sobre el botón de programación. Además la aplicación de telecarga dispone de una interfaz de usuario típica de aplicación de Windows, muy intuitiva y fácil de usar.

En lo relacionado con el desarrollo y programación, se han creado el conjunto de librerías firmware, incluyendo también un ejemplo en que se muestra su utilización: el registrador de trayectorias.

Otro aspecto básico pero importante es que se ha procurado comentar al máximo todo el código de las librerías para que la comprensión y modificación de estas resulte lo más sencilla posible.

A pesar de considerar que se ha alcanzado este objetivo, esto no se puede saber con certeza hasta que algún otro usuario, no relacionado con el desarrollo de este proyecto, la utilice y dé su valoración de la sencillez de su uso.

13.2.2.Placa de registro de trayectorias

"...Funcionar en base a la plataforma de desarrollo GPS creada..."
Esta claro que este objetivo se ha alcanzado, ya que todas las tareas de la placa registradora de trayectorias relacionadas con al información GPS se realizan sobre la placa de desarrollo y las librerías suministradas con esta. A parte ha sido necesario añadir una placa de periféricos a través del bus de expansión por tal de poder disponer de servicios no incluidos en la plataforma como son el display LCD o el zócalo para la tarjeta MultiMediaCard

"...Contar con un dispositivo de memoria eficiente sobre el que almacenar la información de posición..."
Este requerimiento se ha cumplido muy satisfactoriamente, gracias a la elección de memorias Flash MultiMediaCard : son extraíbles, económicas, existen modelos de distintas capacidades ... Así, cambiando solo la tarjeta se puede variar la capacidad de almacenamiento de la placa registradora de trayectorias. Incluso se puede hacer uso de distintas tarjetas en caso de que sea necesario.

El único problema que se ha presentado en este punto ha sido la imposibilidad de aprovechar de forma completa cada uno de los bloques de la tarjeta de memoria, ya que de cada bloque ocupado solo se utilizan 128 de los 512 bytes diponibles desperdiciando así bastante espacio, es decir que únicamente se utiliza una cuarta parte del espacio de tarjeta disponible. Esto es debido a que el microcontrolador utilizado no dispone de suficiente RAM para operar con los bloques completos, por lo que si se sustituyera el microcontrolador utilizado por otro modelo superior de la misma familia este problema desaparecería.

"...Disponer de una interfaz de entrada/salida sencilla para permitir la interacción con el usuario..."

El visualizador alfanumérico y los 3 pulsadores permiten al usuario controlar el sistema de una forma muy sencilla y cómoda parecida a la implementada en muchos teléfonos móviles, en los que con apenas 3 teclas se puede manejar todas sus opciones. La combinación de los pulsadores y el visualizador alfanumérico con un sistema de menús bastante simple e intuitivo facilita y agiliza enormemente la utilización del sistema. Por tanto este objetivo se da también por cumplido. No obstante en un futuro sería interesante dotar al sistema con un LCD gráfico para poder mostrar por este la información de posición combinada con información de tipo cartográfico.

"..Poder conectarse a un ordenador personal para transferir los datos registrados..."
La placa registradora de trayectorias puede conectarse a un ordenador personal para transferir todos los datos almacenados en la tarjeta MultiMediaCard, así que el objetivo se ha alcanzado. No obstante la intención inicial era la de utilizar el puerto paralelo debido al mayor ancho de banda ofrecido por este, pero a causa de problemas de velocidad de la DLL utilizada para acceder a este puerto, se ha tenido que desestimar esta opción y utilizar el puerto serie. La combinación de las rutinas de comunicación serie del microcontrolador con la aplicación cliente del ordenador personal, permiten la transferencia de datos de la placa al ordenador de forma satisfactoria, pero ni mucho menos óptima.

Otro punto interesante sería hacer el firmware compatible con el sistema de ficheros FAT32 para poder leer la información almacenada en la tarjeta directamente mediante un lector de tarjetas en el ordenador.

13.3.Estudio económico

En la siguiente tabla se muestra el coste de los distintos componentes electrónicos utilizados para la elaboración del hardware proyecto, lo que da una idea aproximada del dinero invertido en la realización del este, y también de lo que aproximadamente costaría fabricar otra réplica. La tabla es válida solo cuando se habla de pocas unidades (por pocas se entiende del orden de 10).

El coste se estructura en dos tablas, la primera con los componentes de la placa de desarrollo, y la segunda con los componentes de la placa del registrador de trayectorias:

Placa de desarrollo		
Elemento	**Coste (Euros)**	**%**
Integrados:		
Microcontrolador AVR-AT85151 + Zocalo DIP	5..5	4.7
Módulo receptor GPS Lassen SQ	53.5	45.7
Latguillo conversor Hirose a MCX	6.42	5.53
Antena externa GPS miniwat 3.5v + cable 5m	21.40	18.29
Antena embedded 27dB 80mm cable	14.90	12.73
Max232 + Zócalo DIP	0.75	0.64
Regulador 317	0.25	0.21
Xtal 4mhz	0.45	0.38
Componentes discretos:		
Resistencias	0..3	0.25
Condensadores	1	0.85
Diodos rectificadores	0.02	0.017
Diodos zener 3v3 1W	0.05	0.042
Leds verde	0.08	0.068
Zócalos y Conectores:		
Conector Samtec	1.98	1.69
Weismuller	0.7	0.59
DB9	0.4	0.34
DB25	0.7	0.59
BUS 40	0.91	0.77
Pulsador 6x6 43 mm	0.126	0.10
Otros:		
Portapilas	0.40	0.34
Cable y Estaño	4	3.41
Placa Baquelita Topos	4.04	3.45
TOTAL:	**117.876 Euros**	**100%**

Placa registradora de trayectorias		
Elemento	**Coste (Euros)**	**%**
Integrados (zócalos incluidos):		
LCD compatible HD44708 2x16	9	20.93
MultiMediaCard 64mb	21.5	50
Componentes discretos:		
Resistencias (y potenciómetros)	0.35	0.81
Condensadores	0.3	0.7
Diodos	0.8	1.86
Zócalos y conectores:		
Conector MMC	2.87	6.67
Pulsadores	0.40	0.93
Otros:		
Cable y Estaño	4	9.30
Placa	4.04	9.4
TOTAL:	**43.26 Euros**	**100%**

Si se decidiese realizar una fabricación en serie de esta, el coste de muchas de las partes de este se reduciría considerablemente, lo que también se reflejaía en el precio final del conjunto.

13.4.Dedicación

Es difícil hacer una estimación precisa del tiempo dedicado a este proyecto. Este se inició a principios de enero del año 2004 y se encontró prácticamente listo en septiembre del mismo año, no obstante el tiempo diario dedicado varió en función de la época del año. De enero a junio, época de clases, se invirtió una media de 14 horas semanales, mientras que de julio a septiembre se dedicaron una media de 56 horas semanales. A partir de septiembre y hasta febrero de 2005 la dedicación a este disminuyó a unas 3 horas semanales, por diversas circunstancias. Así el total de horas dedicadas puede ser del orden de unas 1100 horas. De estas 1100 horas unas 165 (15%) se ha dedicado a temas relacionados con el análisis y diseño, unas 715 (65%) a la implementación, y unas 55 (5%) a las pruebas. El tiempo restante, unas 165 horas (15%) , se ha dedicado a la documentación . La tabla siguiente muestra de forma resumida el tiempo dedicado a las distintas tareas:

Tarea:	Tiempo:	Porcentaje:
Diseño y análisis	165	15%
Implementación	715	65%
Pruebas	55	5%
Documentación	165	15%
TOTAL:	1100 horas	100%

Bibliografía

Millman, J. Grabel, A. (1995) *Microeletronica*, Editorial Hispano Europea

Tavernier, C (2004) *Guía del comprador de microcontroladores*, Elektor (edicion española) num 294-noviembre 2004, pag 60-63

¿?, (2004) *ATmega8515(L) Complete*, Atmel
http://www.atmel.com/dyn/resources/prod_documents/doc2512.pdf

¿?, (2004) *Atmega64(L) Complete*, Atmel
http://www.atmel.com/dyn/resources/prod_documents/doc2490.pdf

¿?, (2004) *Max232 Complete Datasheet*, Maxim -Dallas
http://pdfserv.maxim-ic.com/en/ds/MAX220-MAX249.pdf

¿? (2004) Lassen SQ GPS Reciver System Designer Reference Manual, Trimble
http://trl.trimble.com/docushare/dsweb/Get/Document-22183/

¿? (2004) *Intel Hexadecimal File Format Specification* , Intel
www.intel.com/design/intarch/papers/esc_file.pdf

¿? (2004) *MultiMediaCard Product Manual* , SanDisk
www.sandisk.com/pdf/oem/AppNoteMMCQAv1.0.pdf

¿? (2004) Informacion Flash y EEPROM , Intel:
http://www.intel.com/design/Flash/articles/what.htm

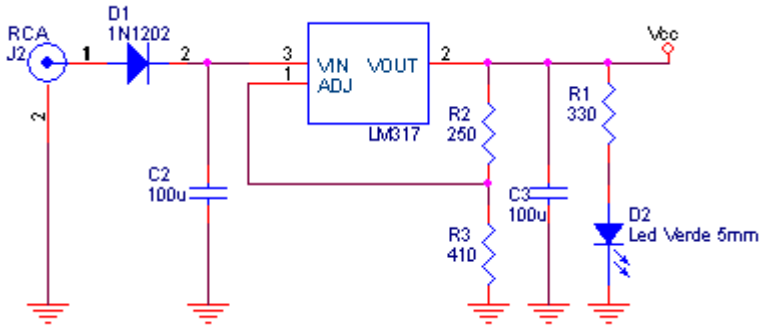
Grier, R. (2004) *Com.NET control*
http://www.hardandsoftware.net/

¿? (2004) *Microsoft .NET FrameworK*, Microsoft
http://msdn.microsoft.com/netframework/

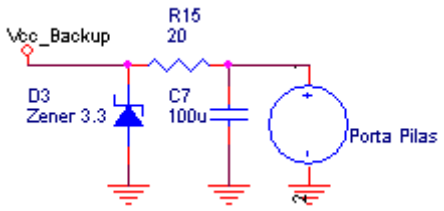
Apéndice A

Esquemáticos

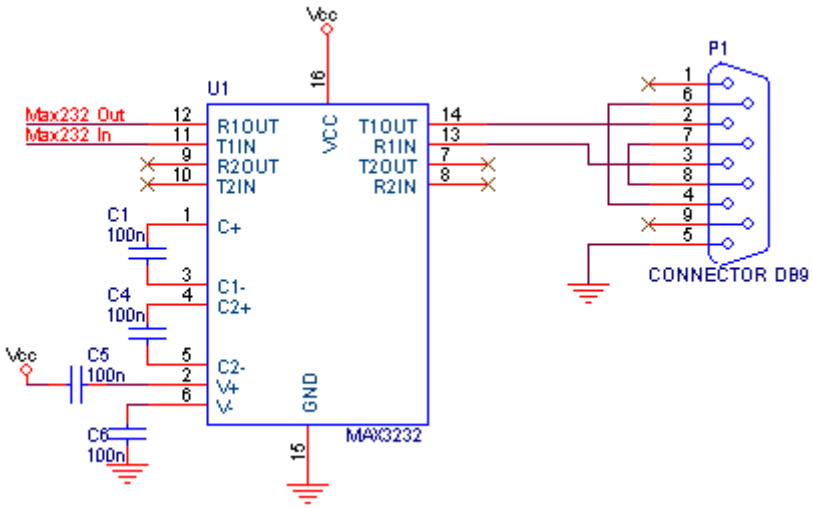
Placa de desarrollo:



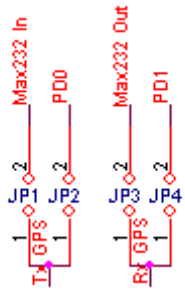
Circuito de alimentación 3V



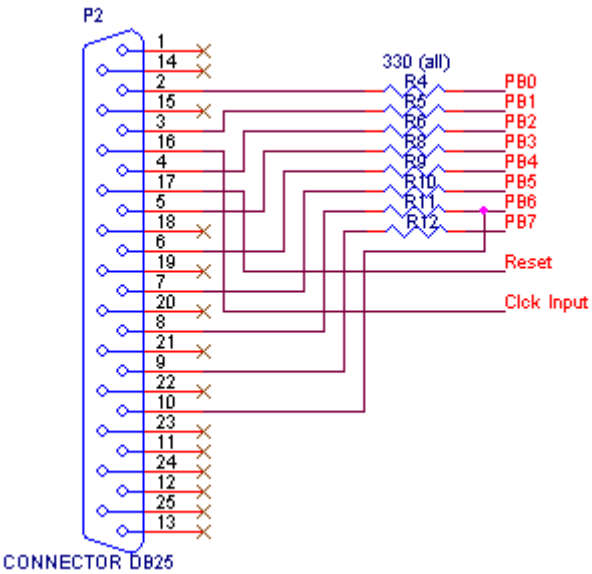
Battery backup del módulo GPS



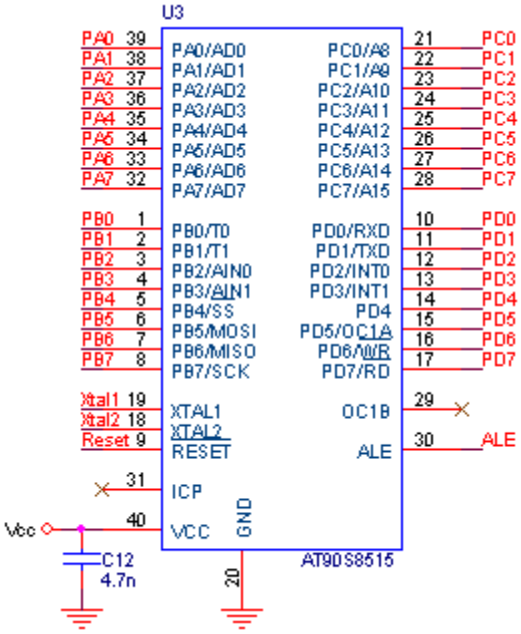
Circuito de conversión RS232 - TTL



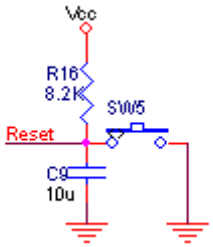
Jumpers de configuración



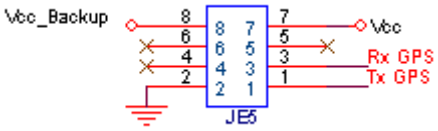
Conector de programación



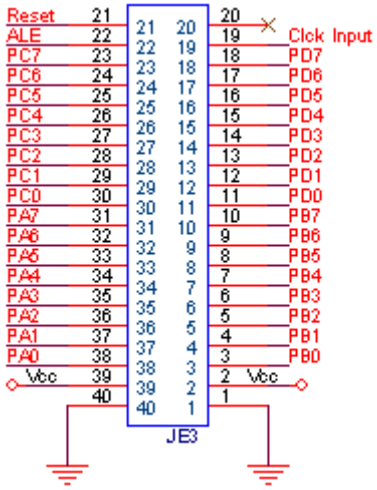
Microcontrolador Atmega-AVR8515L



Pulsador de reset

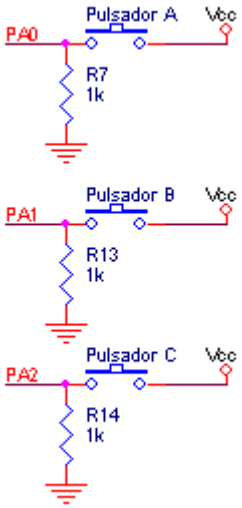


Conector al GPS

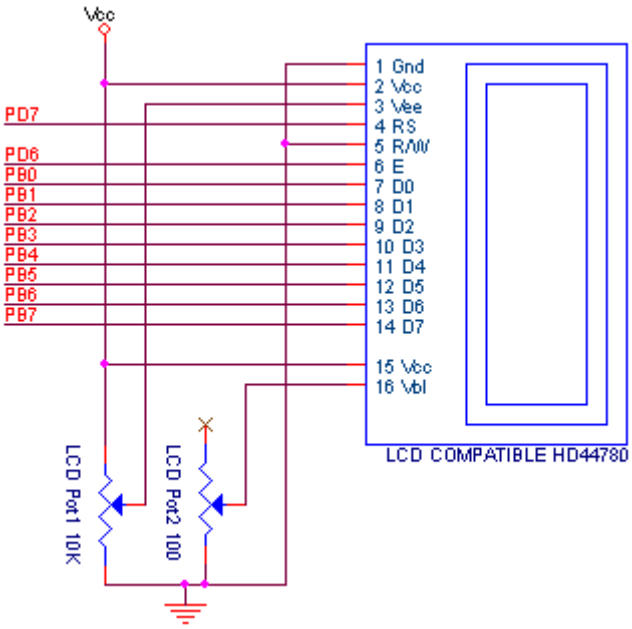


Conector de expansión a la placa de desarrollo

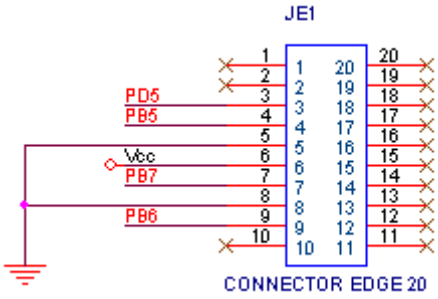
Placa registradora de trayectorias:



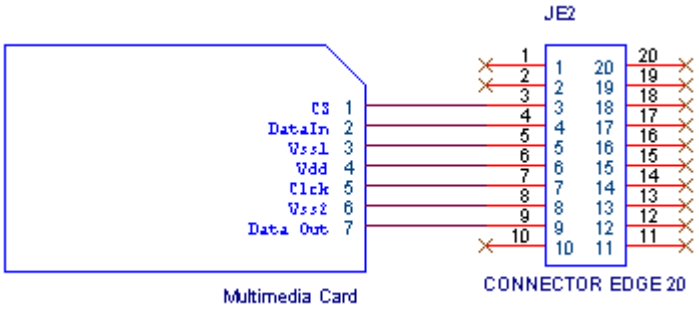
Interfaz de entrada: pulsadores



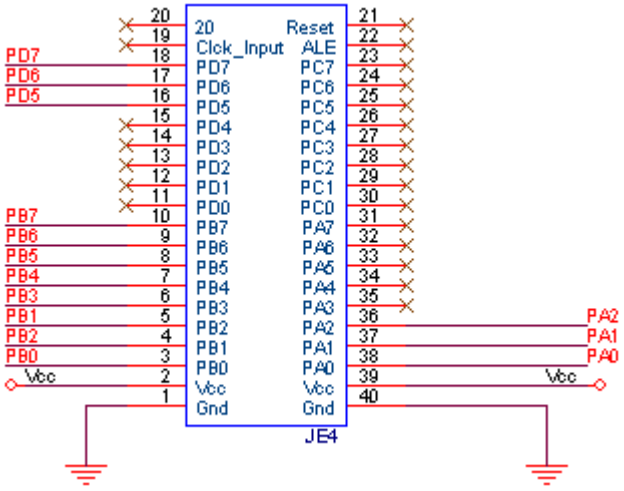
Interfaz de salida: display LCD



Conector hacia la tarjeta multimedia card



Conector de la tarjeta multimedia card



Conector de expansión de la placa del registradora de trayectorias

Apéndice B

Características principales del microcontrolador Atmega-AVR85151

Introducción

Se trata de un microcontrolador de 8 bits muy versátil, que ofrece una muy buena relación prestaciones-precio. A continuación se listan las prestaciones de la CPU, las memorias, y los perifericos.

CPU y memorias:
-Procesador RISC de 8 bits
-Juego de 118 instrucciones todas de 1 ciclo de instrucción y de un ciclo de reloj.
-Un ciclo de instrucción requiere un ciclo de reloj, por tanto este ofrece un throughput de un MIP por Mhz.
-Memoria de programa de 4096 palabras, cada palabra de 16 bits.
-32 registros de propósito general.
-64 registros de E/S para la gestión de los periféricos.
-512 bytes de RAM
-512 bytes de EEPROM
-Las interrupciones se gestionan mediante una tabla de vectores de interrupción de prioridad fija.

Periféricos:
-Temporizador/Contador de 8 bits con preescaler independiente.

-Temporizador/Contador de 16 bits con preescaler independiente, también con PWM.

-Comparador analógico.

-Temporizador Watchdog con oscilador propio.

-UART programable.

-Interfaz SPI.

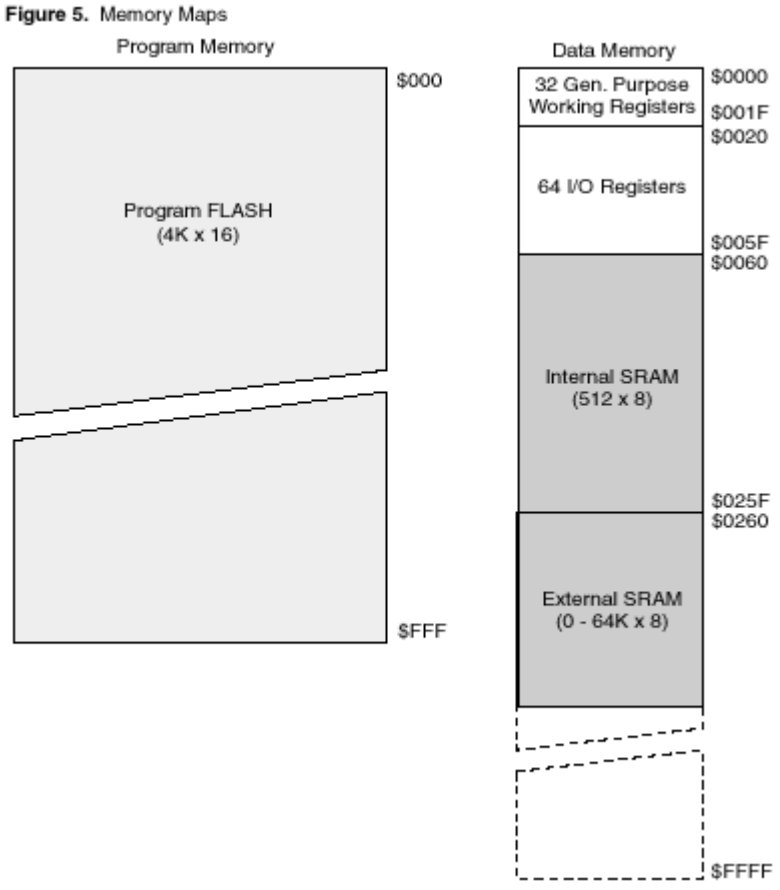
-32 pins E/S: RA0-RA7, RB0-RB7, RC0-RC7, RD0-RD7.

Organización de la memoria

El microcontrolador AVR8515 dispone de dos bloques de memoria diferenciados:

-Bloque de memoria de programa

-Bloque de memoria para datos



-Bloque de memoria de programa:

Esta es la memoria Flash de 8 kbytes, la función de la cual es contener el código que se

va a ejecutar en el microcontrolador. Como máximo puede contener 4096 instrucciones.

-Bloque de memoria para datos:

Este bloque agrupa los siguientes elementos: 32 registros de propósito general, 64 registros de dispositivos E/S, SRAM interna, SRAM externa opcional, y también la memoria EEPROM. Todos estos elementos, menos la memoria EEPROM, a pesar de estar implementados en memorias separadas, están mapeados linealmente.

Memoria de programa

Esta memoria tiene una capacidad de 8kbytes, que se organizan en 4096 posiciones de 16 bits, en cada una de las cuales se almacena una instrucción, a excepción de algunos casos en que la instrucción requiere dos posiciones. Por tanto el microcontrolador permite programas de como mucho 4096 instrucciones. Esto hace que el contador de programa PC sea de 12 bits, los suficientes para direccionar todo el espacio de la memoria de programa.

Registros de propósito general

El conjunto de registros de propósito general lo forman 32 registros de 8 bits, mapeados en la memoria de datos, de la dirección $0000 R0 hasta la $001F R31. La mayoría de instrucciones, excepto algunas como SBCI, SUBI, CPI ..., se pueden utilizar sobre estos registros y se ejecutan en un solo ciclo.

Además, los registros del 26 al 32 pueden utilizarse para realizar direccionamiento indirecto, y disponen de opciones especiales para ello: autoincremento, autodecremento, o desplazamiento fijo. Cuando se usan con este fin se agrupan de la siguiente manera:

Registro X - R27 R26

Registro Y - R29 R28

Registro Z - R31 R30

Registros de E/S

Estos 64 registros, mapeados en la memoria de datos de la dirección $0020 a la dirección $005F, permiten controlar los dispositivos de E/S del microcontrolador. Por tanto los accesos a este rango de direcciones mediante las correspondientes instrucciones de lectura o escritura permitirán escribir o leer sobre los periféricos y otros registros especiales . Estos registros se pueden utilizar como si no estuvieran mapeados en memoria, esto es cuando se usan las instrucciones IN y OUT para su escritura o lectura: en este caso el primer registro es el $00 en lugar del $0020, mientras que el ultimo es el $3F en lugar del $005F. Los 32 primeros registros pueden ser accedidos a nivel de bit mediante las instrucciones SBI, CBI, SBIS ...

De estos 64 registros los mas importantes son:

- STATUS REGISTER SREG: esta mapeado en la dirección $005F de memoria y almacena información sobre aspectos fundamentales del microcontrolador: contiene el estado general de las interrupciones y el estado aritmético de la ALU (flags de carry, overflow...). Al entrar en una RSI este registro no se guarda en ninguna parte, por lo que es una tarea para el programador almacenar el STATUS REGISTER en caso de que sea necesario.

Status Register – SREG

Bit	7	6	5	4	3	2	1	0	
$3F ($5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**
- **Bit 6 – T: Bit Copy Storage**
- **Bit 5 – H: Half-carry Flag**
- **Bit 4 – S: Sign Bit, S = N ⊕ V**
- **Bit 3 – V: Two's Complement Overflow Flag**
- **Bit 2 – N: Negative Flag**
- **Bit 1 – Z: Zero Flag**
- **Bit 0 – C: Carry Flag**

- STACK POINTER SP: es el puntero a la pila y en realidad lo forman dos registros de 8 bits, mapeados en las direcciones consecutivas $005E y $005D. Por tanto el SP es de 16 bits, lo que permite direccionar hasta 64kbytes. La pila crece hacia abajo, por lo que cuando se guarden valores en está, el puntero SP se ira decrementando. Tras ejecutar la instrucción PUSH se guarda un único byte por lo que se decrementa una posición, mientras que cuando se hace una llamada a una subrutina o a una RSI se guardan los 2

bytes de la dirección de retorno, y este se decrementa en dos posiciones. Por tanto se puede ver que en esta arquitectura la pila se implementa en la SRAM, al contrario de lo que sucede en otras arquitecturas como la de las PIC en que la pila es un espacio de memoria independiente.

Stack Pointer – SP

Bit	15	14	13	12	11	10	9	8	
$3E ($5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
$3D ($5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

- CONTROL REGISTER: este registro, mapeado en la dirección $0055, contiene una serie de bits para el control general del microcontrolador.

MCU Control Register – MCUCR

Bit	7	6	5	4	3	2	1	0	
$35 ($55)	SRE	SRW	SE	SM	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – SRE: External SRAM Enable**
- **Bit 6 – SRW: External SRAM Wait State**
- **Bit 5 – SE: Sleep Enable**
- **Bit 4 – SM: Sleep Mode**
- **Bits 3, 2 – ISC11, ISC10: Interrupt Sense Control 1, Bit 1 and Bit 0**
- **Bits 1, 0 – ISC01, ISC00: Interrupt Sense Control 0, Bit 1 and Bit 0**

Los bits 7 y 6 SRE y SRW, permiten activar la memoria externa. Los bits 5 y 4, SE y SM, permiten configurar el modo suspendido SLEEP del microcontrolador. Los bits 3 y 2, ISC11 y ISC10, del registro permiten configurar si la línea de interrupción externa INT1 generará interrupción por flanco de subida, por flanco de bajada o por nivel a 0. Los bits 1 y 0 corresponden a ISC01 y ISC00 y tienen el mismo fin que los bits 3 y 2 pero para la línea de interrupción externa INT0. No hay que olvidar que para que las líneas de interrupción externa funcionen hay que configurar adecuadamente los registros SREG y GIMSK.

- GENERAL INTERRUPT MASK REGISTER: esta mapeado en la dirección $005B de memoria y según su nombre debería controlar aspectos generales de las interrupciones en general, aunque en la realidad solo permite activar o desactivar las líneas de interrupción externa. Los dos últimos bits de este registro, permiten activar o desactivar las líneas de interrupción externas, así cuando estos valen 1 las líneas correspondientes están activadas, mientras que cuando valen 0 las líneas están desactivadas. El resto de

bits no se utilizan.

General Interrupt Mask Register – GIMSK

Bit	7	6	5	4	3	2	1	0	
$3B ($5B)	INT1	INT0	–	–	–	–	–	–	GIMSK
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – INT1: External Interrupt Request 1 Enable**
- **Bit 6 – INT0: External Interrupt Request 0 Enable**
- **Bits 5..0 – Res: Reserved Bits**

- GENERAL INTERRUPT FLAG REGISTER: este registro, mapeado en la dirección $005ª, únicamente contiene los flags de detección de interrupción correspondientes a las líneas de interrupción externa, bits 7 y 6. Así cuando tiene lugar una interrupción externa el flag asociado a esta se pone a 1 y se salta al correspondiente vector de interrupción. Es cuando se retorna de la RSI que el flag se pone a 0 de nuevo. El resto de bits están reservados.

General Interrupt Flag Register – GIFR

Bit	7	6	5	4	3	2	1	0	
$3A ($5A)	INTF1	INTF0	–	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – INTF1: External Interrupt Flag1**
- **Bit 6 – INTF0: External Interrupt Flag0**
- **Bits 5..0 – Res: Reserved Bits**

SRAM interna y externa

La SRAM interna la forman 512 posiciones de 8 bits mapeadas de la dirección $0060 a la dirección $025F. Los accesos por encima de este rango, hará que se acceda a la SRAM externa, siempre y cuando esta este activada. La SRAM externa se conecta a los puertos A, C y D , y para poderla usar, antes hay que configurar adecuadamente el bit SRE del registro MCUCR. El acceso a la SRAM externa requiere un ciclo mas que el acceso a la SRAM interna, o dos o mas en el caso de que se haga uso de un ciclo de WAIT STATE (para dar tiempo a memorias “lentas”).

Memoria EEPROM

Esta consta de 512 bytes, y no esta directamente mapeada en memoria, sino que es

accedida mediante los registros especiales EARH, EARL, EDR. Los dos primeros permiten especificar la parte alta y baja de la dirección a la que se quiere acceder, y el ultimo permite acceder a los datos. Se garantizan unos 100000 ciclos de borrado/escritura antes de que empiece a fallar.

Interrupciones

El microcontrolador AVR8515 dispone de 12 fuentes de interrupción configurables, cada una de las cuales tiene asociada un vector de interrupción individual en la tabla de vectores de interrupción que se encuentra al inicio de la memoria de programa. Así, cada vez que se produce una interrupción se accede a la tabla de vectores de interrupción, desde donde se salta a la correspondiente rutina de servicio de interrupción. La prioridad de una interrupción viene determinada por la posición de su vector dentro de la tabla de vectores, de forma que las mas prioritarias serán las mas bajas (RESET), y las menos prioritarias las más altas (ANA COMP). La siguiente tabla muestra las diferentes fuentes de interrupción y la posición de sus respectivos vectores en la tabla de vectores de interrupción:

Table 2. Reset and Interrupt Vectors

Vector No.	Program Address	Source	Interrupt Definition
1	$000	RESET	External Reset, Power-on Reset and Watchdog Reset
2	$001	INT0	External Interrupt Request 0
3	$002	INT1	External Interrupt Request 1
4	$003	TIMER1 CAPT	Timer/Counter1 Capture Event
5	$004	TIMER1 COMPA	Timer/Counter1 Compare Match A
6	$005	TIMER1 COMPB	Timer/Counter1 Compare Match B
7	$006	TIMER1 OVF	Timer/Counter1 Overflow
8	$007	TIMER0, OVF	Timer/Counter0 Overflow
9	$008	SPI, STC	Serial Transfer Complete
10	$009	UART, RX	UART, Rx Complete
11	$00A	UART, UDRE	UART Data Register Empty
12	$00B	UART, TX	UART, Tx Complete
13	$00C	ANA_COMP	Analog Comparator

Por tanto para que se ejecute la RSI asociada a una interrupción antes hay que activar la fuente de interrupción en el registro de mascara que corresponda, y insertar en la

posición adecuada de la tabla de vectores de interrupción un salto a la dirección donde se encuentra el código de la RSI de esta.

Las causas de interrupción que muestra la tabla son las siguientes:

-Reset: existen tres posibles causas de reset: reset por activación de la alimentación, reset externo, o reset debido al temporizador Watchdog. Así, cuando tiene lugar alguno de estos tres resets, el microcontrolador pasa a ejecutar el código que se encuentra en la posición $0000 (el vector 0), que generalmente es un jump a la rutina de inicialización del sistema.

-INT0 o INT1: estas son las causas asociadas o los cambios en las líneas de interrupción externa. Al producirse esta interrupción se consulta, el vector 2 o 3 de la tabla de vectores de interrupción y se salta a la RSI que corresponda. Se pueden configurar que la interrupción tenga lugar cuando las líneas pasen de 0 a 1 o viceversa mediante el registro MCUCR. Hay que tener en cuenta que aunque los pins se configuren como salida, si alguno de estos pins tiene las interrupciones externas activadas, al variar el valor de esta salida se generará una interrupción. Esto se puede usar como fuente de interrupción software.

-TIMER1 CAPT: tiene lugar cuando se activa el pin de entrada ICP con el fin de capturar el valor del TIMER1. Cuando esto sucede, automáticamente se guarda el contenido del TIMER1 en el registro ICR1 y se consulta el vector de interrupción 4 para ejecutar la RSI asociada a este evento.

-TIMER1 COMPA, TIMER1 COMPB: estas interrupciones tienen lugar cuando el valor del TIMER1 coincide con el contenido del registro OCR1A, o cuando el valor del TIMER1 coincide con el registro OCR1B. Cuando esto sucede se salta a ejecutar la RSI apuntada por los jumps que aparecen en las posiciones 5 o 6 de la tabla de vectores de interrupción.

-TIMER1 OVF, TIMER0 OVF: estas causas de interrupción tienen lugar cuando se desborda el temporizador TIMER1 o el TIMER0. Con las interrupciones activadas, cuando esto sucede, el microcontrolador salta a la tabla de vectores de interrupción para ejecutar el código de las posiciones 7 o 8 de la tabla, que son jumps a las correspondientes RSIs.

-SPI, STC: cuando la interfaz SPI finaliza una transmisión, si el bit SPIE está activado, se genera una interrupción, de tal manera que se consulta la posición 9 de la tabla de

vectores de interrupción donde hay el jump que conduce a la RSI correspondiente.
-UART RX, UART TX, UART URDE: estos vectores controlan el acceso a las RSIs de las interrupciones asociadas a la UART. La interrupción UART RX tiene lugar cuando ha finalizado la recepción de un byte, mientras que la interrupción UART TX tiene lugar tras la transmisión de un byte. En el caso de UART UDRE, esta tiene lugar cuando el registro mediante el que se manda la información al registro de desplazamiento de la UART esta vacío, lo que indica que la UART esta preparada para enviar un nuevo byte. Los vectores asociados a estas fuentes de interrupción son el 10, 11 y 12.
-ANA COMP: es la interrupción asociada al comparador analógico. El comparador analógico consta de dos entradas: un pin positivo y otro negativo. En el momento que la tensión en el pin positivo es mayor que la tensión en el pin negativo se activa el bit ACO. La interrupción se generara en la transición de ACO de 1 a 0, o de 0 a 1 en función de como se hayan configurado los bits ACIS1 y ACIS0 del registro de configuración y control del comparador analógico ACSR. Tal como muestra la tabla, el vector asociado a esta interrupción es el 13.

Cuando tiene lugar una interrupción, el bit Global Interrupt Enable GIE se pone a 0 de forma que mientras se ejecuta su RSI se desactivan todas las interrupciones. Las interrupciones se reactivan al volver de la RSI: al ejecutarse el RETI, el bit GIE se pone de nuevo a 1. Si se desea disponer de interrupciones anidadas, hay que activar vía software el bit GIE al entrar en una interrupción, de lo contrario no se atenderán nuevas interrupciones hasta que no finalice la ejecución de la rutina de servicio de interrupción en curso.

Si una interrupción sucede cuando el bit Global Interrupt Enable esta a 0, es decir cuando las interrupciones están desactivadas, el flag asociado a esa interrupción se mantiene a 1, a la espera de ser detectada al finalizar la ejecución de la RSI en curso y reactivarse las interrupciones. Cuando hay mas de una interrupción esperando ser detectada se sirve primero a las mas prioritaria. Las interrupciones que no disponen de un flag de interrupción no pueden ser recordadas, y por tanto estas solo pueden ser detectadas en el preciso momento que tienen lugar, sino se pierden.

Puertos de E/S

La mayoría de los puertos de E/S están multiplexados con alguno de los periféricos del microcontrolador. Este consta de 4 puertos de E/S:

-PORT A: es un puerto de 8 bits bidireccional, y se controla mediante los registros PORTA($003B), PINA($0039), DDRA($003ª). PORTA es un registro tanto de lectura como de escritura y permite actualizar y consultar los valores que se están mostrando a través del puerto A cuando este está configurado como salida. PINA es solo de lectura y permite leer el valor de los pins del puerto cuando este esta configurado como entrada. Así al consultar PORTA se lee del Data Latch de salida del puerto mientras que al consultar PINA se lee directamente el estado lógico de los pins del puerto. DDRA permite especificar la dirección de los diferentes bits del puerto, es decir que permite determinar que bits actuaran como bits de salida, y cuales como bits de entrada: así los pins con sus bits en DDRA a 0 actúan como entradas, mientras que los pins con sus bits en DDRA a 1 actúan como salidas.

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0
$1B ($3B)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0
$1A ($3A)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0
$19 ($39)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
Read/Write	R	R	R	R	R	R	R	R
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Este puerto dispone de resistencias de pull-up de uso opcional. Cuando los pins de este puerto actúan como salidas, estos pueden suministrar hasta 20 ma. Hay que tener en cuenta que si los pins de este puerto están configurados como entradas, circulara corriente si las entradas valen 0V y las resistencias de pull-up están activadas. Las resistencias de pull-up se activan escribiendo un 1 en su bit del PORTA cuando el puerto esta configurado como entrada, y se desactivan escribiendo un 0 cuando este puerto esta configurado como entrada o simplemente configurando el puerto como salida.

Algunos de los pins de este puerto están multiplexados con algunas de las señales para el control de la SRAM externa.

-PORT B: es otro puerto de 8 bits que se controla mediante los registros PORTB($0038), PINB($0037), DDRB($0036), que tienen exactamente la misma funcionalidad que los registros PORTA, PINA, DDRA del puerto A. También dispone de resistencias de pull-up de uso opcional.

Port B Data Register – PORTB

Bit	7	6	5	4	3	2	1	0	
$18 ($38)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port B Data Direction Register – DDRB

Bit	7	6	5	4	3	2	1	0	
$17 ($37)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port B Input Pins Address – PINB

Bit	7	6	5	4	3	2	1	0	
$16 ($36)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Además algunos de los pins de este puerto están multiplexados con algunos de los pins de la SPI, tal como muestra la siguiente tabla:

Table 20. Port B Pin Alternate Functions

Port Pin	Alternate Functions
PB0	T0 (Timer/Counter 0 External Counter Input)
PB1	T1 (Timer/Counter 1 External Counter Input)
PB2	AIN0 (Analog Comparator positive input)
PB3	AIN1 (Analog Comparator negative input)
PB4	$\overline{SS}$ (SPI Slave Select Input)
PB5	MOSI (SPI Bus Master Output/Slave Input)
PB6	MISO (SPI Bus Master Input/Slave Output)
PB7	SCK (SPI Bus Serial Clock)

-PORT C: se trata de otro puerto bidireccional de 8 bits que se controla mediante los registros PORTC($0035), PINC($0033), DDRC($0034), que tienen exactamente la

misma funcionalidad que los respectivos registros de los puertos PORTA y PORTB. También dispone de resistencias de pull-up de uso opcional.

Port C Data Register – PORTC

Bit	7	6	5	4	3	2	1	0	
$15 ($35)	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port C Data Direction Register – DDRC

Bit	7	6	5	4	3	2	1	0	
$14 ($34)	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port C Input Pins Address – PINC

Bit	7	6	5	4	3	2	1	0	
$13 ($33)	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Algunos de los pins de este puerto también están multiplexados con algunas de las señales para el control de la SRAM externa.

-PORT D: otro puerto bidireccional de 8 bits que se controla con sus respectivos registros de control PORTD ($0032) , PIND ($0030) , DDRD ($0031). También dispone de resistencias de pull-up opcionales, que se configuran igual que en los demás puertos.

Port D Data Register – PORTD

Bit	7	6	5	4	3	2	1	0	
$12 ($32)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port D Data Direction Register – DDRD

Bit	7	6	5	4	3	2	1	0	
$11 ($31)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port D Input Pins Address – PIND

Bit	7	6	5	4	3	2	1	0	
$10 ($30)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Algunos de los pins de este puerto están multiplexados con pins de la UART.

Temporizadores/Contadores

Este microcontrolador dispone de dos temporizadores/contadores de 8 y 16 bits, con preescaler compartido por los dos pero ajustable de forma independiente para cada uno. Además estos se pueden configurar para utilizar como sincronismo una señal externa, en lugar de la señal de sincronismo interna, lo que les permite actuar como contadores. El preescaler puede dividir la frecuencia de la señal de clock principal por 8, 64, 256 y 1024.

Cada temporizador/contador dispone de sus propios registros de control a parte de otros dos registros compartidos. Estos dos registros compartidos por los dos contadores son:
TIMER/COUNTER INTERRUPT MASK REGISTER: este registro permite activar o desactivar las diferentes interrupciones generadas por los timers/contadores, de forma que poniendo a 0 a 1 estos bits se pueden inhibir o desinhibir las diferentes interrupciones asociadas a los dos temporizadores/contadores.

Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
$39 ($59)	TOIE1	OCIE1A	OCIE1B	–	TICIE1	–	TOIE0	–	TIMSK
Read/Write	R/W	R/W	R/W	R	R/W	R	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – TOIE1: Timer/Counter1 Overflow Interrupt Enable**
- **Bit 6 – OCE1A: Timer/Counter1 Output CompareA Match Interrupt Enable**
- **Bit 5 – OCIE1B: Timer/Counter1 Output CompareB Match Interrupt Enable**
- **Bit 4 – Res: Reserved Bit**
- **Bit 3 – TICIE1: Timer/Counter1 Input Capture Interrupt Enable**
- **Bit 2 – Res: Reserved Bit**
- **Bit 1 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**
- **Bit 0 – Res: Reserved Bit**

TIMER/COUNTER INTERRUPT FLAG REGISTER: es el registro que contiene los flags de las diferentes interrupciones asociadas a los timers. En el momento que tiene lugar una interrupción, el bit del registro TIFR asociado a esta se pone a 1. A continuación, si esta interrupción es la mas prioritaria, se ejecuta la RSI asociada, y al terminar la ejecución de esta, el flag se reestablece a 0.

Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
$38 ($58)	TOV1	OCF1A	OCIFB	–	ICF1	–	TOV0	–	TIFR
Read/Write	R/W	R/W	R/W	R	R/W	R	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – TOV1: Timer/Counter1 Overflow Flag**
- **Bit 6 – OCF1A: Output Compare Flag 1A**
- **Bit 5 – OCF1B: Output Compare Flag 1B**
- **Bit 4 – Res: Reserved Bit**
- **Bit 3 – ICF1: Input Capture Flag 1**
- **Bit 2 – Res: Reserved Bit**
- **Bit 1 – TOV: Timer/Counter0 Overflow Flag**
- **Bit 0 – Res: Reserved Bit**

TIMER/COUNTER0 8 bits

Este puede seleccionar que su entrada de sincronismo sea o el clock interno, el clock interno preescalado, o un pin externo. También presenta la posiblidad de poder ser parado por software. Únicamente genera interrupción al producirse overflow.

Se controla mediante los registros TCCR0 y TCTNT0. Los bits CS02, CS01 y CS00 de TCCR0 permiten ajustar el preescaler para el temporizador.

Timer/Counter0 Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
$33 ($53)	–	–	–	–	–	CS02	CS01	CS00	TCCR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..3 – Res: Reserved Bits**
- **Bits 2, 1, 0 – CS02, CS01, CS00: Clock Select0, Bits 2, 1 and 0**

Table 7. Clock 0 Prescale Select

CS02	CS01	CS00	Description
0	0	0	Stop, the Timer/Counter0 is stopped.
0	0	1	CK
0	1	0	CK/8
0	1	1	CK/64
1	0	0	CK/256
1	0	1	CK/1024
1	1	0	External Pin T0, falling edge
1	1	1	External Pin T0, rising edge

Timer Counter0 – TCNT0

Bit	7	6	5	4	3	2	1	0	
$32 ($52)	MSB							LSB	TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TIMER/COUNTER1 16 bits:

Al igual que el timer de 8 bits, este puede seleccionar como sincronismo el clock interno, el clock interno preescalado, o un pin externo. También puede ser parado desde el software. Este, al contrario que el de 8 bits, puede generar diferentes tipos de interrupción: cuando se produce overflow, cuando el valor del contador es igual al de los registros OCR1A o OCR1B, o cuando se activa el pin de captura del contador ICP. Otra funcionalidad de este, es que puede actuar como modulador de ancho de pulso PWM.

Se controla mediante diferentes registros TCCR1A, TCCR1B, TCNT1L, OCR1AH & OCR1AL, OCR1BH & OCR1BL, ICR1H, ICR1L:

Timer/Counter1 Control Register A – TCCR1A

Bit	7	6	5	4	3	2	1	0	
$2F ($4F)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	PWM11	PWM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7, 6 – COM1A1, COM1A0: Compare Output Mode1A, Bits 1 and 0**
- **Bits 5, 4 – COM1B1, COM1B0: Compare Output Mode1B, Bits 1 and 0**

Table 8. Compare 1 Mode Select

COM1X1	COM1X0	Description
0	0	Timer/Counter1 disconnected from output pin OC1X
0	1	Toggle the OC1X output line.
1	0	Clear the OC1X output line (to zero).
1	1	Set the OC1X output line (to one).

Note: X = A or B
In PWM mode, these bits have a different function. Refer to Table 12 on page 40 for a detailed description.

- **Bits 3..2 – Res: Reserved Bits**
- **Bits 1..0 – PWM11, PWM10: Pulse Width Modulator Select Bits 1 and 0**

Table 9. PWM Mode Select

PWM11	PWM10	Description
0	0	PWM operation of Timer/Counter1 is disabled
0	1	Timer/Counter1 is an 8-bit PWM
1	0	Timer/Counter1 is a 9-bit PWM
1	1	Timer/Counter1 is a 10-bit PWM

Timer/Counter1 Control Register B – TCCR1B

Bit	7	6	5	4	3	2	1	0	
$2E ($4E)	ICNC1	ICES1	–	–	CTC1	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ICNC1: Input Capture1 Noise Canceler (4 CKs)**
- **Bit 6 – ICES1: Input Capture1 Edge Select**
- **Bits 5, 4 – Res: Reserved Bits**
- **Bit 3 – CTC1: Clear Timer/Counter1 on Compare Match**
- **Bits 2, 1, 0 – CS12, CS11, CS10: Clock Select1, Bits 2, 1 and 0**

Table 10. Clock 1 Prescale Select

CS12	CS11	CS10	Description
0	0	0	Stop, the Timer/Counter1 is stopped.
0	0	1	CK
0	1	0	CK/8
0	1	1	CK/64
1	0	0	CK/256
1	0	1	CK/1024
1	1	0	External Pin T1, falling edge
1	1	1	External Pin T1, rising edge

Timer/Counter1 – TCNT1H AND TCNT1L

Bit	15	14	13	12	11	10	9	8	
$2D ($4D)	MSB								TCNT1H
$2C ($4C)								LSB	TCNT1L
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Timer/Counter1 Output Compare Register – OCR1AH AND OCR1AL

Bit	15	14	13	12	11	10	9	8	
$2B ($4B)	MSB								OCR1AH
$2A ($4A)								LSB	OCR1AL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Timer/Counter1 Output Compare Register – OCR1BH AND OCR1BL

Bit	15	14	13	12	11	10	9	8	
$29 ($49)	MSB								OCR1BH
$28 ($48)								LSB	OCR1BL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Timer/Counter1 Input Capture Register – ICR1H AND ICR1L

Bit	15	14	13	12	11	10	9	8	
$25 ($45)	MSB								ICR1H
$24 ($44)								LSB	ICR1L
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

UART

La UART (Universal Assynchronous Reciver & Transmitter) permite la comunicación entre el microcontrolador y otros dispositivos provistos de puerto serie convencional. Esta puede operar en modos de 8 o 9 bits , a diferentes velocidades de transmisión, independientemente de la frecuencia de reloj utilizada. Además presenta mecanismos de filtrado de ruido, detección de error, y de interrupciones para el control de la transmisión y recepción de datos.

Antes de iniciar la comunicación hay que configurar adecuadamente la UART. El primer paso es seleccionar la velocidad de transmisión. Para ello hay que escribir el valor adecuado en el registro UBRR. La Tabla 17 del datasheet muestra los valores a escribir en UBRR para las configuraciones de velocidad de transmisión y frecuencia de clock más comunes (el resto de configuraciones se puede calcular mediante una sencilla formula matemática) :

Table 17. UBRR Settings at Various Crystal Frequencies

Baud Rate	1 MHz	%Error	1.8432 MHz	%Error	2 MHz	%Error	2.4576 MHz	%Error
2400	UBRR= 25	0.2	UBRR= 47	0.0	UBRR= 51	0.2	UBRR= 63	0.0
4800	UBRR= 12	0.2	UBRR= 23	0.0	UBRR= 25	0.2	UBRR= 31	0.0
9600	UBRR= 6	7.5	UBRR= 11	0.0	UBRR= 12	0.2	UBRR= 15	0.0
14400	UBRR= 3	7.8	UBRR= 7	0.0	UBRR= 8	3.7	UBRR= 10	3.1
19200	UBRR= 2	7.8	UBRR= 5	0.0	UBRR= 6	7.5	UBRR= 7	0.0
28800	UBRR= 1	7.8	UBRR= 3	0.0	UBRR= 3	7.8	UBRR= 4	6.3
38400	UBRR= 1	22.9	UBRR= 2	0.0	UBRR= 2	7.8	UBRR= 3	0.0
57600	UBRR= 0	7.8	UBRR= 1	0.0	UBRR= 1	7.8	UBRR= 2	12.5
76800	UBRR= 0	22.9	UBRR= 1	33.3	UBRR= 1	22.9	UBRR= 1	0.0
115200	UBRR= 0	84.3	UBRR= 0	0.0	UBRR= 0	7.8	UBRR= 0	25.0

Baud Rate	3.2768 MHz	%Error	3.6864 MHz	%Error	4 MHz	%Error	4.608 MHz	%Error
2400	UBRR= 84	0.4	UBRR= 95	0.0	UBRR= 103	0.2	UBRR= 119	0.0
4800	UBRR= 42	0.8	UBRR= 47	0.0	UBRR= 51	0.2	UBRR= 59	0.0
9600	UBRR= 20	1.6	UBRR= 23	0.0	UBRR= 25	0.2	UBRR= 29	0.0
14400	UBRR= 13	1.6	UBRR= 15	0.0	UBRR= 16	2.1	UBRR= 19	0.0
19200	UBRR= 10	3.1	UBRR= 11	0.0	UBRR= 12	0.2	UBRR= 14	0.0
28800	UBRR= 6	1.6	UBRR= 7	0.0	UBRR= 8	3.7	UBRR= 9	0.0
38400	UBRR= 4	6.3	UBRR= 5	0.0	UBRR= 6	7.5	UBRR= 7	6.7
57600	UBRR= 3	12.5	UBRR= 3	0.0	UBRR= 3	7.8	UBRR= 4	0.0
76800	UBRR= 2	12.5	UBRR= 2	0.0	UBRR= 2	7.8	UBRR= 3	6.7
115200	UBRR= 1	12.5	UBRR= 1	0.0	UBRR= 1	7.8	UBRR= 2	20.0

Baud Rate	7.3728 MHz	%Error	8 MHz	%Error	9.216 MHz	%Error	11.059 MHz	%Error
2400	UBRR= 191	0.0	UBRR= 207	0.2	UBRR= 239	0.0	UBRR= 287	-
4800	UBRR= 95	0.0	UBRR= 103	0.2	UBRR= 119	0.0	UBRR= 143	0.0
9600	UBRR= 47	0.0	UBRR= 51	0.2	UBRR= 59	0.0	UBRR= 71	0.0
14400	UBRR= 31	0.0	UBRR= 34	0.8	UBRR= 39	0.0	UBRR= 47	0.0
19200	UBRR= 23	0.0	UBRR= 25	0.2	UBRR= 29	0.0	UBRR= 35	0.0
28800	UBRR= 15	0.0	UBRR= 16	2.1	UBRR= 19	0.0	UBRR= 23	0.0
38400	UBRR= 11	0.0	UBRR= 12	0.2	UBRR= 14	0.0	UBRR= 17	0.0
57600	UBRR= 7	0.0	UBRR= 8	3.7	UBRR= 9	0.0	UBRR= 11	0.0
76800	UBRR= 5	0.0	UBRR= 6	7.5	UBRR= 7	6.7	UBRR= 8	0.0
115200	UBRR= 3	0.0	UBRR= 3	7.8	UBRR= 4	0.0	UBRR= 5	0.0

UART BAUD Rate Register – UBRR

Bit	7	6	5	4	3	2	1	0	
$09 ($29)	MSB							LSB	UBRR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Tras seleccionar la velocidad de transmisión, hay que configurar el bit de activación de transmisión TXEN y el bit de activación de recepción RXEN del registro UCR. Estos se han de activar o no en función de si se desea solo transmitir (TXEN=1 RXEN=0), solo recibir (TXEN=0 RXEN=1), o transmitir y recibir a la vez (TXEN=1 RXEN=1).

UART Control Register – UCR

Bit	7	6	5	4	3	2	1	0	
$0A ($2A)	RXCIE	TXCIE	UDRIE	RXEN	TXEN	CHR9	RXB8	TXB8	UCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	W	
Initial Value	0	0	0	0	0	0	1	0	

- Bit 7 – RXCIE: RX Complete Interrupt Enable
- Bit 6 – TXCIE: TX Complete Interrupt Enable
- Bit 5 – UDRIE: UART Data Register Empty Interrupt Enable
- Bit 4 – RXEN: Receiver Enable
- Bit 3 – TXEN: Transmitter Enable
- Bit 2 – CHR9: 9-bit Characters
- Bit 1 – RXB8: Receive Data Bit 8
- Bit 0 – TXB8: Transmit Data Bit 8

Tanto para la recepción como para la transmisión, se utiliza el registro UDR que en realidad lo componen dos registros independientes: uno de lectura para la recepción de datos, y otro de escritura para la transmisión de datos. En función de si se lee de UDR o escribe sobre UDR se estará accediendo a uno u otro registro.

UART I/O Data Register – UDR

Bit	7	6	5	4	3	2	1	0	
$0C ($2C)	MSB							LSB	UDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Para la transmisión de un dato únicamente hay que escribirlo en el registro UDR (UART Data Register) de donde el registro de desplazamiento de la UART lo tomará para después enviarlo. No obstante antes de escribir el dato hay que asegurarse de que el bit UDRE (UART Data Register Empty) está a 1: esto indica que UDR realmente esta vacío y preparado para recibir un nuevo valor. Si este no esta a 1 significa que el registro todavía contiene un dato valido pendiente de ser cargado en el registro de desplazamiento, por lo que si se escribe en UDR en ese instante se estará sobrescribiendo sobre un dato válido y por tanto perdiendo información. Por tanto antes de cargar un nuevo valor en UDR hay que asegurarse de que este registro esta listo, del resto se encarga la UART.

Además cuando el contenido del registro de desplazamiento se ha enviado por completo y no hay nuevos datos en UDR, el bit TXC (Transmisión Completa) del registro USR se pone a 1, indicando que ha finalizado la transmisión.

UART Status Register – USR

Bit	7	6	5	4	3	2	1	0	
$0B ($2B)	RXC	TXC	UDRE	FE	OR	–	–	–	USR
Read/Write	R	R/W	R	R	R	R	R	R	
Initial Value	0	0	1	0	0	0	0	0	

- Bit 7 – RXC: UART Receive Complete
- Bit 6 – TXC: UART Transmit Complete
- Bit 5 – UDRE: UART Data Register Empty
- Bit 4 – FE: Framing Error
- Bit 3 – OR: Overrun
- Bits 2..0 – Res: Reserved Bits

Para la recepción de un dato solo hay que leer el contenido del registro UDR después de que el dato haya sido recibido. Para saber si se ha recibido un dato o no, únicamente hay que consultar el bit RXC (Recepción Completa) del registro USR el cual indica si se ha completado la recepción de un dato o no. Así, en el momento en que RXC está a 1 ya se puede leer el dato recibido en UDR. Tras leer UDR este bit se pone automáticamente a 0 y no volverá a valer 1 hasta que se reciba un nuevo dato.

Los bits TXCIEN, RXCIEN del registro UCR permiten activar las interrupciones asociadas a la activación de los bits TXC y RXC. Así con TXCIEN a 1, al activarse el bit TXC tras completar una transmisión, se generará una interrupción. De igual forma, con RXCIEN a 1, al activarse el bit RXC tras completar la recepción, también se generará una interrupción. Por tanto, estos bits permiten la gestión de la transmisión y recepción de datos en la UART mediante interrupciones.

El registro USR también dispone de dos bits para la detección de errores. El bit FE (Framing Error) se pone a 1 cuando se produce un error en la comunicación, mientras que el bit OR (Overrun) se pone a 1 cuando el dato recibido se lee demasiado tarde, es decir cuando se lee después de que haya sido sobrescrito con el siguiente dato.

Además, la UART puede operar con datos de 8 o 9 bits. Cuando el bit CHR9 del registro UCR esta desactivado la UART opera en modo 8 bits, mientras que cuando esta activado opera en modo 9 bits. En el modo de 8 bits se utiliza únicamente el registro UDR para la transmisión y recepción de datos, mientras que en el modo de 9 bits, a parte de usar UDR también se utilizan los bits TXB8 y RXB8 del registro UCR. Estos bits contienen el noveno bit (el de más peso): TXB8 contiene el noveno bit a transmitir,

RXB8 contiene el noveno bit recibido. Así para enviar un dato de 9 bits hay que escribir los 8 bits bajos en UDR y el ultimo bit en TXB8. En cambio para recibir un dato de 9 bits hay que leer los 8 bits bajos en UDR y el ultimo bit en RXB8.

SPI

La SPI (Serial Peripheral Interface) permite la comunicación serie síncrona con otros dispositivos que también dispongan de SPI. En la comunicación a través de un bus SPI se distingue entre dispositivo esclavo y dispositivo maestro y todo el proceso de comunicación se puede hacer mediante 3 o 4 hilos: MOSI, MISO, SCK, SS (obviamente la masa ha de ser común) . El maestro es quién controla todo el proceso de transmisión. Las señales de este son:

-SCK (PB7) es la salida de clock cuando el dispositivo actúa en modo maestro, y la entrada cuando lo hace en modo esclavo. Es esta señal la encargada de marcar el ritmo de la transferencia: así en el flanco de subida de esta, el esclavo captura el valor del bit enviado por el maestro, y en el flanco de bajada el maestro captura el bit respondido por el esclavo.

-MOSI (PB5) actúa como salida de datos serie en el dispositivo maestro y como entrada cuando el dispositivo es el esclavo.

-MISO (PB6) actúa como entrada de datos serie en el dispositivo maestro y como salida en el dispositivo esclavo.

-¡SS (PB4) es la línea que permite seleccionar a un dispositivo como esclavo o como maestro, así cuando el ¡SS de un dispositivo está a 1 significa que este actúa como maestro, mientras que cuando permanece a 0 significa que lo hace como esclavo. En el caso de los dispositivos maestros, el pin 4 correspondiente a la señal ¡SS se puede utilizar como un pin de E/S normal: si se configura como salida no afectará al sistema SPI independientemente del valor que tome, mientras que si utiliza como pin de entrada el usuario debe asegurarse que este siempre está a 1, ya que de lo contrario el sistema SPI pasaría a actuar como esclavo.

Para transmitir un byte únicamente hay que escribir este en el registro de datos de la SPI (SPDR) y tras esto la SPI activará el clock SCK y comenzará a transmitir los bits a través de MOSI que serán recogidos por el esclavo también a través de MOSI. Cuando se transmite un dato del maestro al esclavo este también es transmitido en sentido

inverso simultáneamente, es decir que en un ciclo de transmisión los datos entre el esclavo y el maestro son intercambiados.

SPI Data Register – SPDR

Bit	7	6	5	4	3	2	1	0	
$0F ($2F)	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	x	x	x	x	x	x	x	x	Undefined

Tras finalizar la transmisión del byte, el maestro detiene el clock y activa el bit de final de transmisión SPIF del registro SPSR, de tal forma que se genera una interrupción si el bit de interrupción asociada a este esta activado. Para activar las interrupciones del SPI hay que poner a 1 el bit SPIE en el registro SPCR.

SPI Status Register – SPSR

Bit	7	6	5	4	3	2	1	0	
$0E ($2E)	SPIF	WCOL	–	–	–	–	–	–	SPSR
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7 – SPIF: SPI Interrupt Flag
- Bit 6 – WCOL: Write Collision Flag
- Bits 5..0 – Res: Reserved Bits

SPI Control Register – SPCR

Bit	7	6	5	4	3	2	1	0	
$0D ($2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7 – SPIE: SPI Interrupt Enable
- Bit 6 – SPE: SPI Enable
- Bit 5 – DORD: Data Order
- Bit 4 – MSTR: Master/Slave Select
- Bit 3 – CPOL: Clock Polarity
- Bit 2 – CPHA: Clock Phase
- Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0

Apéndice C
LCDs alfanuméricos compatibles HD44780

Introducción

Existen gran variedad de LCDs en el mercado, pero quizás los más utilizados en los pequeños proyectos de electrónica son los LCDs alfanuméricos compatibles con el controlador HD44780 de Hitachi. Existen otros tipos de LCDs con mejores prestaciones gráficas, pero este controlador ofrece una interfaz muy sencilla y versátil para controlar LCDs alfanuméricos, lo que lo hace ideal para proyectos con microcontrolador en los que no se desea complicar el hardware ni el software.

Los LCDs compatibles con el controlador HD44780 se presentan en el mercado con diferentes configuraciones que se diferencian por el numero de filas y de columnas. Existen modelos con 8,16,20,24,32 o 40 columnas y 1,2 o 4 filas. La mayoría de los modelos disponen de caracteres de 5x7 pixels, mientras que algunos tienen caracteres de 5x10 pixels. La programación de los distintos modelos es prácticamente idéntica, con alguna pequeña diferencia en el proceso de inicialización, configuración y organización de los caracteres en memoria.

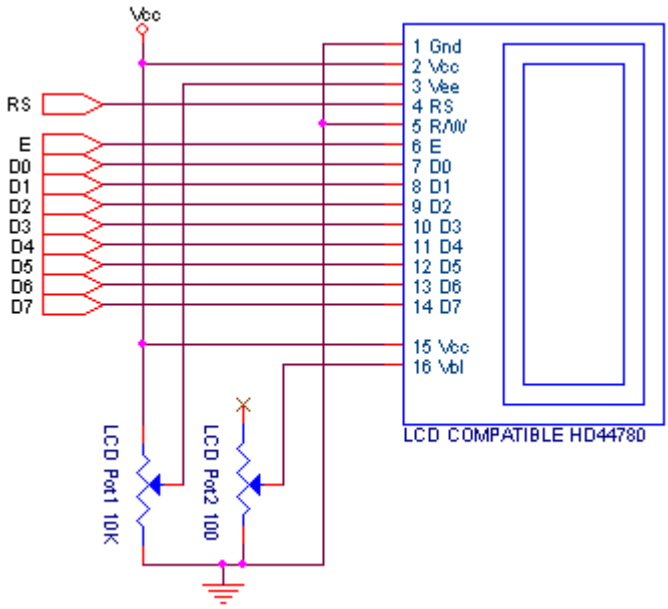
Conexiones

Normalmente la conexión se realiza mediante un conector de 14 o 16 pins. El conector de 16 pins se usa solo en los modelos que disponen de retroiluminación. De estos pins, 8 son para las líneas de datos, 3 para las líneas de control, y el resto son para la alimentación, el control del contraste y el control de la retroiluminación si la tiene. Estos pins no siempre aparecen ordenados en la placa del LCD, por eso, antes de conectar

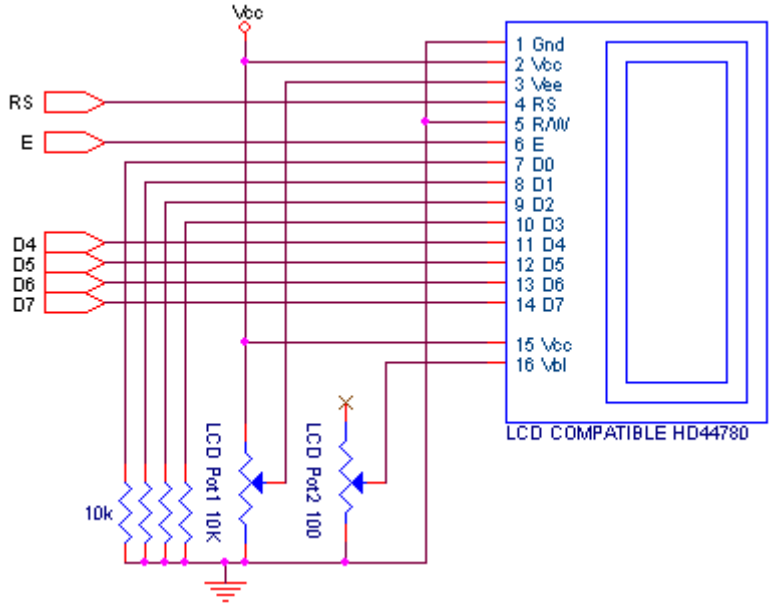
nada es bueno consultar el datasheet para evitar equivocarse y soldar donde no toca. Normalmente estos pins son:

Pin		Nombre Función
1	GND	Masa
2	VCC	Alimentación
3	VEE	Control del contraste
4	RS	Selección de registro de comandos o de datos.
5	R/!W	Escritura o lecutra.
6	E	Enable.
7	D0	Bit 0 de datos
8	D1	Bit 1 de datos
9	D2	Bit 2 de datos
10	D3	Bit 3 de datos
11	D4	Bit 4 de datos
12	D5	Bit 5 de datos
13	D6	Bit 6 de datos
14	D7	Bit 7 de datos
15	VCC	Alimentación de la retroiluminación
16	VBL	Control de la retroiluminación

Estos LCDs pueden operar en modo de 8bits o en modo de 4bits. Este documento únicamente describe el modo de 8 bits, pero con cuatro ideas básicas se puede extender al de 4 bits: la diferencia entre los dos modos es que en el modo de 4bits la transferencia del dato se realiza en dos ciclos de 4 bits mientras que en el modo de 8 bits se realiza en un único ciclo de 8 bits. Si se usa el modo de 4 bits la transferencia de datos tiene lugar por las líneas de datos de D4..7, el resto de líneas, es decir D0..3 se puede conectar a masa mediante una resistencia (p.ej de 10k). Como se puede ver, el primer modo consume menos puertos de E/S pero es mas lento y un poco mas complejo de programar, en cambio el segundo modo consume mas puertos de E/S pero es mas rápido y sencillo de programar. Como todas las operaciones son de escritura, en este documento también se supone que en todo momento la línea R/W esta a 0, es decir directamente conectada a masa.



Esquema de las conexiones en modo 8 bits



Esquema de las conexiones en modo 4 bits

Caracteres y comandos

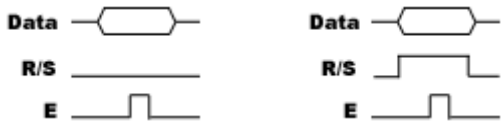
El funcionamiento general del LCD, en el modo de 8 bits, consiste en ejecutar simultaneamente las siguientes acciones:

-mantener el dato en las lineas de datos.

-indicar mediante la linea RS si el dato es un carácter o un comando . Cuando RS está a 0 los datos se interpretan como un comando mientras que cuando esta a 1 se interpretan

como un carácter.

-una vez el dato esta listo y se ha indicado si es un carácter o comando se da un pulso a la señal E para que el LCD capture los datos. En función de si se trata de un comando o un carácter, se ejecutará una operación, o se escribirá un carácter en pantalla. La duración del pulso es una variable importante ya que en LCDs antiguos, con controladores lentos, puede ser que un pulso demasiado rápido no de tiempo al LCD a procesar los datos.



Timing orientativo del envío de un dato. La primera figura muestra el envío de un comando (R/S=0) mientras que la segunda muestra el envío de un carácter (R/S=1)

Por tanto a través de la línea de datos del LCD se pueden transmitir dos tipos de datos: los correspondientes a caracteres y los correspondientes a comandos.

Caracteres

Tal como se comenta en el apartado anterior, el LCD captura el dato en la subida de la señal E y sabe que se trata de un carácter porque la señal R/S esta a 1. Cuando el LCD recibe un carácter, lo único que hace es capturarlo, y guardarlo en la memoria de texto en la posición apuntada por el cursor. Además si el cursor se encuentra una zona visible del LCD este carácter aparecerá por pantalla en la correspondiente posición.

Estos LCDs disponen de una tabla de 256 caracteres distintos la mayoría de los cuales corresponden al alfabeto occidental y al los símbolos fonéticos Katakana japoneses. La distribución de los caracteres es la siguiente:

Ilett, Jullyan, 1997 - *How to use inteligent LCDs*

Tal como se puede ver en la tabla, los 16 primeros códigos corresponden a los caracteres definidos por el usuario (en la CGRAM): debería haber 16 caracteres asociados a estos códigos pero en realidad solo hay 8, lo que es debido a que acceder a los códigos situados entre 08h y 0Fh equivale a acceder a los códigos situados de 00h y 07h. Los códigos que van del 33 al 127 corresponden a los caracteres occidentales, mientras que los que van desde el código 161 hasta el 254 corresponden a caracteres nipones. Los 15 últimos códigos, del 240 al 255, son símbolos comunes entre los que se encuentran el símbolo de la división o las típicas letras griegas. Los códigos que en la tabla aparecen vacíos no tienen caracteres asociados y al mostrar uno de estos simplemente se mostrara un caracteres en blanco.

Comandos

Tal como se comenta en el apartado anterior, el LCD captura el dato en la subida de la señal E y sabe que se trata de un comando porque la señal R/S está a 0. Cuando recibe un comando el LCD ejecuta una acción u otra en función del comando de que se trate. Existen diferentes comandos:

Display & Cursor home: D7..0: 0000 001X
X: puede tomar cualquier valor

Lleva el cursor al inicio, es decir que sitúa el cursor en la dirección 0x00. Esta instrucción es útil para situar el cursor en una posición válida conocida, si se produce algún error de escritura o posicionamiento en el LCD.

*Clear Display:*D7..0: 0000 0001
Hace lo mismo que el comando anterior pero además borra todo el contenido del LCD

Character Entry Mode: D7..0: 0000 01AB
A:1 Incremento 0 Decremento.
B:1 Desplazamiento del cursor activo 0 Desplazamiento del cursor inactivo.

Este comando permite configurar el desplazamiento del cursor tras la inserción de un carácter. Es decir que permite configurar si tras capturar un carácter, el cursor del LCD se ha de desplazar hacia atrás o hacia delante. También permite escoger si se desea que este autodesplazamiento este o no activado. Por tanto, como se puede deducir, este comando lo único que hace es configurar el incremento o decremento automático de la dirección de escritura tras una escritura en la memoria de texto.

El uso de este desplazamiento automático puede simplificar bastante la introducción de caracteres en el display, ya que con este activado, para mostrar un mensaje simplemente hay que ir enviando los caracteres que se quieren mostrar sin preocuparse de la posición del cursor, ya que el mismo cursor se sitúa en la siguiente posición. En cambio si se tiene el desplazamiento del cursor inactivo, antes de enviar cada carácter habrá que especificar la posición del cursor, es decir la dirección de la memoria de texto donde se va a escribir, ya que sino se estarán sobreescribiendo los caracteres anteriormente

escritos.

Display On/Off & Cursor: D7..0: 0000 1ABC
A:1 Display activado 0 Display desactivado
B:1 Underline del cursor activado 0 Underline del cursor desactivado
C:1 Parpadeo del cursor activado 0 Parapadeo del cursor desactivado.

Comando que permite activar el LCD y también inicializar algunos parámetros referentes al cursor. Permite seleccionar si se desea que el cursor este visible en forma de "_", y si se desea que este parapadee o no parpadee.

Display Content/Cursor Shift: D7..0: 0001 ABXX
A:1 Movimiento del contenido 0 Movimiento del cursor:
B:1 Desplazamiento del contenido hacia la derecha 0 Desplazamiento del contenido hacia la izquierda.
X: puede tomar cualquier valor

Este comando sirve para seleccionar si se desea que al insertar un nuevo carácter se desplace todo el contenido del LCD o si solo se desea que se desplace el cursor. También permite seleccionar la dirección de desplazamiento del contenido, es decir si se quiere que al introducir un carácter en una posición el resto del texto se desplace a la derecha o a la izquierda.

Esto permite emular por ejemplo, la entrada de caracteres en las calculadoras, en las que normalmente el cursor aparece en el extremo derecho de la pantalla y al ir añadiendo los diferentes dígitos estos aparecen en el extremo derecho a la vez que los caracteres anteriormente entrados se van desplazando hacia la izquierda a medida que se introducen nuevos dígitos.

Function Set: D7..0: 001A BCXX
A:1 Interfaz en modo 8 bits 0 Interfaz en modo 4 bits
B:1 LCD de 2 líneas 0 LCD de 1 línea
C:1 Caracteres de 5x10 pixels 0 Caracteres de 5x7 pixels

Esta instrucción permite inicializar correctamente el LCD, especificando el modo de trabajo que se va a usar (8 bits o 4 bits), el numero de líneas lógicas del LCD (1 o 2), y el tipo de caracteres que se utilizan en este (5x10 o 5x7 pixels).

Set CGRAM Address: D7..0: 01DD DDDD
D: bits de la dirección CGRAM

Este comando permite seleccionar la dirección de la CGRAM sobre la que se quiere escribir. La CGRAM es la Caracter Generator RAM la cual permite almacenar hasta 8 caracteres definidos por el usuario. Cada 8 posiciones de la CGRAM contienen un carácter de forma que, cada posición de 8 bits contiene la tira de pixels de cada una de las filas del carácter. De cada tira de bits solo se utilizan los 5 bits mas bajos, debido a que el ancho del carácter es de 5 pixels. De estas 8 filas, la primera fila, es decir la situada en la dirección más baja de las 8, corresponde a la parte superior del carácter, mientras que la última, es decir la situada en la posición más alta de las 8, corresponde a la parte inferior del carácter. Además de estas 8 filas solo se muestran las 7 primeras debido a que la fila 8, la correspondiente a la parte inferior, suele estar ocupada por el cursor.

Así para definir un carácter, solo hay que enviar un comando de establecimiento de dirección CGRAM con una dirección múltiple de 8 y a continuación enviar las 8 tiras de bits que forman el carácter, teniendo en cuenta las especificaciones anteriores. El envío de los datos correspondientes a la tira de bits se realiza como si se trataran de caracteres.

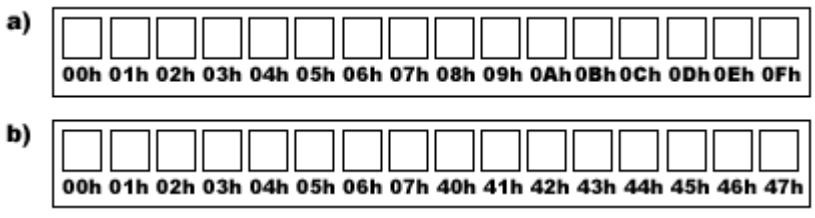
Set Display Address: D7..0: 1DDD DDDD
D: bits de la dirección del cursor
Este comando permite situar el cursor donde se escribirá el siguiente carácter, es decir en la posición del LCD deseada. Cada posición del LCD tiene asociada una dirección concreta en la memoria de texto, de forma que para mostrar un carácter en esa posición concreta hay que escribir sobre su dirección de memoria asociada.

A primera vista, como disponemos de 7 bits de dirección, podemos pensar que podemos

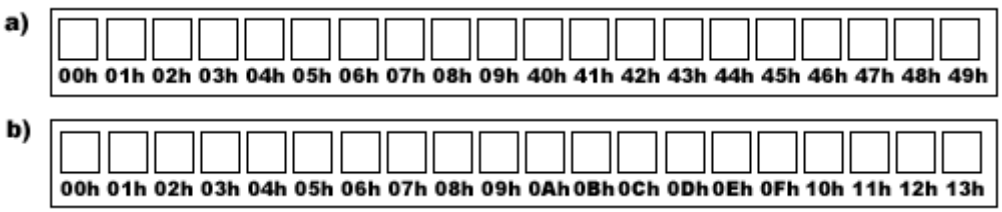
direccionar y escribir en 128 posiciones de la memoria de texto, pero en realidad no todas estas direcciones tienen asociadas una posición en el LCD. De estas 128 direcciones solo están disponibles 80 . Estas 80 direcciones se organizan en 40 direcciones por línea cuando el display dispone de 2 líneas, o en 80 direcciones por línea cuando el LCD dispone de 1 línea. Únicamente los displays de 40 columnas y 2 líneas, y los de 20 columnas y 4 líneas son capaces de mostrar las 80 direcciones. El resto de modelos solo muestran las primeras posiciones de cada línea, o sea, las que caben en el LCD. De hecho se puede visualizar el contenido de la memoria de texto que queda fuera del área visible del LCD haciendo uso de los comandos de desplazamiento.

Por lo general en los LCDs que solo hacen uso de una línea esta comienza en la dirección 00h de la memoria de texto. En cambio, en los LCDs que hacen uso de dos líneas la primera línea también comienza en la dirección 00h, mientras que la segunda línea comienza en la dirección 40h. Los LCDs de 4 lineas en realidad solo hacen uso de 2 líneas, lo que sucede es que estas se disponen físicamente entrelazadas tal como muestran las figuras del documento.

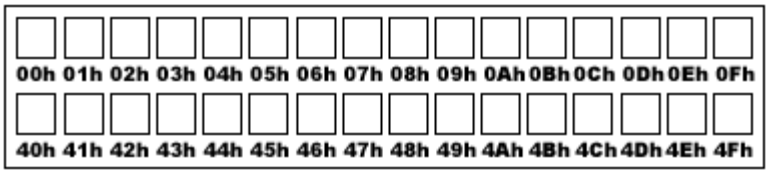
Además la correspondencia posición del carácter vs. dirección de memoria depende del modelo de LCD. Incluso en LCDs con el mismo numero de columnas y filas esta correspondencia puede ser diferente dependiendo del fabricante, de hecho existen infinidad de disposiciones diferentes. Los siguientes esquemas muestran las correspondencias posición carácter vs. dirección de memoria, mas frecuentes:



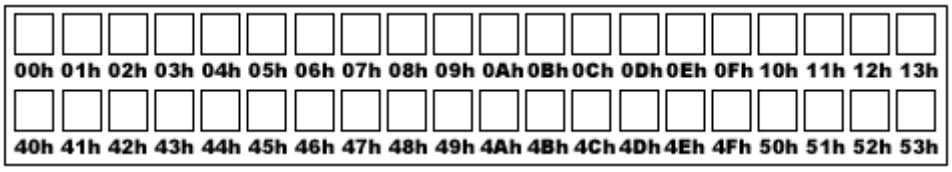
Modelo 16 columnas x 1 fila



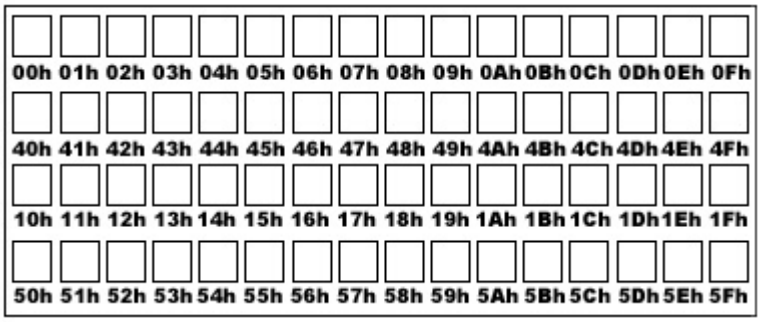
Modelo 20 columnas x 1 fila



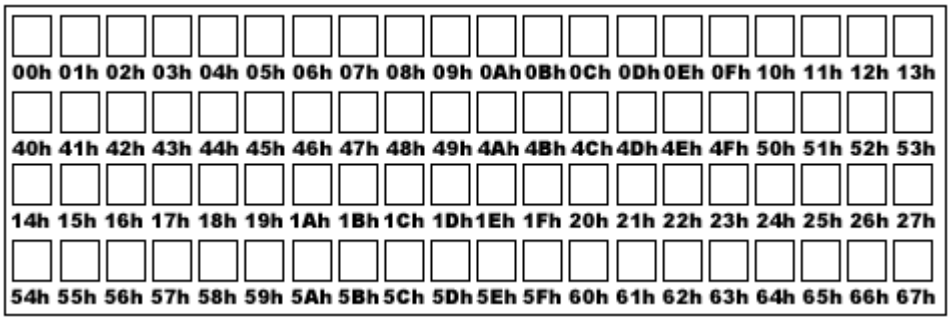
Modelo 16 columnas x 2 filas



Modelo 20 columnas x 2 filas



Modelo 16 columnas x 4 filas



Modelo 20 columnas x 4 filas

Inicialización

Para que el LCD funcione correctamente hay que inicializarlo adecuadamente. Para inicializar adecuadamente el LCD hay que configurar el modo de trabajo, el número de

líneas y el tamaño de carácter mediante el comando *Function Set*. También hay que ejecutar el comando *Display On/Off & Cursor* con el fin de activar el LCD y el cursor.

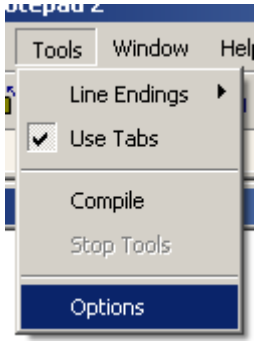
Apéndice D

Configuración básica del entorno de programación WinAVR

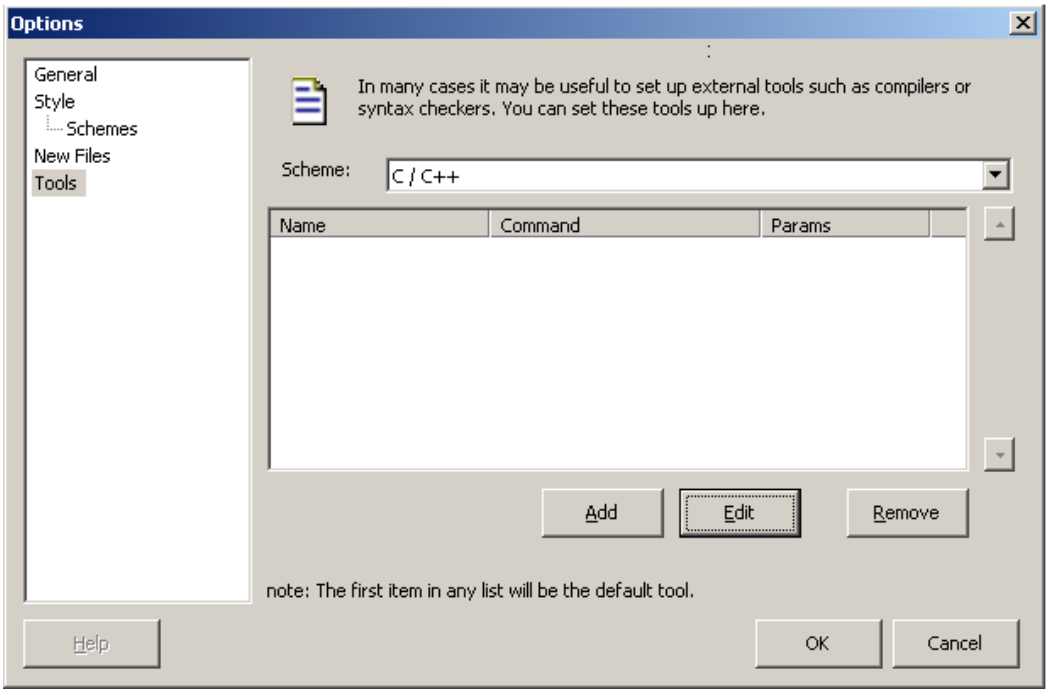
Pasos:

1-Tras descomprimir el paquete WinAVR, se copia el archivo make.exe de la carpeta "...\WinAVR\utils\bin" a la carpeta "...\WinAVR\bin".

2-Se ejecuta la aplicación Programmers Notepad 2 que se encuentra en "...\WinAVR\pn" y en el menú Tools se selecciona la opción Options.



3-Tras abrirse el cuadro de dialogo Options se selecciona Tools, en Scheme se selecciona C/C++ y se hace Add.

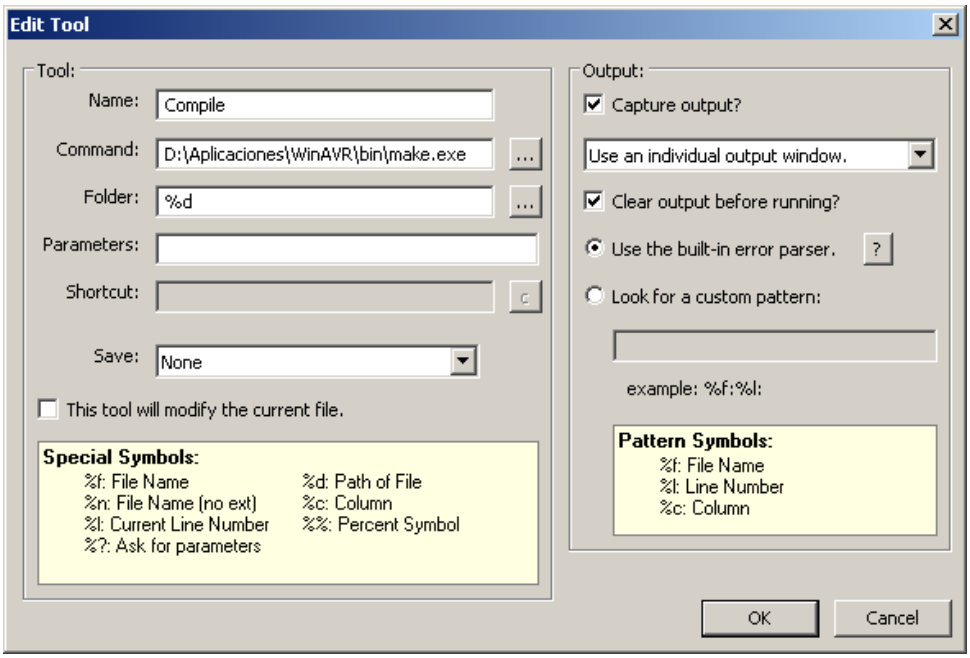


4-Aparece el cuadro de dialogo Edit Tool, cuyos campos se rellenan tal como muestra la imagen.

Name: se introduce el nombre que se le dará a la nueva herramienta

Command: contiene la ubicación del archivo make.exe

Folder: es la carpeta donde se encuentran los archivos fuente (*.c y makefile) y donde se guardaran los resultados.



Después de crear los correspondientes archivos **.c** y el **makefile**, solo hay que pulsar sobre la opción añadida Compile del menú tools para construir el proyecto. Asegurarse de que el makefile esta bien configurado y contiene referencias a los ficheros de nuestro proyecto.