**UNIVERSIDAD AUTONOMA DE MADRID**

**ESCUELA POLITECNICA SUPERIOR**




# PROYECTO FIN DE CARRERA

## Sistema lector de Tarjetas-Chip con acceso USB


**José Rubén Ibáñez Sánchez**


**Enero 2013**

# Sistema lector de Tarjetas-Chip con acceso USB

**AUTOR: José Rubén Ibáñez Sánchez**
**TUTOR: Guillermo González de Rivera Peces**

**Human Computer Technology Laboratory**

HCTLab

**Dpto. de Tecnología Electrónica y de las Comunicaciones**
**Escuela Politécnica Superior**
**Universidad Autónoma de Madrid**

**Enero de 2013**

# Resumen

En el proyecto que se presenta a continuación se ha llevado a cabo el diseño y construcción de un lector de tarjetas-chip con acceso USB. Su diseño busca aunar un reducido tamaño, bajo consumo, sencillez e independencia entre los componentes que lo forman, es decir, modularidad.

Siguiendo esta pauta, el lector esta formado por un microcontrolador como sistema central que comunica por un lado con un chip especializado en la lectura de tarjetas y, por el otro, se comunica con el sistema gestor (PC, sistema embebido, etc) al que se encuentra conectado por medio de una conexión USB.

Para el intercambio de datos entre los dos chips principales, el microcontrolador y el lector de tarjetas, se utiliza un bus $I^2C$ por el que se envían tramas de comandos y datos formateadas según el protocolo ALPAR y, dentro de este, según el protocolo APDU.

Además de la fabricación del prototipo se ha desarrollado un software para la programación del microcontrolador que actúa como sistema central del dispositivo. El cometido de este software es configurar el sistema y mantener el dispositivo listo y a la espera de que se inserte una tarjeta para proceder a su lectura. Una vez terminada la lectura los datos recogidos son enviados por USB al PC y la tarjeta es desactivada para así poder ser extraída de forma segura.

Este es el primer proyecto en el que se utiliza una nueva herramienta de diseño electrónico, utilizada para la captura de esquemáticos y la generación de ficheros para la fabricación de circuitos impresos adquirida en el laboratorio, *Altium Designer*.

# Palabras Clave

Lector de tarjetas, tarjeta-chip, tarjeta inteligente, $I^2C$, USB, ALPAR, APDU, modularidad.

# Abstract

The project presented below shows the design and construction of a chip-card reader with USB access. This design seeks to join a small size, low power, simplicity and independence between the components that constitute it, namely, modularity.

Following this guideline, the designed reader consists of a microcontroller as central system which communicates on one side with a specialized chip card reading and, on the other side, with the PC connected via the USB connection.

An $I^2C$ bus is used to exchange data between the two main chips, microcontroller and card reader. Swapped command and data frames are encapsulated according firstly to the APDU protocol, and then by the ALPAR protocol.

In addition to the manufacture of prototype, software developed programs the central system of the device: the microcontroller. The main purpose of this software is to configure the system and keep the device ready and waiting to insert a card to proceed with its reading. Once finished the reading, the collected data is sent via USB to the PC and the card is deactivated in order to be remove safely.

This is the first project using a new electronic design tool, used to schematic capture and file generation for printed circuit manufacture acquired in laboratory, Altium Designer.



# Key words

Card reader, chip-card, smart card, $I^2C$, USB, ALPAR, APDU, modularity.

# *Agradecimientos*

En primer lugar quiero dar las gracias a mi tutor, Guillermo González de Rivera Peces, por darme la oportunidad de realizar este proyecto y por compartir conmigo su tiempo y conocimientos.

También quiero dar las gracias a todos los miembros del grupo HCTLab, en especial a Alberto Sánchez González, por prestarme su ayuda siempre que la he necesitado.

Gracias a todos los compañeros con los que he compartido estos años, gracias a Chema, Pablo, Rober y Javi por las risas, los ánimos y el apoyo en los momentos de apuro. Y, en especial, gracias a Oscar, por todo lo que me has enseñado y que nunca olvidaré, y a Santi, a quien conocí al principio de esta aventura y ahora es uno de mis mejores amigos.

Dejo para el final de estos agradecimientos lo más importante que hay en mi vida, mi familia y mi novia sin los que jamás habría llegado hasta aquí.

Gracias a mis padres, por todo el cariño que me habéis dado, por el apoyo en los malos momentos y la alegría e ilusión compartida en los buenos. Gracias por la educación y valores que me habéis enseñado y que me han convertido en la persona que soy hoy. Gracias a mi hermana Sandra, por estar siempre a mi lado cuando la necesito y por los buenos ratos que pasamos juntos, os quiero.

Y, en especial, gracias a mi Yadi, por estar siempre a mi lado, por darme todo tu apoyo, ayuda y cariño, por compartir tu vida conmigo. Te quiero.

J. Rubén Ibáñez Sánchez

Enero 2013

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# 1 Introducción

En los últimos años la utilización de tarjetas inteligentes en muchos ámbitos de la vida se ha convertido en algo habitual. Desde la tarjeta SIM (*Subscriber Identity Module*) de los teléfonos móviles hasta las tarjeta de crédito, pasando por DNI-e (Documento Nacional de Identidad electrónico), tarjeta sanitaria, tarjetas de identificación, tarjetas de transporte… Todas ellas son tarjetas inteligentes que nos ahorran trámites burocráticos, portan de forma segura nuestros datos personales y bancarios, son resistentes y duraderas, son fáciles de transportar y de usar y, por encima de todo, nos hacen la vida más fácil y cómoda.

De forma paralela a la expansión de las tarjetas inteligentes surgen los dispositivos necesarios para el uso de estas tarjetas, los lectores o lectores/escritores de tarjetas inteligentes. Estos dispositivos actúan como puente entre la tarjeta inteligente y el sistema anfitrión, es decir, el sistema que alberga la aplicación final para la que se creo esa tarjeta.

El hecho de que las tarjetas inteligentes sean capaces de cubrir un rango de necesidades tan amplio viene determinado por su gran capacidad operativa, dado que integran una CPU (*Central Processing Unit*), memorias, buses de comunicación, sistema operativo y sistema de archivos. Si a todo esto se le añaden unos sistemas de seguridad que protegen los datos que portan, se está ante una tecnología con un sin fin de posibilidades.

## 1.1 Motivación

Desde el grupo HCTLab de la Escuela Politécnica Superior (EPS) de la Universidad Autónoma de Madrid (UAM) se muestra gran interés por desarrollar un lector de tarjetas-chip propio que pueda sustituir a los dispositivos comerciales que actualmente se están utilizando en sus proyectos en los que intervienen este tipo de dispositivos.

Este lector forma parte de un proyecto mucho mayor que consiste en un sistema completo de control de acceso para las zonas restringidas de la Escuela. Una vez terminado el lector, si satisface todas las condiciones que requiere este sistema de control, sustituirá al lector comercial que se encuentra instalado actualmente. Es por esta razón que debe ser capaz de realizar las mismas tareas que uno comercial pero debe tener un diseño modular de forma que si alguno de los chips que lo componen se dejara de fabricar este podría ser sustituido por otro de características similares sin alterar el resultado final.

A su vez y como complemento a este proyecto, se presenta la oportunidad de comenzar a trabajar con una nueva herramienta adquirida por el HCTLab, un nuevo software para el diseño de circuitos impresos llamado *Altium Designer*. Debido a que este software no ha sido empleado con anterioridad en el laboratorio, su aprendizaje y utilización suponen un aliciente en la realización del proyecto.

Todas estas variantes hacen de este proyecto un reto muy interesante y atractivo para poner punto y final a todos los años de entrega, estudio y sacrificio dedicados a la realización de la carrera de Ingeniería de Telecomunicación.

## 1.2 Objetivos

El principal objetivo de este proyecto fin de carrera es el diseño, fabricación y programación de un lector de tarjetas-chip con acceso USB (*Universal Serial Bus*) que permita el envío de datos desde el lector al sistema anfitrión por medio de dicha conexión USB.

Uno de los principales requisitos de este proyecto es que el lector de tarjetas que se va a diseñar sea modular, esto es, que se diseñe a partir de componentes independientes, sin importar fabricante, tipo de encapsulados… Esta idea conduce a un prototipo final que, en caso de que una parte de los componentes que lo forman se dejaran de fabricar, estos podrían ser

reemplazaos por otros similares sin que el resultado final cambiase de manera significativa.

La persecución de esta especificación concreta lleva a una búsqueda de componentes de distintos tipos como chips lectores, cristales, conectores… de distintos fabricantes de los cuales se seleccionan los que mejor se ajustan a las necesidades del proyecto. El microcontrolador a emplear viene impuesto por las especificaciones del proyecto, por lo que no se busca ninguna alternativa.

Ya que se pretende crear un lector lo más reducido y sencillo posible, se busca que los distintos componentes posean una característica común: componentes de montaje superficial SMD (*Surface-Mount Device*). Esto es debido a que su tamaño es mucho más reducido y, de esta forma, se eliminan los taladros necesarios para el montaje de componentes tipo *through-hole*.

La realización de este proyecto conlleva la utilización de diversos equipos de laboratorio como osciloscopio, estación de soldadura, analizador lógico, fuentes de alimentación… además del estudio de varios estándares de transmisión de datos como RS-232, $I^2C$ (*Inter-Integrated Circuit*)… conocimientos respecto a microcontroladores en cuanto a su funcionamiento y programación, y de los distintos accesorios que puedan ser necesarios en esta tarea como programadores externos, software especifico…

Como objetivo añadido de este proyecto se propone el estudio y aprendizaje del uso de una nueva herramienta para el diseño y fabricación de circuitos impresos: *Altium Designer*. La parte de la memoria donde se describe el proceso de diseño hardware del lector se explica de modo que puede tomarse como una guía de uso de este software.

## 1.3 Organización de la memoria

La documentación de este proyecto se estructura de la siguiente manera:

El capitulo primero hace una pequeña introducción sobre el tema que nos ocupará durante la ejecución de este proyecto. Además se añaden los objetivos que se persiguen, así como un vistazo general sobre todo lo que puede aportar al estudiante la realización de este proyecto.

El capitulo segundo abarca el estado del arte. Incluye un repaso histórico, se describe la estructura y funcionamiento básico de las tarjetas, se listan las distintas clasificaciones en las que se agrupan las tarjetas inteligentes, se examinan los distintos tipos de lectores existentes, su funcionamiento y se presentan algunos ejemplos de los lectores comerciales disponibles actualmente y, por último, se realiza un repaso a toda la normativa que debe tenerse en cuenta a la hora de realizar este proyecto.

El capitulo tercero engloba todo el estudio de tecnologías, estándares de comunicación y componentes disponibles para concluir presentando una solución para el proyecto. Como su nombre indica, se trata del proceso de diseño del lector de tarjetas.

El capitulo cuarto describe con detalle el proceso de desarrollo del lector, desde la creación del esquemático del circuito impreso que dará lugar al lector, pasando por su rutado y construcción, hasta su ensamblaje y programación.

El capitulo quinto presenta los resultados obtenidos y las conclusiones alcanzadas durante todo el desarrollo del proyecto. Así mismo, se ofrece una serie de ideas para los futuros trabajos y ampliaciones que se pueden efectuar sobre el prototipo obtenido como resultado de este proyecto.

# 2 Estado del arte

## 2.1 Historia.

El desarrollo de las tarjetas inteligentes tiene su origen en la unión de dos aportaciones básicas. Por un lado, en 1968, los inventores alemanes Jurgen Dethloff y Helmut Grotrupp desarrollan y patentan la idea de incorporar circuitos integrados a las tarjetas plásticas de identificación. Por otro lado, en 1970, el Dr. Kunitaka Arimura de Japón consigue unir en una sola pieza de silicio el almacenamiento de datos y la lógica aritmética, registrando su trabajo como la única patente en la que aparece el concepto de tarjeta inteligente.

Con la aparición del primer ordenador completo en un solo chip en 1971 de Intel, se produce el gran avance y, en 1974, el inventor francés Roland Moreno presenta la primera tarjeta plástica con circuito integrado denominándola “Sistema de Transferencia de Datos”. Posteriormente será rebautizada con el nombre de tarjeta inteligente. La patente creada por Roland Moreno incluye tanto la tarjeta creada con un dispositivo para operar con ella, el primer lector de tarjetas.

Continuando la evolución, en 1979 aparece la primera tarjeta con microprocesador operativa. Incorpora 1kByte de memoria programable y un microprocesador 6805 de Motorola. Es considerada la primera tarjeta inteligente debido a que combina el poder del microprocesador y la memoria programable, siendo capaz de tomar sus propias decisiones basándose en las necesidades del usuario para modificar, añadir, modificar y eliminar datos guardados. Este diseño integraba el microprocesador y la memoria en chips diferentes lo que ocasionó graves problemas de seguridad.

Los avances tecnológicos en los años 80 permiten que se puedan unir en un mismo chip un microprocesador y una memoria, lo que significa el lanzamiento definitivo de las tarjetas inteligentes.

En 1983 aparece en Europa la tarjeta telefónica empleada en los teléfonos públicos. Un año después tiene lugar el hito que marca un antes y un después, cuando la industria bancaria francesa decide crear una tarjeta inteligente estándar como tarjeta de crédito y débito para sus clientes. Esto ocasiona que se distribuyan más de 16 millones de tarjetas inteligentes. Acto seguido esta tecnología se comienza a distribuir también por Estados Unidos, aunque no es acogida con gran entusiasmo.

Las tarjetas inteligentes continúan ampliando su campo de uso y mejorando la tecnología que implementan durante los años siguientes, pero realmente experimentan su gran expansión con la aparición de la tarjeta SIM para teléfonos móviles GSM (*Global System for Mobile Communication*) en Europa en 1995.

A partir de este momento, las tarjetas inteligentes se distribuyen a todos los sectores del mercado, haciendo de su uso algo habitual e indispensable.

## 2.2 Tarjetas inteligentes.

Una tarjeta inteligente se podría describir, de una manera sencilla, como un chip encapsulado en un rectángulo de PVC (*PolyVinyl Chloride*) con unos contactos exteriores que permiten la comunicación de la tarjeta con los dispositivos lectores. Esta somera descripción permite hacerse una idea de las características básicas que poseen las tarjetas inteligentes.


Figura 2- 1: Composición de una tarjeta inteligente con contactos.

En este apartado se enumeran los distintos tipos de tarjetas inteligentes que existen, la estructura interna que normalmente poseen y qué sistemas de archivos y seguridad implementan.

### 2.2.1 Clasificación de las tarjetas inteligentes.

Las tarjetas inteligentes se pueden clasificar según cuatro características básicas. A continuación se describen los grupos y los rasgos característicos que dan origen a estas clasificaciones:

Según las capacidades del chip se pueden clasificar como:

- Tarjetas de memoria: tarjetas que únicamente son capaces de almacenar datos. Por tanto, no ejecutan ni albergan aplicaciones ejecutables.
- Tarjetas microprocesadas: tarjetas con una estructura muy similar a la de un ordenador, con CPU, memorias y sistema operativo completo. Son capaces de almacenar datos y ejecutar aplicaciones.
- Tarjetas criptográficas: evolución de las tarjetas microprocesadas en las que se incorporan módulos hardware para la ejecución de algoritmos de cifrado y firma digital.

Según la estructura del sistema operativo se clasifican como:

- Tarjetas de memoria: disponen de un sistema operativo muy limitado con una serie de comandos básicos de lectura y escritura de las distintas secciones de memoria.
- Basadas en sistemas de ficheros, aplicaciones y comandos: Estas tarjetas disponen de un sistema de ficheros acorde al estándar ISO/IEC 7816 parte 4 y un sistema operativo que incorpora una o más aplicaciones que incluyen una serie de comandos que se pueden invocar a través de APIs (*Application Programming Interface*) de programación.

- Java *Cards*: tarjetas donde el sistema operativo es una pequeña máquina virtual Java. Pueden ejecutar pequeñas aplicaciones desarrolladas específicamente para este entorno.

Según el tamaño de la tarjeta existen:

- ID-000: tarjetas SIM.
- ID-00: tamaño poco común.
- ID-1: tamaño estándar tarjeta de crédito.

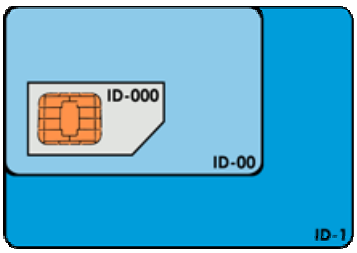



Figura 2- 2: Tamaños estándar para tarjetas inteligentes.

Por último, se pueden clasificar según su interfaz:

- Con contactos: dispone de 8 contactos metálicos accesibles por los que se realiza la comunicación entre dispositivo lector y tarjeta inteligente. Se encuentran estandarizadas en la ISO 7816.

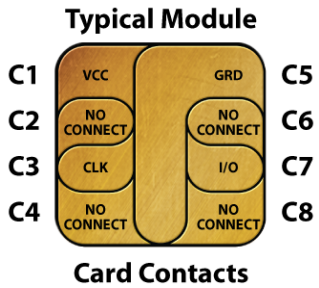



Figura 2- 3: Esquema de los contactos de una tarjeta chip.

- Sin contactos: se comunica con el sistema lector mediante radiofrecuencia ya que incorporan etiquetas RFID (*Radio-Frequency Identification*). Facilita su uso ya que no es necesario introducirla en

el lector, y alarga la vida útil de la tarjeta al no sufrir estrés mecánico debido a su empleo. Se encuentran estandarizadas en la ISO 14443.

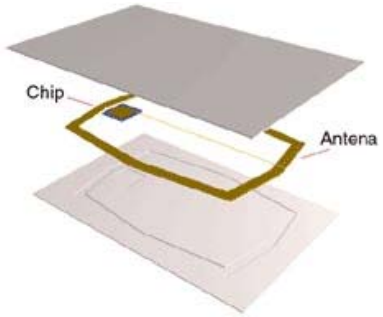



Figura 2- 4: Composición de una tarjeta inteligente tipo RFID.

- Hibridas: tarjetas sin contactos a las que se añade un segundo chip con contactos. Cada uno de los chips puede ser de cualquiera de las categorías antes mencionadas.
- Duales: iguales a las tarjetas hibridas con la salvedad de que solo portan un circuito integrado que realiza las funciones de chip con y sin contactos.

### 2.2.2 Estructura de la tarjeta microprocesada.

El chip de una tarjeta inteligente microprocesada está compuesto generalmente por una CPU, una memoria ROM (*Read-Only Memory*) donde se encuentra el sistema operativo, las instrucciones del protocolo de comunicación y los algoritmos de seguridad, una memoria RAM (*Random-Access Memory*) y una memoria EEPROM (*Electrically Erasable Programmable Read-Only Memory*) utilizada para el almacenamiento de datos y aplicaciones. En ocasiones puede contener algún modulo hardware destinado a operaciones criptográficas.

Todos estos elementos se comunican mediante un bus interno al que no se puede acceder desde el exterior, lo que hace imposible introducir comandos falsos o códigos maliciosos que produzcan un error en la seguridad. Este aspecto refuerza la seguridad global de las tarjetas inteligentes.

Respecto al sistema de archivos se puede decir que se encuentra dividido en tres zonas: la zona abierta, donde se encuentran los datos de usuario y aquella información no confidencial, la zona protegida por claves, donde la información se encuentra encriptada por distintas claves y la zona protegida por el PIN (*Personal Identification Number*), que contiene todos aquellos ficheros que necesitan el código PIN para poder acceder a ellos. Este código es solicitado al usuario cuando se intenta acceder a esta información.

La organización jerárquica del árbol de directorios consiste en un directorio raíz (MF o *Master File*) del que cuelgan distintos directorios (DF o *Dedicated File*) o fichero (EF o *Elementary File*).

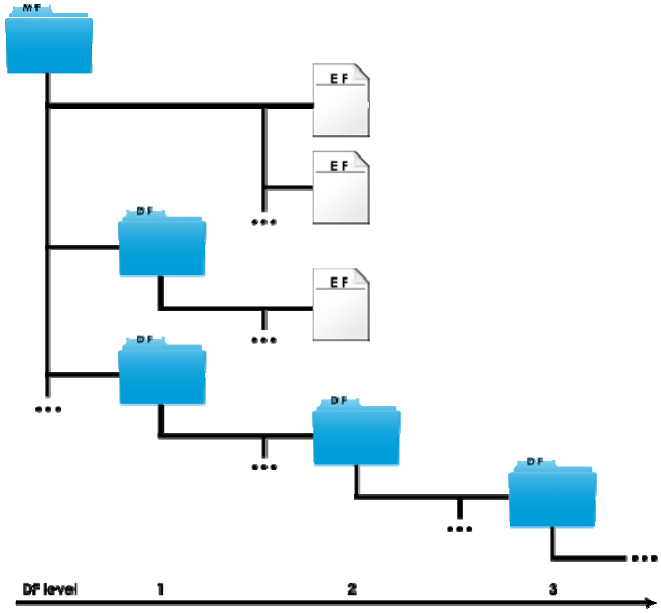



Figura 2- 5: Esquema del árbol de directorios de una tarjeta microprocesada.

Para acceder a los distintos ficheros que contienen la información deseada se debe recorrer el árbol de directorios. Para ello se deben intercambiar comandos con la tarjeta solicitando el acceso a cada uno de los directorios y, una vez llegados al fichero buscado, se solicita la extracción de los datos que contiene. Los comandos utilizados son en forma de APDU (*Aplication Protocol Data Unit*) que es la unidad de comunicación entre un lector de tarjetas inteligente y la propia tarjeta. Su forma y detalles se verán más adelante en esta memoria.

## 2.3 Lectores de tarjetas inteligentes.

Un lector de tarjetas es un dispositivo intermedio entre la tarjeta inteligente y el sistema que interactúa con ella. Permite la lectura y escritura en las tarjetas inteligentes y, como consecuencia de la gran expansión que están experimentando las tarjetas en todos los sectores, cada vez resultan más útiles e imprescindibles este tipo de dispositivos.

Existen distintos tipos de lectores de tarjetas dependiendo de sus características principales y de su capacidad operativa. A continuación se clasifican según diversas características:

Según su capacidad operativa:

- Solo lectores: son dispositivos que solo son capaces de leer datos de una tarjeta. Mantiene un proceso de comunicación que termina con una extracción de datos.
- Lectores/grabadores: además de leer datos también son capaces de grabarlos en la memoria de la tarjeta inteligente. Tienen un precio superior a los anteriores pero permiten un mayor rango de operaciones con las tarjetas.

Según su conexión con el sistema anfitrión:

- Integrados o internos: son lectores diseñados para ser instalados de forma permanente dentro del sistema que hará uso de ellos. El sistema puede ser un cajero automático, un torno de acceso…
- Externos: se trata de lectores portátiles. Son fáciles de transportar dado su reducido tamaño y son más económicos que los lectores fijos. Su conexión con el sistema anfitrión suele ser bien por USB o bien por medio de una interfaz PCMCIA (*Personal Computer Memory Card International Association*).

Según su compatibilidad con las tarjetas:

- Específicos: son lectores específicos para trabajar con un solo tipo de tarjeta. Normalmente son para uso doméstico o situaciones en las que todos los usuarios posean el mismo tipo de tarjeta. Son lectores asequibles y sencillos de usar.
- Multi-tarjeta: lectores capaces de operar con tarjetas que poseen distintas tecnologías. Las más habituales son tarjetas con contactos, sin contactos o RFID y tarjetas de banda magnética. Son lectores diseñados para entornos empresariales o comerciales. Debido a su gran capacidad operativa, son lectores más complejos y caros que los anteriores.

Estas son las tres grandes características que se deben tener en cuenta a la hora de diseñar o adquirir un lector de tarjetas inteligentes. Tomando una de las opciones de cada uno de los grupos se puede obtener el lector acorde con las necesidades del proyecto que lo requiere.

### 2.3.1 Lectores de tarjetas comerciales.

A continuación se enumeran algunos ejemplos de lectores de tarjetas chip que se pueden encontrar en el mercado. Estos lectores combinan alguna de las características descritas con anterioridad y se suministran junto con los *drivers* y software necesario para hacerlos funcionar correctamente.

A continuación se muestran algunos de los modelos disponibles de la compañía C3PO S.A. [14], de los cuales se comentarán sus características principales, extraídas directamente de las hojas de datos de los dispositivos, y precios de venta al publico vía *online*, que no incluyen ni impuestos ni gastos de envío.

- **LTC31 USB:**

Figura 2- 6: Lector LTC31 USB.

*El lector LTC31 USB, es el más popular y económico de su gama. Diseñado especialmente para su utilización en entornos de firma electrónica, identificación y autenticación. Es totalmente compatible con el DNI electrónico, la tarjeta criptográfica CERES-FNMT y las tarjetas chip que cumplan con el estándar ISO 7816 (1,2,3 y 4).*

Precio: 12.89 €

- **LTC36 PRO USB externo:**

Figura 2- 7: Lector LTC36 PRO USB externo.

*La línea LTC36 PRO de lectores de tarjeta inteligente está diseñada especialmente para su uso en entornos de firma electrónica, identificación y autenticación, siendo totalmente compatibles con el DNI electrónico y las tarjetas chip más utilizadas actualmente en el mercado. Ofrecen grandes prestaciones y facilidad de uso a un coste reducido. Incorporan un contactor de tarjetas de larga duración que permite más de 150.000 inserciones, así como un microprocesador de gran capacidad de memoria de programa que ofrece la posibilidad de ampliar las funcionalidades de los dispositivos.*

Precio: 18.97 €

- **LTC36 PRO USB interno:**

Figura 2- 8: Lector LTC36 PRO USB interno.

*Los lectores LTC36 PRO componen la línea profesional de C3PO, S.A.. Están pensados para un uso intensivo sin dañar la tarjeta y para ofrecer funcionalidades adicionales según las necesidades de instituciones, organismos y empresas. El lector LTC36 PRO USB interno, está diseñado para ser integrado en PCs, kioskos, máquinas expendedoras,...*

Precio: 20.37 €

- **Teclado KBR36 USB Negro:**

Figura 2- 9: Lector con teclado KBR36 USB.

*Teclado USB con lector de tarjeta chip y DNIe integrado. Diseñado especialmente para su uso en entornos de firma electrónica, identificación y autenticación, siendo totalmente compatible con el DNI electrónico y las tarjetas chip más utilizadas actualmente en el mercado. Incorpora un contactor de tarjetas de larga duración que permite más de 150.000 inserciones, así como un microprocesador de gran capacidad de memoria de programa que ofrece la posibilidad de ampliar las funcionalidades de los dispositivos.*

Precio: 25.00 €

- **Express Card 4321:**

Figura 2- 10: Lector Express Card 4321.

*Dispositivo lector y grabador de tarjeta chip integrado en la bahía Express Card de los ordenadores portátiles. Diseñado especialmente para su utilización en entornos de certificación con firma digital.*

Precio: 41.00 €

- **PCMCIA 4040:**


Figura 2- 11: Lector PCMCIA 4040.

*Dispositivo lector y grabador de tarjeta chip que se integra completamente en la bahía PCMCIA de los ordenadores portátiles. Diseñado especialmente para su utilización en entornos de certificación con firma digital.*

Precio: 35.99 €

Ahora de la compañía Bit4it [13] se presentan los modelos disponibles en su tienda online. Se da una pequeña descripción por parte del fabricante y se indican sus precios de venta que no incluyen ni impuestos ni gastos de envío.

- **miniLector EVO:**

Figura 2- 12: miniLector EVO.

*Lector y grabador de tarjeta inteligente (DNIe), de escritorio, diseñado para cumplir con los requisitos técnicos y funcionales más exigentes en materia de firma electrónica.*

Precio: 12,99 €

- **ID-ONE:**

Figura 2- 13: Lector ID-ONE.

*Lector de DNI electrónico de bolsillo con 2GB de memoria integrada. Compatible Windows, Linux y Mac OS X.*

Precio: 39,99 €

- **miniLector KEYBOARD:**

Figura 2- 14: miniLector KEYBOARD.

*Teclado con lector de tarjeta inteligente (DNIe) integrado, que cumple con los requisitos técnicos y funcionales más exigentes en materia de firma electrónica.*

Precio: 28,99 €

- **miniLector BAY:**

Figura 2- 15: nimiLector BAY.

*Lector de tarjeta inteligente (DNIe) diseñado para una perfecta integración en bahías de 3,5" de ordenadores o quioscos de internet. Permite la conexión directa al puerto USB de la placa base del ordenador o mediante un cable USB externo.*

Precio: 17,99 €

- **miniLector PCMCIA:**

Figura 2- 16: miniLector PCMCIA.

*Lector de DNIe para puerto PCMCIA tipo II. Diseñado para ordenadores portátiles.*

Precio: 7,99 Euros

- **miniLector XPRESS:**

Figura 2- 17: miniLector XPRESS.

*Lector de DNIe para puerto XPRESS Card. Diseñado para ordenadores portátiles.*

Precio: 43,99 €

## 2.4 Normativa: ISO/IEC 7816.

La norma ISO/IEC 7816 [20] define los estándares para la fabricación y uso de las tarjetas inteligentes. Está compuesta por 15 apartados que tratan cada uno de los aspectos a tener en cuanta a la hora de diseñar, fabricar u operar con esta tecnología.

A continuación se ha realizado un breve resumen de las distintas partes que componen esta norma, prestando especial atención a las 4 primeras ya que describen los aspectos de mayor interés para este proyecto.

**ISO 7816-1: Características físicas.**

En este apartado se describen las características físicas de las tarjetas inteligentes con contactos. Se analizan los valores límite de exposición a fenómenos electromagnéticos y de temperatura. Además se definen las pruebas de esfuerzo mecánico que deben superar las tarjetas para cumplir con la normativa.

**ISO 7816-2: Tamaño y localización de los contactores.**

En este punto se define la dimensión y ubicación de los contactores en la tarjeta de PVC. También se describe el número de contactos que deben existir, así como su función y posición.

**ISO 7816-3: Señales electrónicas y protocolos de transmisión.**

Potencia, forma de señal e intercambio de información entre una tarjeta inteligente y un sistema lector. Incluye los siguientes subapartados: ISO 7816-3.1 Valores de corriente y tensión, ISO 7816-3.2 Procedimiento operativo para tarjetas con circuitos integrados, ISO 7816-3.3 Respuesta a un reset o ATR (*Answer to Reset*), ISO7816-3.4 Selección de tipo de protocolo (PTS, *Protocol Type Selection*), ISO 7816-3.5 Tipo de protocolo T=0, protocolo de transmisión de caracteres asíncrono *half-duplex*.

**ISO 7816-4: Organización, seguridad y comandos para el intercambio de información.**

Contenido de los mensajes intercambiados entre tarjeta inteligente y dispositivo lector, así como los comandos, la estructura del sistema de archivos y los datos que albergan, métodos de acceso a los datos y métodos de seguridad.

**ISO 7816-5: Registro de la solicitud de los proveedores.**

Modo de utilización de un identificador de aplicación para determinar la presencia y/o realizar la recuperación de una aplicación en una tarjeta.

**ISO 7816-6: Interoperabilidad en los elementos de datos para el intercambio**

Elementos de datos (DEs) utilizados para el intercambio interindustrial basado en tarjetas de circuitos integrados (ICC) con contactos y sin contactos. Se proporciona el identificador, nombre, descripción, formato, la codificación y la disposición de cada DE y define los medios de recuperación de las DE de la tarjeta.

**ISO 7816-7: Interoperabilidad en los comandos de la tarjeta (SCQL).**

Método seguro de base de datos relacional para tarjetas inteligentes basadas en interfaces SQL (SCQL).

**ISO 7816-8: Comandos para operaciones de seguridad.**

Comandos para tarjetas de circuitos integrados, ya sean con contactos o sin contactos, que se pueden utilizar para operaciones criptográficas. Estos comandos son complementarios y se basan en los comandos descritos es el apartado ISO 7816-4.

**ISO 7816-9: Comandos para la gestión de la tarjeta.**

Comandos para tarjetas de circuitos integrados, con contactos y sin contactos, para la gestión de archivos. Estos comandos abarcan todo el ciclo de vida completo de la tarjeta y, por lo tanto, algunos comandos pueden ser utilizados antes de que la tarjeta haya sido expedida a su titular o después de la tarjeta haya caducado.

**ISO 7816-10: Señales electrónicas para operación síncrona.**

Métodos utilizados por las tarjetas de memoria para aplicaciones tales como tarjetas telefónicas prepago o máquinas expendedoras.

**ISO 7816-11: Verificación de la identidad personal a través de métodos biométricos.**

Uso de los comandos y objetos de datos relacionados con la verificación personal a través de métodos biométricos en tarjetas inteligentes. Los comandos utilizados se definen en la norma ISO 7816-4. Los objetos de datos están parcialmente definidos en la parte importada de la norma ISO / IEC 19785-1.

**ISO 7816-12 Tarjetas con contactos. Interfaz eléctrica USB y procedimientos operativos.**

Condiciones de funcionamiento de una tarjeta inteligente que proporciona una interfaz USB.

**ISO 7816-13: Comandos de administración de aplicaciones en múltiples aplicaciones entorno.**

Comandos para la gestión de aplicaciones en un entorno multi-aplicación.

**ISO 7816-15: Aplicación de información criptográfica.**

Aplicación que contiene información sobre la funcionalidad criptográfica. Por otra parte, se define una sintaxis común (en ASN.1) y formato de la información criptográfica y mecanismos para compartir esta información cuando proceda.

Este resumen únicamente recoge la normativa que se aplica directamente sobre las tarjetas inteligentes y dispositivos estándar que operan con ellas. Para aplicaciones específicas sobre tarjetas inteligentes existen normas que se deben tener en cuenta como la EMV (*Europay MasterCard VISA*) para trabajar con sistema de pago o GSM para trabajos basados en tarjetas SIM.

# 3 Diseño del prototipo

## 3.1 Introducción

La idea inicial para el diseño del lector de tarjetas se puede resumir perfectamente mediante el siguiente diagrama:

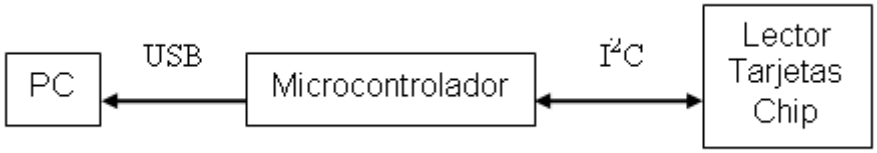



Figura 3- 1: Diagrama del diseño básico para el lector de tarjetas.

Se trata de un microcontrolador como parte central que, por un lado envía y extrae información de las tarjetas inteligentes mediante un chip especializado en ese apartado y, por otro lado, envía la información obtenida al PC (*Personal Computer*) al que se encuentra conectado.

Otra posible configuración sería aquella que aunara en un solo chip microcontrolador con acceso USB y lector de tarjetas, de forma que con un solo chip se resuelve todo el problema. Esta idea se implementa en un primer diseño, pero es descartada por las limitaciones que implica este modelo, además de que elimina una de las premisas iniciales: la modularidad.

Partiendo de esta idea se buscan en el mercado los elementos que mejor se adaptan al diseño, por sus características principales así como por otras propiedades que, aunque no sean críticas, sí son deseables, como puede ser su encapsulado, su consumo… Además de todas estas características nunca se debe olvidar un punto importante: que sea fácilmente adquirible y lo mas económico posible, ya se que trata de diseñar un lector sencillo y económico.

Seguidamente se describen las características principales exigidas a cada uno de los dispositivos que conforman el lector:

- Microcontrolador: se requiere un chip con conexión USB, que implemente los protocolos $I^2C$ y SPI (*Serial Peripheral Interface*) de comunicación, con puertos de entrada/salida estándar, pines de interrupción externa y una velocidad de operación aceptable que no ralentice el funcionamiento del conjunto, sin dejar de lado características como número de pines, tipo de encapsulado (lo mas sencillo posible para su instalación) y la tensión de alimentación para su funcionamiento a plena capacidad.

- Lector de Tarjetas Chip: se busca un chip lo más sencillo posible que sea capaz de operar con todas las tarjetas chip del mercado, es decir, que opere con los tres voltajes estándar (5v, 3v y 1,8v), que cumpla con todos los parámetros de la ISO 7816 y que sea capaz de comunicarse por $I^2C$. También es deseable que su rango de tensiones de alimentación coincida al menos en parte con las posibles tensiones de alimentación del microcontrolador.

## 3.2 Búsqueda de componentes.

En este apartado se describen los diferentes chips disponibles en el mercado que encajan en mayor o menos medida con las especificaciones básicas buscadas para el diseño del lector de tarjetas. Algunos se desechan por no disponer de alguna de las funcionalidades requeridas para el diseño y otros, aunque sus características son suficientes, se desechan por no encontrarse disponibles en los suministradores habituales o porque su precio está fuera de lugar en comparación con chips similares.

A continuación se muestra un resumen de los chips estudiados junto con un extracto de sus principales características.

- De la empresa Atmel se tiene el chip lector de tarjetas AT83C24 [4]. Se trata de un chip que encaja sobradamente con el diseño ideado en cuanto a sus principales características, dado que es capaz de generar los tres voltajes estándar para la lecturas de tarjetas, tiene capacidad para la programación de tarjetas, implementa el protocolo TWI para la comunicación, bajo consumo y dispone de pines de interrupción.

- Del fabricante Maxim IC se encuentran dos chips interesantes. Por un lado, se tiene un chip lector de tarjetas. Se trata del modelo DS8113 [25], un chip simple solo lector de tarjetas con capacidad para generar los tres voltajes estándares de lectura, 28 pines y una estructura sencilla, pero que no dispone de protocolos de comunicación suficientes para permitir cierta flexibilidad a la hora de diseñar el modelo final ya que solo dispone de una línea serie para la transmisión de datos con el controlador. Por otro lado, se encuentra un dispositivo de los denominados *System-on-Chip*, es decir, un chip que engloba microcontrolador y lector de tarjetas. En este caso, el modelo es el 73S1215F [24]. Incluye un microcontrolador de 8 bits muy completo que, además de la capacidad de lectura de tarjetas, también posee múltiples opciones de comunicación con otros dispositivos por medio de USB, $I^2C$ y una interfaz *full-duplex* serie. Posee una frecuencia de operación de hasta 25 MHz, memoria para el almacenamiento de programas y datos, gestor de interrupciones, *timers*, bajo consumo, etc. Se trata de un chip muy completo pero con un gran inconveniente: para su desarrollo y máxima utilización es necesario disponer de ciertos elementos como programadores y software especifico exclusivos de esta marca comercial.

- De la marca STMicroelectronics se encuentra de nuevo las opciones de chip lector de tarjetas y chip todo en uno. El modelo ST8024 [42] es un chip lector de tarjetas muy simple y con unas prestaciones muy limitadas, ya que tan solo tiene capacidad de lectura sobre tarjetas de 3 V y 5 V. Está especialmente indicado para tarjetas de banca y tarjetas de pago por visión para televisión. En cuanto a los chips todo en uno

existen dos de modelos de la familia ST7, el ST7GEME4 [40] y el ST7SCR1E4 [41]. Se componen de un microcontrolador de 8 bits con USB y RS-232 como protocolos de comunicación, puertos I/O (*Input/Output*) estándares, control de interrupciones y completa capacidad de lectura de tarjetas.

- Por último, del fabricante NXP se dispone de toda la familia TDA80XX (TDA8007BHL, TDA8020HL, TDA8023TT, TDA8024, TDA8024AT, TDA8024T, TDA8024TT, TDA8025HN, TDA8026ET, TDA8029HL, TDA8034, TDA8034AT, TDA8034HN, TDA8034T, TDA8035HN) [32]. Se trata de una familia de chips lectores de tarjetas con diferentes características. Entre las diferencias se pueden encontrar chips que solo disponen de parte de los voltajes estándares de lectura o de todos ellos, distintos protocolos de comunicación para la transmisión de datos con el microcontrolador, encapsulados, velocidades de funcionamiento, etc. Estudiando las hojas de características de los chips que componen la familia se puede encontrar uno que se ajuste a las características deseadas para un diseño en particular.

En cuanto a los microcontroladores, no se realiza ninguna búsqueda dado que se dispone de un modelo de microcontrolador en el laboratorio con el que se desea comenzar a trabajar en varios proyectos y esta parece una buena oportunidad para empezar a conocer las capacidades y posibilidades que ofrece este chip.

## 3.3 Selección de componentes

En esta sección se describen las principales características de los componentes seleccionados para formar parte del lector de tarjetas que se esta diseñando.

### 3.3.1 Microcontrolador:

El chip disponible es el ATmega32U4 [5], un chip de la marca comercial Atmel que cumple con todas las exigencias previas y que además incluye características extras que, en determinados momentos, pueden resultar muy útiles e interesantes. A continuación se analizan punto por punto las características imprescindibles:

- Microcontrolador de 8 bits de bajo consumo con 32 kBytes de memoria flash ISP (*In-System Self-Programmable*) y 2,5 kBytes de memoria SRAM (*Static Random Access Memory*).
- Voltaje de alimentación ente 2,7 V y 5,5 V.
- Frecuencia de reloj de hasta 16 MHz con cristal externo y alimentación a 5 V u 8 MHz con solo reloj interno.
- USB 2.0 completo.
- Interfaz SPI maestro/esclavo
- Interfaz TWI (*Two Wire Interface*).
- 26 líneas programables de entrada/salida.
- Encapsulado TQFP (*Thin Quad Flat Package*), 44 pines (10x10 mm).
- Gestor de interrupciones tanto internas como externas.
- Watchdog Timer programable.

A continuación se detallan las características complementarias:

- 12 canales de 10 bits para el conversor analógico/digital.
- USART *(universal synchronous/asynchronous receiver/transmitter*) serie programable.
- Interfaz JTAG (*Joint Test Action Group*).

La programación del microcontrolador se realiza mediante lenguaje C y un software de apoyo que proporciona el propio fabricante del chip. Se trata de "*AVR Studio 5.0*" [8]. Este software compila el código C y genera una serie de archivos, entre los que se encuentra el archivo ".HEX" que es el encargado de portar el programa de usuario listo para el microcontrolador.

También se debe destacar de este chip su versatilidad a la hora de ser programado. La programación se puede llevar a cabo de tres formas distintas:

- USB: mediante un software específico gratuito que se puede descargar de la página Web de Atmel llamado “Flip” [9] se pueden programar las memorias del chip con tan solo conectarlo al PC por USB e instalar los drivers que acompañan al programa. Este software no permite modificar los *fuses* ni bits de seguridad del microcontrolador.

- ISP (*In-System Programming*): con la ayuda de un pequeño programador y mediante el protocolo de comunicación SPI se puede programar tanto las memorias del chip con sus *fuses* y bits de seguridad.

- JTAG: también permite su programación mediante este sistema que además aporta respecto del anterior la posibilidad de usar un *debugger* en línea.

Es importante indicar que, como es previsible, todas las funcionalidades del chip no pueden estar accesibles en todo momento, es decir, existe una multiplexación de los pines que conectan cada modulo del chip con el exterior:

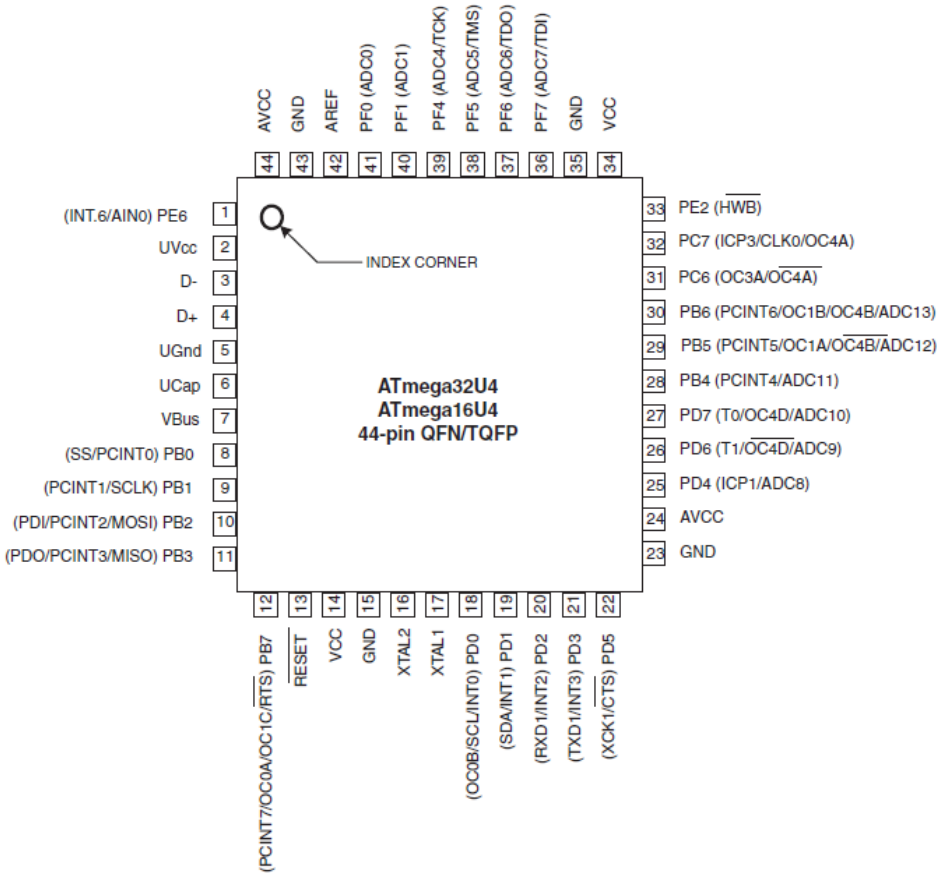



Figura 3- 2: *Pinout* ATmega32U4

Para este proyecto en particular no supone un problema ya que todas las funcionalidades necesarias se encuentran en pines separados, por lo que no es necesario añadir señales control sobre esta multiplexación de pines.

Todas estas características hacen que este microcontrolador tenga capacidad más que suficiente para ser el centro neurálgico del lector de tarjetas que se describe en este proyecto.

### 3.3.2 Chip lector de tarjetas:

El chip escogido para esta función es el TDA8029HL [31] del fabricante NXP Semiconductors (anteriormente Philips Semiconductors). Es un chip muy simple pero con grandes prestaciones y que cumple sobradamente con las exigencias previas. Estas son sus características con algo más de detalle:

- Bajo consumo, bajo coste y eficiente lector de tarjetas chip.

- Chip controlado mediante un núcleo 80C51 con 16 kBytes de ROM, 256 bytes de RAM y 512 bytes de XRAM.
- Implementación y funcionamiento según ISO7816.
- Capacidad para generar voltajes para lectura de tarjetas de 5V, 3V y 1,8V con una corriente máxima de 65 mA, 50 mA y 30 mA respectivamente.
- Genera señal de reloj para lectura de tarjetas de hasta 20 MHz.
- Soporta tarjetas sincronías que no utilicen los pines C4 y C8.
- Modo ahorro de energía.
- Preprogramado para una fácil integración.
- Comunicación con controlador principal mediante RS-232 e $I^2C$.
- Dispone de una entrada de interrupción y 4 pines entrada/salida.
- Encapsulado LQFP (*Low-profile Quad Flat Package*) de 32 pines (7x7 mm).
- Voltaje de alimentación entre 2,7V a 6V.

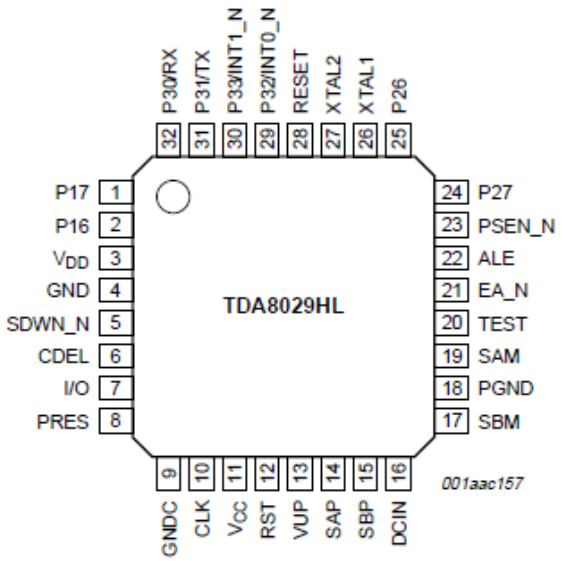



Figura 3- 3: *Pinout* TDA8029HL

Resulta interesante comentar que, aunque sí es capaz de comunicarse mediante $I^2C$ no lo tiene implementado como tal, sino que está simulado por el software que lleva instalado de fábrica. Para hacerlo funcionar se deben dar una serie de pasos y conectar ciertos pines de determinada manera para que el software interno reconozca la situación y active la comunicación por $I^2C$.

Esta situación no supone un problema grave pero si un pequeño inconveniente ya que limita bastante la velocidad a la que puede recibir/enviar datos mediante este protocolo de comunicación. Según el fabricante, esta velocidad estaría limitada a un máximo de 60 kHz en la línea de reloj, aunque pruebas posteriores demuestran que se debe bajar esa velocidad si se quiere una comunicación sin errores.

Por lo demás cumple con las especificaciones. Es un chip muy sencillo de instalar, con un funcionamiento simple y que encaja perfectamente con el microcontrolador elegido tanto en protocolos de comunicación, ambos tienen RS-232 e $I^2C$, como en los rangos de las tensiones de alimentación ya que ambos funcionan a pleno rendimiento a 5V.

## 3.4 Protocolos de comunicación

En este apartado se analizan los protocolos de comunicación que serán necesarios en el desarrollo del proyecto. Se estudiarán el bus $I^2C$ con su protocolo de comunicación asociado, el protocolo ALPAR para el encapsulado de datos de manera que sean comprendidos por el chip TDA8029 y el protocolo APDU como encapsulado de datos para el acceso a la tarjeta inteligente desde el chip lector.

### 3.4.1 Bus $I^2C$

Se trata de un bus de comunicaciones bidireccional, serie y síncrono de 2 hilos desarrollado por Philips Semiconductors (ahora NXP Semiconductors) en 1982 con la idea de simplificar lo máximo posible la comunicación entre componentes de una misma placa base. El estándar de facto $I^2C$ es conocido como *Inter-Integrated Circuit*, Inter-IC o $I^2C$ –Bus [29][19][33].

Algunos fabricantes emplean otro bus que en esencia es $I^2C$ pero con alguna pequeña particularidad, como por ejemplo el caso de Atmel y su TWI.

Este bus es compatible con $I^2C$ casi en su totalidad y las herramientas y modo de utilización son iguales en ambos casos.

El bus $I^2C$ consta de 2 hilos, la línea que trasporta los datos SDA (*Serial Data Line*) y la línea de reloj SCL (*Serial Clock Line*) que aporta el sincronismo necesario para la comunicación entre componentes. En un inicio no se contemplaba la utilización de este bus para comunicaciones entre distintas placas base, pero dada su simplicidad y buen funcionamiento fue inmediata su aplicación para este uso, tan solo hay que añadir una línea más, la línea de tierra de referencia (GND).

Características Principales del bus $I^2C$:

- Solo 2 líneas, línea de datos SDA y línea de reloj SCL.
- Todos los dispositivos conectados al bus son direccionables por software mediante una dirección única. En todo momento existe una relación maestro/esclavo entre dispositivos.
- Se incluyen técnicas de detección de colisiones y de arbitraje para evitar la corrupción de datos si 2 o más dispositivos configurados como maestros intentan transmitir datos al mismo tiempo.
- Velocidades de transmisión serie bidireccional: hasta 100 kbit/s en modo "*Standard*", hasta 400 kbit/s en modo "*Fast*", hasta 1 Mbit/s en modo "*Fast Plus*" y hasta 3,4 Mbit/s en modo "*High-speed*".
- Velocidad de transmisión serie unidireccional: hasta 5 Mbit/s en modo "*Ultra Fast*".
- Filtrado de picos en la línea de datos para preservar su integridad.
- El número máximo de circuitos integrados que se pueden conectar al bus viene determinado por la capacitancia máxima del bus (aprox. 400 pF).

#### 3.4.1.1 Protocolo del bus $I^2C$:

El bus $I^2C$ consta de 2 hilos, uno para la transmisión de datos serie (SDA) y otro para la transmisión del reloj de sincronización (SCL), que portan toda información necesaria para una comunicación completa entre los

dispositivos conectados al bus. Hay que distinguir entre dos tipos de dispositivos, los denominados maestros y los denominados esclavos. La comunicación siempre es iniciada por un dispositivo maestro, que envía o solicita datos a uno o varios dispositivos esclavos, además de generar la señal de sincronismo (reloj). Los dispositivos esclavos tienen una dirección única en el bus, por lo que no aparecen conflictos cuando un maestro intenta comunicarse con un esclavo en particular.

Las líneas SDA y SCL son líneas bidireccionales que conectan todos los dispositivos y que además están conectadas siempre al positivo del generador mediante unas resistencias. Esto hace que si en el bus no esta en uso, las líneas se mantengan a nivel alto.

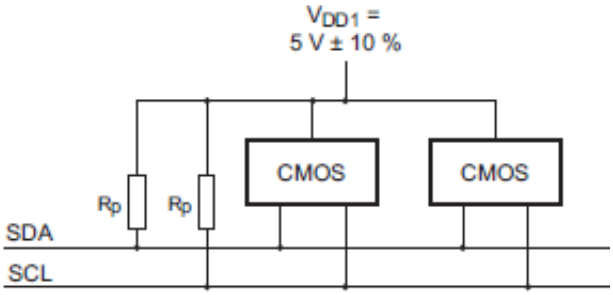



Figura 3- 4: Configuración básica con *pull-up* de un bus $I^2C$.

Por otra parte se considera nivel alto cuando el voltaje en la línea es al menos el 70% del voltaje de la fuente que alimenta el sistema y, nivel bajo cuando esta por debajo del 30%.

Dicho esto también indicar que para que los datos se transmitan sin errores se deben cumplir ciertas condiciones entre la línea de reloj y el estado de la línea de datos. Como se observa en la Figura 3-5, la línea de datos solo puede cambiar de estado H→L o L→H cuando la línea de reloj se encuentra a nivel bajo.

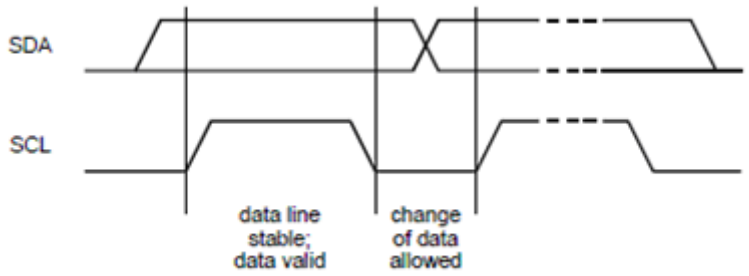



Figura 3- 5: Condiciones para transmisión de datos valida.

Tras este resumen de términos y condiciones se va a proceder a describir los distintos formatos de trama y como se realiza una comunicación completa.

Todas las transmisiones comienzan con la “condición de inicio (S)” y finalizan con la “condición de parada (P)” que, por supuesto, son generadas por el dispositivo que actúa como maestro. En la Figura 3-6 se observan las condiciones necesarias para generar estas señales.

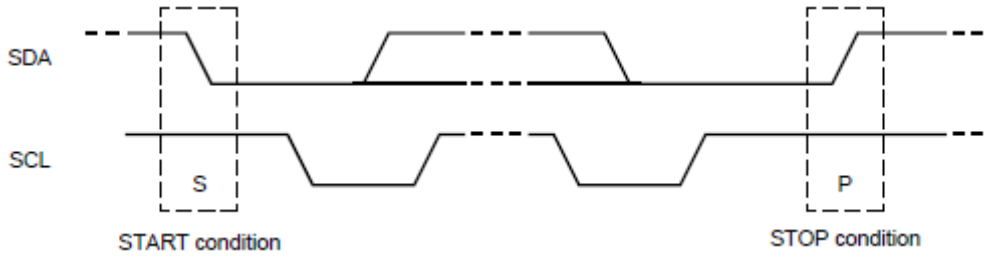



Figura 3- 6: Condiciones de START y STOP.

Una vez envía la condición de inicio el bus se considerara ocupado para cualquier dispositivo que quiera acceder a él, y no se considera libre de nuevo hasta que el maestro envía la condición de parada. También se puede dar el caso de que el maestro envíe una nueva señal de inicio después de un envío previo de dicha señal. Esto se conoce como “repetición de condición de inicio (Sr)”, y a efectos de validez se considera como una condición de inicia más.

A continuación se analiza el formato de las tramas que se envían en una comunicación cualquiera por el bus $I^2C$, ilustrado en la Figura 3-7.

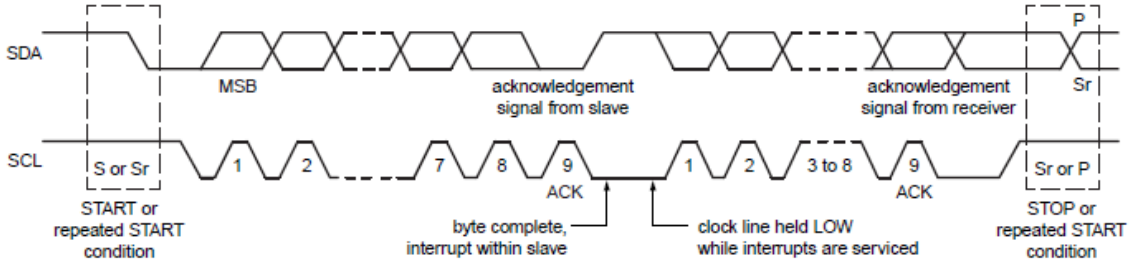



Figura 3- 7: Comunicación completa mediante $I^2C$.

La transmisión se realiza byte a byte seguido por un bit de asentimiento. No hay límites de bytes a transmitir y siempre se comienza la transmisión por el bit más significativo. Si durante la transmisión el dispositivo esclavo necesita parar el envío de datos, este puede retener la línea de reloj a nivel bajo para producir que el dispositivo maestro entre en un estado de espera. Cuando se quiera reanudar la transmisión, tan solo se debe liberar la línea de reloj por parte de esclavo. Esto se conoce como "*Clock stretching*" y es una función opcional que muy pocos dispositivos esclavos implementan. Por tanto es muy poco utilizada.

Los bits de asentimiento (ACK o *ACKnowledgement*) o no asentimiento (NACK o *Negative ACKnowledgement*) que siguen a cada byte transmitido sirven para indicar el estado de la comunicación, ya sea para confirmar el que un byte se ha recibido correcta o incorrectamente, además de para indicar problemas en la transmisión o que alguno de los dispositivos no puede continuar.

En toda transmisión el primer byte que sigue a la señal de comienzo es el byte que incluye la dirección del esclavo con el que se quiere realizar la comunicación. Los siete primeros bits dan la dirección del esclavo y el octavo indica si se trata de una operación de escritura o lectura.

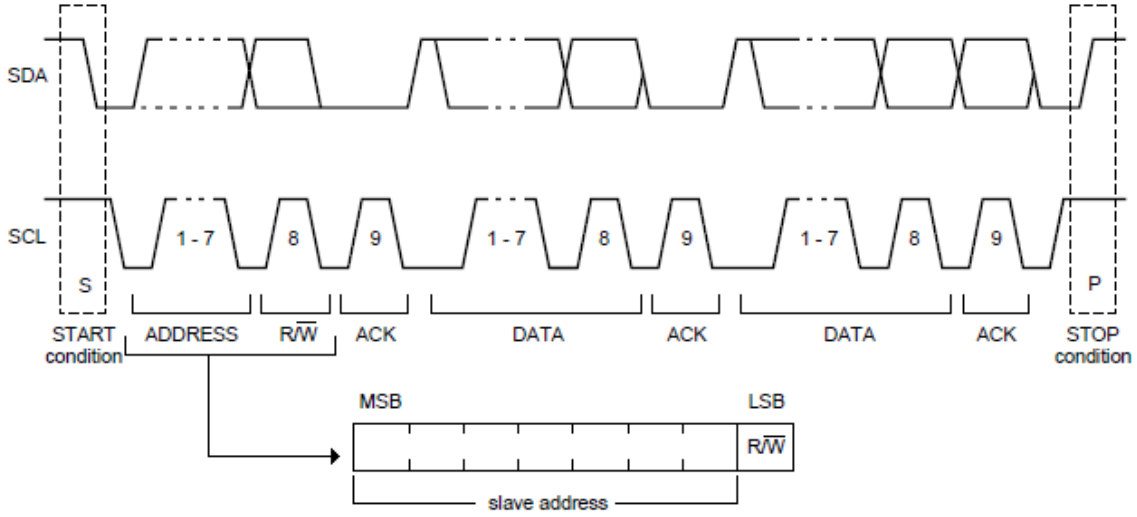



Figura 3- 8: Análisis del primer byte de una comunicación $I^2C$.

En cualquier momento el maestro puede generar una nueva señal de comienzo y direccionar la comunicación a otro esclavo.

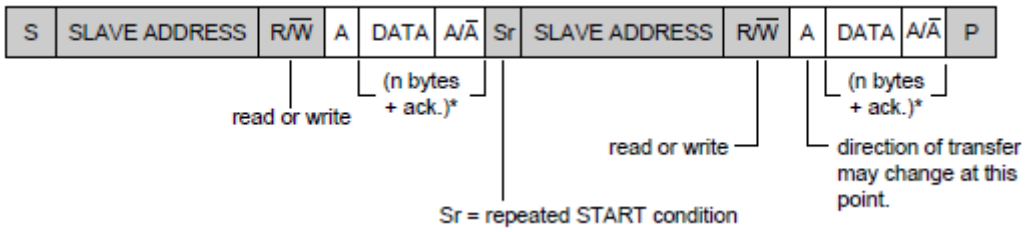



Figura 3- 9: Ejemplo de redireccionamiento de esclavo en comunicación $I^2C$.

Todas las transmisiones terminan con la señal de parada que es generada siempre por el dispositivo maestro.

Existe la posibilidad de direccionamiento con 10 bits, pero actualmente no es muy utilizada. Ambos modos de direccionar, 7-bits y 10-bits pueden convivir en el sistema sin ningún problema.

### 3.4.1.2 Hardware del bus $I^2C$:

Para que el bus funcione correctamente y la comunicación sea lo mas rápida posible y con el menor número de errores, es necesario tener en cuenta ciertos factores físicos que hay que subsanar añadiendo algunos componentes

hardware. A continuación se muestra un esquema simple de conexión de dos dispositivos y los componentes que se deberán tener en cuenta.

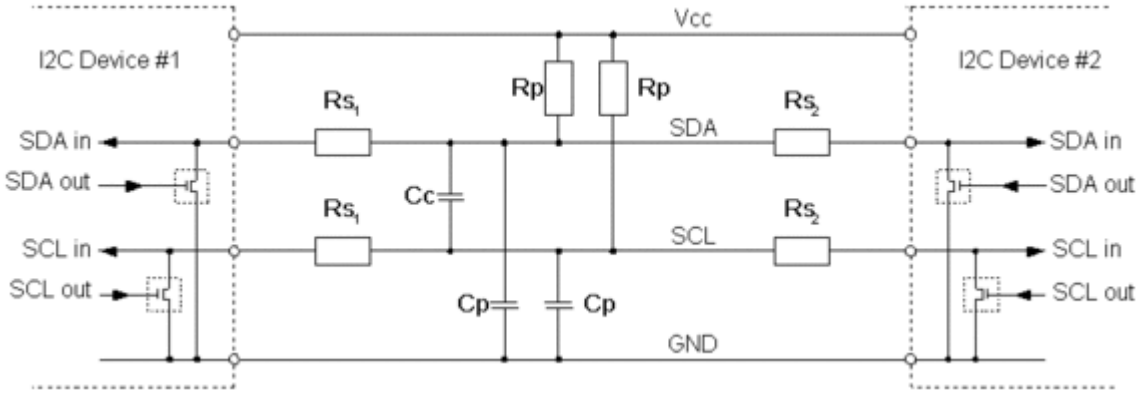



Figura 3- 10: Esquema hardware de una conexión $I^2C$.

El voltaje típico de alimentación (Vcc) oscila entre los 1.2 hasta los 5.5 voltios y siempre debe existir una línea de tierra (GND) común para todos los dispositivos.

Lo primero a tener en cuenta es que las líneas SDA y SCL son del tipo drenaje abierto, es decir, los dispositivos conectados solo pueden actuar sobre las líneas llevándolas a nivel bajo o dejándolas libres. Para llevar las líneas a nivel alto se deben colocar unas resistencias conocidas como “resistencias de polarización (Rp)” que conectan SDA y SCL a Vcc haciendo que si el bus no esta siendo utilizado o ningún dispositivo tiene a nivel bajo alguna de las líneas estas vuelvan a nivel alto de forma automática.

Estas resistencias, junto con la suma de capacitancias (Cp) de conductores, pines, conexiones… afectan al comportamiento temporal de las señales que viajan por SDA y SCL, además de que los posibles valores para Rp vienen determinados también por la capacitancia del bus, una alta Cp debe compensarse con una Rp pequeña y viceversa. Los valores para Rp suelen estar comprendidos entre 1KΩ y 10 KΩ pero su margen real de utilización varia desde los 1K8Ω hasta los 47 KΩ.

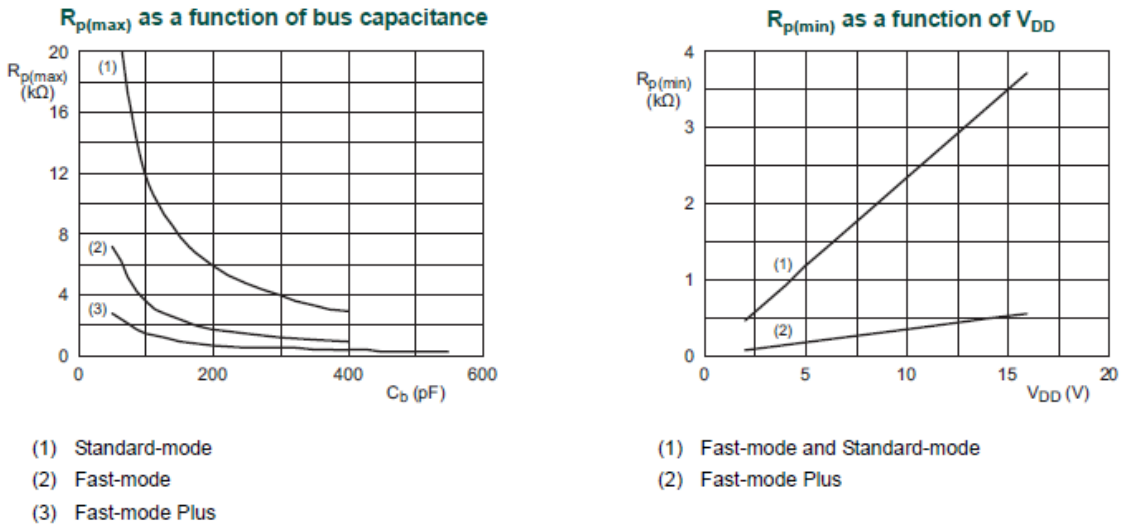



Figura 3- 11: Rango de valores de Rp en función de $C_b$ y $V_{DD}$.

A continuación se muestra un ejemplo de cómo afectan a las señales los distintos valores que puede tomar Rp [44]:

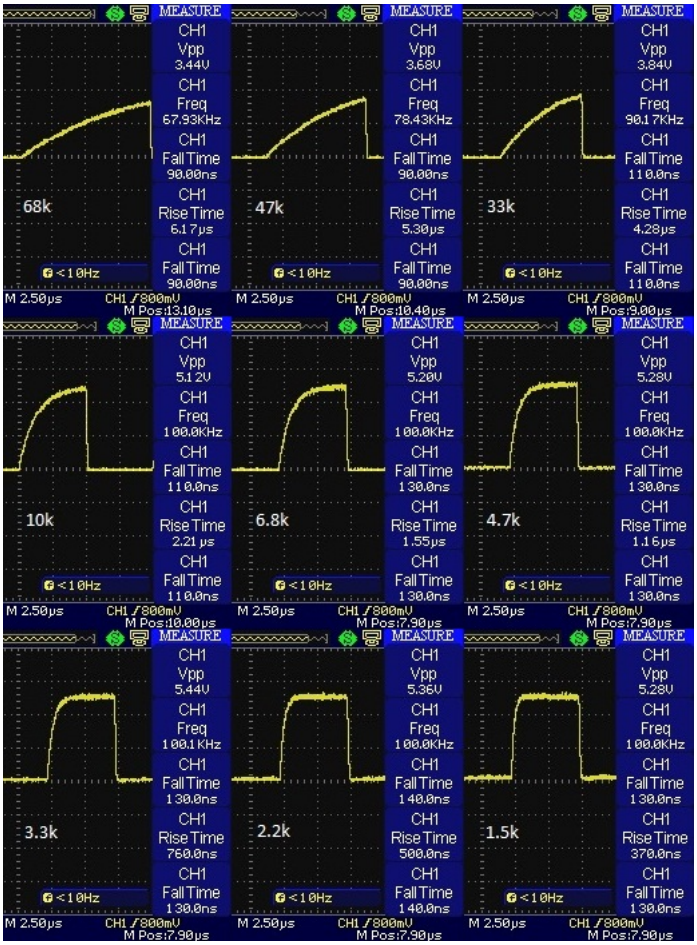



Figura 3- 12: Variación en la forma de la señal en función del valor de Rp.

Por otra parte, es importante indicar que las longitudes de las líneas del bus afectan de forma crítica a la capacitancia del bus y, por tanto se deben diseñar buses lo mas cortos posibles. También recordar que esta capacitancia marca el número máximo de dispositivos que se podrán conectar al bus.

Otro punto a tener en cuenta son las resistencias serie (Rs) que se deben instalar en las líneas del bus si estas vienen de otra placa por medio de algún conector. Se inserta en el sistema para proteger las líneas de posibles sobrecorrientes. Si todo el bus se encuentra dentro de una misma placa no son necesarias.

Las resistencias serie junto con las resistencias de polarización afectan a las líneas del bus cuando estas se encuentran a nivel bajo. El efecto se puede calcular siguiendo la siguiente relación:

$$\frac{Rs}{Rs + Rp} * Vcc$$

Si se colocan resistencias serie muy grandes la tensión del nivel bajo podría ser lo suficientemente alta como para interpretarse como un nivel alto ocasionando graves errores en los datos transmitidos. En la Figura 3-13 se muestra la relación recomendada entre Rp y Rs:

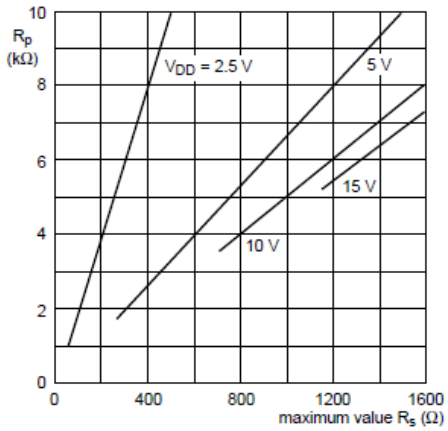



Figura 3- 13: Relación entre los valores de Rp y Rs.

Por ultimo, se describe el efecto de "*Crosstalk*" (Cc) entre las líneas SDA y SCL. Se trata de pequeñas interferencias en las señales producidas por acoplamiento magnético entre las dos líneas del bus. No resulta un problema a

tener en cuenta siempre y cuando los picos no sean demasiado grandes. Su efecto se minimiza acortando todo lo posible el bus y aumentando en la medida de lo posible el valor de Rs y Rp.

### 3.4.2 Protocolo ALPAR

La comunicación entre el ATmega32U4 y el TDA8029 se realizará mediante $I^2C$ pero toda la información que se envíe al TDA8029 deberá estar encapsulada dentro de una trama invariante. El protocolo que rige esta estructuración de los mensajes se conoce como “Protocolo ALPAR” [30].

Las tramas que se envían por el canal de comunicación deben tener la siguiente forma:

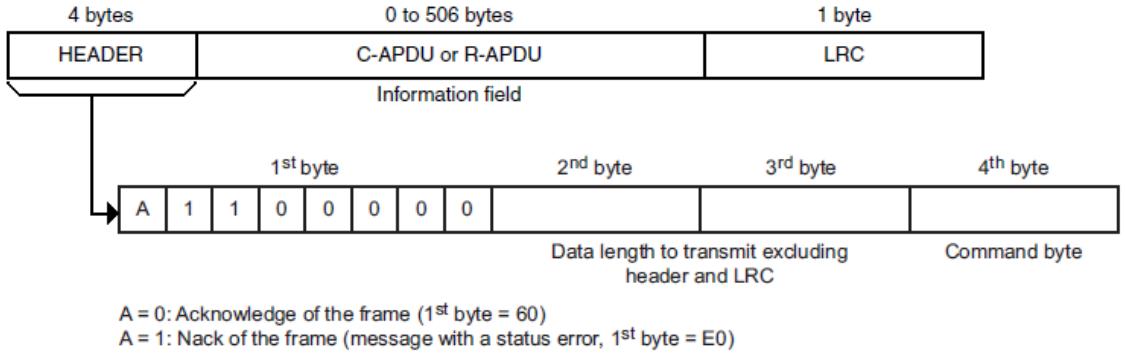



Figura 3- 14: Formato de trama estándar en protocolo ALPAR.

El campo LRC (*Longitudinal Redundancy Check*) es la OR-Exclusiva de todos los bytes de la trama incluyendo el propio campo como nulo.

El protocolo estipula que en toda comunicación el microcontrolador es el maestro y que cuando envía una trama el TDA8029 responde con otra que debe incluir el byte de comando, ACK o NACK y los datos correspondientes a la respuesta, ya sea positiva o mensaje de error. Estas son las estructuras básicas de las tramas:

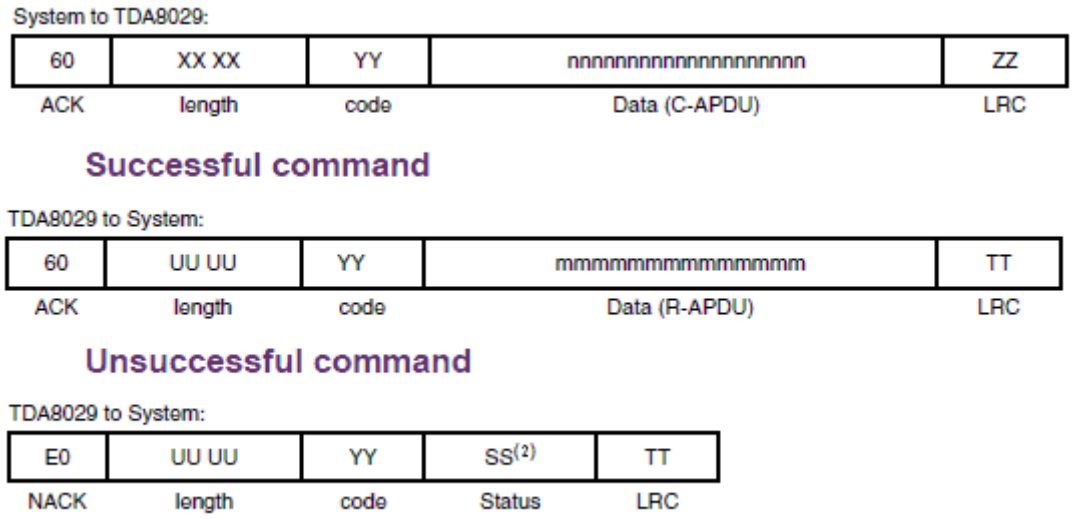



Figura 3- 15: Comunicación mediante protocolo ALPAR. Estructuras básicas de trama.

Donde *length* es la longitud del campo de datos incluido en la trama, *code* es el byte de comando, "*Data* (R-APDU)" es el campo de datos que se envían junto al comando, pudiendo ser una trama APDU si el comando así lo indica, o los datos de la respuesta al comando y, por último, *Status* es el byte de estado que indica el código del error producido.

Una vez vista la forma genérica que deben tener las tramas, a continuación se muestra cómo se realiza una comunicación sobre $I^2C$ usando este protocolo:

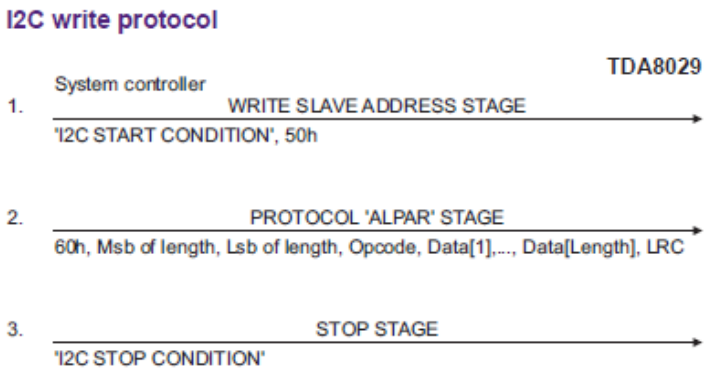



Figura 3- 16: Comunicación ALPAR mediante $I^2C$ en proceso de escritura.

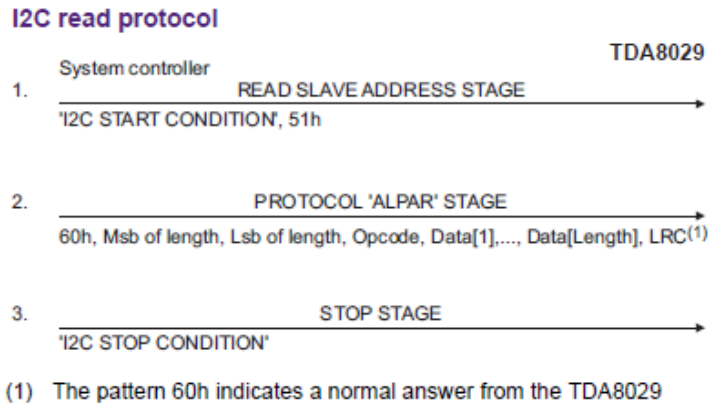



Figura 3- 17: Comunicación ALPAR mediante $I^2C$ en proceso de lectura.

Se debe tener en cuenta que la dirección única asignada dentro del bus para este dispositivo lector de tarjetas es la 0x50.

Partiendo de este punto, lo primero que se envía es la condición de inicio seguida de la dirección del dispositivo “0x50” si se trata de una operación de escritura, o la dirección del dispositivo con el bit menos significativo a “1”, es decir, 0x51, si se trata de una operación de lectura. Seguidamente se envía la trama de datos usando el protocolo ALPAR y, una vez terminada la transferencia de todos los datos por parte del maestro, si es una operación de escritura, o por parte del esclavo, si es una operación de lectura, el maestro coloca en el bus la condición de parada.

Por tanto se trata de una transferencia estándar sobre $I^2C$ con la salvedad de que las tramas que portan los comandos y datos deben ser encapsuladas siguiendo el protocolo descrito.

Para ver todos los comandos permitidos, funciones disponibles y el listado de errores se debe consultar la nota de aplicación AN10207 [23].

### 3.4.3 Protocolo APDU

Define el formato de trama necesario para el envío y recepción de comandos y datos entre lectores y tarjetas inteligentes [27]. Se encuentra definido en el estándar ISO/IEC 7816 apartado 4. Existen dos tipos de tramas, las de comando y las de respuesta.

La estructura de una trama de comando es la siguiente:

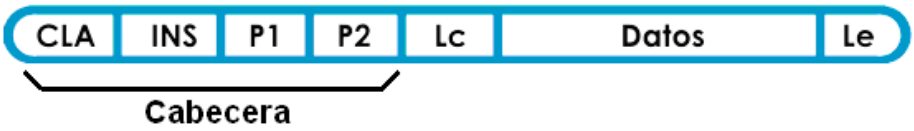



Figura 3- 18: Formato estándar de trama de comando en protocolo APDU.

- CLA: clase de instrucción (1 byte).
- INS: código de instrucción (1 byte).
- P1, P2: parámetros de la instrucción (2 bytes).
- Lc: longitud campo de datos (0, 1 o 3 bytes).
- Datos: datos a enviar.
- Le: número de bytes de datos esperados en la respuesta.

La estructura de la trama de respuesta es:

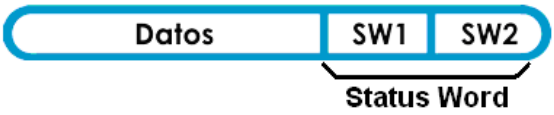



Figura 3- 19: Formato estándar de trama de respuesta en protocolo APDU.

- Datos: datos de la respuesta. Contiene “Le” bytes.
- SW1, SW2: indican el estado del proceso (2 bytes).

La forma de operar con estos comandos es muy simple. Primero se envía a la tarjeta el comando deseado, esta lo procesa y, si es correcto, devuelve los datos demandados y un “*Status Word*” indicando todo correcto o alguna advertencia. Si el comando no es valido solo devuelve el “*Status Word*” con el error correspondiente.

En este proyecto, en el que el chip lector tiene su propio protocolo de comunicación, cuando se envían tramas APDU desde el microcontrolador, estas viajan encapsuladas dentro del protocolo ALPAR, en el campo de datos y previa indicación en el campo de comando de que se envía una trama APDU.

## 3.5 Solución propuesta

Una vez vistos los principales dispositivos a utilizar, las tecnologías que implementan y los protocolos de comunicación que mejor se adaptan al problema que presenta este proyecto se puede proponer una solución que satisfaga los objetivos del mismo. A continuación se muestra un diagrama de bloques que explica de forma sencilla los principales detalles de esta solución:

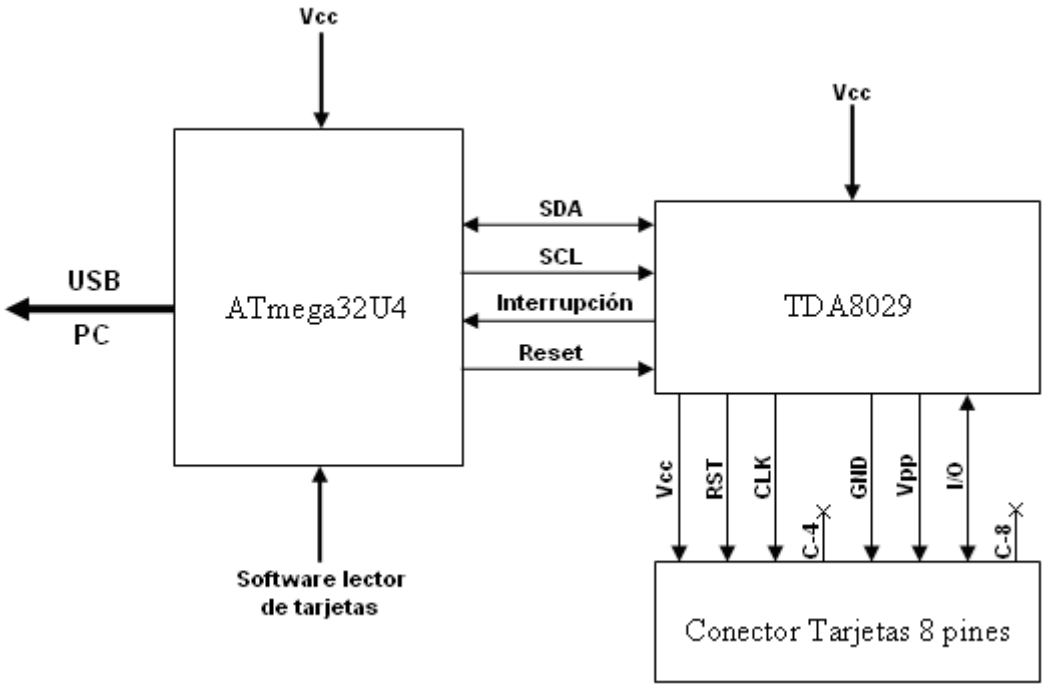



Figura 3- 20: Diagrama de la solución propuesta para el lector de tarjetas.

La solución que se propone es un PCB (*Printed Circuit Board*) con las dimensiones más reducidas posibles en el que se instalen tanto el microcontrolador como el chip lector de tarjetas y su conector, así como todos los componentes discretos necesarios para el funcionamiento del sistema: condensadores, resistencias, cristales, conectores… De esta manera se crea un pequeño lector independiente que solo necesita del exterior, una vez programado, una línea de alimentación y una conexión USB a un PC donde descargar los datos leídos.

En los siguientes apartados se describe con detalle el diseño del esquemático del circuito impreso: rutado, construcción, montaje, programación y puesta en funcionamiento.

# 4 Desarrollo del prototipo

## 4.1 Introducción

En este apartado se explica con detalle el proceso que sigue la solución propuesta, pasando por las fases de diseño del esquemático, rutado del circuito impreso, fabricación del circuito impreso, instalación de componentes y creación e instalación del software de usuario.

Para llevar a cabo estos puntos se emplean distintas aplicaciones software de las que se incluye una pequeña guía de uso a medida que se explica el proceso de desarrollo del proyecto. Estas aplicaciones utilizadas son:

- *Altium Designer* [2]: conjunto de herramientas y programas para el diseño electrónico en todas sus fases. Con él se realizara la parte del diseño de esquemático, creación y rutado del PCB, testeo de errores y, por último, generación de los archivos Gerber y de taladros para la maquina de fabricado de PCBs.

- *AVR Studio 5.0* [8]: entorno de desarrollo integrado para la creación, compilación y depuración de aplicaciones en código C/C++ y ensamblador para microcontroladores de Atmel. Con este software se lleva a cabo la creación de la aplicación de usuario y la programación y configuración del microcontrolador con la ayuda del programador externo AVRISP mkII [7].

Estas aplicaciones software son totalmente desconocidas para mí al comienzo del proyecto, por lo que es necesario un estudio detallado de sus características, usos, particularidades y funcionalidades que pueden desarrollar, con especial atención en el software *Altium Designer*, ya que es el más extenso y complejo de utilizar.

Durante el proceso de desarrollo se hará uso de otros equipos auxiliares como son la estación de soldadura, osciloscopio, fuentes de alimentación, analizador lógico, multímetro…

## 4.2 Esquemático

Para la realización de este apartado se hace uso del software *Altium Designer* anteriormente presentado. Los siguientes apartados constituyen una pequeña guía de cómo se ha desarrollo el esquemático del PCB mediante *Altium Designer* que dará lugar al lector de tarjetas.

Tras un estudio detallado de las hojas de características de los chips principales y notas de aplicación, prestando particular atención a todos los elementos que rodean a estos chips, y sin los cuales no podrían funcionar, se comienza el desarrollo del esquemático [5][31][6][21][28][30][37].

Se crea un nuevo proyecto del tipo "*PCB Project*" y se añade una nueva hoja de esquemático sobre la que se colocan los componentes y conexiones necesarias. Esta es la apariencia del programa en este punto:

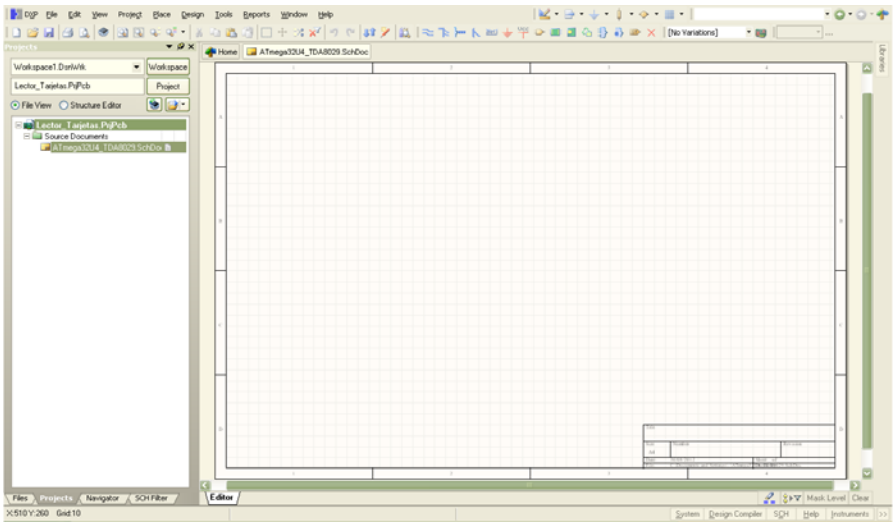


Figura 4- 1: Interfaz de *Altium Designer* para el editor de esquemáticos.

Se comienzan a añadir componentes. Los más estándares, como condensadores, resistencias, etc. se encuentran en las librerías que vienen por defecto con el software, pero los chips particulares que se han elegido para el

proyecto no están, por lo que hay que crearlos con el editor que incluye el software para esta tarea. También se debe comprobar el *footprint* que incluye por defecto cada componente de la librería y, si no es el deseado, cambiarlo. En caso de crear el componente se le debe adjudicar un *footprints* estándar, si existe, o se debe crear nuevo con otra herramienta de la que se dispone en el software.

Para saber qué *footprints* de los instalados en el software se corresponde a cada componente se puede acceder a la página Web de *Altium* y, en el apartado de librerías [3], hay un buscador que permite introducir los parámetros característicos de cada chip y buscar cuál es el *footprints* estándar que encaja con él.

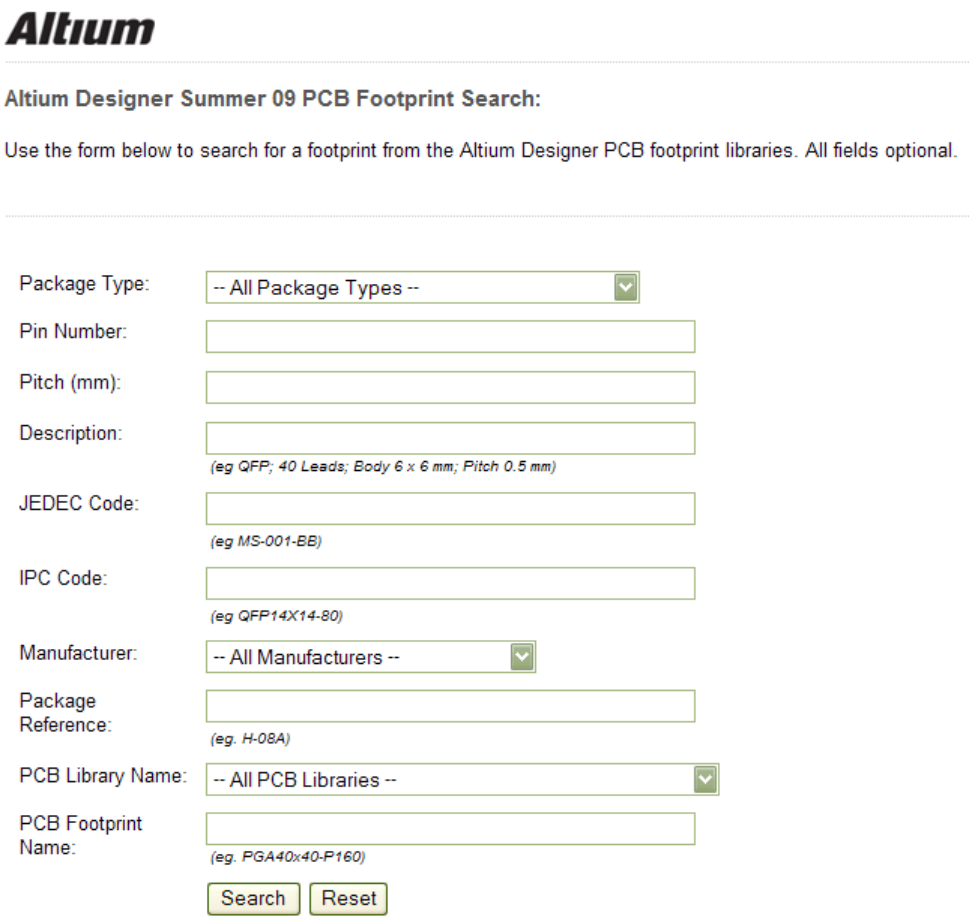



Figura 4- 2: Formulario de búsqueda de *footprints* disponible en la Web *Altium Designer*.

En caso de que no sea un *footprints* estándar y no aparezca dentro de los disponibles en las librerías se tienen dos opciones. Por un lado se puede hacer a mano sobre un archivo tipo "*PCB Library*", añadiendo los pines con su forma y la capa a la que pertenecen, contorno del componente, marcas en las distintas capas del PCB, etc. y todo ello con el tamaño exacto del componente.

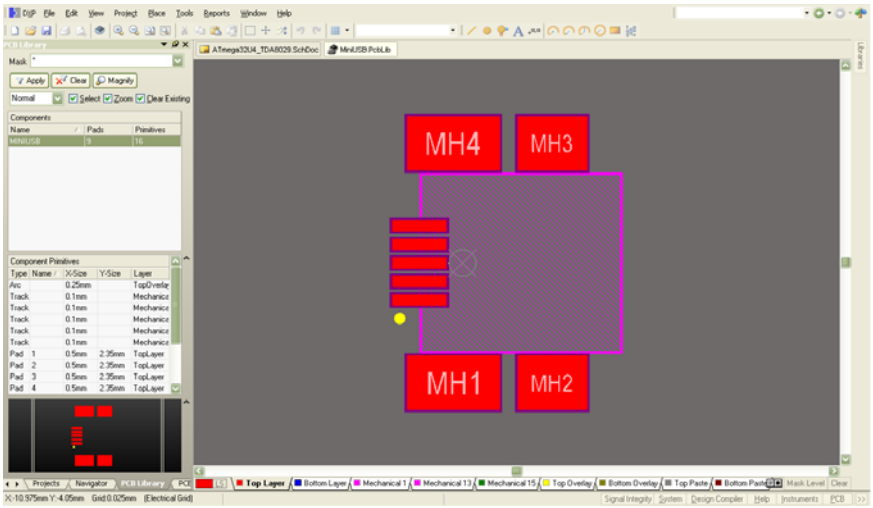



Figura 4- 3: Interfaz *Altium Designer* para editor de *footprints* (vista 2D).

También se puede construir el modelo 3D para que aparezca de esta manera cuando se visualiza el PCB con el simulador 3D.

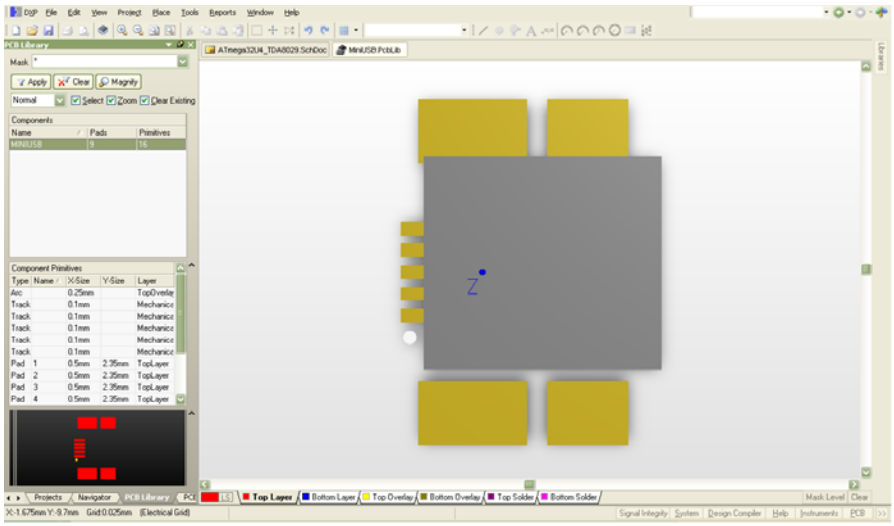


Figura 4- 4: Interfaz *Altium Designer* para editor de *footprints* (vista 3D).

Por otro lado se puede utilizar un asistente que incluye este software para llevar a cabo esta clase de tareas. Para acceder al asistente se debe

añadir al proyecto un archivo tipo "*PCB Library*" y una vez en él, en el menú "*Tools*" se selecciona la opción "*Component Wizard*". Siguiendo los pasos del asistente se consigue crear el componente deseado con un *footprint* a medida.


Figura 4- 5: Pantalla inicial del asistente para creación de *footprints*.

Para el PCB del lector de tarjetas se han tenido que crear los esquemáticos de los chips ATmega32U4 y TDA8029, así como los *footprints* del conector USB, conector de tarjetas, cristales, pulsador y algún condensador. Es importante indicar que todos los *footprints* utilizados son del tipo montaje superficial excepto el conector de tarjetas, lo que facilita la construcción del PCB y reduce de manera muy significativa el tamaño de los componentes y en consecuencia el del PCB.

Se colocan los componentes sobre la hoja y se van haciendo las conexiones oportunas ya sea mediante un "hilo" o mediante etiquetas que conectan dos puntos sin representación visual sobre la hoja de esquemático.


Figura 4- 6: Funcionalidad de los botones del editor de esquemático en *Altium Designer*.

Seguidamente se resume cómo evoluciona el esquemático y cómo finalmente queda todo conectado:

Se comienza por la entrada de alimentación del PCB. Se tiene el conector de entrada, filtro para la señal de alimentación y un diodo Led con su resistencia limitadora de corriente que indica si el sistema está alimentado o no.

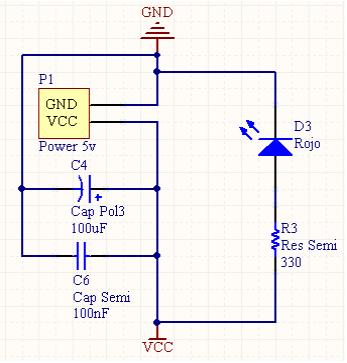



Figura 4- 7: Esquemático del bloque de alimentación del PCB en diseño.

A continuación se muestra el bloque formado por el chip TDA8029 y todos los elementos necesarios para su correcto funcionamiento. Es importante destacar la existencia de dos conectores para tarjetas. Esto se debe a que se prepara el PCB con dos *footprints* solapados para poder usar un tipo de conector u otro sin tener que volver a construirlo.

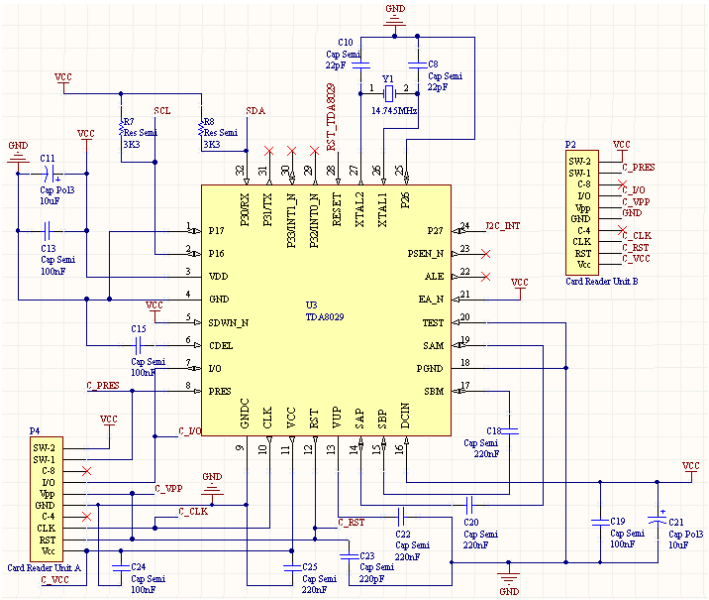



Figura 4- 8: Esquemático del bloque del chip lector de tarjetas del PCB en diseño.

Por ultimo se tiene el bloque del microcontrolador. Está compuesto, principalmente, por el chip ATmega32U4, el conector de miniUSB y dos conectores de comunicación, uno donde se conecta el programador del microcontrolador y otro que se deja preparado para una conexión SPI de cualquier otro dispositivo o PCB al microcontrolador. En principio este conector tiene su razón de ser en un trabajo futuro planeado en inicio para comunicar por SPI este PCB con otro que albergara un lector de tarjetas RFID.

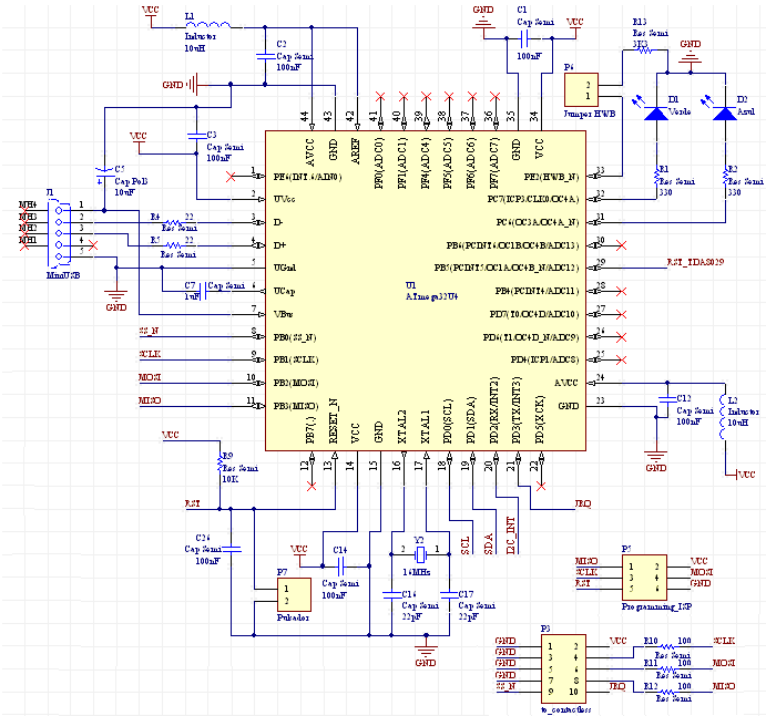


Figura 4- 9: Esquemático del bloque del microcontrolador del PCB en diseño.

Una vez expuestos cada uno de los bloques principales se expone una panorámica de cómo queda el esquemático del lector de tarjeta que se está diseñando:

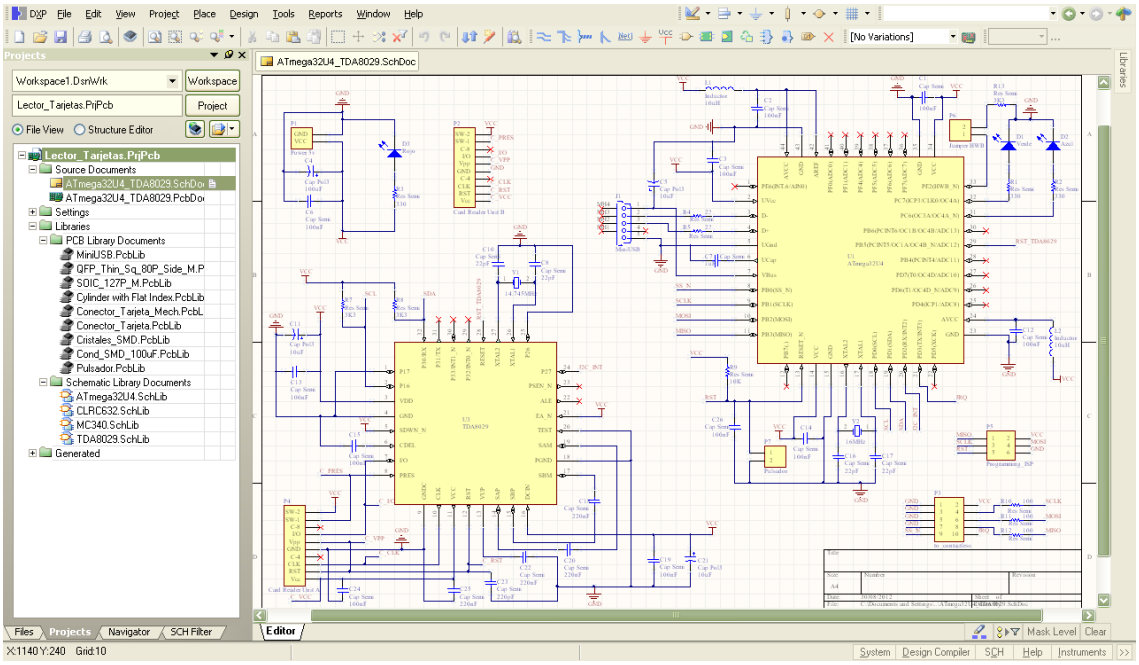


Figura 4- 10: Interfaz del editor de esquemáticos con el esquema terminado.

Por último se debe compilar el proyecto para detectar posibles errores en conexiones mal hechas o perdidas, pines que se hayan quedado sin conectar, cortocircuitos, etc. También al compilar se comprueba si los pines configurados como salida funcionan como salida y los de entrada como entrada. En caso contrario se notifica un error indicando el dispositivo y el pin que causan ese error.

Después de este paso se puede crear el documento PCB donde se colocan los componentes y se rutan las pistas. Este es un listado de todos los componentes utilizados con sus valores y *footprints* asociados:

Tabla 1.- Listado de componentes que forman el lector de tarjetas.

| Comment | Description | Designator | *Footprint* | Quantity | Value |
|---|---|---|---|---|---|
| Cap Semi | Capacitor (Surface Mount) | C1, C2, C3, C6, C12, C13, C14, C15, C19, C24, C26 | CAPC2012L | 11 | 100nF |
| Cap Pol3 | Polarized Capacitor (Surface Mount) | C4 | Cond_SMD_100uF | 1 | 100uF |
| Cap Pol3 | Polarized Capacitor (Surface Mount) | C5, C11, C21 | CAPC3216L | 3 | 10uF |
| Cap Semi | Capacitor (Surface Mount) | C7 | CAPC2012L | 1 | 1uF |
| Cap Semi | Capacitor (Surface Mount) | C8, C10, C16, C17 | CAPC2012L | 4 | 22pF |
| Cap Semi | Capacitor (Surface Mount) | C18, C20, C22, C25 | CAPC2012L | 4 | 220nF |
| Cap Semi | Capacitor (Surface Mount) | C23 | CAPC2012L | 1 | 220pF |
| Verde | Typical BLUE SiC LED | D1 | CAPC2012L | 1 | |
| Azul | Typical BLUE SiC LED | D2 | CAPC2012L | 1 | |
| Rojo | Typical BLUE SiC LED | D3 | CAPC2012L | 1 | |
| MiniUSB | Mini-B Receptacle, Right Angle, SMT, 0.80mm (.031") Pitch, Solder Tabs with Back Cover, Recessed Type | J1 | MiniUSB | 1 | |
| Inductor | Inductor | L1, L2 | INDC2012L | 2 | 10uH |
| Power 5v | Header, 2-Pin | P1 | HDR1X2 | 1 | |
| Card Reader Unit B | Header, 10-Pin | P2 | Conector Tarjeta Mecanico | 1 | |
| to_contactless | Header, 5-Pin, Dual row | P3 | HDR2X5 | 1 | |
| Card Reader Unit A | Header, 10-Pin | P4 | Conector_Tarjeta | 1 | |
| Programming_ISP | Header, 3-Pin, Dual row | P5 | HDR2X3 | 1 | |
| Jumper HWB | Header, 2-Pin | P6 | HDR1X2 | 1 | |
| Pulsador | Header, 2-Pin | P7 | Pulsador | 1 | |
| Res Semi | Semiconductor Resistor | R1, R2, R3 | RESC2012L | 3 | 330 |
| Res Semi | Semiconductor Resistor | R4, R5 | RESC2012L | 2 | 22 |
| Res Semi | Semiconductor Resistor | R7, R8, R13 | RESC2012L | 3 | 3K3 |
| Res Semi | Semiconductor Resistor | R9 | RESC2012L | 1 | 10K |
| Res Semi | Semiconductor Resistor | R10, R11, R12 | RESC2012L | 3 | 100 |
| ATmega32U4 | Low power CMOS 8-bit Microcontroller | U1 | TQFP80P1200X1200-44AM | 1 | |
| TDA8029 | Contact Smart Card Reader | U3 | TQFP80P900X900-32AM | 1 | |
| 14.745MHz | Crystal Oscillator | Y1 | PCBComponent_1 | 1 | |
| 16MHz | Crystal Oscillator | Y2 | PCBComponent_1 | 1 | |

## 4.3 Rutado PCB

Lo primero que se debe hacer es añadir al proyecto un archivo tipo “PCB”. Una vez añadido desde la hoja de esquemático, en el menú de diseño, se toma la opción “*Update PCB Document*” de manera que todos los *footprints* de los componentes aparecen en el archivo de PCB junto con todas las conexiones realizadas.

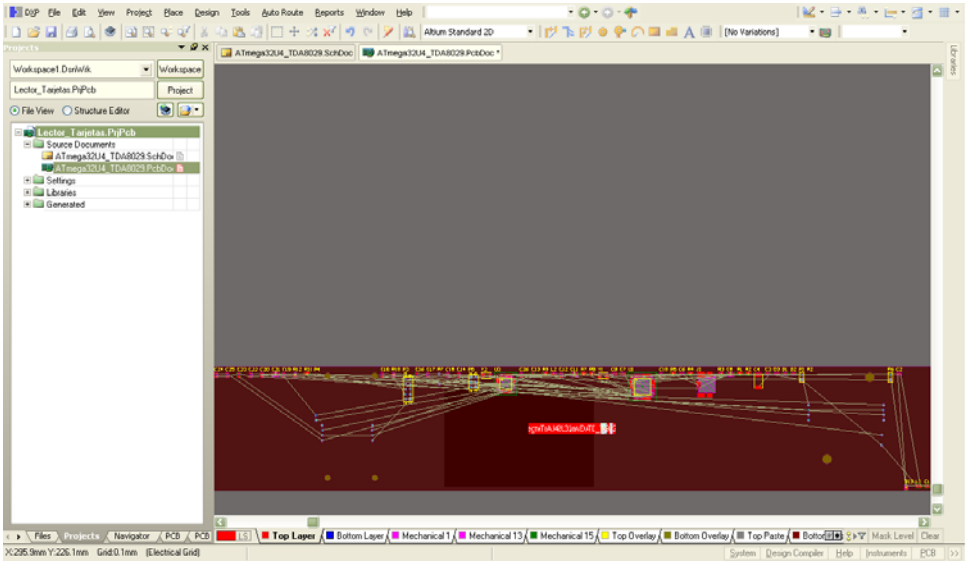


Figura 4- 11: Interfaz de *Altium Designer* para el editor de PCBs.

Una vez en este punto, se deben organizar los componentes de manera que las pistas que se rutarán después sean lo mas cortas posibles, se generen el menor numero de vías y que el PCB quede lo mas compacto posible.

Como detalle, es importante indicar que todos los componentes van sobre la cara superior del PCB excepto el conector de tarjetas que se sitúa en la cara inferior, haciendo así un diseño compacto que aprovecha al máximo el espacio.

También es importante saber y habilitar las capas que, de entre todas las disponibles, se deben utilizar. Para este PCB solo se utilizan tres capas que son "*Top*" y "*Bottom*" para la instalación de componentes y pistas y la capa "*Mechanical 1*" para el contorno del PCB.

Por último, y antes de empezar el rutado, se deben crear las reglas de diseño que se deben cumplir. Para ello, en el menú "*Design*" se selecciona la opción "*Rules*…". Se deben ajustar parámetros como el ancho de las pistas, distancia entre componentes, tamaño de las vías y taladros… Una vez introducidos en el editor de reglas, el software indica mediante código de colores sobre el PCB y errores en el chequeo de reglas si se cumplen todos los parámetros establecidos o no. Este es el aspecto del editor:

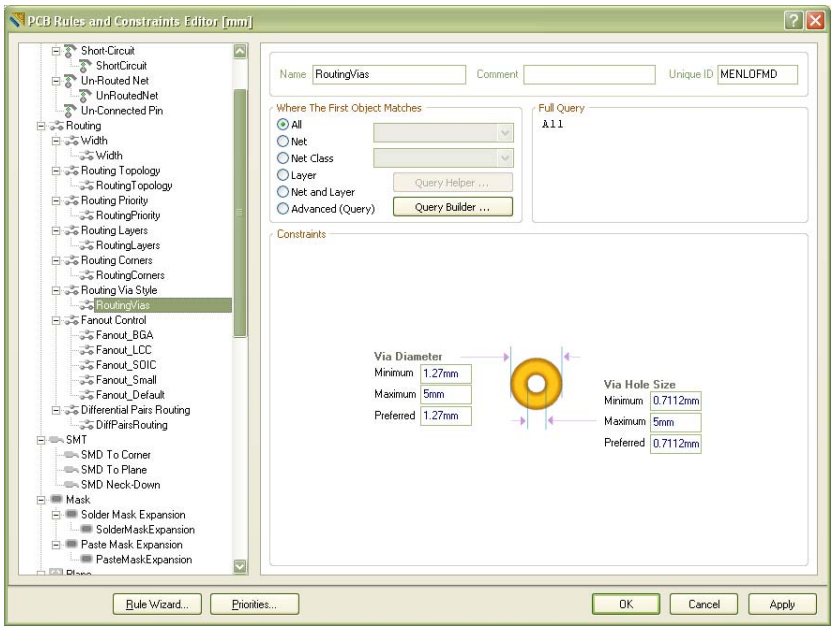



Figura 4- 12: Editor de reglas para el diseño del PCB.

Se comienza con la colocación teniendo en cuenta todos los factores anteriormente mencionados y se obtiene el siguiente resultado:

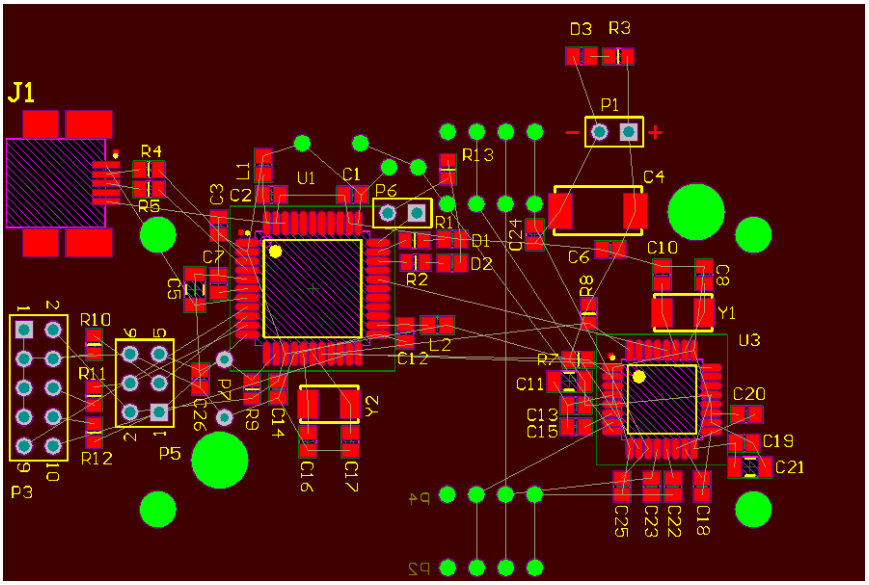



Figura 4- 13: Composición del PCB antes de comenzar el rutado.

Los puntos verdes son los taladros pertenecientes a los pines y puntos de sujeción de los conectores de tarjetas. El software los muestra de este color indicando un error, diciendo que se encuentran solapados, hecho que no se debe tener en cuenta ya que se han colocado así intencionadamente.

En los bordes se debe dejar espacio suficiente para colocar cuatro taladros que servirán para instalar unos apoyos para el PCB. Para definir el tamaño de la zona útil para la instalación de componentes y pistas se debe utilizar la opción "*Board Shape*" del menú "*Design*". Esta opción facilita una herramienta con la que se dibuja el contorno del espacio útil.

Se realiza el rutado de forma manual. Aunque el software dispone de una herramienta de rutado automático, los resultados obtenidos con ella no han sido satisfactorios debido a que se generan demasiadas vías y el rutado de las pistas resulta un poco enrevesado. Esta herramienta se encuentra en el menú "*Auto Route*" y opción "*All*…".Este es el aspecto del PCB una vez finalizada esta tarea:

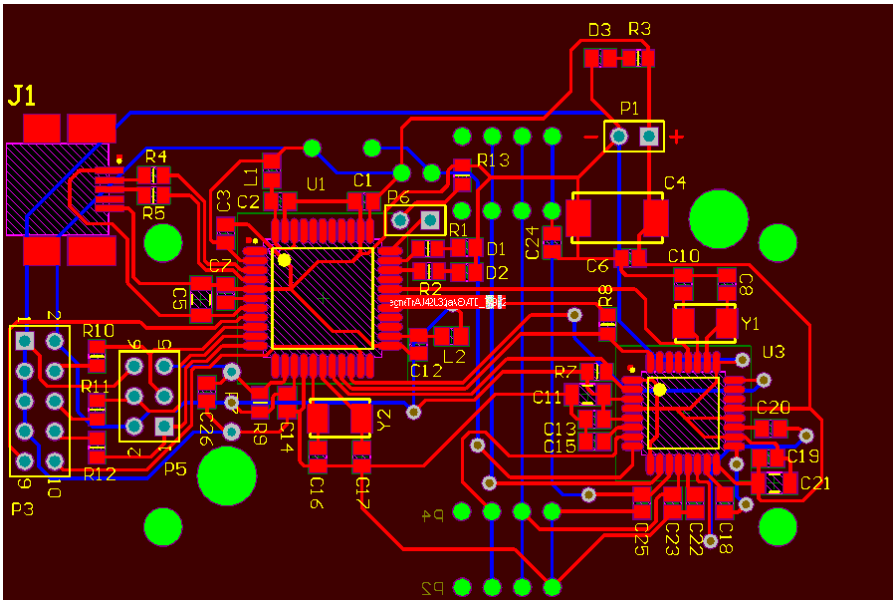



Figura 4- 14: PCB con rutado completo.

Las pistas de color rojo son las que se encuentran en la cara superior de PCB y las azules las que están en la cara inferior. El siguiente paso es añadir los taladros para los apoyos, el contorno del PCB, los "*teardrops*" en las conexiones para mejorar la integridad de las señales y personalizar el PCB. La opción para añadir los "*teardrops*" se encuentra en el menú "*Tools*" y su efecto es el siguiente:

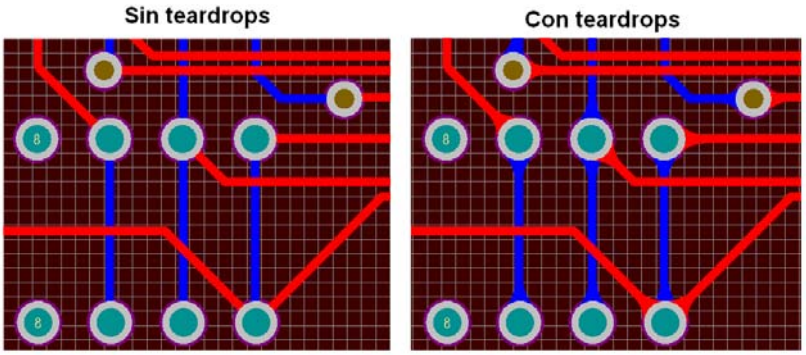



Figura 4- 15: Detalle del PCB sin *teardrops* y con *teardrops*.

Este es el aspecto después de todos estos cambios:

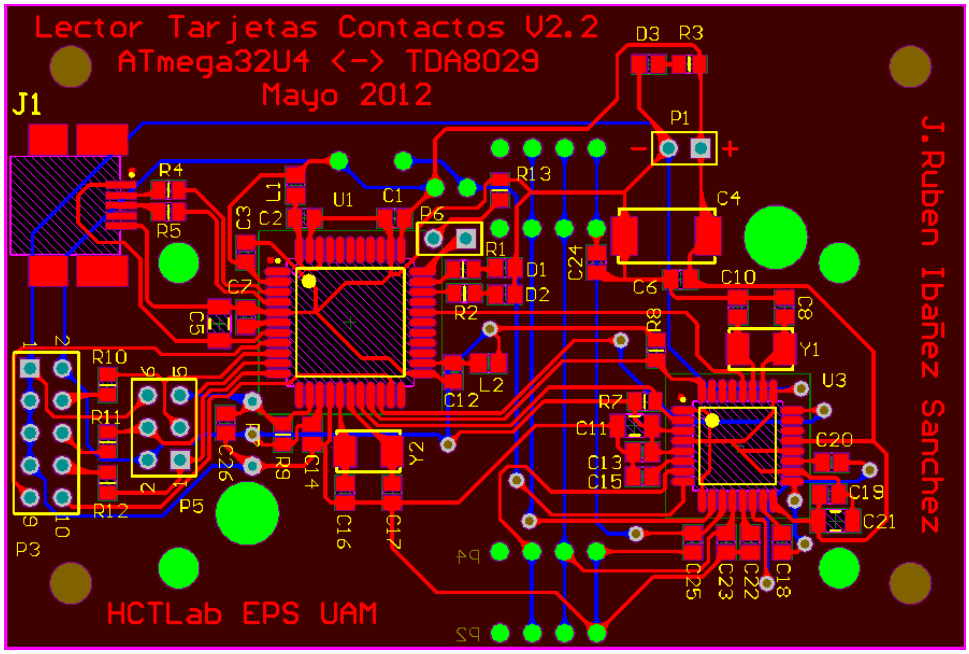



Figura 4- 16: PCB terminado.

Una vez terminado se comprueba que se cumplan todas las reglas de diseño preestablecidas mediante la opción "*Design Rule Check*" del menú "*Tools*".

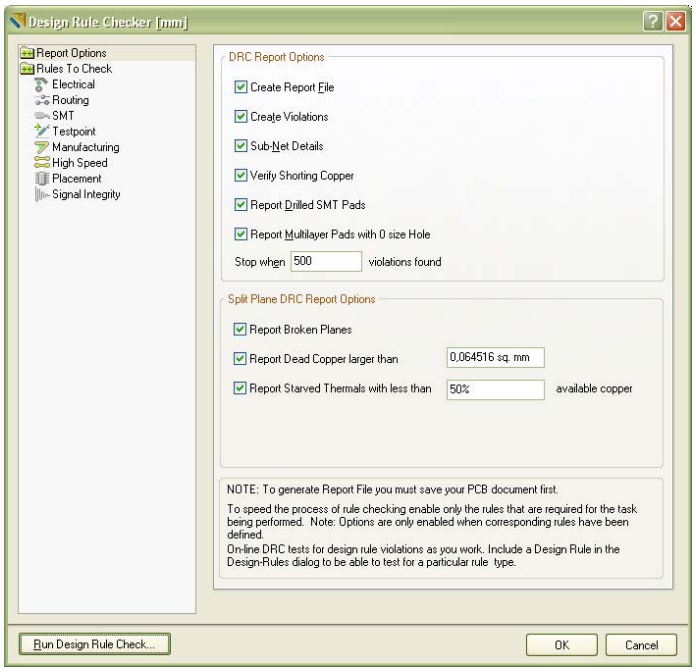
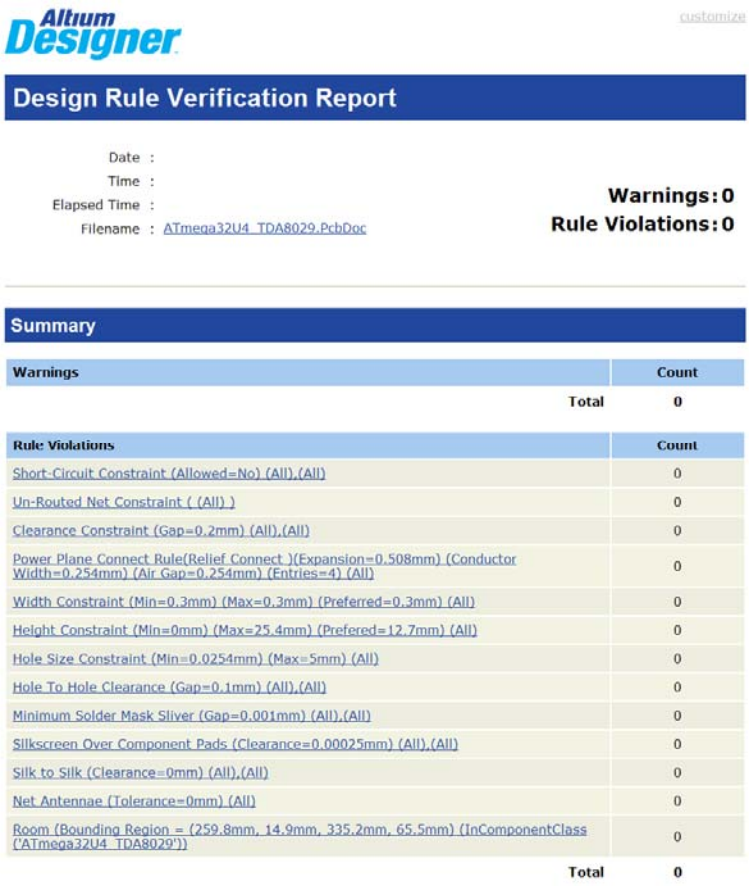



Figura 4- 17: Ventana de opciones de *Design Rule Check*.

Y el resultado obtenido después de todas las correcciones es el siguiente:



Altium Designer

customize

**Design Rule Verification Report**

Date :
Time :
Elapsed Time :
Filename : ATmega32U4_TDA8029.PcbDoc

**Warnings: 0**
**Rule Violations: 0**

**Summary**

| Warnings | Count |
|---|---|
| Total | 0 |

| Rule Violations | Count |
|---|---|
| Short-Circuit Constraint (Allowed=No) (All),(All) | 0 |
| Un-Routed Net Constraint ( (All) ) | 0 |
| Clearance Constraint (Gap=0.2mm) (All),(All) | 0 |
| Power Plane Connect Rule(Relief Connect )(Expansion=0.508mm) (Conductor Width=0.254mm) (Air Gap=0.254mm) (Entries=4) (All) | 0 |
| Width Constraint (Min=0.3mm) (Max=0.3mm) (Preferred=0.3mm) (All) | 0 |
| Height Constraint (Min=0mm) (Max=25.4mm) (Prefered=12.7mm) (All) | 0 |
| Hole Size Constraint (Min=0.0254mm) (Max=5mm) (All) | 0 |
| Hole To Hole Clearance (Gap=0.1mm) (All),(All) | 0 |
| Minimum Solder Mask Sliver (Gap=0.001mm) (All),(All) | 0 |
| Silkscreen Over Component Pads (Clearance=0.00025mm) (All),(All) | 0 |
| Silk to Silk (Clearance=0mm) (All),(All) | 0 |
| Net Antennae (Tolerance=0mm) (All) | 0 |
| Room (Bounding Region = (259.8mm, 14.9mm, 335.2mm, 65.5mm) (InComponentClass ('ATmega32U4_TDA8029')) | 0 |
| Total | 0 |

Figura 4- 18: Reporte de errores de *Design Rule Check*.

La mayoría de errores corregidos se deben a problemas con los textos de los nombres de los componentes, números de los pines, dibujos de los contornos… Como en este caso no se van a imprimir, no se presta atención a la colocación adecuada en el PCB sino que simplemente se colocan donde menos interfieran. Por tanto, en el editor de reglas se debe retocar los parámetros que controlan estos errores y adjudicarles un valor que haga desaparecer el error o si es posible desactivarlos.

Terminado el PCB, el software ofrece la posibilidad de obtener una vista simulada del resultado una vez fabricado y con los componentes montados:

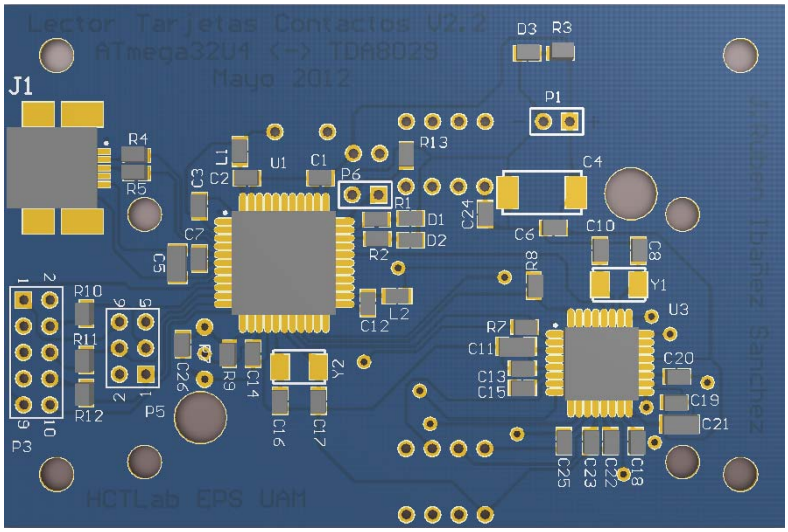



Figura 4- 19: Vista simulada del PCB terminado con componentes.

También ofrece la posibilidad de observar el PCB con y sin sus componentes mediante una vista 3D:

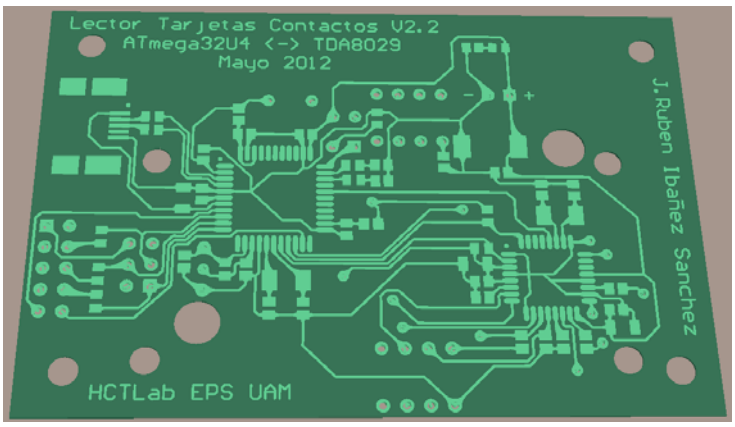



Figura 4- 20: Vista 3D del PCB terminado sin componentes.

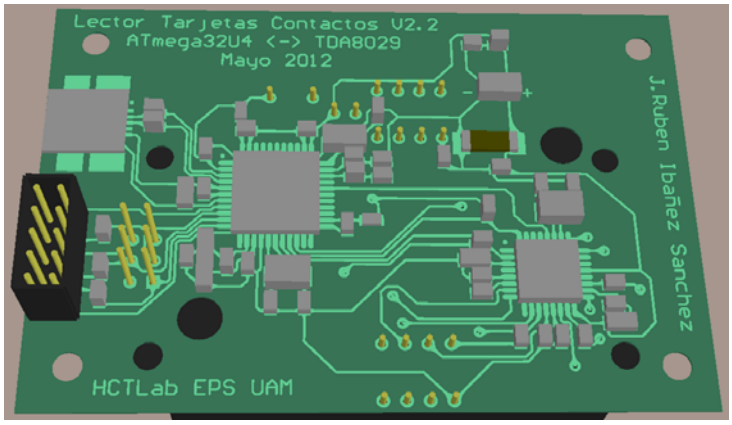



Figura 4- 21: Vista 3D del PCB terminado con componentes.

## 4.4 Construcción PCB

Una vez finalizado el proceso de rutado se deben generar los archivos necesarios para la fabricación del PCB. Para ello se añade un nuevo archivo al proyecto de *Altium* denominado "*Output Job File*".

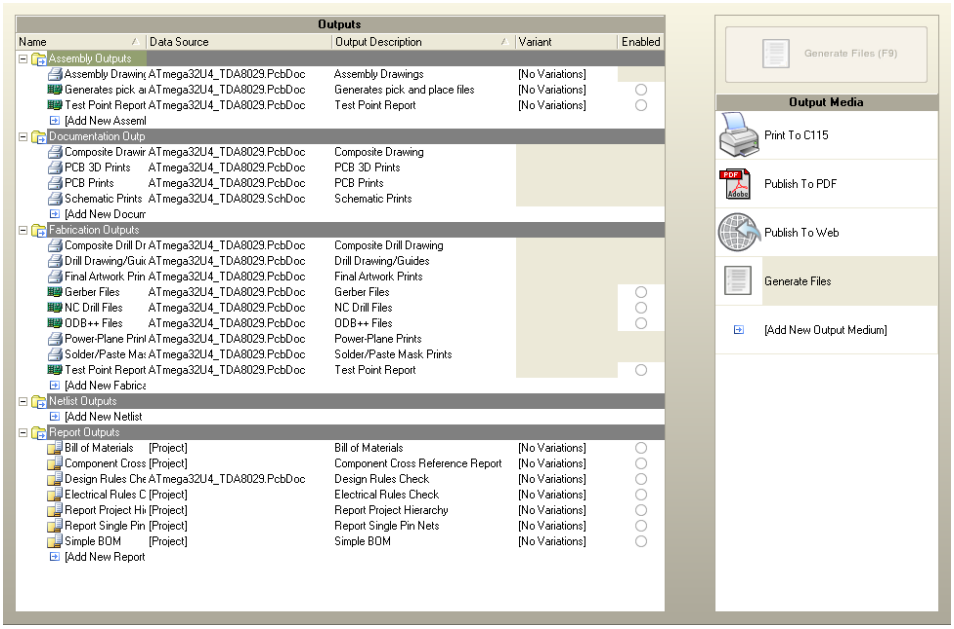



Figura 4- 22: Interfaz de *Altium Designer* del editor para extracción de documentos y archivos.

Con la ayuda de este editor se pueden seleccionar todos los archivos que se quieren extraer, ya sean documentos de texto o planos, o los archivos Gerber y de taladros para la fabricación.

Para obtener los documentos relativos al proyecto y los planos de rutado y componentes, se debe seleccionar en el menú de la derecha "*Publish To PDF*", lo que activa los distintos documentos en el menú de la izquierda que se pueden sacar en este formato. Se opta por generar los siguientes:

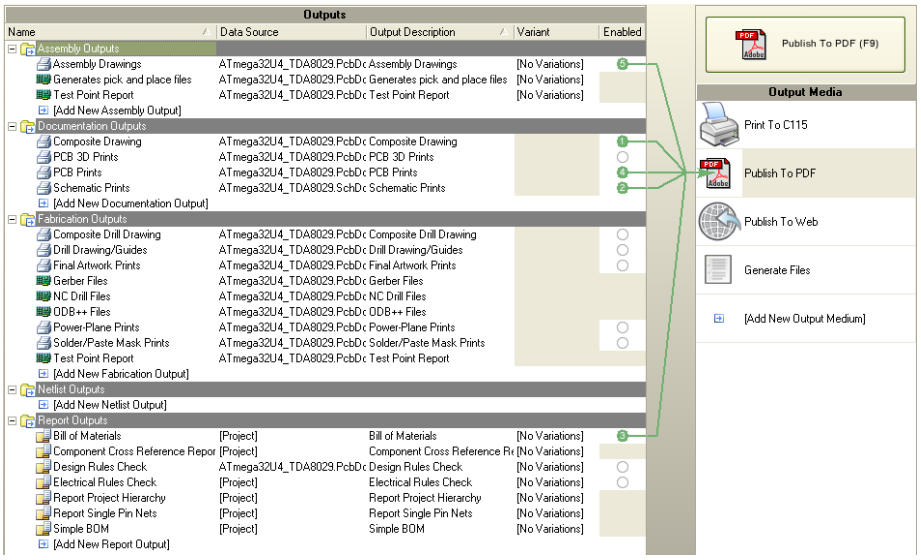



Figura 4- 23: Configuración para extracción de documentos en pdf.

Una vez seleccionados se pulsa el botón "*Publish To PDF (F9)*" que se encuentra en la parte superior del menú de la derecha. Todos estos documentos se añaden a un mismo documento PDF donde se encuentra recogida toda la información deseada.

Seguidamente se generan los archivos de construcción, los Gerber de las capas superior, inferior y contorno, y un archivo de texto en formato *.txt* que contiene toda la información necesaria sobre los taladros y agujeros existentes en el PCB. Para ello se toma la opción "*Generate Files*" del menú de la derecha y se seleccionan los archivos deseados, en este caso, "*Gerber Files*" y "*NC Drill Files*".

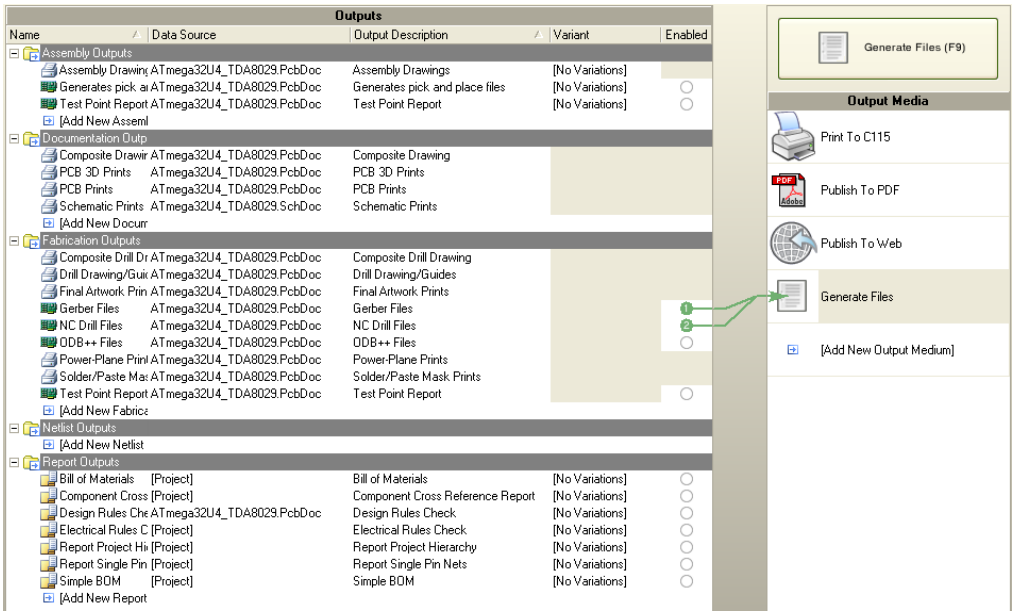



Figura 4- 24: Configuración para extracción de los archivos de fabricación.

Antes de generar estos archivos se deben configurar con las condiciones que se consideran óptimas para este tipo de PCB y con las restricciones que impone la fresadora con la que se construirá el PCB (ProtoMat S100). Para ello clic botón derecho sobre el nombre y "*configure*".

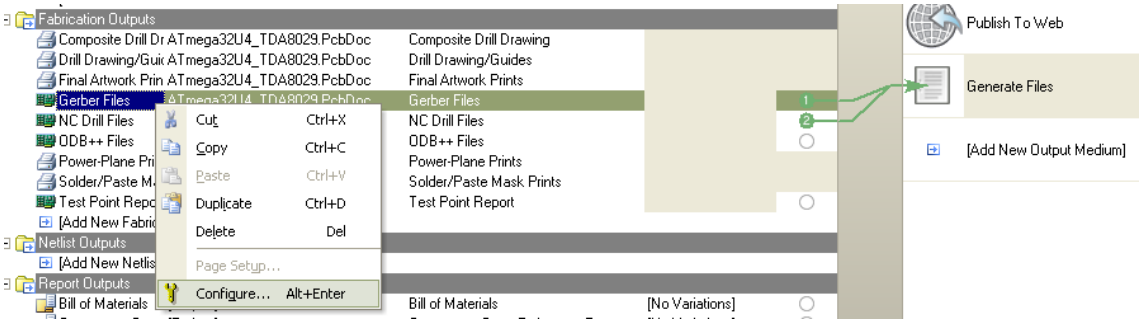



Figura 4- 25: Acceso a la configuración de los archivos Gerber.

La configuración de los archivos Gerber:

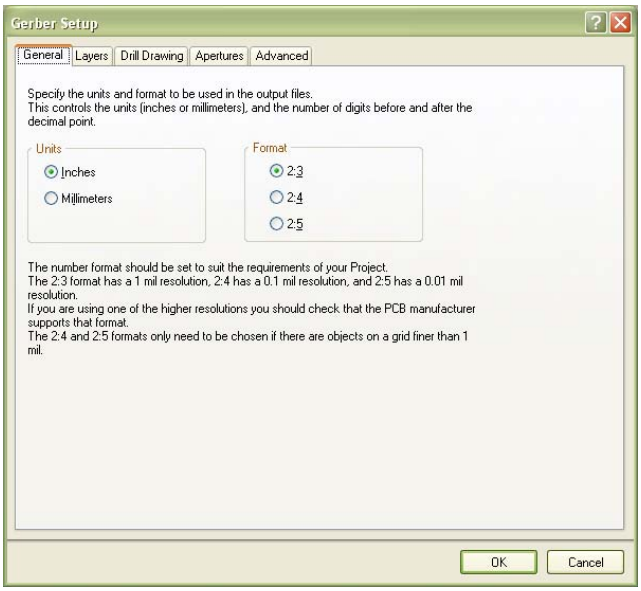



Figura 4- 26: Configuración Gerber (*General*).

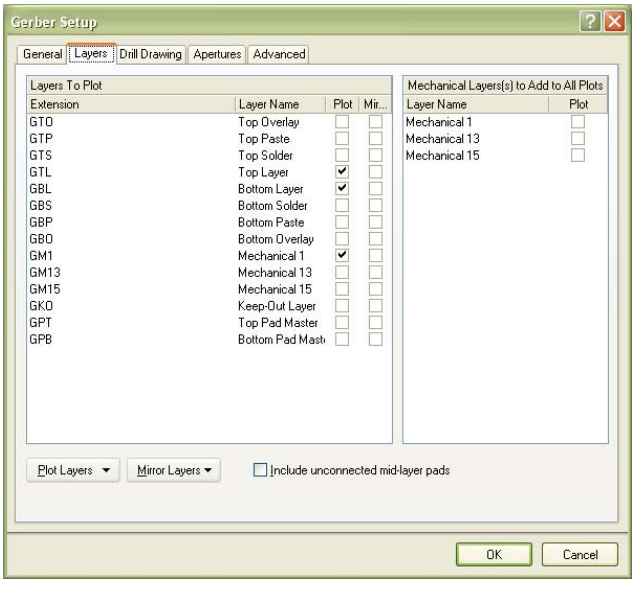



Figura 4- 27: Configuración Gerber (*Layers*).

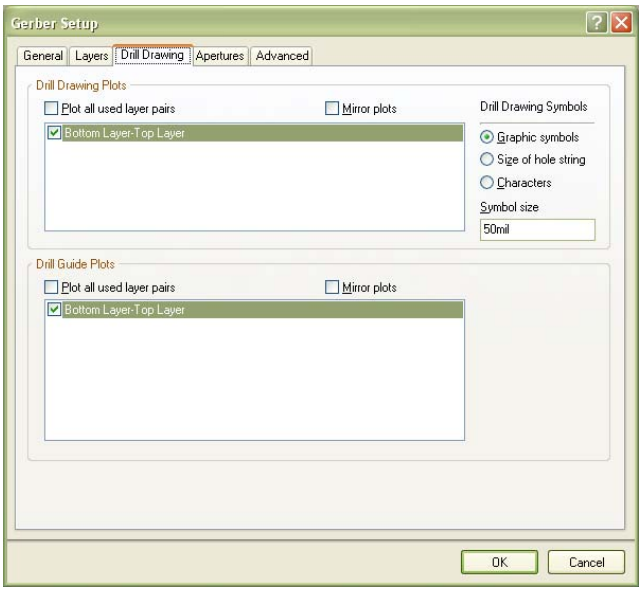



Figura 4- 28: Configuración Gerber (*Drill Drawing*).

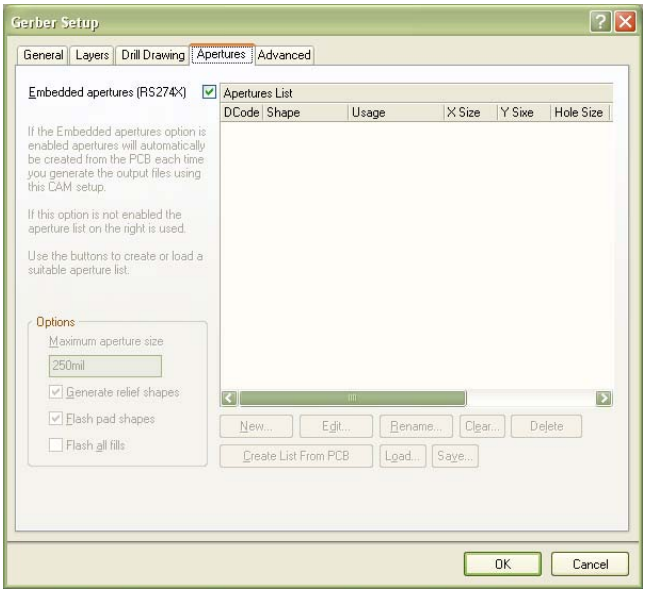



Figura 4- 29: Configuración Gerber (*Apertures*).

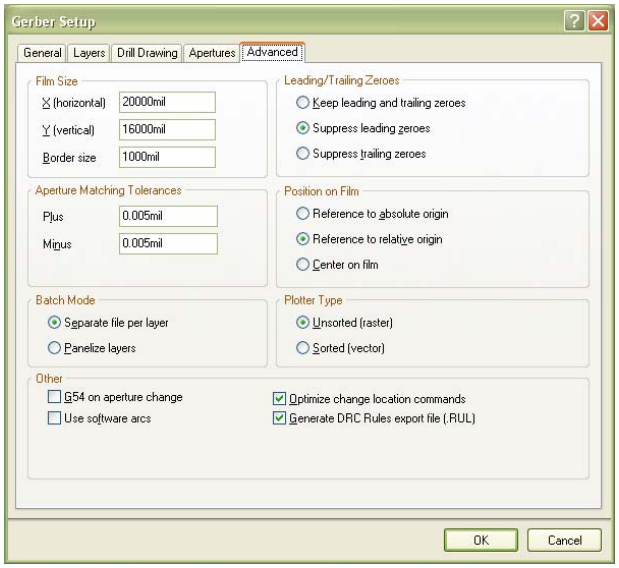



Figura 4- 30: Configuración Gerber (*Advanced*).

La configuración del archivo de taladros:

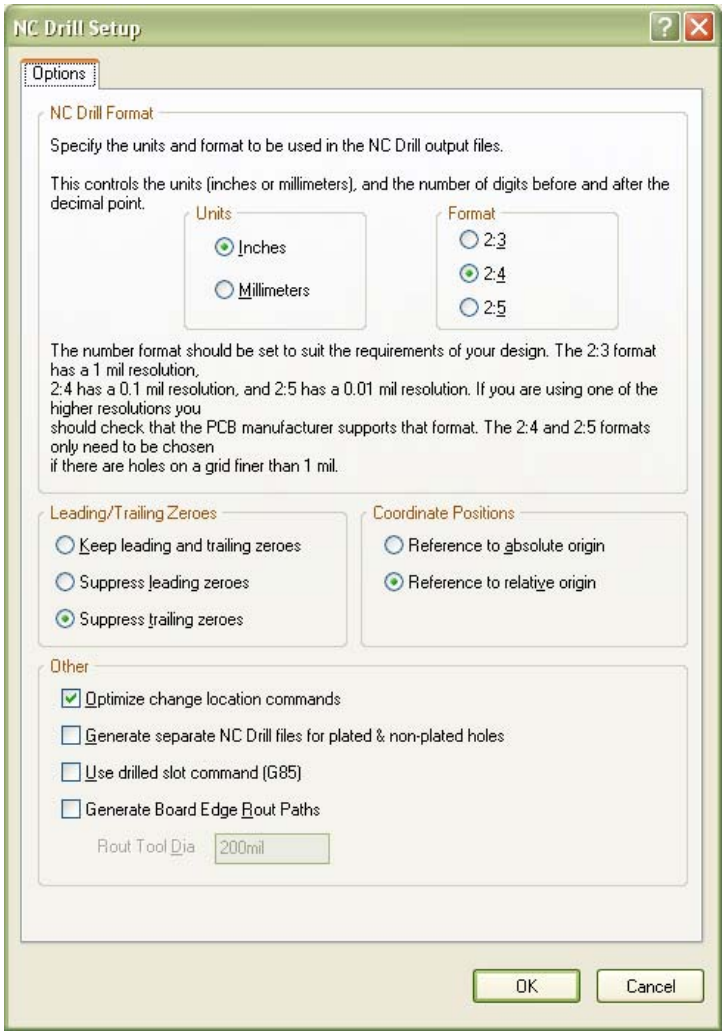



Figura 4- 31: Configuración *NC Drill*.

Una vez finalizada la configuración, como en el caso anterior, para generar los archivos se pulsa el botón "*Generate Files (F9)*" que se encuentra en la parte superior del menú de la derecha. Este es el resultado obtenido:

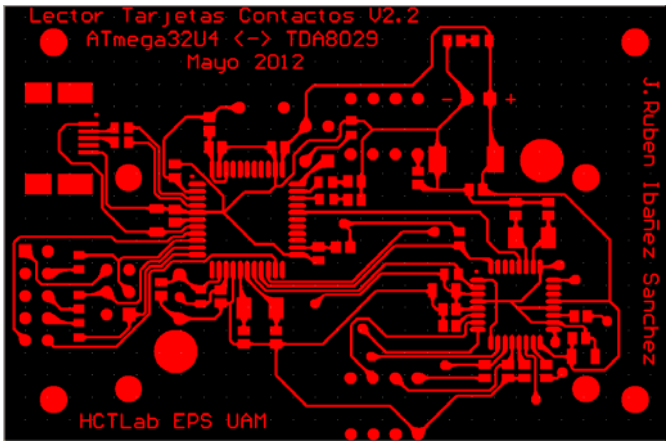



Figura 4- 32: Gerber capa superior (*Top*).

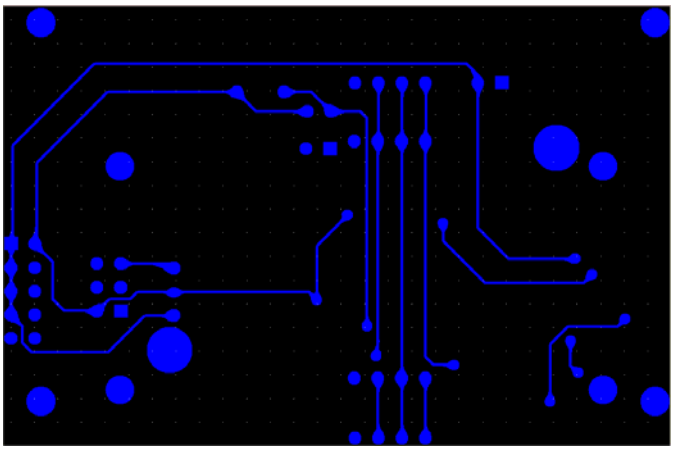


Figura 4- 33: Gerber capa inferior (*Bottom*).

Figura 4- 34: Gerber capa contorno (*Mechanical 1*).

Y el contenido del archivo de taladros:

```
M48
;Layer_Color=9474304
;FILE_FORMAT=2:4
INCH,LZ
;TYPE=PLATED
T1F00S00C0.0280
T2F00S00C0.0300
T3F00S00C0.0394
T4F00S00C0.1260
T5F00S00C0.1280
T6F00S00C0.1969
%
T01
X109921Y01248
X115984Y012165
X11811Y011063
X118504Y009803
X121811Y009409
X121339Y015394
X126929Y013898
X127638Y013228
X129055Y011378
X126732Y010433
X127047Y009094
X125866Y007874
X117283Y015748
T02
X109921Y011504
Y013504
T03
X103032Y010528
X104031
Y011528
X103032
Y012528
X104031
Y013528
X103032
Y014528
X104031
X106677Y013693
Y012693
Y011693
X107677
Y012693
Y013693
X117583Y008858
```

```
X118583
X119583
X120583
X120598Y006339
X119598
X118598
X117598
X117583Y018858
X118583
X119583
X120583
X120598Y021339
X119598
X118598
X117598
X116583Y020118
X115583
X115535Y018543
X116535
X114598Y020945
X112598
X122819Y021339
X123819
T04
X107638Y008346
X12811
Y017795
X107638
T05
X104291Y007874
X130315
Y023858
X104291
T06
X109764Y010039
X126142Y018583
M30
```

Estos tres archivos son los que se deben pasar a la fresadora que construirá el PCB. El resultado obtenido es el siguiente:

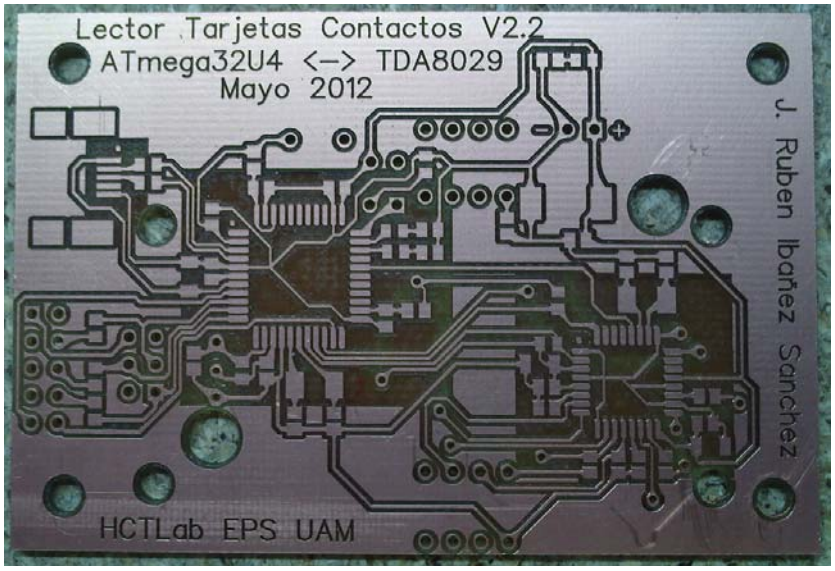



Figura 4- 35: Fotografía de la cara superior PCB construido.

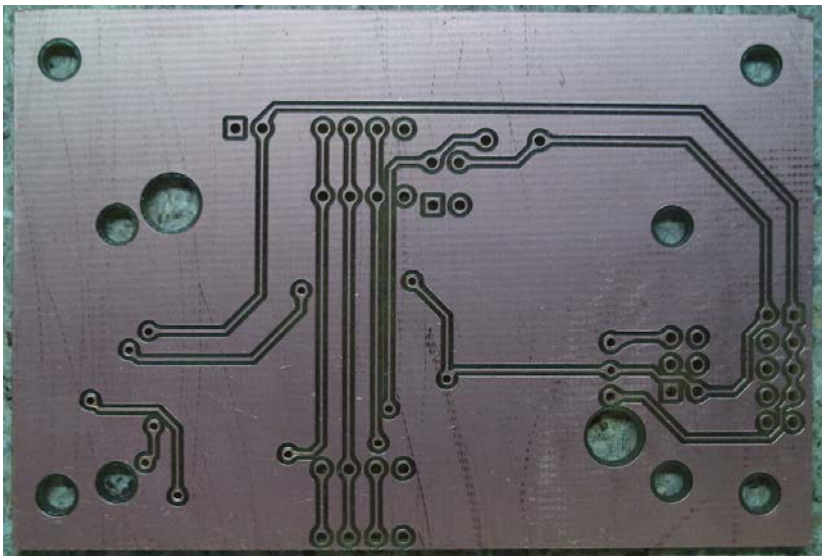


Figura 4- 36: Fotografía de la cara inferior PCB construido.

Con el PCB ya construido y tras comprobar que no tiene ningún error de fabricación ni ninguna pista abierta o cortocircuitada, se da comienzo a la fase de montaje.

## 4.5 Ensamblaje del prototipo

Con todos los componentes disponibles y el PCB fabricado se comienza la fase de montaje. Antes de comenzar se impregna el PCB con una sustancia que evita que se oxide y facilita la soldadura ya que contiene *flux*.

Se dispone de una estación de soldadura marca JBC que dispone de todas las herramientas necesarias para llevar a cabo este proyecto.

Con todo preparado se comienza el montaje. El orden seguido a la hora de soldar los componentes es el siguiente. Primero, aquellos que tienen una mayor dificultad de acceso por lo pequeño de sus pines o su ubicación, en este caso, el ATmega32U4, el TDA8029, el conector miniUSB y los osciladores.

A continuación, todas las resistencias, condensadores, bobinas y leds seguidos de los conectores de la cara superior del PCB. Antes de soldar el conector de tarjetas en la cara inferior se deben crear las vías existentes en la placa, ya que la fresadora utilizada realiza los taladros pero no los metaliza, por tanto se debe conectar cara superior con inferior mediante un pequeño trozo de conductor. Una vez conectadas y comprobadas todas las vías se procede a la instalación del conector de tarjetas.

Del conector de tarjetas utilizado merece la pena destacar un detalle. Se trata de un conector en el que las patillas que hacen contacto con los pines de la tarjeta se encuentran recogidos dentro de la estructura del conector. Al insertar una tarjeta, esta empuja el mecanismo que controla la posición de las patillas, haciendo que solo se produzca contacto con los pines de la tarjeta cuando estos están ubicados en el lugar preciso. Da esta forma se evita el estrés mecánico que pueda sufrir la tarjeta y roces y falsos contactos que puedan sufrir los pines del chip.

Siguiendo estos pasos se termina la instalación de todos los componentes. Se obtiene el siguiente resultado:

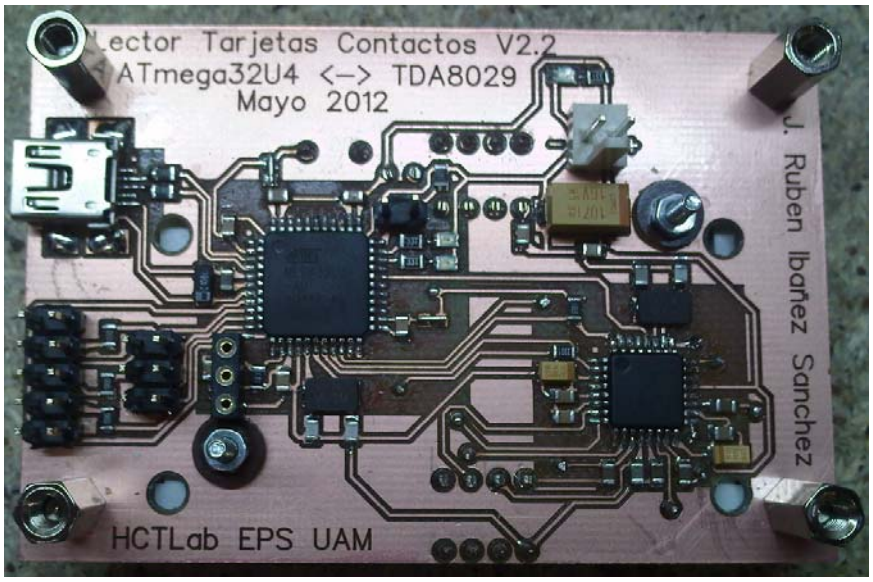



Figura 4- 37: Fotografía de la cara superior del PCB tras el montaje de componentes.

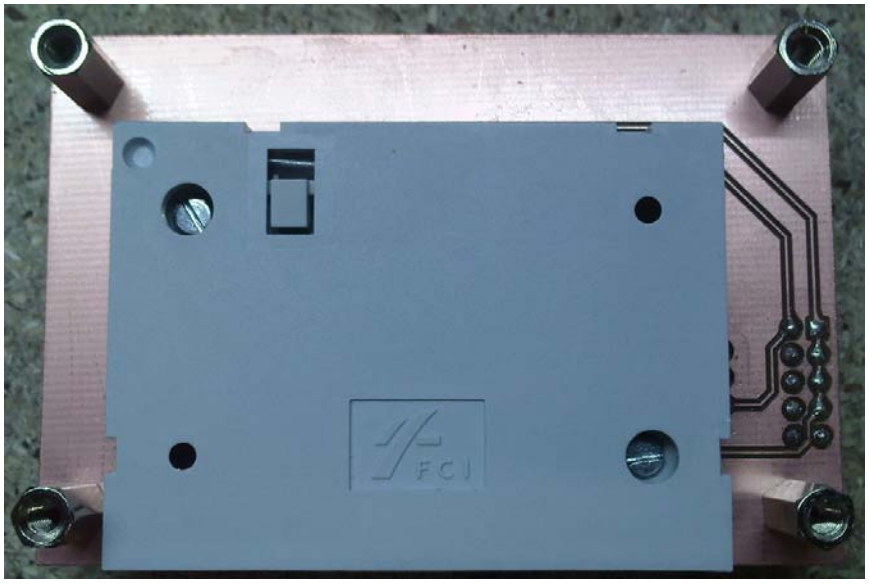



Figura 4- 38: Fotografía de la cara inferior del PCB tras el montaje de componentes.

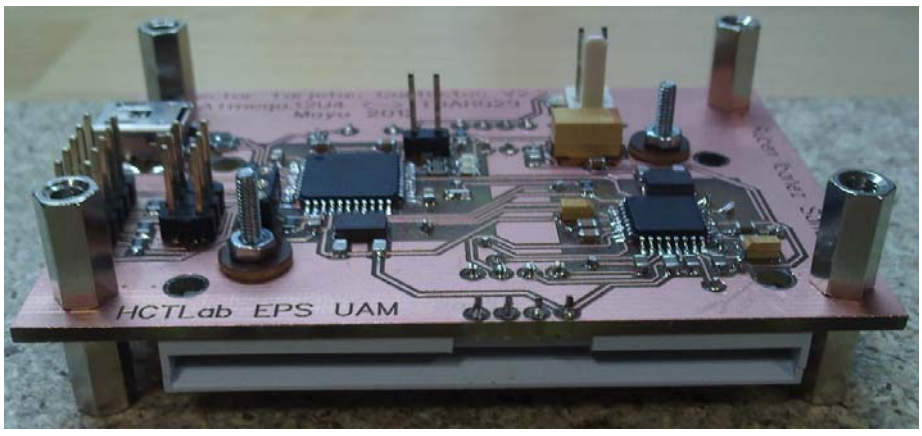



Figura 4- 39: Fotografía del lector de tarjetas terminado.

Por último se muestra una aclaración de donde están finalmente situados cada uno de los puntos importantes del lector de tarjetas:

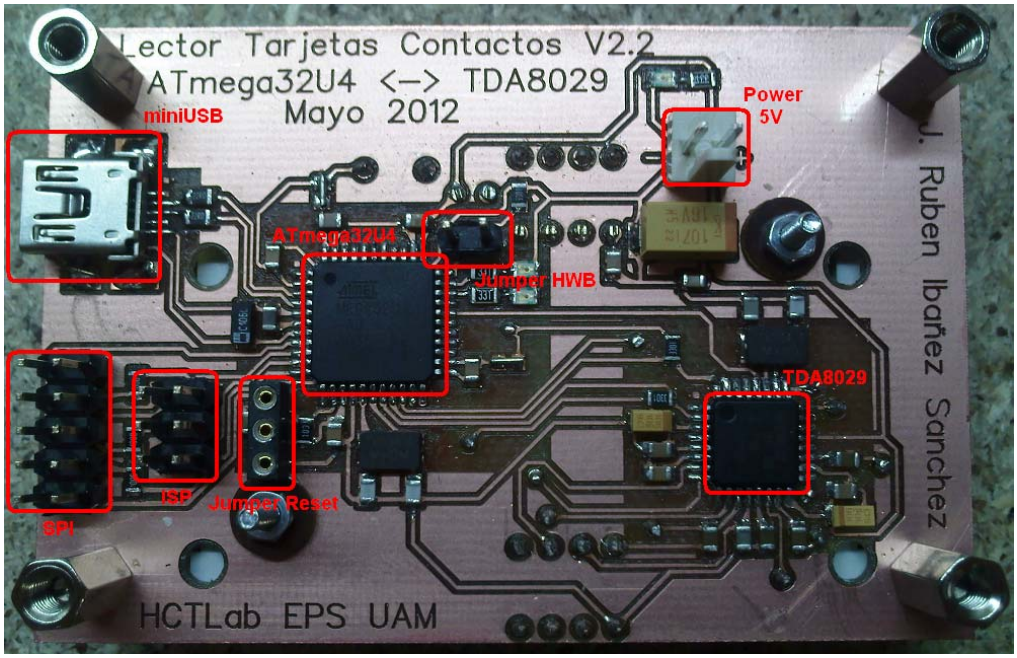



Figura 4- 40: Componentes principales del lector de tarjetas.

En los siguientes apartados se dará una visión más específica de la utilidad y forma de uso de cada uno de estos elementos.

## 4.6 Software lector de tarjetas

Una vez terminada la construcción del prototipo llega el momento de su programación. Se debe desarrollar una aplicación que permita al microcontrolador tomar el control del dispositivo completo y hacer que este realice las tareas para el que ha sido diseñado.

La aplicación sigue un sencillo esquema de funcionamiento que se puede resumir con el siguiente diagrama:

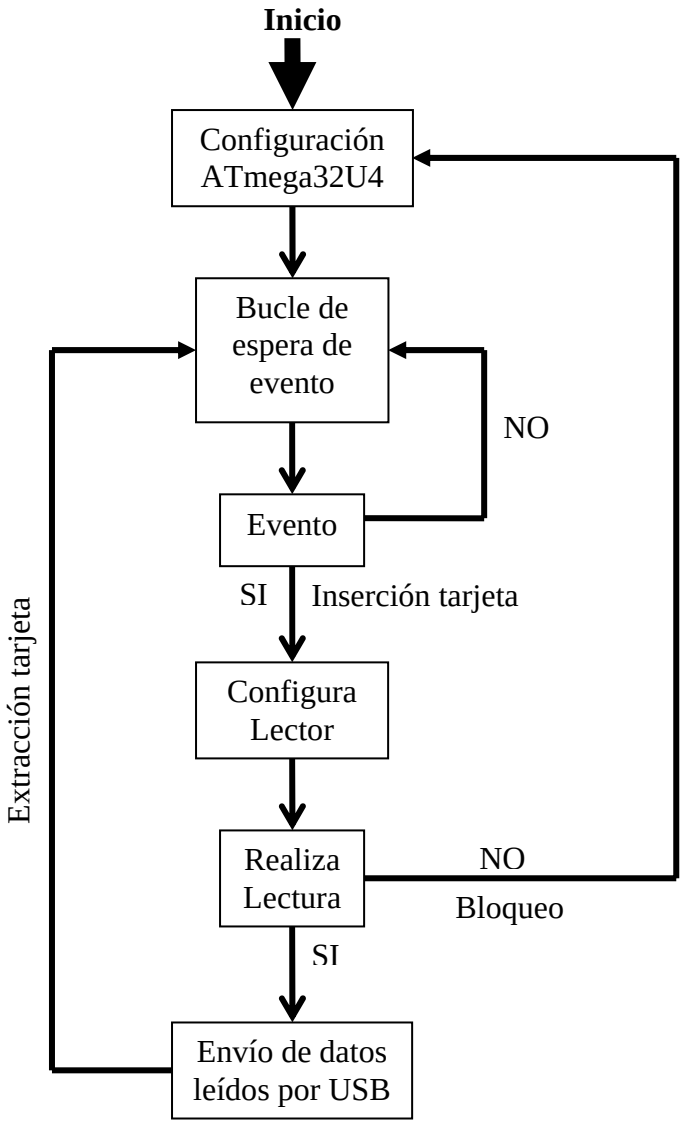



Figura 4- 41: Diagrama de flujo del software instalado en el lector de tarjetas.

Cada uno de los bloques hace uso de diferentes funciones creadas para realizar cada una de las tareas. Estas funciones se muestran con mayor detalle posteriormente.

Para llevar a cabo la tarea de programación se utiliza un software del fabricante Atmel, el "*AVR Studio 5.0*".

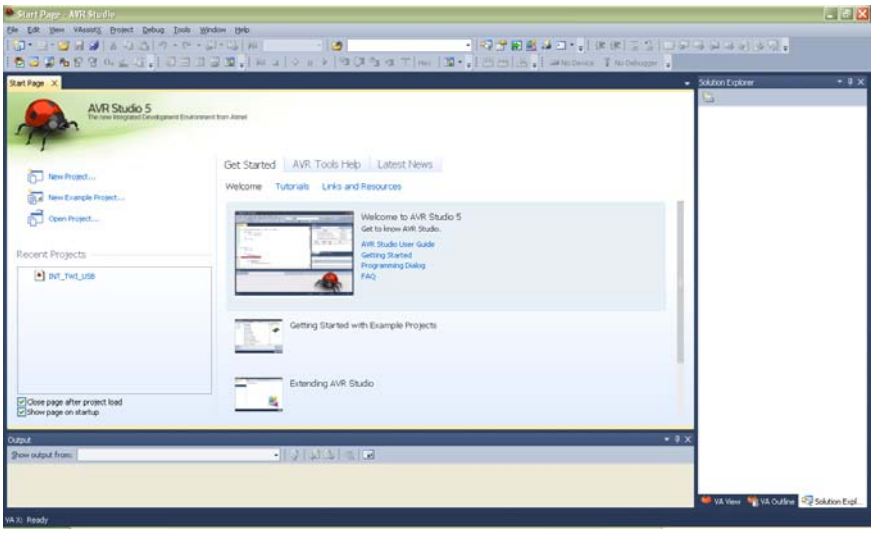



Figura 4- 42: Interfaz inicial de *AVR Studio 5.0*.

La utilización de este software viene determinada por la elección del microcontrolador de esta misma marca comercial, ya que incluye librerías y múltiples herramientas para la programación, depuración, compilación y modificación de las memorias y opciones del microcontrolador. A su vez simplifica mucho el uso de programadores externos específicos para microcontroladores de Atmel.

La programación del microcontrolador se realiza en código C, haciendo uso de librerías estándar y especificas de de los microcontroladores AVR. Las librerías específicas se pueden consultar en [12].

Una vez compilado todo el código de la aplicación, si éste está libre de errores, se generan varios archivos fruto de esta compilación. El más interesante de todos es aquel con el mismo nombre del proyecto y una extensión ".hex" ya que este es el archivo que contiene la aplicación de usuario en un lenguaje entendible por el microcontrolador. Es con este archivo con el que se debe programar el microcontrolador para que ejecute la aplicación creada.

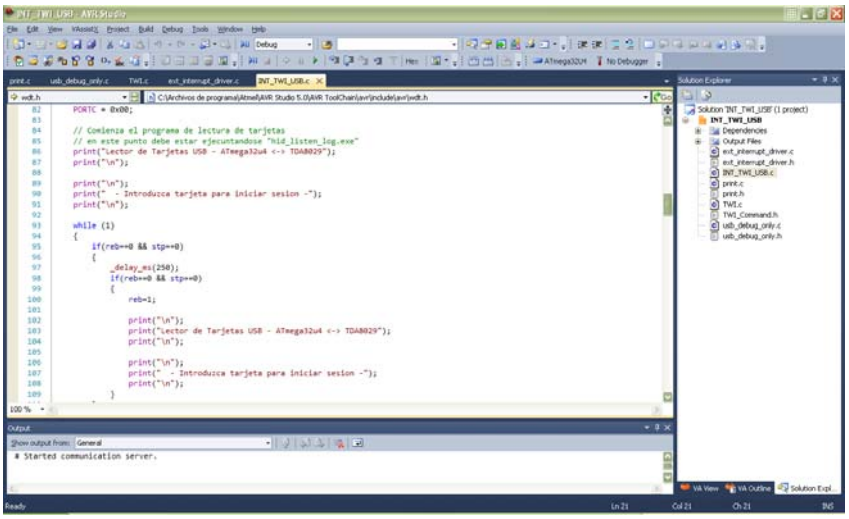


Figura 4- 43: Interfaz de *AVR Studio 5.0* para el editor de código.

A continuación se describen las principales funciones que componen el programa de usuario:

- **INT_TWI_USB.c**: archivo principal del programa, incluye el “main” y unifica toda la estructura de funciones y variables. Se encarga de llevar el control de la ejecución del programa completo, controlando cuándo se ejecuta cada una de las funciones que realizan cada una de las tareas. También se encarga del control de errores y bloqueos que puedan aparecer durante el funcionamiento del programa.

- **ext_interrupt_driver.c:** archivo que contiene las funciones que configuran y habilitan/deshabilitan las interrupciones externas del microcontrolador. Se acompaña de un archivo de cabecera “ext_interrupt_driver.h” donde se definen unas macros que facilitan el uso de las funciones que implementa. Estas son las funciones que se encuentran:

```
/*****************************************************************************
*
 * Esta función configura las interrupciones externas de forma que sean sensibles
 * a flanco de subida o bajada y solicita una interrupción.
 *
 * Parámetros:
 *     INT_NO Interrupción que se quiere configurar.
 *     INT_MODE     Modo de detección de INT_NO.

*****************************************************************************
/
void Configure_Interrupt(uint8_t INT_NO, uint8_t INT_MODE);
```

```
/*******************************************************
 * Esta función habilita las interrupciones externas.
 * Parámetros:
 *     INT_NO Interrupción que se desea habilitar.
 *******************************************************/
void Enable_Interrupt(uint8_t INT_NO);
```

```
/*******************************************************
 * Esta función deshabilita las interrupciones externas.
 * Parámetros:
 *     INT_NO Interrupción que se desea deshabilitar.
 *******************************************************/
void Disable_Interrupt(uint8_t INT_NO);
```

Estas interrupciones son utilizadas por el chip lector de tarjetas para indicar al microcontrolador que se ha producido un evento, en este caso, una inserción de tarjeta y, por tanto, se debe poner en marcha la rutina de lectura.

- **TWI.c:** archivo que contiene las funciones necesarias para la comunicación TWI ($I^2C$) entre el microcontrolador (maestro) y cualquier dispositivo conectado al bus (esclavos), en este caso, el chip lector de tarjetas. Se acompaña de un archivo de cabecera, “TWI_Command.h”, donde se incluyen unas macros con todos los comandos que acepta el chip lector de tarjetas, facilitando de esta forma el uso de las funciones de comunicación. También incluye una macro con el valor de un registro del microcontrolador que es el encargado de establecer la velocidad máxima del bus. Las funciones que contiene son:

```
/*************************************************************************
 Función para configurar el maestro TWI a su estado de espera inicial.
 Habilitar las interrupciones globales después de la inicialización.
*************************************************************************/
void TWI_Master_Initialise(void);
```

```
/*************************************************************************
* Función para la escritura/lectura de datos mediante TWI.
*
* Parámetros:
*    address: dirección esclavo con el que se desea comunicar.
*    num_data_TX: número de bytes de datos que se envían.
*    *data: puntero al array de datos para enviar.
*    num_data_RX: número de bytes de datos que se van ha leer.
*
* La función devuelve un puntero al array de datos leídos "read_data" del bus
* I2C.
*
* Esta función solo puede ser utilizada por el dispositivo maestro.
*************************************************************************/
unsigned char* TWI_TX_RX(char address, char num_data_TX, unsigned char *data,
char num_data_RX);
```

El registro antes mencionado, denominado "*TWI Bit Rate*" o TWBR, controla la velocidad de la línea de reloj SCL mediante la siguiente fórmula:

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

De esta manera, imponiendo el valor deseado a "*SCL frequency*", se obtiene el valor del registro despejando de la formula TWBR. El valor de TWPS, "*TWI Prescaler Bits*", es cero. A modo de ejemplo estos son los valores utilizados durante el proceso de pruebas y ajustes:

```
TWI_TWBR = 0x7D -> Vel 60 kHz (máximo)
TWI_TWBR = 0x98 -> Vel 50 kHz
TWI_TWBR = 0xA9 -> Vel 45 kHz
TWI_TWBR = 0xC0 -> Vel 40 kHz (Óptimo)
TWI_TWBR = 0xDC -> Vel 35 kHz
```

Con estas funciones se controla todo el tráfico de datos entre el ATmega32U4 y el TDA8029.

- **usb_debug_only.c:** archivo que contiene todas las funciones y estructuras de datos necesarias para la configuración y correcto funcionamiento del USB. Su cometido es configurar el USB de tal forma que un PC cualquiera detecte el lector diseñado como un dispositivo de interfaz humana o HID. Junto a este archivo tenemos otro de tipo cabecera, "usb_debug_only.h" que contiene variables y macros para hacer más claro y sencillo su funcionamiento.

- **print.c:** por último este archivo contiene las funciones que dan formato válido a los datos leídos por las funciones anteriores y los envía por el USB con dirección al PC. De nuevo viene con un archivo de cabecera "print.h" que contiene macros para facilitar su uso. Sus principales funciones son:

```
/**************************************************************************
Función para el envío de cadenas de caracteres por USB. Da el formato
necesario a los datos para su correcta transmisión.
**************************************************************************/
void print_P(const char *s);
```

```
/**************************************************************************
Función para la transmisión de valores hexadecimales de 8 bits por USB.
**************************************************************************/
void phex(unsigned char c);
```

```
/**************************************************************************
Función para la transmisión de valores hexadecimales de 16 bits por USB.
**************************************************************************/
void phex16(unsigned int i);
```

Estos dos últimos archivos pertenecen a un proyecto de software libre publicado en la Web de "*PJRC: Electronic Projects*" [43] y no han sido modificados salvo algún comentario aclaratorio y una pequeña personalización en el archivo usb_debug_only.c, donde se añaden las siguientes líneas:

```
#define STR_MANUFACTURER    L"HCTLab - Ruben"
#define STR_PRODUCT         L"USB Smart Card Reader"
```

El proyecto estudiado, llamado “Teensy”, se basa en microcontroladores de la marca Atmel y se trata de un pequeño PCB con un ATmega32U4 o un AT90USB1286 con los elementos mínimos para hacerlos funcionar, una conexión USB y todos los demás pines con un fácil acceso tipo Arduino.

El software desarrollado para ese proyecto, debido a que se trata del mismo microcontrolador, resulta de gran utilidad y su utilización es inmediata sobre el software que se desarrolla para este PFC. Puesto que el cometido de las funciones adoptadas es el de configurar y hacer funcionar el USB, tareas fuera de las bases de este PFC, no hay necesidad de modificarlas dado que su funcionamiento es del todo correcto y mas que suficiente para finalizar el software que se esta diseñando.

Para ver con detalle el código perteneciente a cada una de las funciones mencionadas se debe acudir al Anexo B.

## 4.7 Software PC

Este PFC incluye un software para el PC donde se quiera hacer uso del lector de tarjetas.

Se trata de un programa que monitorea los dispositivos HID con una configuración determinada cuando se conectan al PC, mostrando por pantalla los datos que se transmiten desde el lector al PC por medio de la conexión USB.

Este programa procede del proyecto libre anteriormente mencionado (*Teensy” de “PJRC: Electronic Projects*). Ya que se distribuye con los archivos fuente, el software original ha sido modificado y adaptado para un uso especifico.

Las modificaciones realizadas sobre el software buscan crear un archivo de registro que guarde toda la información que el lector transmite hacia el PC mientras se encuentra funcionando. Además en cada lanzamiento del programa se debe retomar el mismo archivo de registro añadiendo una nueva entrada con la fecha y hora del sistema que lo ejecuta.

Es importante mencionar que este programa no solo proporciona un buen complemento al lector de tarjetas, sino que ha resultado extremadamente útil durante la fase de desarrollo y prueba del software para el prototipo, funcionando como una herramienta de depuración de código.

Esta aplicación esta compuesta por 2 archivos fuente más un *makefile*:

- **hid_listen.c:** archivo principal del programa. Contiene el código necesario para tomar el control de la ejecución del programa, incluyendo el "*main*". Este es el archivo que se ha modificado para obtener el resultado buscado.

- **rawhid.c:** archivo de código que incluye la definición de todas las funciones y estructuras necesarias para trabajar con los puertos USB. Todas las funciones específicas utilizadas en "hid_listen.c" se encuentran aquí desarrolladas. Este archivo se acompaña de un archivo de cabecera, "rawhid.h", que incluye los prototipos de todas las funciones presentes en el archivo "rawhid.c".

- **Makefile:** archivo de texto utilizado para llevar a cabo la compilación de los archivos fuente que forman este programa. Incluye todas variables, reglas e instrucciones necesarias para compilar el código de programa y generar el archivo ejecutable de la aplicación para un sistema operativo específico.

# 5 Conclusiones y trabajos futuros

## 5.1 Conclusiones

Como resultado de este proyecto se obtiene un lector de tarjetas chip con conexión USB capaz de comunicarse con un amplio rango de tarjetas que posean esta tecnología y realizar cualquier operación de lectura deseada.

Gracias a su reducido tamaño y ligereza se puede trasportar o adaptar a un sistema fácilmente y, si a esto se le añade su conexión USB por la que se vuelcan todos los datos obtenidos de la lectura de una tarjeta, resulta mas sencillo si cabe su introducción en un sistema anfitrión que necesite esta capacidad operativa.

Junto al lector se desarrolla una aplicación software que, instalada en el microcontrolador del dispositivo, permite al lector llevar a cabo las tareas para las que se diseñó. Este software está escrito en código C, lo que permite realizar modificaciones sobre él sin ser necesarios conocimientos en lenguajes específicos para microcontroladores.

Además de llevar a cabo con éxito el objetivo principal del proyecto también se han conseguido los objetivos secundarios. Se alcanza un buen nivel de manejo sobre el nuevo software para diseño de circuitos impresos *Altium Designer* y se consigue que el diseño esté centralizado sobre el microcontrolador deseado de forma que se empieza a conocer la forma y peculiaridades en su funcionamiento.

En resumen, se ha diseñado, fabricado y programado un lector de tarjetas chip que cumple sin problemas su cometido principal, además de presentar una buena fiabilidad, bajo consumo, un tamaño reducido y facilidad en su manejo y posible evolución, tanto hardware como software. Además se alcanza un buen nivel de manejo con la nueva herramienta *Altium Disigner* tras su estudio y utilización durante la realización de este proyecto.

## 5.2 Trabajos futuros

Como trabajo a corto plazo se propone añadir un nuevo módulo que capacite al lector de la tecnología necesaria para poder realizar lecturas de tarjetas sin contactos. Esta mejora convertiría este lector en un lector de cualquier tipo de tarjeta inteligente, con un amplio campo de aplicación.

En este mismo sentido, es importante añadir que en el momento del diseño que se presenta ya se tiene en cuenta esta posible evolución, por ello se dota al prototipo de un microcontrolador con suficiente capacidad operativa y se añade un conector para la conexión y comunicación mediante SPI de un nuevo módulo.

Como trabajos posibles a largo plazo se propone convertir este lector en un dispositivo autónomo y portátil de lectura de tarjetas inteligentes. Para ello se debería unir el lector a una *Single-board computer* o SBC, es decir, un completo PC construido sobre una sola placa. De esta forma se obtendría un sistema completo para la adquisición y procesado de datos, además se dispondría de diferentes tecnologías de comunicación como *wifi* y *Ethernet*, y se podrían añadir elemento como pantallas táctiles, dispositivos de almacenamiento de datos, etc. Para finalizar si todos estos elementos se alimentan con una batería recargable con capacidad suficiente y se conjunta todo dentro de una estructura, se conseguiría un dispositivo portátil con un amplio rango de posibilidades.

Por otro lado, la evolución del software que porta el lector presentado en este proyecto puede seguir dos caminos. Se puede evolucionar el software ya existente para mejorar el rendimiento o añadir alguna nueva funcionalidad al lector presentado y se debe evolucionar siempre de forma conjunta a las mejoras hardware que se añadan, ya que el microcontrolador siempre será el sistema central que unifica y controla todos los posibles módulos que formen el lector.

# Bibliografía


[1] AENOR. *Tarjetas de identificación*. *UNE 71-107-91, UNE 71-1071-94, UNE 71-107-3*. Disponibles en la Biblioteca de la EPS, UAM.

[2] Altium. *Altium Designer* .Available from: http://products.live.altium.com/ *[11 de diciembre de 2012]*

[3] Altium Designer footprint libraries. Available from: http://www.altium.com/community/libraries/altium-designer-libraries/altium-designer-footprint-libraries/en/altium-designer-footprint-libraries_home.cfm *[11 de diciembre de 2012]*

[4] Atmel. *AT83C24: Smart Card Reader Interface with Power Management*. Available from: http://www.atmel.com/dyn/products/product_card.asp?part_id=2502#dataSheets *[21 de febrero de 2012]*

[5] Atmel. *ATmega32U4: 8-bit AVR Microcontroller with 32K Bytes of ISP Flash and USB Controller*. Available from: http://www.atmel.com/Images/doc7766.pdf *[11 de diciembre de 2012]*

[6] Atmel. *AVR Hardware Design Considerations*. AVR042; [Application Note]. Available from: http://www.atmel.com/Images/doc2521.pdf *[11 de diciembre de 2012]*

[7] Atmel. *AVRISP mkII can program all AVR 8-bit RISC microcontrollers with ISP interface*. Available from: http://www.atmel.com/tools/avrispmkii.aspx *[11 de diciembre de 2012]*

[8] Atmel. *AVR Studio 5.0, now Atmel Studio 6*. Available from: http://www.atmel.com/microsite/atmel_studio6/ *[11 de diciembre de 2012]*

[9] Atmel. *Flip, Supports in-system programming of flash devices through RS232, USB or CAN*. Available from: http://www.atmel.com/tools/FLIP.aspx *[11 de diciembre de 2012]*

[10] Atmel. *Using External Interrupts for megaAVR Devices*. AVR1200; [Application Note]. Available from: http://www.atmel.com/Images/doc8468.pdf *[11 de diciembre de 2012]*

[11] Atmel. *Using the TWI module as I2C master*. AVR315; [Application Note]. Available from: http://www.atmel.com/Images/doc2564.pdf *[11 de diciembre de 2012]*

[12] *AVR Library Reference.* Available from: http://www.nongnu.org/avr-libc/user-manual/modules.html *[11 de diciembre de 2012]*

[13] Bit4it – the Best Information Technology fo(u)r IDentification. *Lectores para PC.* Available from: http://www.bit4id.com/online_store_es_pt/Bit4id_Tienda_es/index.php?gen=10 *[11 de diciembre de 2012]*

[14] C3PO S.A. Seguridad e Identificación digital. *Lectores para PC.* Available from: https://www.c3po.es/products-page/lectores-para-pc/*[11 de diciembre de 2012]*

[15] *CardWerk's - Smarter Card Solutions*. Available from : http://www.cardwerk.com/ *[11 de diciembre de 2012]*

[16] Chhabra, Tina and Chindaphorn, Piya. *Smart Cards*. *Department of Computer Engineering*. May 17, 2004, Santa Clara University. Available from: http://www.cse.scu.edu/~jholliday/COEN150Sp04/projects/Smart_Cards.doc *[11 de diciembre de 2012]*

[17] García, Vicente. *Introducción al I2C en Arduino*. Available from: http://www.hispavila.com/3ds/atmega/intro_i2c.html *[11 de diciembre de 2012]*

[18] Guthery, Scott B. and Jurgensen, Timothy M. *Smart card developers kit.* Indianapolis: Macmillan Technical, 1998.

[19] I2C-Bus.org. *I2C Bus*. Available from: http://www.i2c-bus.org/i2c-bus/ *[11 de diciembre de 2012]*

[20] ISO/IEC, *Identification cards* in *ISO/IEC 7816-1:2011, ISO/IEC 7816-2:2007, ISO/IEC 7816-3:2006, ISO/IEC 7816-4:2005, ISO/IEC 7816-5:2004, ISO/IEC7816-6:2004, ISO/IEC 7816-7:1999, ISO/IEC 7816-8:2004, ISO/IEC7816-9:2004, ISO/IEC 7816-10:1999, ISO/IEC 7816-11:2004, ISO/IEC7816-12:2005, ISO/IEC 7816-13:2007, ISO/IEC 7816-15:2004*. Available from: http://www.iso.org/iso/home.htm *[11 de diciembre de 2012]*

[21] JMN. *Probando el ATmega(32/16)u4*. C.I.r.E. - Club de Informática, robótica y Electrónica. Available from: http://webdelcire.com/wordpress/archives/269 *[11 de diciembre de 2012]*

[22] Ladino, Enrique. *Tarjetas Inteligentes*. Monografias.com. Available from: http://www.monografias.com/trabajos10/tarin/tarin.shtml *[11 de diciembre de 2012]*

[23] *Manual de Altium DXP*. Disponible en Internet. Manual libre, no posee derechos de autor.

[24] Maxim Integrated. *73S1215F: 80515 System-on-Chip with USB, ISO 7816/EMV, PINpad, and More.* Available from: http://datasheets.maximintegrated.com/en/ds/73S1215F.pdf *[11 de diciembre de 2012]*

[25] Maxim Integrated. *DS8113: Smart Card Interface*. Available from: http://datasheets.maximintegrated.com/en/ds/DS8113.pdf *[11 de diciembre de 2012]*

[26] Medaglia, Diego. *Tarjetas Inteligentes*. Monografias.com. Available from: http://www.monografias.com/trabajos16/tarjetas-inteligentes/tarjetas-inteligentes.shtml#SISTEMA *[11 de diciembre de 2012]*

[27] Ministerio de Industria, Energía y Turismo. *Introducción en el uso de tarjetas inteligentes*. Available from: https://zonatic.usatudni.es/es/aprendizaje/aprende-sobre-el-dnie/57-dnie-tecnico/204-introduccion-en-el-uso-de-tarjetas-inteligentes.html *[11 de diciembre de 2012]*

[28] NXP Semiconductors. *Description of the TDA8029 I2C Demo Board.* UM1033811. January 2011; Rev. 1.0. Available from: http://www.nxp.com/documents/user_manual/UM10338.pdf *[11 de diciembre de 2012]*

[29] NXP Semiconductors. *I2C-bus specification and user manual*. UM10204. 13 February 2012; Rev. 4. Available from: http://www.nxp.com/documents/user_manual/UM10204.pdf *[11 de diciembre de 2012]*

[30] NXP Semiconductors. *Smart Card reader application with TDA8029 Mask 06 and Mask 07.* AN10207; [Application Note]. 2 February 2011; Rev. 1.0. Available from: http://www.nxp.com/documents/application_note/AN10207.pdf *[11 de diciembre de 2012]*

[31] NXP Semiconductors. *TDA8029: Low power single card reader*. 22 February 2005; Rev. 03. Available from: http://www.nxp.com/documents/data_sheet/TDA8029.pdf *[11 de diciembre de 2012]*

[32] NXP Semiconductors. *TDA80xx: contact smart card reader ICs family.* Available from: http://www.nxp.com/products/identification_and_security/reader_ics/contact_smart_card_reader_ics/#products *[11 de diciembre de 2012]*

[33] Paret, Dominique. *El bus I2C de la teoría a la práctica.* Madrid : Paraninfo, D.L. 1995.

[34] Petri, Steve. *An Introduction to SMART CARDS*. Available from: http://artofconfusion.org/smartcards/docs/intro.pdf *[11 de diciembre de 2012]*

[35] Rankl, W.; *Smart card applications design models for using and programming smart cards.* Chichester, West Sussex: Wiley, 2007.

[36] Rankl, W. and Effing, W. *Smart card handbook.* Chichester [etc.]: Wiley & Sons, 2000.

[37] Reusch Elektronik. *U4DIL: AVR USB Module*. 2011-07-12; Rev. 2.0. Available from: http://re.reworld.eu/common/u4dil/U4DIL_Manual.pdf *[11 de diciembre de 2012]*

[38] Robot Electronics - ATMEGA32 Examples. *Ejemplos de comunicación entre ATmega32 y distintos módulos mediante $I^2C$*. Available from: http://www.robot-electronics.co.uk/htm/ATMEGA32_examples.htm#SRF01 *[11 de diciembre de 2012]*

[39] Smart Card Basics; Available from: http://www.smartcardbasics.com/ *[11 de diciembre de 2012]*

[40] STMicroelectronics. *ST7GEME4: Full-speed USB MCU with smartcard firmware and EMV/non-EMV interface*. Available from: http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00159344.pdf *[11 de diciembre de 2012]*

[41] STMicroelectronics. *ST7SCR1E4: 8-bit low-power, full-speed USB MCU with 16-Kbyte Flash, 768-byte RAM, smartcard interface and timer*. Available from: http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00003037.pdf *[11 de diciembre de 2012]*

[42] STMicroelectronics. *ST8024: Smart Card Interface*. Available from: http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00049310.pdf *[11 de diciembre de 2012]*

[43] Stoffregen, Paul and Coon, Robin. *Teensy USB Development Board*. PJRC – Electronic Projects; Available from: http://www.pjrc.com/teensy/ *[11 de diciembre de 2012]*

[44] Truchsess, Wayne. *Effects of Varying I2C Pull-Up Resistors*. Available from: http://dsscircuits.com/articles/effects-of-varying-i2c-pull-up-resistors.html *[11 de diciembre de 2012]*

[45] Wikipedia. *Tarjeta inteligente*. Available from: http://es.wikipedia.org/wiki/Tarjeta_inteligente *[11 de diciembre de 2012]*

# Glosario

| | |
|---|---|
| ***ACK*** | *ACKnowledgement* |
| ***APDU*** | *Application Protocol Data Unit* |
| ***API*** | *Application Programming Interface* |
| ***ATR*** | *Answer To Reset* |
| ***CPU*** | *Central Processing Unit* |
| ***DF*** | *Dedicated File* |
| ***DNI-e*** | *Documento Nacional de Identidad electrónico* |
| ***EEPROM*** | *Electrically Erasable Programmable Read-Only Memory* |
| ***EF*** | *Elementary File* |
| ***EPS*** | *Escuela Politécnica Superior* |
| ***GSM*** | *Global System for Mobile Communications* |
| ***HID*** | *Human Interface Device* |
| ***$I^2C$*** | *Inter-Integrated Circuit* |
| ***I/O*** | *Input/ Output* |
| ***IEC*** | *International Electrotechnical Commission* |
| ***ISO*** | *International Organization for Standardization* |
| ***ISP*** | *In-System Programming* |
| ***JTAG*** | *Joint Test Action Group* |
| ***LRC*** | *Longitudinal Redundancy Check* |
| ***MF*** | *Master File* |
| ***NACK*** | *Negative ACKnowledgement* |
| ***PC*** | *Personal Computer* |
| ***PCB*** | *Printed Circuit Board* |
| ***PCMCIA*** | *Personal Computer Memory Card International Association* |
| ***PIN*** | *Personal Identification Number* |
| ***PTS*** | *Protocol Type Selection* |
| ***PVC*** | *PolyVinyl Chloride* |
| ***RAM*** | *Random-Access Memory* |
| ***RFID*** | *Radio-Frequency IDentification* |
| ***ROM*** | *Read-Only Memory* |
| ***SCL*** | *Serial Clock Line* |
| ***SDA*** | *Serial Data Line* |
| ***SIM*** | *Subscriber Identity Module* |

| | |
|---|---|
| ***SMD*** | *Surface-Mount Device* |
| ***SPI*** | *Serial Peripheral Interface* |
| ***TWI*** | *Two Wire Interface* |
| ***UAM*** | *Universidad Autónoma de Madrid* |
| ***USB*** | *Universal Serial Bus* |

# Anexos

## A. Esquemático y *Layout* del PCB

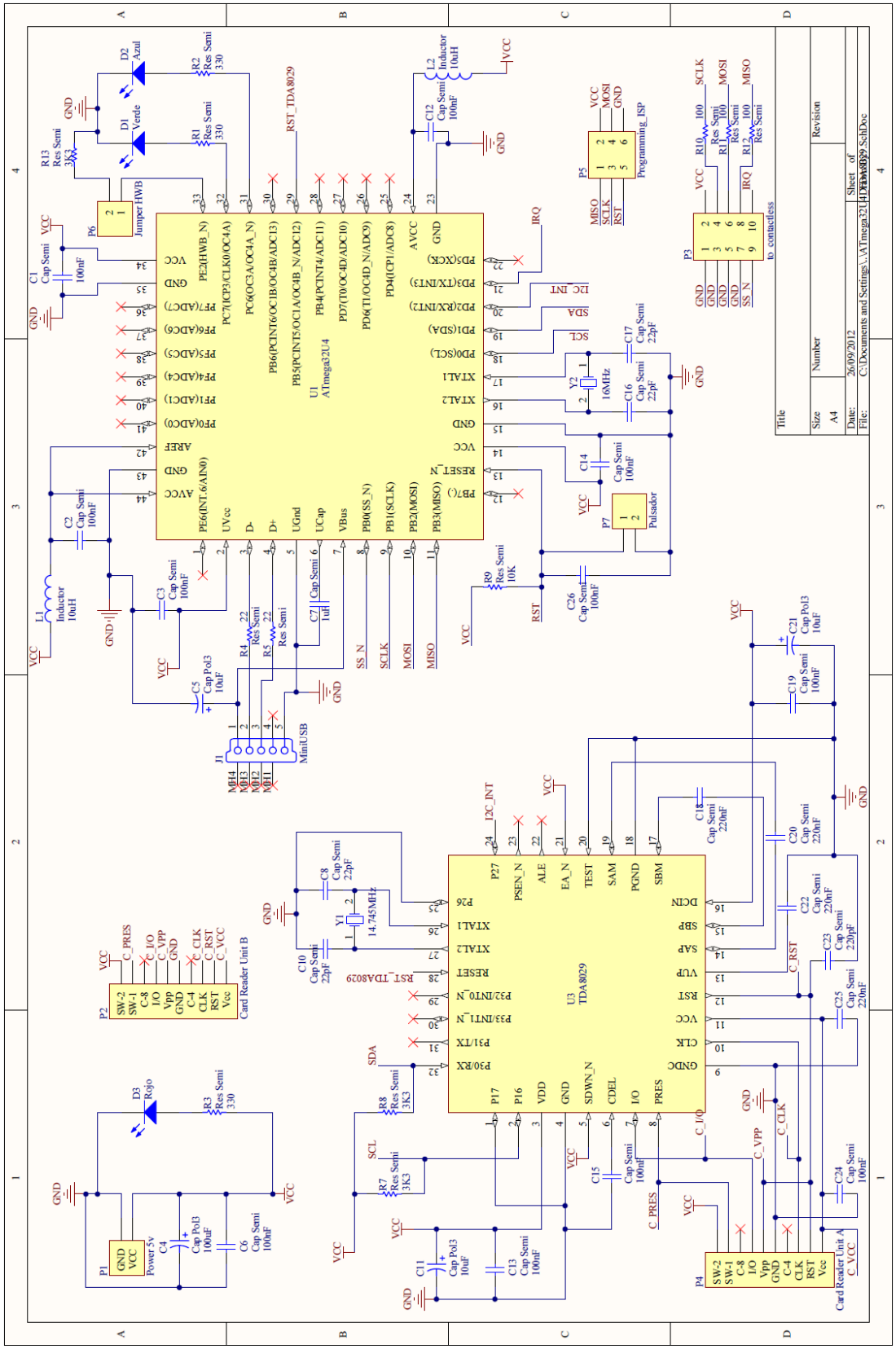


Figura A- 1: Esquemático PCB lector de tarjetas.

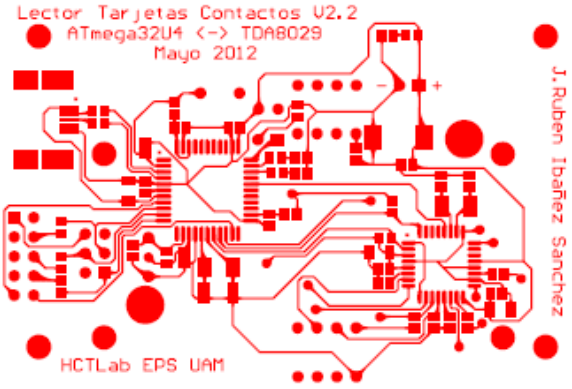



Figura A- 2: *Layout* de la capa superior del PCB a tamaño real.

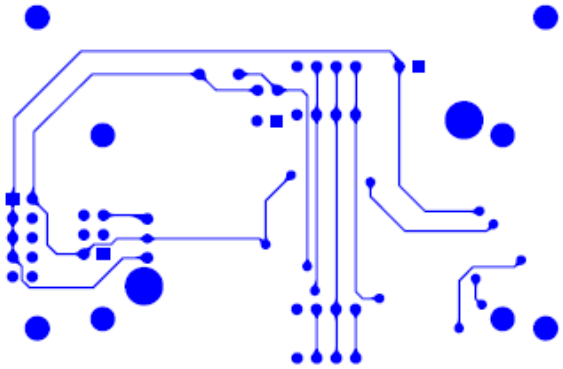


Figura A- 3: *Layout* de la capa inferior del PCB a tamaño real

Figura A- 4: *Layout* de la capa de contorno del PCB a tamaño real.

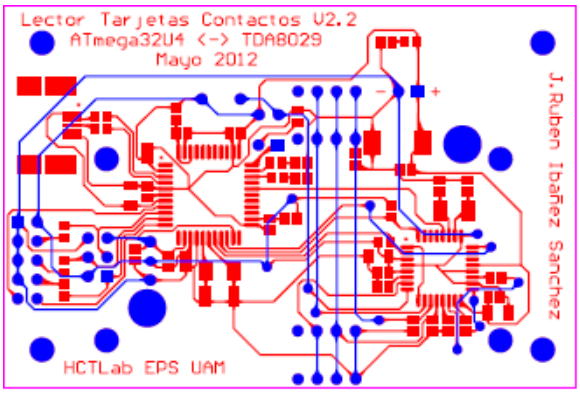



Figura A- 5: *Layout* completo del PCB a tamaño real.

## B. Códigos fuente

En este anexo se presentan los códigos fuente de todas las funciones que componen la aplicación de usuario creada para el lector de tarjetas. En su desarrollo intervienen diversas fuentes además del autor de este PFC.

En la cabecera de cada función se puede encontrar el nombre archivo, una pequeña descripción, la fecha de la última modificación, autor del archivo, título del PFC y las fuentes de las que procede este código. En algunos casos también se incorpora la licencia de uso del código cedida por su creador.

Las fuentes utilizadas para el desarrollo de esta aplicación son: la hoja de características del ATmega32U4 [5], las notas de aplicación AVR315 [11] y AVR1200 [10], la *Web Robot Electronics* (*ATMEGA32 Examples*) [38], la Web PJRC.com (*Teensy USB Development Board*) [43] y la nota de aplicación AN10207 [30].

```
/*****************************************************************************
 * Nombre del archivo: INT_TWI_USB.c
 *
 * Función principal de la aplicación de usuario para el lector de tarjetas basado
 * en el microcontrolador ATmega32U4 y el chip lector de tarjetas TDA8029.
 *
 * Código basado en la hoja de características de ATmega32U4 y notas de aplicación
 * de ATMEL AVR315 y AVR1200.
 *
 * Autor: J. Rubén Ibáñez Sánchez.
 *
 * PFC: Lector de Tarjetas Chip con Acceso USB.
 *
 * Última modificación: 18/09/12
 *****************************************************************************/

// Establece reloj de referencia
#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <math.h>
#include <avr/wdt.h>
#include <avr/pgmspace.h>

#include "ext_interrupt_driver.h"
#include "TWI_Command.h"
#include "usb_debug_only.h"
#include "print.h"

//Semaforo rutina de interrupcion
char reb=1,stp=1;

int main(void)
{
        //Deshabilita bit interrupción global
        cli();

        // Reset variables semaforo interrupción
        reb=1;
        stp=1;

        //Configura PORTC (PIN 6 y 7) como salida
        DDRC = 0xC0;

        //Configura PORTB 5 como salida
        DDRB = 0x20;

        //Enciende los leds -> Inicia Conf.
        PORTC = 0xC0;

        //Watchdog 8 segundos
        wdt_reset();
        wdt_enable(WDTO_8S);

        // Reset TDA8029
        PORTB = 0x20;
        _delay_ms(100);
        PORTB = 0x00;

        //Habilita pull up en PORTD
        PORTD = 0xFF;

        //Configura INT2 sensible a flanco de bajada
        Configure_Interrupt(INTR2,FALLING);

        //Habilita INT2 interrupt
        Enable_Interrupt(INTR2);

        // Inicializa y configura USB, espera respuesta confirmación
        usb_init();
        while (!usb_configured()) /* wait */ ;

        // Se deja un segundo de margen para la configuración por parte del PC
        _delay_ms(250);
        _delay_ms(250);
        _delay_ms(250);
```

```
        _delay_ms(250);

        //Habilita bit interrupción global
        sei();

        //Apaga los leds -> Fin Conf.
        PORTC = 0x00;

        // Comienza el programa de lectura de tarjetas
        // En este punto debe estar ejecutándose "hid_listen_log.exe"
        print("Lector de Tarjetas USB - ATmega32u4 <-> TDA8029");
        print("\n");

        print("\n");
        print("  - Introduzca tarjeta para iniciar sesion -");
        print("\n");

        while (1)
        {
                if(reb==0 && stp==0)
                {
                        _delay_ms(250);
                        if(reb==0 && stp==0)
                        {
                                reb=1;

                                print("\n");
                                print("Lector de Tarjetas USB - ATmega32u4 <-> TDA8029");
                                print("\n");

                                print("\n");
                                print("  - Introduzca tarjeta para iniciar sesion -");
                                print("\n");
                        }
                }
                wdt_reset();
                continue;
        }
}

//Rutina de Interrupción
ISR(INT2_vect)
        {
                if(reb==1)
                {
                        reb=0;

                        char i=0;
                        unsigned char TX[20];
                        unsigned char *RX,*ATR,*DATOS;
                        unsigned char TWI_targetSlaveAddress = 0x50;

                        _delay_ms(250);

                        // Inicializa TWI y las interrupciones
                        TWI_Master_Initialise();

                        //Deshabilita bit interrupción global
                        cli();

                        // Enciende leds -> Inicio TX.
                        PORTC=0xC0;

                        //Comando 0: activado y ATR.
                        TX[0]=ack;
                        TX[1]=0x00;
                        TX[2]=0x00;
                        TX[3]=power_up_iso;
                        TX[4]=(TX[0]^TX[1]^TX[2]^TX[3]);

                        //TWI_TX_RX(char   address,   char   num_data_TX,   unsigned   char   *data,   char
                        num_data_RX);
                        ATR = TWI_TX_RX(TWI_targetSlaveAddress, 5, TX, 32);

                        print("\n");
                        print("ATR: ");
                        while(i!=32)
                        {
```

```
        phex(ATR[i]);
        print(" ");
        i++;
}
print("\n");

_delay_ms(250);

//Comando 1: 0x00, 0xA4, 0x04, 0x00, 0x09, 0xF7, 0x24, 0x00, 0x00, 0x04, 0x07,
0xF6, 0x00, 0x0C
TX[0]=ack;
TX[1]=0x00;
TX[2]=0x0E;
TX[3]=card_command;
TX[4]= 0x00;
TX[5]= 0xA4;
TX[6]= 0x04;
TX[7]= 0x00;
TX[8]= 0x09;
TX[9]= 0xF7;
TX[10]= 0x24;
TX[11]= 0x00;
TX[12]= 0x00;
TX[13]= 0x04;
TX[14]= 0x07;
TX[15]= 0xF6;
TX[16]= 0x00;
TX[17]= 0x0C;
TX[18]=(TX[0]^TX[1]^TX[2]^TX[3]^TX[4]^TX[5]^TX[6]^TX[7]^TX[8]^TX[9]^TX[10]^
TX[11]^TX[12]^TX[13]^TX[14]^TX[15]^TX[16]^TX[17]);

//TWI_TX_RX(char   address,   char   num_data_TX,   unsigned   char   *data,   char
num_data_RX);
RX = TWI_TX_RX(TWI_targetSlaveAddress, 19, TX, 7);

//Comando 2: 0x00, 0xB2, 0x01, 0x0C, 0x02
TX[0]=ack;
TX[1]=0x00;
TX[2]=0x05;
TX[3]=card_command;
TX[4]= 0x00;
TX[5]= 0xB2;
TX[6]= 0x01;
TX[7]= 0x0C;
TX[8]= 0x02;
TX[9]=(TX[0]^TX[1]^TX[2]^TX[3]^TX[4]^TX[5]^TX[6]^TX[7]^TX[8]);

//TWI_TX_RX(char   address,   char   num_data_TX,   unsigned   char   *data,   char
num_data_RX);
RX = TWI_TX_RX(TWI_targetSlaveAddress, 10, TX, 128);

//Comando 3: 0x00, 0xB2, 0x01, 0x0C, 0x60
TX[0]=ack;
TX[1]=0x00;
TX[2]=0x05;
TX[3]=card_command;
TX[4]= 0x00;
TX[5]= 0xB2;
TX[6]= 0x01;
TX[7]= 0x0C;
TX[8]= 0x60;
TX[9]=(TX[0]^TX[1]^TX[2]^TX[3]^TX[4]^TX[5]^TX[6]^TX[7]^TX[8]);

//TWI_TX_RX(char   address,   char   num_data_TX,   unsigned   char   *data,   char
num_data_RX);
DATOS = TWI_TX_RX(TWI_targetSlaveAddress, 10, TX, 128);

i=0;

print("\n");
print("Datos Propietario: ");
while(i!=128)
{
        phex(DATOS[i]);
        print(" ");
        i++;
}
print("\n");
```

```
                _delay_ms(250);

                TX[0]=ack;
                TX[1]=0x00;
                TX[2]=0x00;
                TX[3]=power_off;
                TX[4]=(TX[0]^TX[1]^TX[2]^TX[3]);

                //TWI_TX_RX(char   address,   char   num_data_TX,   unsigned   char   *data,   char
                num_data_RX);
                RX = TWI_TX_RX(TWI_targetSlaveAddress, 5, TX, 5);

                wdt_reset();

                // Parpadeo de led indica extraer tarjeta (4 segundos)
                PORTC=0x00;
                _delay_ms(250);
                _delay_ms(250);
                PORTC=0x80;
                _delay_ms(250);
                _delay_ms(250);
                PORTC=0x40;
                _delay_ms(250);
                _delay_ms(250);
                PORTC=0x80;
                _delay_ms(250);
                _delay_ms(250);
                PORTC=0x40;
                _delay_ms(250);
                _delay_ms(250);
                PORTC=0x80;
                _delay_ms(250);
                _delay_ms(250);
                PORTC=0x40;
                _delay_ms(250);
                _delay_ms(250);
                PORTC=0x00;
                _delay_ms(250);
                _delay_ms(250);
                PORTC=0xC0;

                // Reset TDA8029
                PORTB = 0x20;
                _delay_ms(100);
                PORTB = 0x00;

                //Apaga leds -> Fin TX.
                PORTC=0x00;

                //Habilita bit interrupción global
                sei();

                stp=0;
        }

        return;
}
```

```
/*****************************************************************************
 * Nombre del archivo: ext_interrupt_driver.c
 *
 * ATmega2560 External interrupts driver source file.
 * Application note:
 *      AVR1200: Using external interrupts for megaAVR devices
 *
 *      Atmel Corporation: http://www.atmel.com
 *      Support email: avr@atmel.com
 *
 * Copyright (c) 2011, Atmel Corporation All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * 3. The name of ATMEL may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * Modificado por:
 *      J. Rubén Ibáñez Sánchez.
 *
 * PFC: Lector de Tarjetas Chip con Acceso USB.
 *
 * Última modificación: 10/01/12
 *****************************************************************************/

#include "ext_interrupt_driver.h"

void Configure_Interrupt(uint8_t INT_NO, uint8_t INT_MODE)
{
        switch(INT_NO)
        {
        case 0: switch(INT_MODE)
                {
                case 0:EICRA=(EICRA&(~(1<<ISC01|1<<ISC00)))|(0<<ISC01|0<<ISC00);
                        break;
                case 1:EICRA=(EICRA&(~(1<<ISC01|1<<ISC00)))|(0<<ISC01|1<<ISC00);
                        break;
                case 2:EICRA=(EICRA&(~(1<<ISC01|1<<ISC00)))|(1<<ISC01|0<<ISC00);
                        break;
                case 3:EICRA=(EICRA&(~(1<<ISC01|1<<ISC00)))|(1<<ISC01|1<<ISC00);
                        break;
                default:break;
                }
                break;

        case 1: switch(INT_MODE)
                {
                case 0:EICRA=(EICRA&(~(1<<ISC11|1<<ISC10)))|(0<<ISC11|0<<ISC10);
                        break;
                case 1:EICRA=(EICRA&(~(1<<ISC11|1<<ISC10)))|(0<<ISC11|1<<ISC10);
                        break;
                case 2:EICRA=(EICRA&(~(1<<ISC11|1<<ISC10)))|(1<<ISC11|0<<ISC10);
                        break;
                case 3:EICRA=(EICRA&(~(1<<ISC11|1<<ISC10)))|(1<<ISC11|1<<ISC10);
                        break;
                default:break;
                }
                break;

        case 2: switch(INT_MODE)
                {
                case 0:EICRA=(EICRA&(~(1<<ISC21|1<<ISC20)))|(0<<ISC21|0<<ISC20);
                        break;
                case 1:EICRA=(EICRA&(~(1<<ISC21|1<<ISC20)))|(0<<ISC21|1<<ISC20);
                        break;
                case 2:EICRA=(EICRA&(~(1<<ISC21|1<<ISC20)))|(1<<ISC21|0<<ISC20);
                        break;
                case 3:EICRA=(EICRA&(~(1<<ISC21|1<<ISC20)))|(1<<ISC21|1<<ISC20);
                        break;
                default:break;
```

```
                }
                break;

        case 3: switch(INT_MODE)
                {
                case 0:EICRA=(EICRA&(~(1<<ISC31|1<<ISC30)))|(0<<ISC31|0<<ISC30);
                        break;
                case 1:EICRA=(EICRA&(~(1<<ISC31|1<<ISC30)))|(0<<ISC31|1<<ISC30);
                        break;
                case 2:EICRA=(EICRA&(~(1<<ISC31|1<<ISC30)))|(1<<ISC31|0<<ISC30);
                        break;
                case 3:EICRA=(EICRA&(~(1<<ISC31|1<<ISC30)))|(1<<ISC31|1<<ISC30);
                        break;
                default:break;
                }
                break;

        case 4: switch(INT_MODE)
                {
                case 0:EICRB=(EICRB&(~(1<<ISC41|1<<ISC40)))|(0<<ISC41|0<<ISC40);
                        break;
                case 1:EICRB=(EICRB&(~(1<<ISC41|1<<ISC40)))|(0<<ISC41|1<<ISC40);
                        break;
                case 2:EICRB=(EICRB&(~(1<<ISC41|1<<ISC40)))|(1<<ISC41|0<<ISC40);
                        break;
                case 3:EICRB=(EICRB&(~(1<<ISC41|1<<ISC40)))|(1<<ISC41|1<<ISC40);
                        break;
                default:break;
                }
                break;

        case 5: switch(INT_MODE)
                {
                case 0:EICRB=(EICRB&(~(1<<ISC51|1<<ISC50)))|(0<<ISC51|0<<ISC50);
                        break;
                case 1:EICRB=(EICRB&(~(1<<ISC51|1<<ISC50)))|(0<<ISC51|1<<ISC50);
                        break;
                case 2:EICRB=(EICRB&(~(1<<ISC51|1<<ISC50)))|(1<<ISC51|0<<ISC50);
                        break;
                case 3:EICRB=(EICRB&(~(1<<ISC51|1<<ISC50)))|(1<<ISC51|1<<ISC50);
                        break;
                default:break;
                }
                break;

        case 6: switch(INT_MODE)
                {
                case 0:EICRB=(EICRB&(~(1<<ISC61|1<<ISC60)))|(0<<ISC61|0<<ISC60);
                        break;
                case 1:EICRB=(EICRB&(~(1<<ISC61|1<<ISC60)))|(0<<ISC61|1<<ISC60);
                        break;
                case 2:EICRB=(EICRB&(~(1<<ISC61|1<<ISC60)))|(1<<ISC61|0<<ISC60);
                        break;
                case 3:EICRB=(EICRB&(~(1<<ISC61|1<<ISC60)))|(1<<ISC61|1<<ISC60);
                        break;
                default:break;
                }
                break;

        case 7: switch(INT_MODE)
                {
                case 0:EICRB=(EICRB&(~(1<<ISC71|1<<ISC70)))|(0<<ISC71|0<<ISC70);
                        break;
                case 1:EICRB=(EICRB&(~(1<<ISC71|1<<ISC70)))|(0<<ISC71|1<<ISC70);
                        break;
                case 2:EICRB=(EICRB&(~(1<<ISC71|1<<ISC70)))|(1<<ISC71|0<<ISC70);
                        break;
                case 3:EICRB=(EICRB&(~(1<<ISC71|1<<ISC70)))|(1<<ISC71|1<<ISC70);
                        break;
                default:break;
                }
                break;

        default:break;
        }
}

void Enable_Interrupt(uint8_t INT_NO)
```

```
{
        switch(INT_NO)
        {
        case 0:EIMSK|=(1<<INT0);
                break;
        case 1:EIMSK|=(1<<INT1);
                break;
        case 2:EIMSK|=(1<<INT2);
                break;
        case 3:EIMSK|=(1<<INT3);
                break;
        case 4:EIMSK|=(1<<INT4);
                break;
        case 5:EIMSK|=(1<<INT5);
                break;
        case 6:EIMSK|=(1<<INT6);
                break;
        case 7:EIMSK|=(1<<INT7);
                break;
        default:break;
        }
}

void Disable_Interrupt(uint8_t INT_NO)
{
        switch(INT_NO)
        {
        case 0:EIMSK=(EIMSK&(~(1<<INT0)));
                break;
        case 1:EIMSK=(EIMSK&(~(1<<INT1)));
                break;
        case 2:EIMSK=(EIMSK&(~(1<<INT2)));
                break;
        case 3:EIMSK=(EIMSK&(~(1<<INT3)));
                break;
        case 4:EIMSK=(EIMSK&(~(1<<INT4)));
                break;
        case 5:EIMSK=(EIMSK&(~(1<<INT5)));
                break;
        case 6:EIMSK=(EIMSK&(~(1<<INT6)));
                break;
        case 7:EIMSK=(EIMSK&(~(1<<INT7)));
                break;
        default:break;
        }
}
```

```
/*****************************************************************************
 * Nombre del archivo: ext_interrupt_driver.h
 *
 * \brief  ATmega2560 external interrupts driver header file.
 *
 *      This file contains the enumerator definitions for various
 *      external interrupts for the ATmega2560 external interrupt driver.
 *
 *      The driver is not intended for size and/or speed critical code, since
 *      most functions are just a few lines of code, and the function call
 *      overhead would decrease code performance. The driver is intended for
 *      rapid prototyping and documentation purposes for getting started with
 *      the ATmega2560 external interrupts.
 *
 *      For size and/or speed critical code, it is recommended to copy the
 *      function contents directly into your application instead of making
 *      a function call.
 *
 * \par Application note:
 *      AVR1200: Using external interrupts for megaAVR devices
 *
 * \par Documentation
 *      For comprehensive code documentation, supported compilers, compiler
 *      settings and supported devices see readme.html
 *
 * \author
 *      Atmel Corporation: http://www.atmel.com \n
 *      Support email: avr@atmel.com
 *
 * Copyright (c) 2011, Atmel Corporation All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * 3. The name of ATMEL may not be used to endorse or promote products derived
 * from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY AND
 * SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * Modificado por:
 *      J. Rubén Ibáñez Sánchez.
 *
 * PFC: Lector de Tarjetas Chip con Acceso USB.
 *
 * Última modificación: 10/01/12
 *****************************************************************************/
#include<avr/io.h>

// Prototipos

/*****************************************************************************
 * Esta función configura las interrupciones externas de forma que sean sensibles
 * a flanco de subida o bajada y solicita una interrupción.
 *
 * Parámetros:
 *     INT_NO   Interrupción que se quiere configurar.
 *     INT_MODE Modo de detección de INT_NO.
 *****************************************************************************/
void Configure_Interrupt(uint8_t INT_NO, uint8_t INT_MODE);

/*****************************************************
 * Esta función habilita las interrupciones externas.
```

```
 * Parámetros:
 *     INT_NO   Interrupción que se desea habilitar.
 ******************************************************/
void Enable_Interrupt(uint8_t INT_NO);

/******************************************************
 * Esta función deshabilita las interrupciones externas.
 * Parámetros:
 *     INT_NO   Interrupción que se desea deshabilitar.
 ******************************************************/
void Disable_Interrupt(uint8_t INT_NO);

// Definición de Macros para interrupción
#define INTR0 0
#define INTR1 1
#define INTR2 2
#define INTR3 3
#define INTR4 4
#define INTR5 5
#define INTR6 6
#define INTR7 7

// Definición de Macros para modo de detección
#define LEVEL 0
#define EDGE 1
#define FALLING 2
#define RISING 3
```

```
/*****************************************************************
 * Nombre del archivo: TWI.c
 *
 * Funciones para TWI: inicialización y escritura/lectura.
 *
 * Creado a partir de la hoja de datos de ATmega32U4, AVR315 y Web:
 * http://www.robot-electronics.co.uk/htm/ATMEGA32_examples.htm#SRF01
 *
 * Autor: J. Rubén Ibáñez Sánchez.
 *
 * PFC: Lector de Tarjetas Chip con Acceso USB.
 *
 * Última modificación: 20/06/12

 *****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <math.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#include "TWI_Command.h"

void TWI_Master_Initialise(void)
{
  TWBR = TWI_TWBR;                                  // Set bit rate register (Baudrate). Defined
                                                        in header file.
// TWSR = TWI_TWPS;                                  // Not used. Driver presumes prescaler to be
                                                        00.
  TWDR = 0xFF;                                      // Default content = SDA released.
  TWCR = (1<<TWEN)|                                 // Enable TWI-interface and release TWI
                                                        pins.
         (0<<TWIE)|(0<<TWINT)|                      // Disable Interupt.
         (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|           // No Signal requests.
         (0<<TWWC);                                 //
}

unsigned char* TWI_TX_RX(char address, char num_data_TX, unsigned char *data, char num_data_RX)
{
        char i=0,j=0;
        unsigned char read_data[num_data_RX];

        TWCR = 0xA4;                              // Envio del bit de inicio.
        while(!(TWCR & 0x80));                    // Espera confirmación de transmisión.

        TWDR = address;                           // Carga la dirección del dispositivo esclavo.
        TWCR = 0x84;                              // Transmisión.
        while(!(TWCR & 0x80));                    // Espera confirmación de transmisión.

        while(i<num_data_TX)
        {
                TWDR = data[i];                   // Carga byte a enviar.
                TWCR = 0x84;                      // Transmisión.
                while(!(TWCR & 0x80));            // Espera confirmación de transmisión.

                i++;
        }

        TWCR = 0x94;                              // Envio del bit de parada.

        //============================================================================================//

        _delay_ms(200); // Delay entre envio de datos y lectura de respuesta.

        //============================================================================================//

        TWCR = 0xA4;                             // Envio del bit de inicio.
        while(!(TWCR & 0x80));                   // Espera confirmación de transmisión.

        TWDR = address+1;                        // Carga la dirección del dispositivo esclavo
                                                  indicando lectura.
        TWCR = 0x84;                             // Transmisión.
        while(!(TWCR & 0x80));                   // Espera confirmación de transmisión.

        while(j<num_data_RX-1)
```

```
        {
                TWCR = 0xC4;                    // Acepta byte anterior y solicita siguiente
                                                byte.
                while(!(TWCR & 0x80));          // Espera confirmación de transmisión.
                read_data[j] = TWDR;            // Guarda byte recibido.

                j++;
        }

        TWCR = 0x84;                            // Acepta byte anterior y solicita el último
                                            byte.
        while(!(TWCR & 0x80));                 // Espera confirmación de transmisión.
        read_data[j] = TWDR;                    // Guarda byte recibido (último).

        TWCR = 0x94;                            // Envio del bit de parada.

        return read_data;                       // La función devuelve los datos recibidos.
}
```

```
/*************************************************************
 * Nombre del archivo: TWI_Command.h
 *
 * Autor: J. Rubén Ibáñez Sánchez.
 *
 * Comando extraídos de la nota de aplicación AN10207.
 *
 * PFC: Lector de Tarjetas Chip con Acceso USB.
 *
 * Comandos para la comunicación TWI (I2C) ATmega32U4 <-> TDA8029
 *
 * Última modificación: 5/09/12
 *************************************************************/

/*************************************************************************
 Función para configurar el maestro TWI a su estado de espera inicial.
 Habilitar las interrupciones globales después de la inicialización.
*************************************************************************/
void TWI_Master_Initialise(void);

/*****************************************************************************************
* Función para la escritura/lectura de datos mediante TWI.
*
* Parámetros:
*    address: dirección esclavo con el que se desea comunicar.
*    num_data_TX: número de bytes de datos que se envían.
*    *data: puntero al array de datos para enviar.
*    num_data_RX: número de bytes de datos que se van ha leer.
*
* La función devuelve un puntero al array de datos leídos "read_data" del bus I2C.
*
* Esta función solo puede ser utilizada por el dispositivo maestro.
*****************************************************************************************/
unsigned char* TWI_TX_RX(char address, char num_data_TX, unsigned char *data, char num_data_RX);

/*************************************************************************
* Registro ATmega32U4 de control de la velocidad en línea SCL
* Controla velocidad SCL mediante formula:
*
*       SCL frequency = CPU Clock frequency/(16 + 2(TWBR)*4^(TWPS))
*
* TWI_TWBR = 0x7D -> Vel 60 KHz (max.)
* TWI_TWBR = 0x98 -> Vel 50 KHz
* TWI_TWBR = 0xA9 -> Vel 45 KHz
* TWI_TWBR = 0xC0 -> Vel 40 KHz
* TWI_TWBR = 0xDC -> Vel 35 KHz
*************************************************************************/
#define TWI_TWBR                        0xC0


// Comandos I2C TDA8029.
#define ack                             0x60
#define nack                            0xE0

#define card_command                    0x00
#define process_T_1                     0x01
#define write_I2C                       0x02
#define read_S9                         0x03
#define read_S9_protect                 0x04
#define write_S9_protect                0x05
#define write_S9_unprotec               0x06
#define verify_pin_S9                   0x07
#define compare_S9                      0x08
#define check_pres_card                 0x09
#define send_num_mask                   0x0A
#define set_card_baud_rate              0x0B
#define ifsd_request                    0x0C
#define set_serial_baud_rate            0x0D
#define negotiate                       0x10
#define set_clock_card                  0x11
#define read_I2C                        0x12
#define read_2C_extended                0x13
#define read_current_I2C                0x23
#define power_off                       0x4D
#define power_up_iso                    0x69
#define power_up_S9                     0x6B
#define power_up_I2C                    0x6C
#define power_up_1_8V                   0x68
```

```
#define power_up_3V                     0x6D
#define power_up_5V                     0x6E
#define idle_mode_clk_L                 0xA2
#define power_down_mode                 0xA3
#define idle_mode_clk_H                 0xA4
#define set_nad                         0xA5
#define get_card_param                  0xA6
#define get_reader_status               0xAA
#define power_up_S10                    0xC1
#define process_S10                     0xC2
#define read_IO                         0xCE
#define set_IO                          0xCF
```

```
/**************************************************************************
 * Nombre del archivo: usb_debug_only.c
 *
 * USB Debug Channel Example for Teensy USB Development Board
 * http://www.pjrc.com/teensy/
 * Copyright (c) 2008 PJRC.COM, LLC
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * Modificado por:
 *      J. Rubén Ibáñez Sánchez.
 *
 * PFC: Lector de Tarjetas Chip con Acceso USB.
 *
 * Última modificación: 18/08/12
 **************************************************************************/

// Version 1.0: Initial Release
// Version 1.1: Add support for Teensy 2.0

#define USB_SERIAL_PRIVATE_INCLUDE
#include "usb_debug_only.h"

/**************************************************************************
 *
 *  Configurable Options
 *
 **************************************************************************/

// You can change these to give your code its own name.  On Windows,
// these are only used before an INF file (driver install) is loaded.
#define STR_MANUFACTURER        L"HCTLab - Ruben"
#define STR_PRODUCT             L"USB Smart Card Reader"


// Mac OS-X and Linux automatically load the correct drivers.  On
// Windows, even though the driver is supplied by Microsoft, an
// INF file is needed to load the driver.  These numbers need to
// match the INF file.
#define VENDOR_ID               0x16C0
#define PRODUCT_ID              0x0479

// USB devices are supposed to implment a halt feature, which is
// rarely (if ever) used.  If you comment this line out, the halt
// code will be removed, saving 102 bytes of space (gcc 4.3.0).
// This is not strictly USB compliant, but works with all major
// operating systems.
#define SUPPORT_ENDPOINT_HALT

/**************************************************************************
 *
 *  Endpoint Buffer Configuration
 *
 **************************************************************************/

// you might want to change the buffer size, up to 64 bytes.
// The host reserves your bandwidth because this is an interrupt
// endpoint, so it won't be available to other interrupt or isync
// endpoints in other devices on the bus.

#define ENDPOINT0_SIZE          32
#define DEBUG_TX_ENDPOINT       3
```

```
#define DEBUG_TX_SIZE           32
#define DEBUG_TX_BUFFER         EP_DOUBLE_BUFFER

static const uint8_t PROGMEM endpoint_config_table[] = {
        0,
        0,
        1, EP_TYPE_INTERRUPT_IN,  EP_SIZE(DEBUG_TX_SIZE) | DEBUG_TX_BUFFER,
        0
};

/**************************************************************************
 *
 *  Descriptor Data
 *
 **************************************************************************/

// Descriptors are the data that your computer reads when it auto-detects
// this USB device (called "enumeration" in USB lingo).  The most commonly
// changed items are editable at the top of this file.  Changing things
// in here should only be done by those who've read chapter 9 of the USB
// spec and relevant portions of any USB class specifications!

static uint8_t PROGMEM device_descriptor[] = {
        18,                                     // bLength
        1,                                      // bDescriptorType
        0x00, 0x02,                             // bcdUSB
        0,                                      // bDeviceClass
        0,                                      // bDeviceSubClass
        0,                                      // bDeviceProtocol
        ENDPOINT0_SIZE,                         // bMaxPacketSize0
        LSB(VENDOR_ID), MSB(VENDOR_ID),         // idVendor
        LSB(PRODUCT_ID), MSB(PRODUCT_ID),       // idProduct
        0x00, 0x01,                             // bcdDevice
        1,                                      // iManufacturer
        2,                                      // iProduct
        0,                                      // iSerialNumber
        1                                       // bNumConfigurations
};

static uint8_t PROGMEM hid_report_descriptor[] = {
        0x06, 0x31, 0xFF,                       // Usage Page 0xFF31 (vendor defined)
        0x09, 0x74,                             // Usage 0x74
        0xA1, 0x53,                             // Collection 0x53
        0x75, 0x08,                             // report size = 8 bits
        0x15, 0x00,                             // logical minimum = 0
        0x26, 0xFF, 0x00,                       // logical maximum = 255
        0x95, DEBUG_TX_SIZE,                    // report count
        0x09, 0x75,                             // usage
        0x81, 0x02,                             // Input (array)
        0xC0                                    // end collection
};

#define CONFIG1_DESC_SIZE (9+9+9+7)
#define HID_DESC2_OFFSET  (9+9)
static uint8_t PROGMEM config1_descriptor[CONFIG1_DESC_SIZE] = {
        // configuration descriptor, USB spec 9.6.3, page 264-266, Table 9-10
        9,                                      // bLength;
        2,                                      // bDescriptorType;
        LSB(CONFIG1_DESC_SIZE),                 // wTotalLength
        MSB(CONFIG1_DESC_SIZE),
        1,                                      // bNumInterfaces
        1,                                      // bConfigurationValue
        0,                                      // iConfiguration
        0xC0,                                   // bmAttributes
        50,                                     // bMaxPower
        // interface descriptor, USB spec 9.6.5, page 267-269, Table 9-12
        9,                                      // bLength
        4,                                      // bDescriptorType
        0,                                      // bInterfaceNumber
        0,                                      // bAlternateSetting
        1,                                      // bNumEndpoints
        0x03,                                   // bInterfaceClass (0x03 = HID)
        0x00,                                   // bInterfaceSubClass
        0x00,                                   // bInterfaceProtocol
        0,                                      // iInterface
        // HID interface descriptor, HID 1.11 spec, section 6.2.1
        9,                                      // bLength
        0x21,                                   // bDescriptorType
```

```
        0x11, 0x01,                             // bcdHID
        0,                                      // bCountryCode
        1,                                      // bNumDescriptors
        0x22,                                   // bDescriptorType
        sizeof(hid_report_descriptor),          // wDescriptorLength
        0,
        // endpoint descriptor, USB spec 9.6.6, page 269-271, Table 9-13
        7,                                      // bLength
        5,                                      // bDescriptorType
        DEBUG_TX_ENDPOINT | 0x80,               // bEndpointAddress
        0x03,                                   // bmAttributes (0x03=intr)
        DEBUG_TX_SIZE, 0,                       // wMaxPacketSize
        1                                       // bInterval
};

// If you're desperate for a little extra code memory, these strings
// can be completely removed if iManufacturer, iProduct, iSerialNumber
// in the device desciptor are changed to zeros.
struct usb_string_descriptor_struct {
        uint8_t bLength;
        uint8_t bDescriptorType;
        int16_t wString[];
};
static struct usb_string_descriptor_struct PROGMEM string0 = {
        4,
        3,
        {0x0409}
};
static struct usb_string_descriptor_struct PROGMEM string1 = {
        sizeof(STR_MANUFACTURER),
        3,
        STR_MANUFACTURER
};
static struct usb_string_descriptor_struct PROGMEM string2 = {
        sizeof(STR_PRODUCT),
        3,
        STR_PRODUCT
};

// This table defines which descriptor data is sent for each specific
// request from the host (in wValue and wIndex).
static struct descriptor_list_struct {
        uint16_t wValue;
        uint16_t wIndex;
        const uint8_t   *addr;
        uint8_t         length;
} PROGMEM descriptor_list[] = {
        {0x0100, 0x0000, device_descriptor, sizeof(device_descriptor)},
        {0x0200, 0x0000, config1_descriptor, sizeof(config1_descriptor)},
        {0x2200, 0x0000, hid_report_descriptor, sizeof(hid_report_descriptor)},
        {0x2100, 0x0000, config1_descriptor+HID_DESC2_OFFSET, 9},
        {0x0300, 0x0000, (const uint8_t *)&string0, 4},
        {0x0301, 0x0409, (const uint8_t *)&string1, sizeof(STR_MANUFACTURER)},
        {0x0302, 0x0409, (const uint8_t *)&string2, sizeof(STR_PRODUCT)}
};
#define NUM_DESC_LIST (sizeof(descriptor_list)/sizeof(struct descriptor_list_struct))

/**************************************************************************
 *
 *  Variables - these are the only non-stack RAM usage
 *
 **************************************************************************/

// zero when we are not configured, non-zero when enumerated
static volatile uint8_t usb_configuration=0;

// the time remaining before we transmit any partially full
// packet, or send a zero length packet.
static volatile uint8_t debug_flush_timer=0;

/**************************************************************************
 *
 *  Public Functions - these are the API intended for the user
 *
 **************************************************************************/

// initialize USB
void usb_init(void)
```

```
{
        HW_CONFIG();
        USB_FREEZE();                           // enable USB
        PLL_CONFIG();                           // config PLL
        while (!(PLLCSR & (1<<PLOCK))) ;// wait for PLL lock
        USB_CONFIG();                           // start USB clock
        UDCON = 0;                              // enable attach resistor
        usb_configuration = 0;
        UDIEN = (1<<EORSTE)|(1<<SOFE);
        sei();
}

// return 0 if the USB is not configured, or the configuration
// number selected by the HOST
uint8_t usb_configured(void)
{
        return usb_configuration;
}

// transmit a character.  0 returned on success, -1 on error
int8_t usb_debug_putchar(uint8_t c)
{
        static uint8_t previous_timeout=0;
        uint8_t timeout, intr_state;

        // if we're not online (enumerated and configured), error
        if (!usb_configuration) return -1;
        // interrupts are disabled so these functions can be
        // used from the main program or interrupt context,
        // even both in the same program!
        intr_state = SREG;
        cli();
        UENUM = DEBUG_TX_ENDPOINT;
        // if we gave up due to timeout before, don't wait again
        if (previous_timeout) {
                if (!(UEINTX & (1<<RWAL))) {
                        SREG = intr_state;
                        return -1;
                }
                previous_timeout = 0;
        }
        // wait for the FIFO to be ready to accept data
        timeout = UDFNUML + 4;
        while (1) {
                // are we ready to transmit?
                if (UEINTX & (1<<RWAL)) break;
                SREG = intr_state;
                // have we waited too long?
                if (UDFNUML == timeout) {
                        previous_timeout = 1;
                        return -1;
                }
                // has the USB gone offline?
                if (!usb_configuration) return -1;
                // get ready to try checking again
                intr_state = SREG;
                cli();
                UENUM = DEBUG_TX_ENDPOINT;
        }
        // actually write the byte into the FIFO
        UEDATX = c;
        // if this completed a packet, transmit it now!
        if (!(UEINTX & (1<<RWAL))) {
                UEINTX = 0x3A;
                debug_flush_timer = 0;
        } else {
                debug_flush_timer = 2;
        }
        SREG = intr_state;
        return 0;
}

// immediately transmit any buffered output.
void usb_debug_flush_output(void)
{
        uint8_t intr_state;

        intr_state = SREG;
```

```
        cli();
        if (debug_flush_timer) {
                UENUM = DEBUG_TX_ENDPOINT;
                while ((UEINTX & (1<<RWAL))) {
                        UEDATX = 0;
                }
                UEINTX = 0x3A;
                debug_flush_timer = 0;
        }
        SREG = intr_state;
}

/**************************************************************************
 *
 *  Private Functions - not intended for general user consumption....
 *
 **************************************************************************/

// USB Device Interrupt - handle all device-level events
// the transmit buffer flushing is triggered by the start of frame
//
ISR(USB_GEN_vect)
{
        uint8_t intbits, t;

        intbits = UDINT;
        UDINT = 0;
        if (intbits & (1<<EORSTI)) {
                UENUM = 0;
                UECONX = 1;
                UECFG0X = EP_TYPE_CONTROL;
                UECFG1X = EP_SIZE(ENDPOINT0_SIZE) | EP_SINGLE_BUFFER;
                UEIENX = (1<<RXSTPE);
                usb_configuration = 0;
        }
        if (intbits & (1<<SOFI)) {
                if (usb_configuration) {
                        t = debug_flush_timer;
                        if (t) {
                                debug_flush_timer = -- t;
                                if (!t) {
                                        UENUM = DEBUG_TX_ENDPOINT;
                                        while ((UEINTX & (1<<RWAL))) {
                                                UEDATX = 0;
                                        }
                                        UEINTX = 0x3A;
                                }
                        }
                }
        }
}

// Misc functions to wait for ready and send/receive packets
static inline void usb_wait_in_ready(void)
{
        while (!(UEINTX & (1<<TXINI))) ;
}
static inline void usb_send_in(void)
{
        UEINTX = ~(1<<TXINI);
}
static inline void usb_wait_receive_out(void)
{
        while (!(UEINTX & (1<<RXOUTI))) ;
}
static inline void usb_ack_out(void)
{
        UEINTX = ~(1<<RXOUTI);
}

// USB Endpoint Interrupt - endpoint 0 is handled here.  The
// other endpoints are manipulated by the user-callable
// functions, and the start-of-frame interrupt.
//
ISR(USB_COM_vect)
{
        uint8_t intbits;
        const uint8_t *list;
```

```
const uint8_t *cfg;
uint8_t i, n, len, en;
uint8_t bmRequestType;
uint8_t bRequest;
uint16_t wValue;
uint16_t wIndex;
uint16_t wLength;
uint16_t desc_val;
const uint8_t *desc_addr;
uint8_t desc_length;

UENUM = 0;
intbits = UEINTX;
if (intbits & (1<<RXSTPI)) {
        bmRequestType = UEDATX;
        bRequest = UEDATX;
        wValue = UEDATX;
        wValue |= (UEDATX << 8);
        wIndex = UEDATX;
        wIndex |= (UEDATX << 8);
        wLength = UEDATX;
        wLength |= (UEDATX << 8);
        UEINTX = ~((1<<RXSTPI) | (1<<RXOUTI) | (1<<TXINI));
        if (bRequest == GET_DESCRIPTOR) {
                list = (const uint8_t *)descriptor_list;
                for (i=0; ; i++) {
                        if (i >= NUM_DESC_LIST) {
                                UECONX = (1<<STALLRQ)|(1<<EPEN);  //stall
                                return;
                        }
                        desc_val = pgm_read_word(list);
                        if (desc_val != wValue) {
                                list += sizeof(struct descriptor_list_struct);
                                continue;
                        }
                        list += 2;
                        desc_val = pgm_read_word(list);
                        if (desc_val != wIndex) {
                                list += sizeof(struct descriptor_list_struct)-2;
                                continue;
                        }
                        list += 2;
                        desc_addr = (const uint8_t *)pgm_read_word(list);
                        list += 2;
                        desc_length = pgm_read_byte(list);
                        break;
                }
                len = (wLength < 256) ? wLength : 255;
                if (len > desc_length) len = desc_length;
                do {
                        // wait for host ready for IN packet
                        do {
                                i = UEINTX;
                        } while (!(i & ((1<<TXINI)|(1<<RXOUTI))));
                        if (i & (1<<RXOUTI)) return;    // abort
                        // send IN packet
                        n = len < ENDPOINT0_SIZE ? len : ENDPOINT0_SIZE;
                        for (i = n; i; i--) {
                                UEDATX = pgm_read_byte(desc_addr++);
                        }
                        len -= n;
                        usb_send_in();
                } while (len || n == ENDPOINT0_SIZE);
                return;
        }
        if (bRequest == SET_ADDRESS) {
                usb_send_in();
                usb_wait_in_ready();
                UDADDR = wValue | (1<<ADDEN);
                return;
        }
        if (bRequest == SET_CONFIGURATION && bmRequestType == 0) {
                usb_configuration = wValue;
                usb_send_in();
                cfg = endpoint_config_table;
                for (i=1; i<5; i++) {
                        UENUM = i;
                        en = pgm_read_byte(cfg++);
```

```
                        UECONX = en;
                        if (en) {
                                UECFG0X = pgm_read_byte(cfg++);
                                UECFG1X = pgm_read_byte(cfg++);
                        }
                }
                UERST = 0x1E;
                UERST = 0;
                return;
        }
        if (bRequest == GET_CONFIGURATION && bmRequestType == 0x80) {
                usb_wait_in_ready();
                UEDATX = usb_configuration;
                usb_send_in();
                return;
        }

        if (bRequest == GET_STATUS) {
                usb_wait_in_ready();
                i = 0;
                #ifdef SUPPORT_ENDPOINT_HALT
                if (bmRequestType == 0x82) {
                        UENUM = wIndex;
                        if (UECONX & (1<<STALLRQ)) i = 1;
                        UENUM = 0;
                }
                #endif
                UEDATX = i;
                UEDATX = 0;
                usb_send_in();
                return;
        }
        #ifdef SUPPORT_ENDPOINT_HALT
        if ((bRequest == CLEAR_FEATURE || bRequest == SET_FEATURE)
          && bmRequestType == 0x02 && wValue == 0) {
                i = wIndex & 0x7F;
                if (i >= 1 && i <= MAX_ENDPOINT) {
                        usb_send_in();
                        UENUM = i;
                        if (bRequest == SET_FEATURE) {
                                UECONX = (1<<STALLRQ)|(1<<EPEN);
                        } else {
                                UECONX = (1<<STALLRQC)|(1<<RSTDT)|(1<<EPEN);
                                UERST = (1 << i);
                                UERST = 0;
                        }
                        return;
                }
        }
        #endif
        if (bRequest == HID_GET_REPORT && bmRequestType == 0xA1) {
                if (wIndex == 0) {
                        len = wLength;
                        do {
                                // wait for host ready for IN packet
                                do {
                                        i = UEINTX;
                                } while (!(i & ((1<<TXINI)|(1<<RXOUTI))));
                                if (i & (1<<RXOUTI)) return;    // abort
                                // send IN packet
                                n = len < ENDPOINT0_SIZE ? len : ENDPOINT0_SIZE;
                                for (i = n; i; i--) {
                                        UEDATX = 0;
                                }
                                len -= n;
                                usb_send_in();
                        } while (len || n == ENDPOINT0_SIZE);
                        return;
                }
        }
    }
    UECONX = (1<<STALLRQ) | (1<<EPEN);      // stall
}
```

```
/**************************************************************************
 * Nombre del archivo: usb_debug_only.h
 *
 * USB Debug Channel Example for Teensy USB Development Board
 * http://www.pjrc.com/teensy/
 * Copyright (c) 2008 PJRC.COM, LLC
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * PFC: Lector de Tarjetas Chip con Acceso USB.
 *
 **************************************************************************/

#ifndef usb_serial_h__
#define usb_serial_h__

#include <stdint.h>

void usb_init(void);                    // initialize everything
uint8_t usb_configured(void);           // is the USB port configured

int8_t usb_debug_putchar(uint8_t c);    // transmit a character
void usb_debug_flush_output(void);      // immediately transmit any buffered output
#define USB_DEBUG_HID


// Everything below this point is only intended for usb_serial.c
#ifdef USB_SERIAL_PRIVATE_INCLUDE
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>

#define EP_TYPE_CONTROL                 0x00
#define EP_TYPE_BULK_IN                 0x81
#define EP_TYPE_BULK_OUT                0x80
#define EP_TYPE_INTERRUPT_IN            0xC1
#define EP_TYPE_INTERRUPT_OUT           0xC0
#define EP_TYPE_ISOCHRONOUS_IN          0x41
#define EP_TYPE_ISOCHRONOUS_OUT         0x40

#define EP_SINGLE_BUFFER                0x02
#define EP_DOUBLE_BUFFER                0x06

#define EP_SIZE(s)      ((s) == 64 ? 0x30 :     \
                        ((s) == 32 ? 0x20 :     \
                        ((s) == 16 ? 0x10 :     \
                                     0x00)))

#define MAX_ENDPOINT                    4

#define LSB(n) (n & 255)
#define MSB(n) ((n >> 8) & 255)

#if defined(__AVR_AT90USB162__)
#define HW_CONFIG()
#define PLL_CONFIG() (PLLCSR = ((1<<PLLE)|(1<<PLLP0)))
#define USB_CONFIG() (USBCON = (1<<USBE))
#define USB_FREEZE() (USBCON = ((1<<USBE)|(1<<FRZCLK)))
#elif defined(__AVR_ATmega32U4__)
#define HW_CONFIG() (UHWCON = 0x01)
#define PLL_CONFIG() (PLLCSR = 0x12)
#define USB_CONFIG() (USBCON = ((1<<USBE)|(1<<OTGPADE)))
```

```
#define USB_FREEZE() (USBCON = ((1<<USBE)|(1<<FRZCLK)))
#elif defined(__AVR_AT90USB646__)
#define HW_CONFIG() (UHWCON = 0x81)
#define PLL_CONFIG() (PLLCSR = 0x1A)
#define USB_CONFIG() (USBCON = ((1<<USBE)|(1<<OTGPADE)))
#define USB_FREEZE() (USBCON = ((1<<USBE)|(1<<FRZCLK)))
#elif defined(__AVR_AT90USB1286__)
#define HW_CONFIG() (UHWCON = 0x81)
#define PLL_CONFIG() (PLLCSR = 0x16)
#define USB_CONFIG() (USBCON = ((1<<USBE)|(1<<OTGPADE)))
#define USB_FREEZE() (USBCON = ((1<<USBE)|(1<<FRZCLK)))
#endif

// standard control endpoint request types
#define GET_STATUS                      0
#define CLEAR_FEATURE                   1
#define SET_FEATURE                     3
#define SET_ADDRESS                     5
#define GET_DESCRIPTOR                  6
#define GET_CONFIGURATION               8
#define SET_CONFIGURATION               9
#define GET_INTERFACE                   10
#define SET_INTERFACE                   11
// HID (human interface device)
#define HID_GET_REPORT                  1
#define HID_GET_PROTOCOL                3
#define HID_SET_REPORT                  9
#define HID_SET_IDLE                    10
#define HID_SET_PROTOCOL                11
// CDC (communication class device)
#define CDC_SET_LINE_CODING             0x20
#define CDC_GET_LINE_CODING             0x21
#define CDC_SET_CONTROL_LINE_STATE      0x22
#endif
#endif
```

```c
/**************************************************************************
 * Nombre del archivo: print.c
 *
 * Very basic print functions, intended to be used with usb_debug_only.c
 * http://www.pjrc.com/teensy/
 * Copyright (c) 2008 PJRC.COM, LLC
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * PFC: Lector de Tarjetas Chip con Acceso USB.
 *
 **************************************************************************/

// Version 1.0: Initial Release

#include <avr/io.h>
#include <avr/pgmspace.h>

#include "print.h"

void print_P(const char *s)
{
        char c;

        while (1) {
                c = pgm_read_byte(s++);
                if (!c) break;
                if (c == '\n') usb_debug_putchar('\r');
                usb_debug_putchar(c);
        }
}

void phex1(unsigned char c)
{
        usb_debug_putchar(c + ((c < 10) ? '0' : 'A' - 10));
}

void phex(unsigned char c)
{
        phex1(c >> 4);
        phex1(c & 15);
}

void phex16(unsigned int i)
{
        phex(i >> 8);
        phex(i);
}
```

```
/*****************************************************************************
 * Nombre del archivo: print.h
 *
 * Very basic print functions, intended to be used with usb_debug_only.c
 * http://www.pjrc.com/teensy/
 * Copyright (c) 2008 PJRC.COM, LLC
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * PFC: Lector de Tarjetas Chip con Acceso USB.
 *
 *****************************************************************************/

#ifndef print_h__
#define print_h__

#include <avr/pgmspace.h>
#include "usb_debug_only.h"

// this macro allows you to write print("some text") and
// the string is automatically placed into flash memory :)
#define print(s) print_P(PSTR(s))
#define pchar(c) usb_debug_putchar(c)

void print_P(const char *s);
void phex(unsigned char c);
void phex16(unsigned int i);

#endif
```

## C. Manual de usuario

Los elementos necesarios para la configuración, puesta en marcha y funcionamiento del lector de tarjetas son: fuente de alimentación regulada a 5V, cable de alimentación, programador *AVRISP mkII* con su cable para conectarlo al PC, cable USB a miniUSB y el archivo ".hex" de la aplicación de usuario.

El primer paso que se debe realizar es conectar el lector a la alimentación y, seguidamente conectar el programador y el miniUSB y, por último, conectar la fuente.

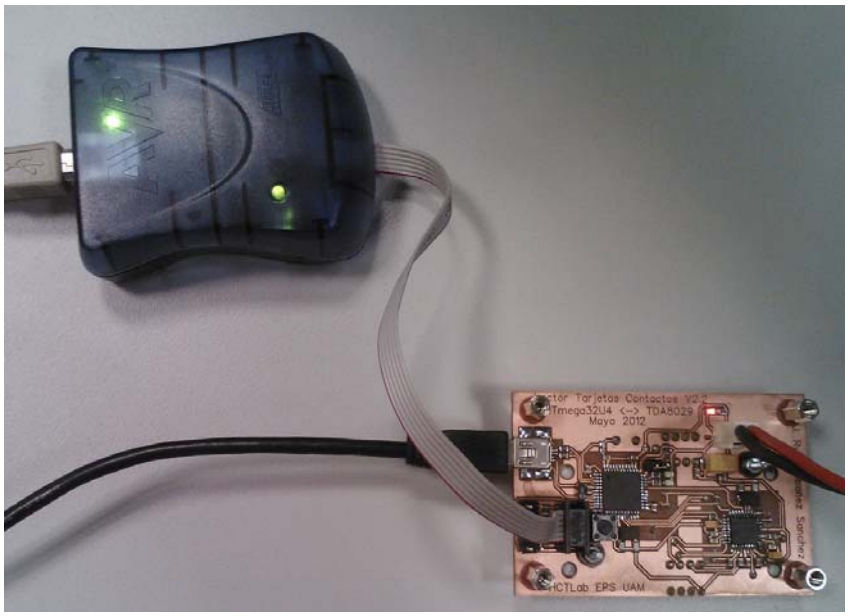



Figura C- 1: Conexión del lector al programador, PC y alimentación.

Mediante el software *AVR Studio 5.0* y el programador ISP se instala la aplicación de usuario en la memoria interna del microcontrolador. Además permite acceder a las configuraciones internas del chip, como *fuses* y bits de bloqueo. A continuación se muestran las opciones disponibles.

Accedemos al menú "*AVR Programming*" pulsando el siguiente botón:

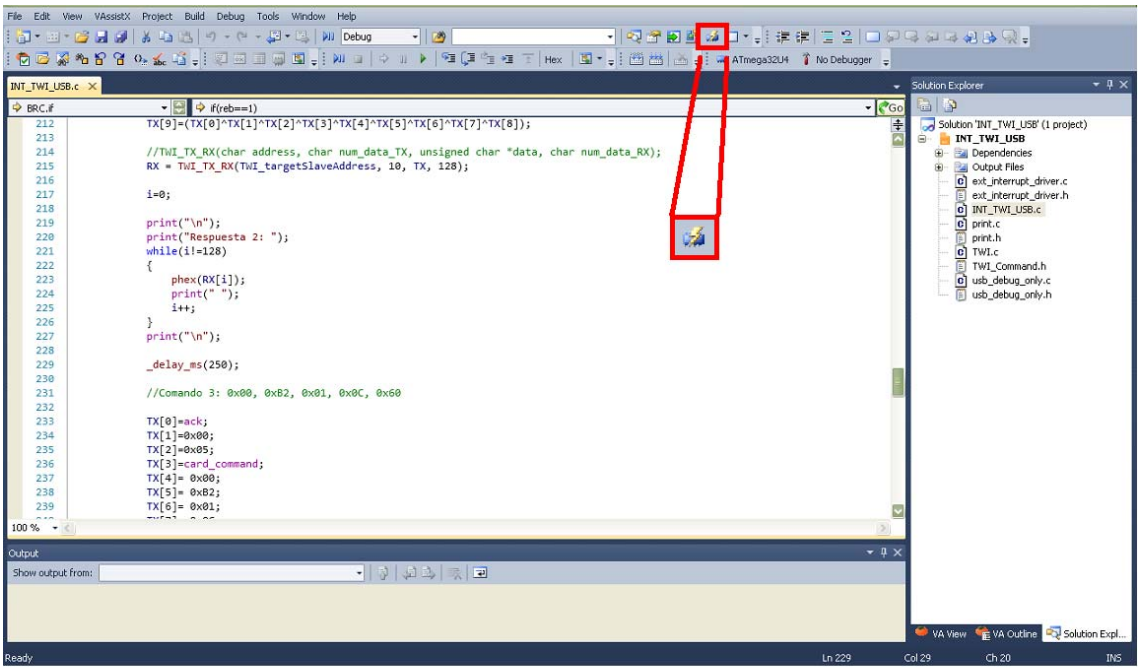


Figura C- 2: Ubicación del botón de programación del dispositivo en *AVR Studio 5*.

Estas son las opciones de las que se dispone:

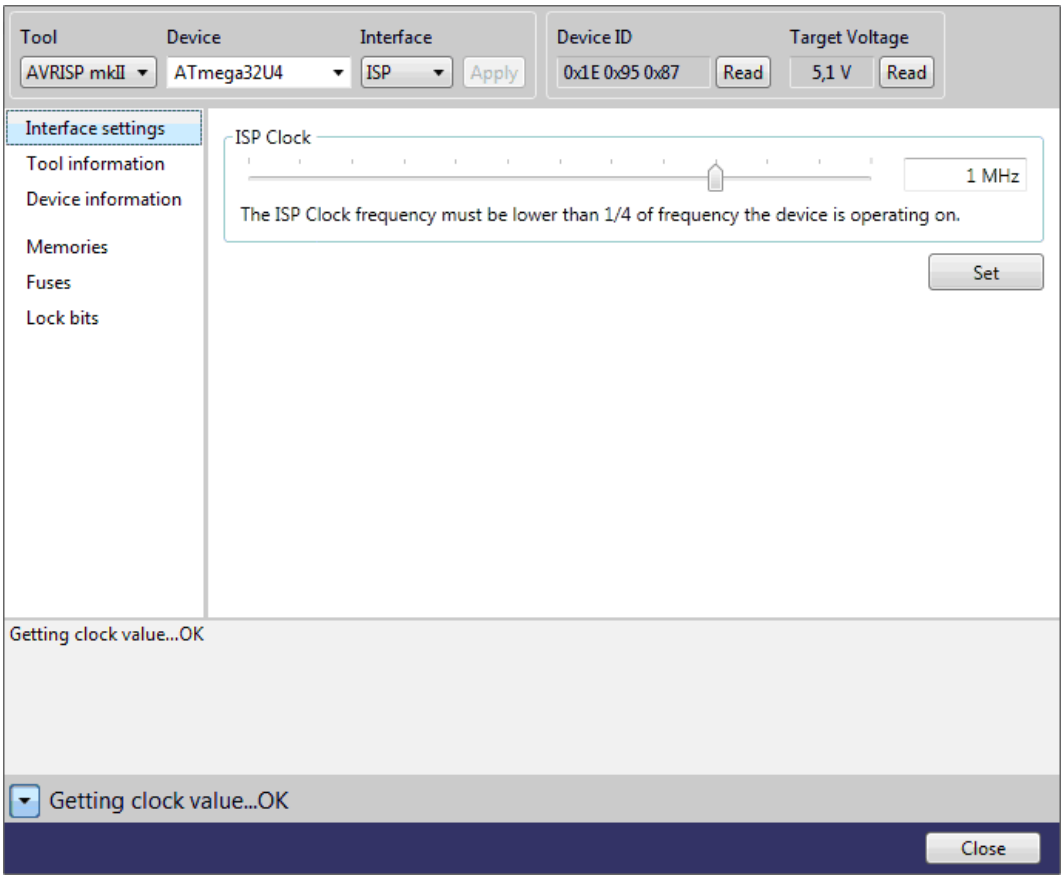



Figura C- 3: Ventana de opciones de programación (*Interface settings*).

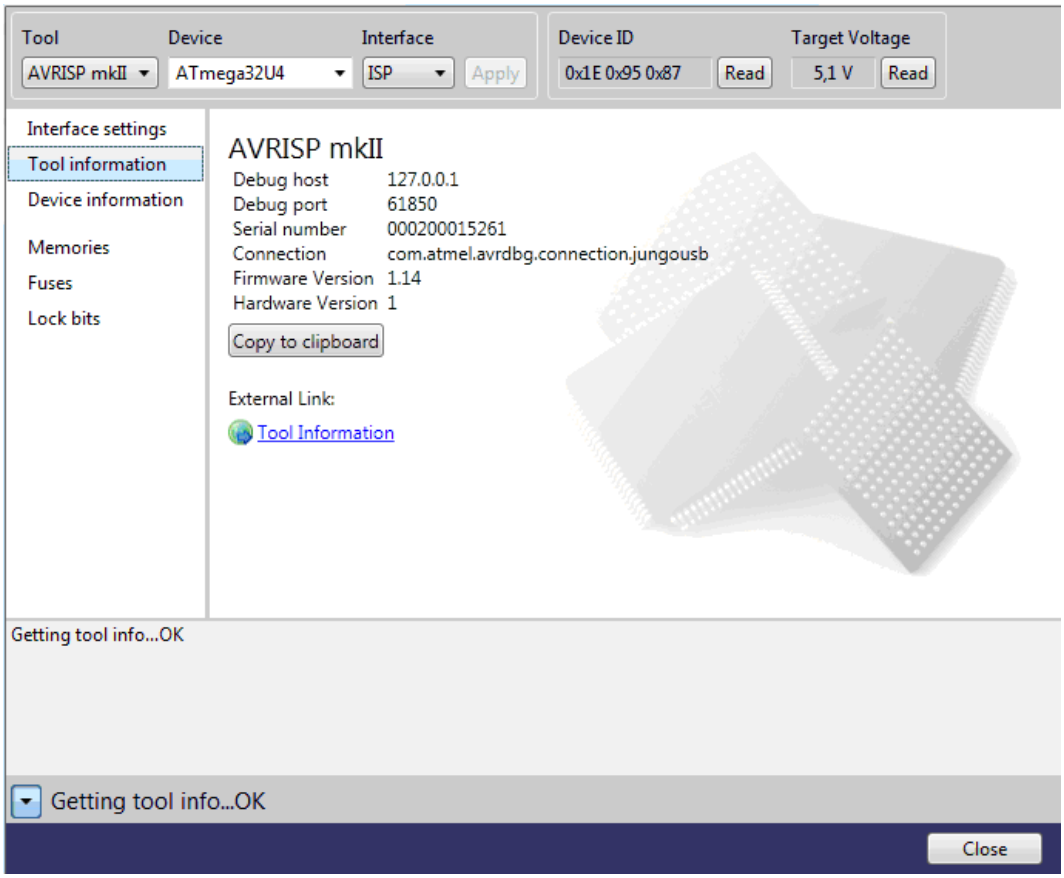



Figura C- 4: Ventana de opciones de programación (*Tool information*).

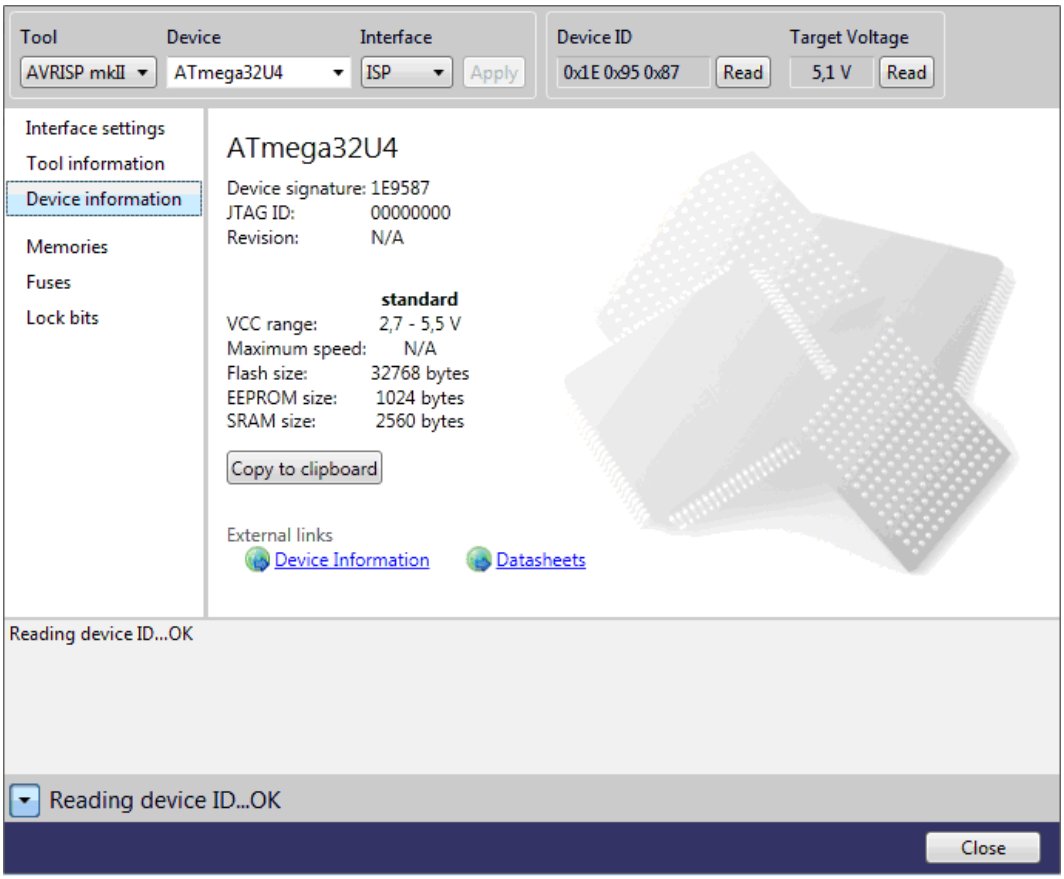



Figura C- 5: Ventana de opciones de programación (*Device information*).

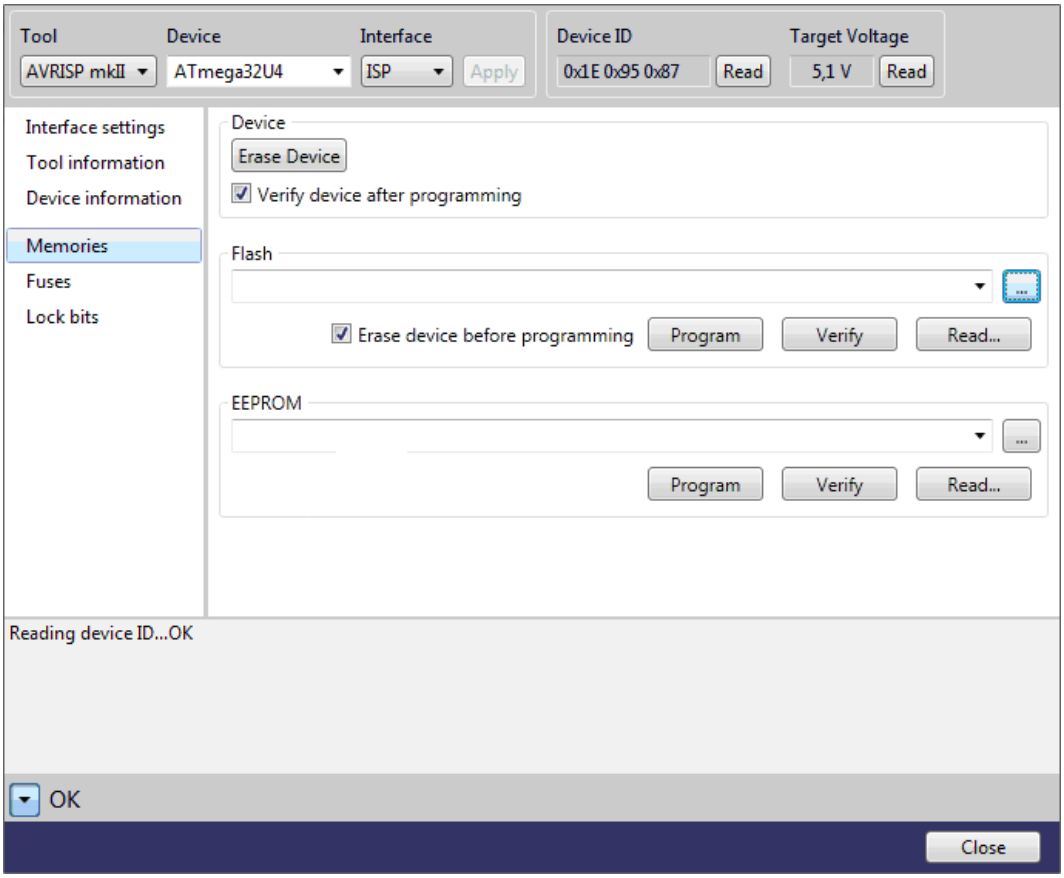



Figura C- 6: Ventana de opciones de programación (*Memories*).

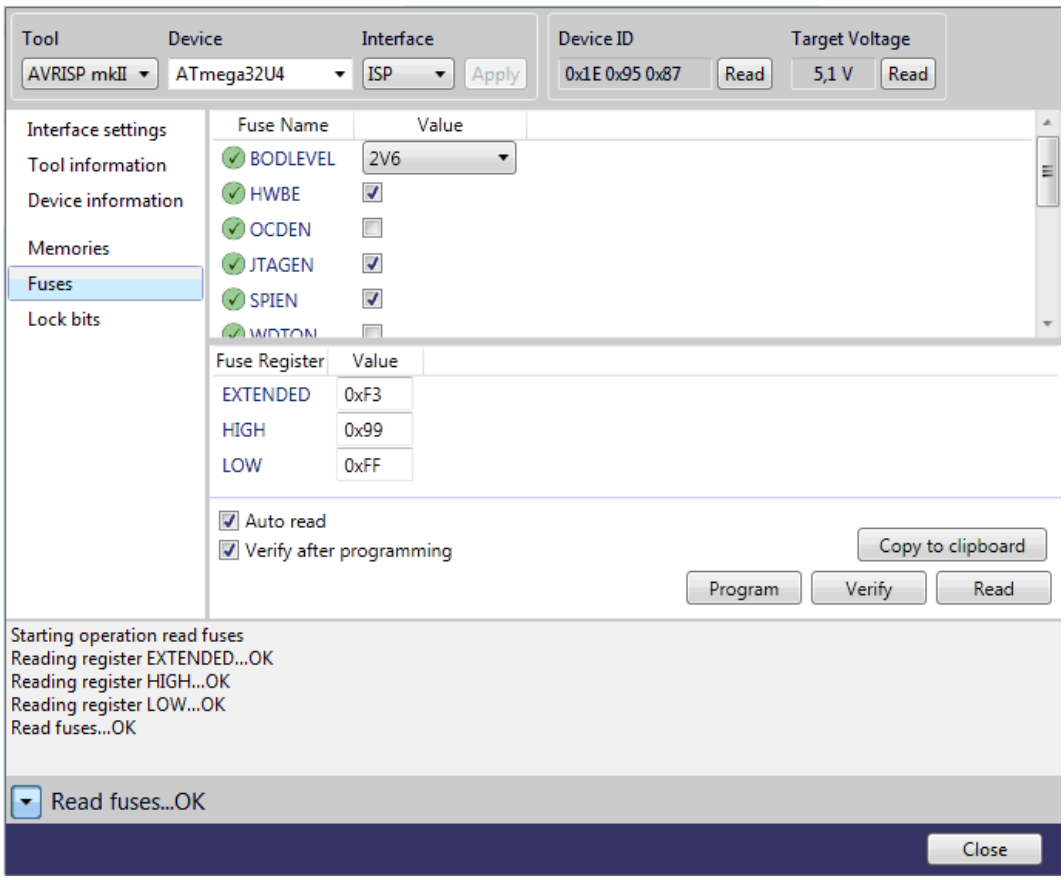



Figura C- 7: Ventana de opciones de programación (*Fuses*).

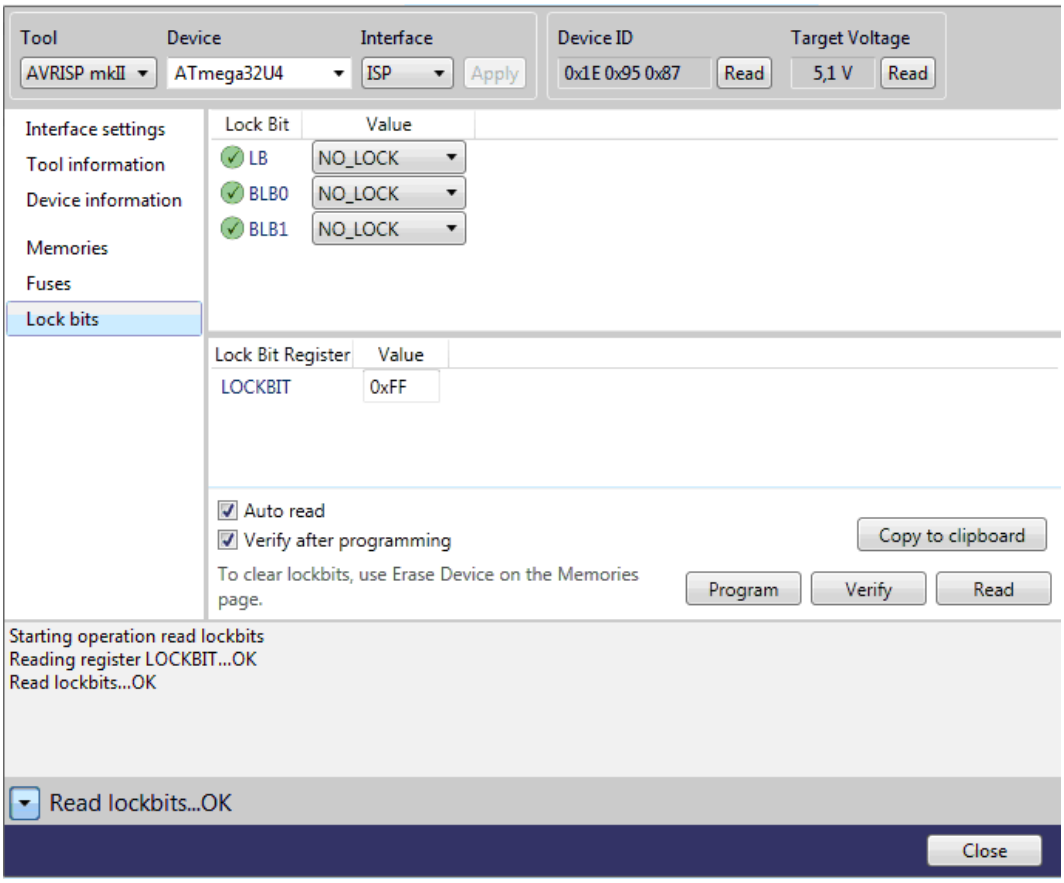



Figura C- 8: Ventana de opciones de programación (*Lock bits*).

A continuación se encuentran todas las opciones que facilita la herramienta. La primera vez que se conecta el microcontrolador a la herramienta se deben configurar los *fuses* y los bits de bloqueo y, si todo es correcto, no se vuelven a modificar. Esta es la configuración elegida y grabada en el chip:

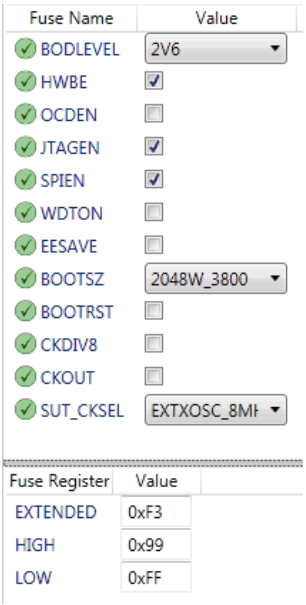



Figura C- 9: Configuración de los *fuses*.

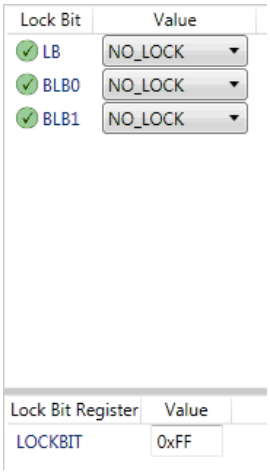



Figura C- 10: Configuración de los *Lock bits*.

Esta configuración permite activar la opción de oscilador externo al que se conecta un cristal de 16 MHz y desactivar la opción de división por 8 de la señal de reloj. También se activa la programación mediante USB del microcontrolador, que se podrá usar siempre y cuando se conserve o reinstale el "*bootloader*" preciso para ello, ya que para poder modificar los *fuses* es necesario borrar las memorias del chip por lo que este programa que viene preinstalado se pierde.

Si se conserva o reinstala el *bootloader* USB es necesario tener en cuenta un par de detalles para poder utilizarlo. Lo primero es instalar el software *Flip* del fabricante Atmel en el PC junto con sus *drivers*. Una vez instalado se ejecuta y, para que reconozca que el microcontrolador está conectado por USB, se debe colocar primero el *Jumper* HWB cerrando el circuito y, a continuación, cortocircuitar el *Jumper* de reset. Esto produce que el microcontrolador entre en la zona de memoria reservada para este *bootloader* y sea reconocido por la aplicación de PC. Para más información al respecto consultar [21].

Con esta configuración grabada en el chip, el siguiente paso es la programación del mismo, es decir, instalar la aplicación de usuario desarrollada para llevar acabo la lectura de tarjetas. Para ello se introduce la ruta del archivo ".hex" generado en apartados anteriores en la pestaña "*memories*" en la parte de memoria *flash*. A continuación se pulsa el botón "*Program*" y si todo va correctamente ya se tiene el microcontrolador programado y listo para ejecutar la aplicación diseñada.

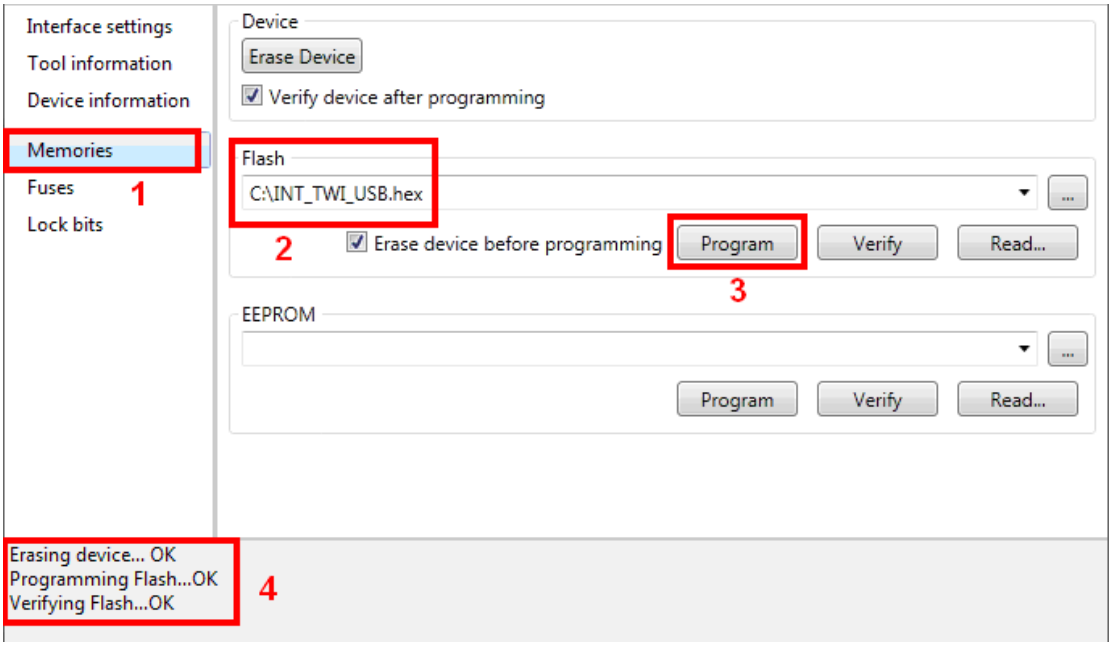



Figura C- 11: Proceso de programación del dispositivo.

Con la programación terminada se puede desconectar del lector el programador manteniendo conectado solamente la alimentación y el cable USB<->miniUSB.

Vamos a ver los pasos a seguir desde la conexión del lector a la alimentación hasta la realización de una lectura con éxito, explicando en cada momento qué respuesta se puede esperar por pantalla y que significado tiene el estado de los leds instalados en el lector.

Antes de conectar la alimentación se abre la aplicación “hid_listen.exe” en el PC que permite ver y registrar todos los eventos que suceden en el lector de tarjetas y que este envía por USB.


Figura C- 12: Pantalla de espera de la aplicación rastreadora del puerto USB.

Se conecta la alimentación. Lo primero que llama la atención es que un led rojo situado al lado del conector de alimentación se enciende. Esto significa que el sistema esta recibiendo tensión.

A continuación se encienden los dos led verdes situados junto al microcontrolador. Esto significa que se está realizando la configuración del sistema, por tanto no se debe hacer nada hasta que los dos leds se apaguen. Una vez apagados el sistema esta listo para trabajar y la aplicación del PC mostrara el siguiente mensaje:

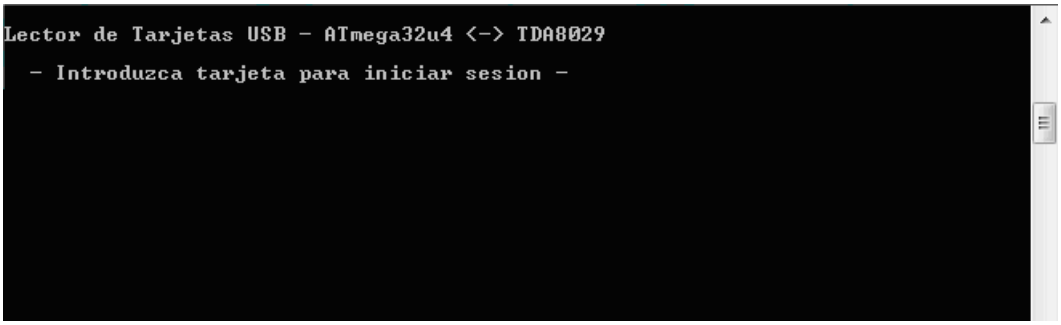



Figura C- 13: Pantalla de detección del dispositivo de la aplicación rastreadora del puerto USB.

Mientras no se produzca ningún error al ejecutar el código de aplicación por parte del microcontrolador ni se introduzca ninguna tarjeta, los leds verdes deben permanecer apagados y la aplicación del PC no debe mostrar ningún cambio.

Cuando se introduce una tarjeta en el lector, automáticamente se procede a su lectura y envío de los datos obtenidos por USB al PC. Esto se indica nuevamente mediante los leds verdes, cuando se comienza la lectura se encienden los dos y permanecen fijos hasta que se finaliza la lectura. En ese momento y para indicar que la tarjeta debe ser extraída, los leds comienzan a parpadear. Cuando el lector este listo para una nueva lectura lo indica manteniendo los dos leds verdes apagados.

En caso de lectura con éxito esto es lo que muestra la aplicación del PC por pantalla:


Figura C- 14: Pantalla de lectura completada de la aplicación rastreadora del puerto USB.

Estos códigos hexadecimales esconden los datos de libre acceso que poseen las tarjetas chip que funcionan como carné universitario. Se obtienen los siguientes datos:

ATR: respuesta al reset o "*answer to reset*". Como su nombre indica, son los datos que la tarjeta devuelve cuando es sometida a una acción de reset; en este caso, un reset posterior a su activación. Estos datos incluyen información sobre los parámetros de los circuitos integrados, características de los protocolos que utiliza y algunos datos históricos como fabricante, tiempo de vida…

Estos datos no son de interés para la aplicación desarrollada, tan solo se tiene en cuenta como un indicativo de que la tarjeta insertada ha sido inicializada y funciona correctamente.

Datos propietario:

- Nombre y apellidos.
- DNI.
- ID de usuario.
- Contraseña inicial de usuario.

Para una lectura donde se produce un error pero no se bloquea el sistema:


Figura C- 15: Pantalla de error de lectura de la aplicación rastreadora del puerto USB.

En caso de que la lectura no se lleve a cabo por algún motivo, el dispositivo está protegido contra bloqueos con una cuenta atrás que si no se resetea cada 8 segundos como máximo reinicia el lector (*Watchdog timer* 8 segundos). En caso de reseteo los

leds actúan como en un inicio normal del sistema. La aplicación del PC mostrará por pantalla algo de este estilo:

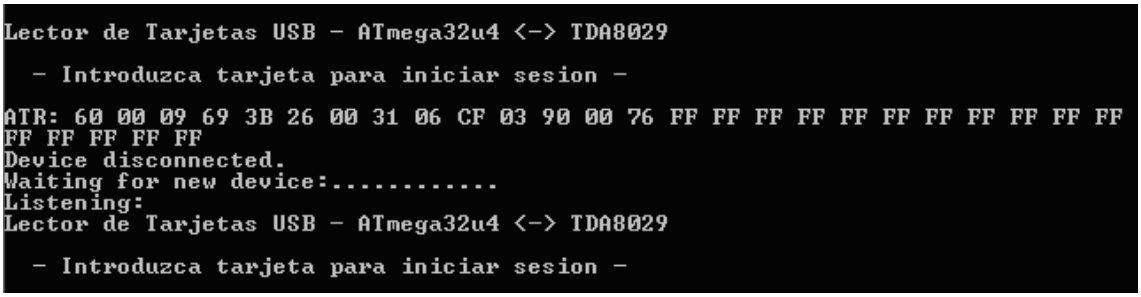



Figura C- 16: Pantalla tras reset del sistema de la aplicación rastreadora del puerto USB.

La imagen muestra cómo después de obtener el ATR de la tarjeta se produce un bloqueo y no continúa con la lectura. Al cabo de 8 segundos el sistema se reinicia y se restablece.

De forma simultánea, todo lo que la aplicación del PC muestra por pantalla está siendo registrado en un archivo de texto llamado log_data.txt, que se crea si no existía en el momento que se lanza la aplicación o que se continúa escribiendo en caso de ya existir. Este archivo se creará o se debe colocar para seguir escribiendo sobre el en la misma localización en la que se encuentre el archivo “.exe” de la aplicación.

Estos son los pasos a seguir y los detalles a tener en cuenta a la hora de poner en marcha el lector y hacer un uso correcto del mismo.

## D. Control de versiones

Durante la realización de este proyecto se han superando distintos problemas, contratiempos, averías y ajustes de diversa índole. Esto da lugar a una serie de versiones del prototipo que finalmente se presenta.

A continuación se presentan estas versiones con una pequeña explicación de sus componentes principales, su diseño básico y por qué fueron desechas o evolucionadas.

### Versión 1.0:

Se trata de un diseño centralizado en un chip del fabricante TERIDIAN, el 73S1215F. Este chip es del tipo “todo en uno”. En otras palabras, es un chip que incorpora un pequeño microcontrolador, la capacidad de lectura de tarjetas, USB, comunicaciones por puerto serie e $I^2C$ y pines de entrada/salida programables.

Incluye todo lo necesario para hacer un primer prototipo de prueba en el mismo chip, lo que supone un ahorro de componentes, espacio y tiempo muy considerable.

El prototipo se diseña y construye, usando este chip acompañado de sus elementos periféricos necesarios para su funcionamiento, conector para tarjetas, regulador de voltaje, USB, puerto para $I^2C$ y puerto serie. Este es su esquemático y PCB resultante:

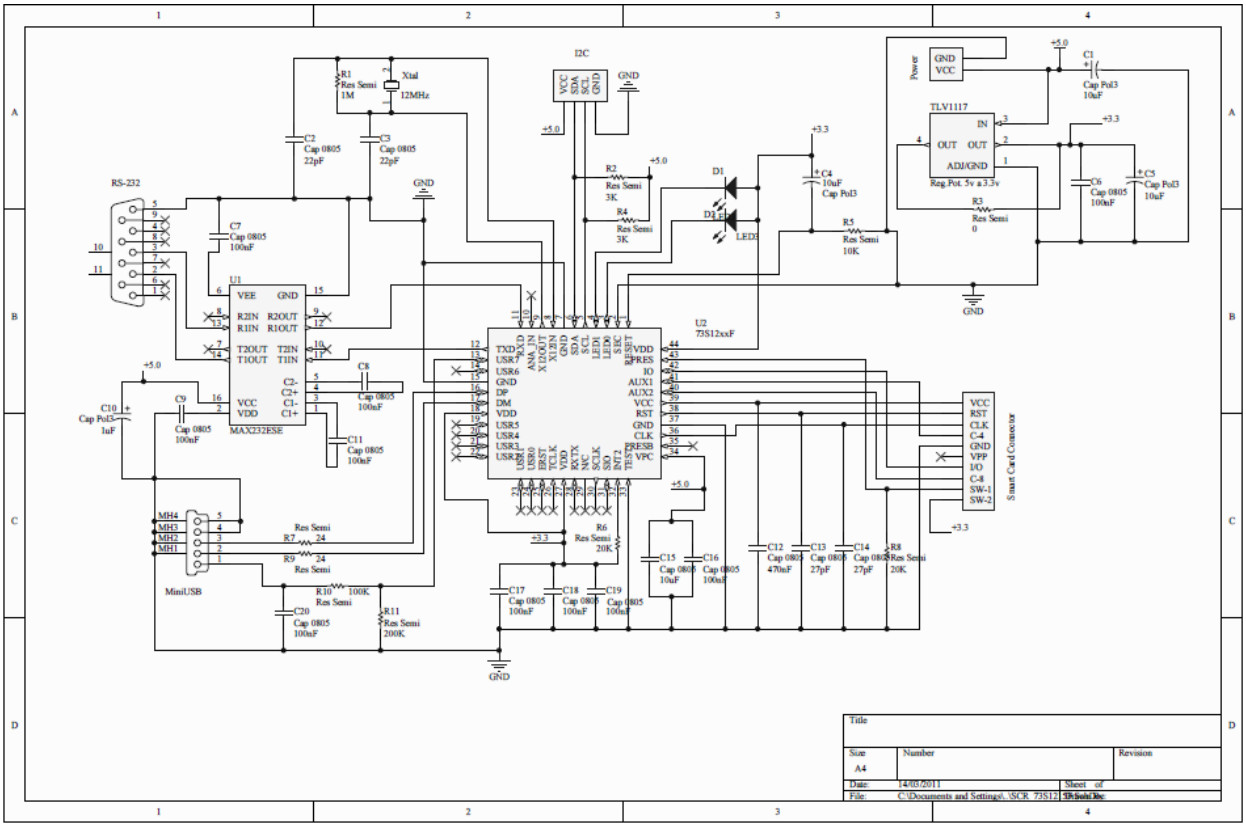



Figura D- 1: Esquemático PCB V1.0.

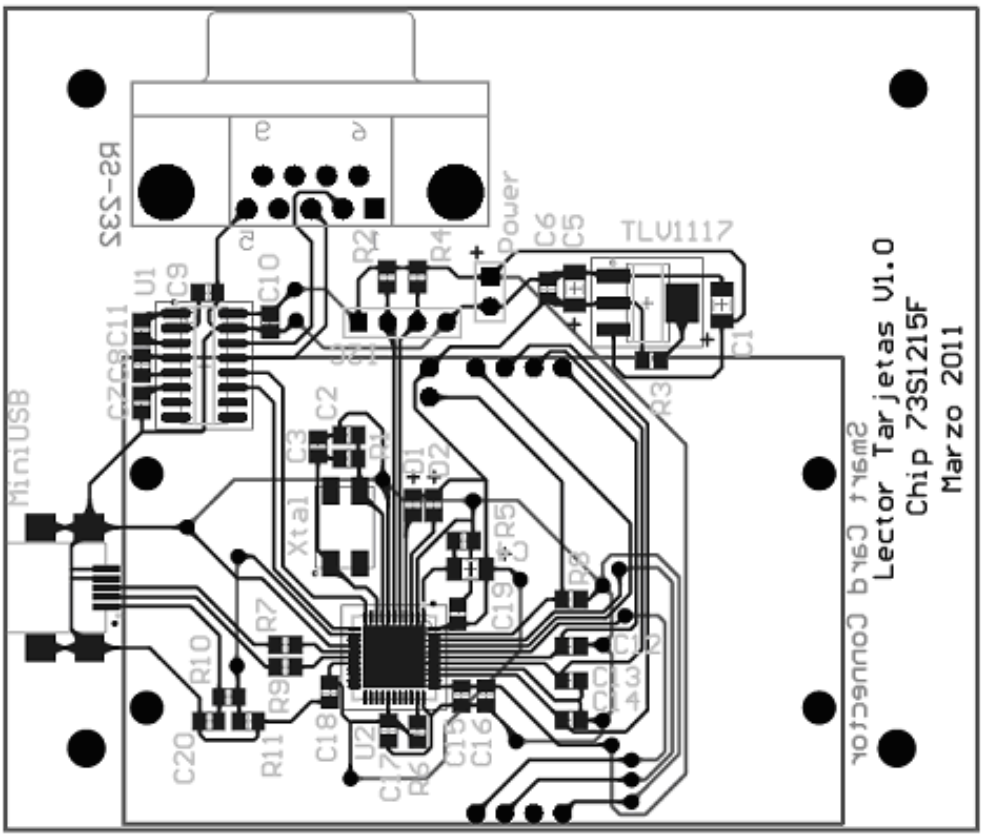



Figura D- 2: *Layout* completo PCB V1.0.

Tras la construcción y el montaje llega el momento de probar el funcionamiento. El prototipo no responde a ningún estímulo y no realiza ninguna operación. Se testean todas las líneas, tanto observando si reciben la alimentación correcta con un multímetro, como con el osciloscopio para ver si hay algún movimiento de datos. Tras muchas pruebas se llega a la conclusión de que el chip principal no viene programado de fábrica, lo que complica mucho su utilización y añade un coste extra en la compra de los elementos necesarios para esta tarea. Por este motivo se decide abandonar este diseño y enfocar el proyecto con una idea distinta, elementos separados y lo más estándares posibles.

Indicar también que este diseño supone el estudio y comprensión del funcionamiento del estándar de comunicación RS-232 y el uso de los reguladores de voltaje, tecnologías que no se vuelven a usar después.

## Versión 2.0:

Esta nueva versión representa la ideología anteriormente comentada. Se basa en un microcontrolador ATmega32U4 de la marca Atmel y un chip lector de tarjetas de la marca Philips, el TDA8029.

Estos chips se rodean de los elementos necesarios para su funcionamiento más un conector para tarjetas, un controlador de reset para el microcontrolador, un puerto USB y un puerto SPI. Estos son su esquemático y PCB asociado:

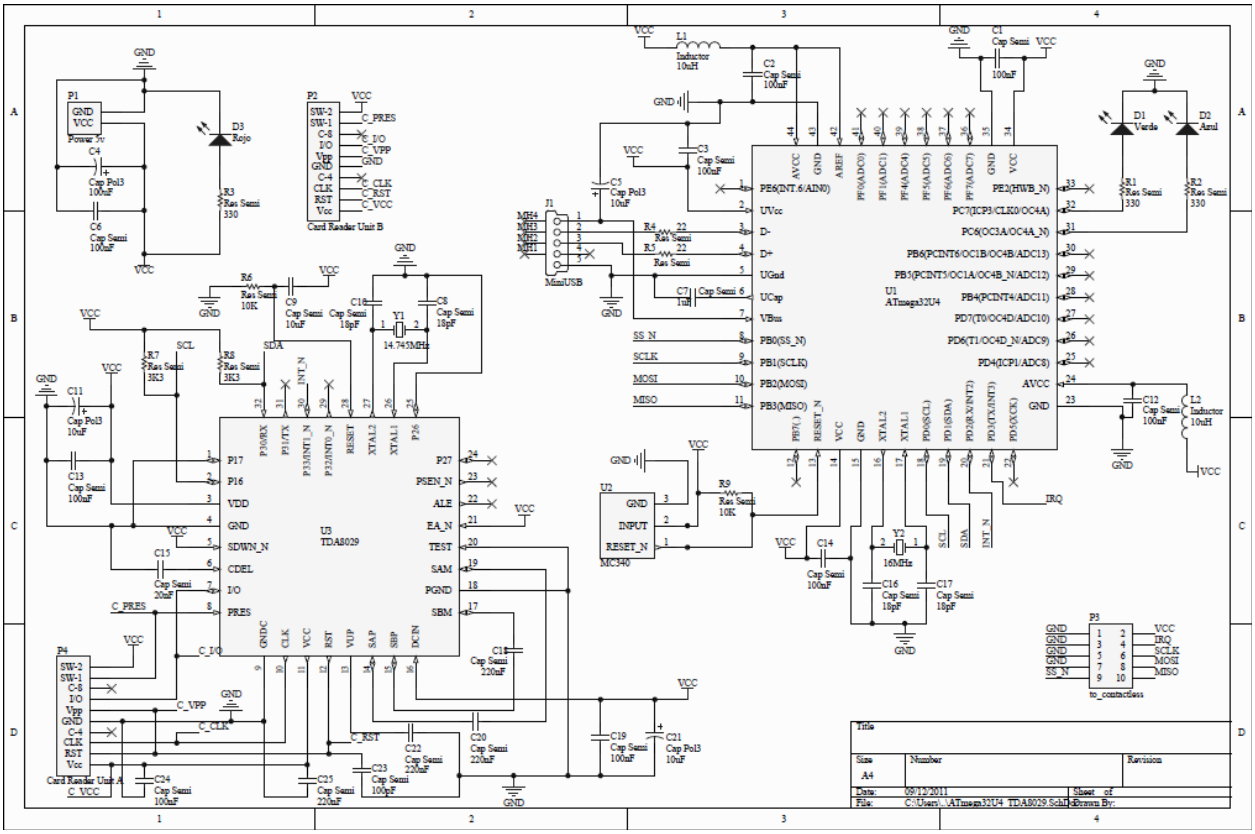


Figura D- 3: Esquemático PCB V2.0.

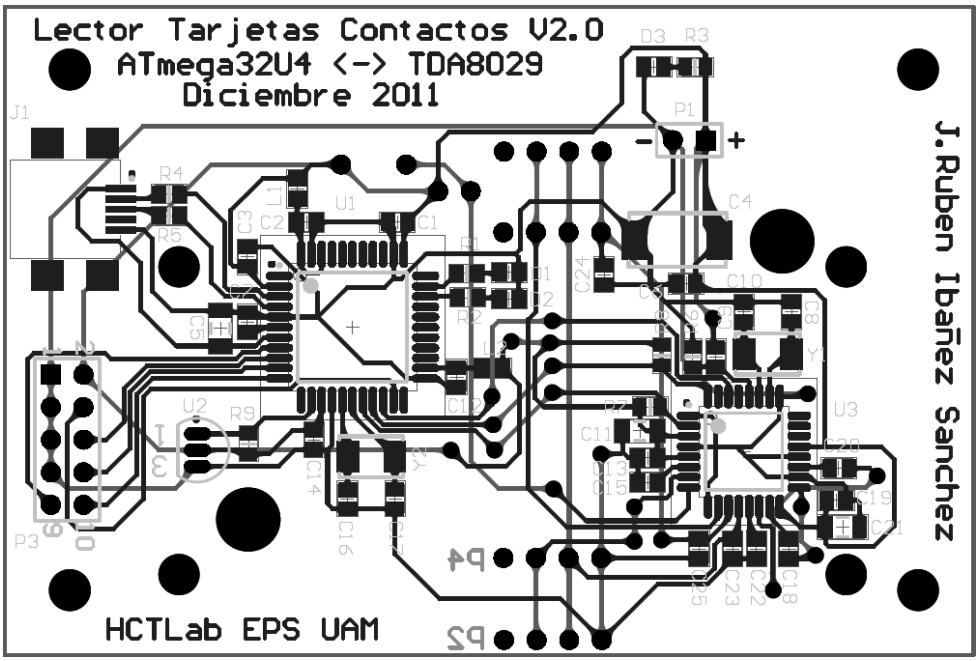



Figura D- 4: *Layout* completo PCB V2.0.

En principio tiene todo lo necesario para cumplir con las exigencias de este proyecto pero, tras el estudio en profundidad de algunos ejemplos de aplicación del microntrolador se detecta la necesidad de realizar algunos cambios y añadir y quitar algunos elementos. Esto da paso a la siguiente versión del prototipo.

## Versión 2.1:

Evolución del diseño anterior donde se realizan los siguientes cambios:

- Supresión del controlador de reset. Se sustituye por reset hardware RC más un pulsador.
- Se añade el puerto para el programador ISP.
- Cambio del pin de interrupción $I^2C$ en el TDA8029, pasando del pin 30 al pin 24.
- Se añade el *jumper* HWB al pin 33 del microcontrolador que realiza la función de direccionamiento de memoria al "*bootloader*" para USB.

Tras la realización de estos cambios se procede a la construcción y ensamblaje del prototipo. Estos son su esquemático y PCB asociado:

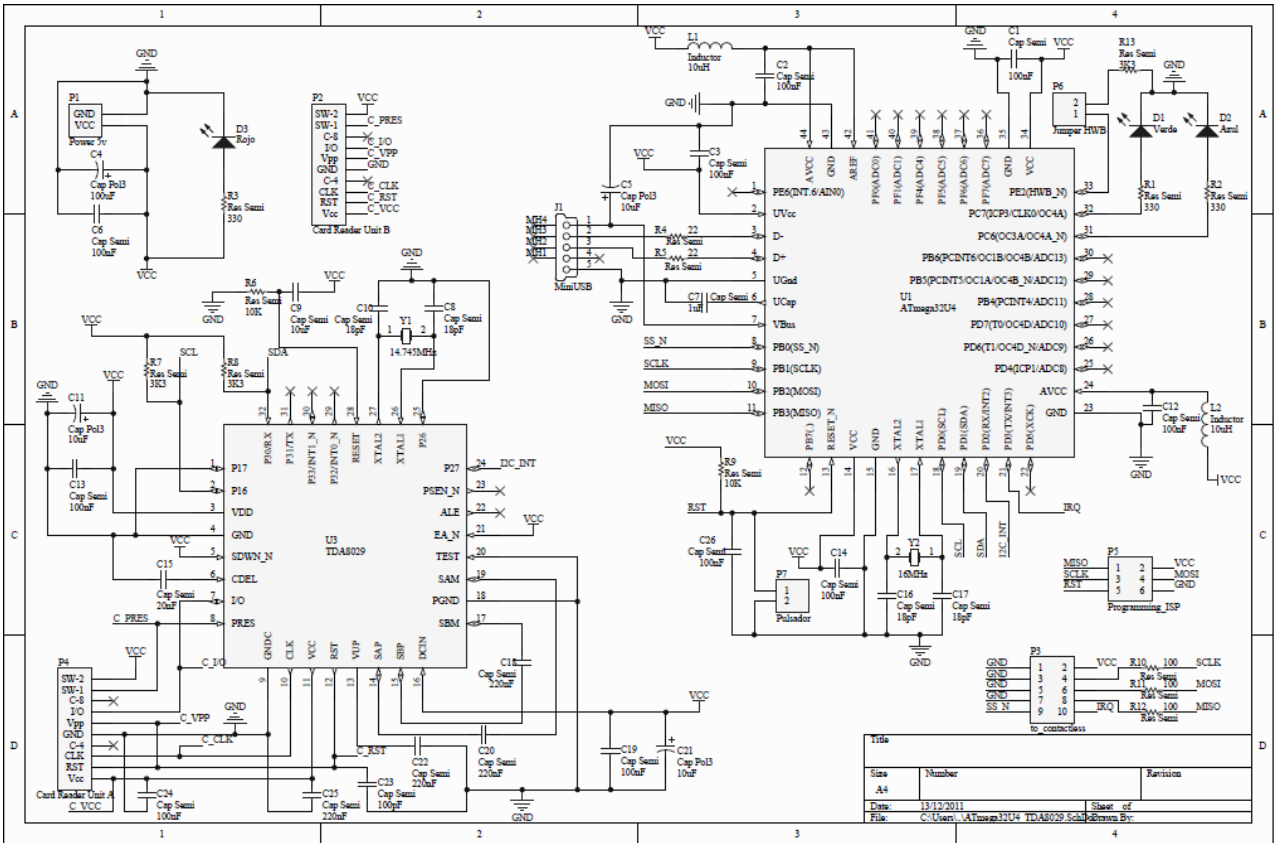


Figura D- 5: Esquemático PCB V2.1.

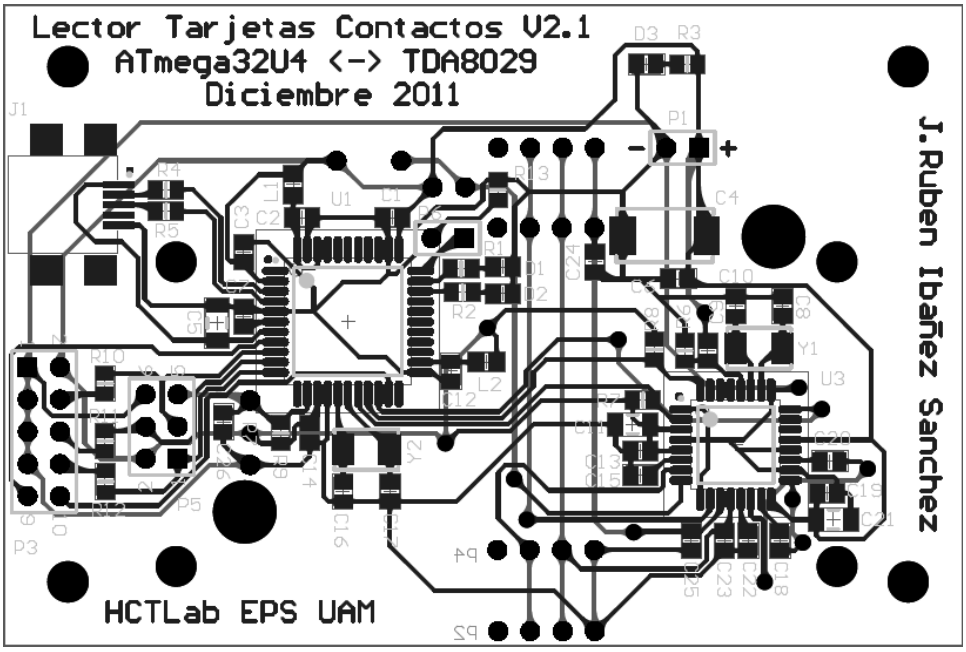



Figura D- 6: *Layout* completo PCB V2.1.

Tras su construcción y ensamblaje, el prototipo se programa para ver si es capaz de realizar todas las tareas para el que es diseñado. Después de muchas pruebas se detectan una serie de problemas y posibles mejoras que permitirían al dispositivo cumplir con todos los requerimientos. Por ello se vuelve a la fase de diseño y se realizan los siguientes cambios:

- Ampliación del *footprint* del conector miniUSB. Se amplían los *pads* de los anclajes pertenecientes a la carcasa del conector.
- Se elimina el reset RC del TDA8029 y se entrega el control del mismo al microcontrolador por medio de una línea exclusiva.
- Se modifica el rutado de alguna pista como consecuencia del cambio anterior.

## Versión 2.2:

Versión final del lector de tarjetas-chip con acceso USB. Esta versión es presentada y descrita con detalle a lo largo de esta memoria.

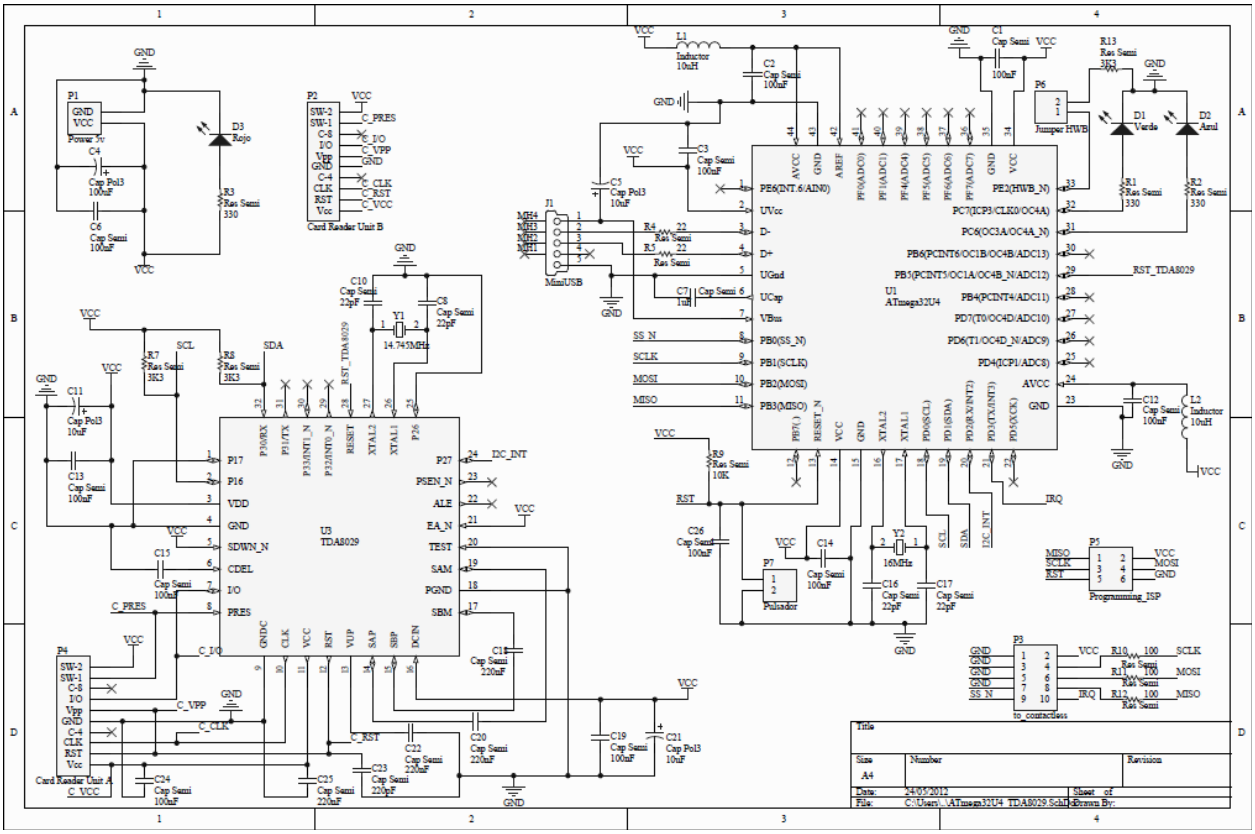


Figura D- 7: Esquemático PCB V2.2.

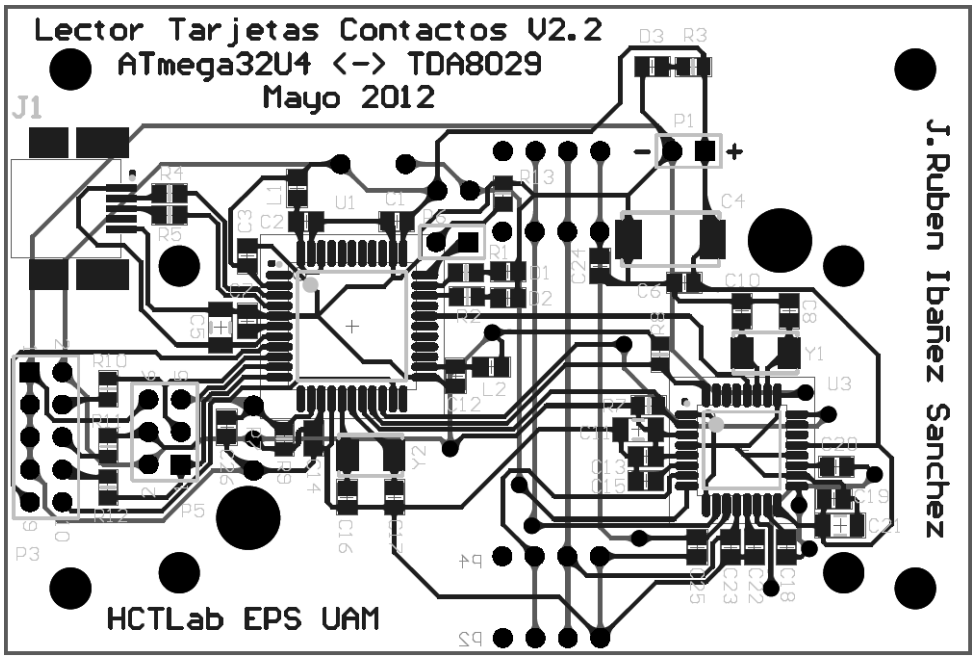



Figura D- 8: *Layout* completo PCB V2.2.

# Presupuesto

**1) Ejecución Material**

- Compra de ordenador personal (S.O. incluido)..... 1.000 €
- Licencia de *Altium Designer*..... 1.200 €
- Material de oficina..... 150 €
- Total de ejecución material ..... 2.350 €

**2) Gastos generales**

- 16 % sobre Ejecución Material..... 376 €

**3) Beneficio Industrial**

- 6 % sobre Ejecución Material..... 141 €

**4) Honorarios Proyecto**

- 750 horas a 30 € / hora..... 22.500 €

**5) Material fungible**

- Gastos de impresión ..... 200 €
- Encuadernación ..... 50 €

**6) Subtotal del presupuesto**

- Subtotal Presupuesto..... 25.617 €

**7) I.V.A. aplicable**

- 21% Subtotal Presupuesto..... 5.379,57 €

**8) Total presupuesto**

- Total Presupuesto.....30.996,57 €

Madrid, Enero de 2013

El Ingeniero Jefe de Proyecto

Fdo.: José Rubén Ibáñez Sánchez
Ingeniero Superior de Telecomunicación

# Pliego de condiciones

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un sistema lector de Tarjetas-Chip con acceso USB. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

## Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente llamado ”Presupuesto de Ejecución Material” que hoy designa otro concepto.

## Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.