Universidad de la República
Facultad de Ingeniería

Proyecto de fin de carrera

Recarga Fácil por Radio Frecuencia

$\mathbf{RF}^2$

Daniel Aicardi, Melina Rabinovich, Edgardo Vaz

Tutores: Ing. Juan Pablo Oliver, Ing. Andrés Aguirre

Montevideo, Uruguay
Julio 2011

Recarga Fácil por Radio Frecuencia

Resumen

El presente documento describe el prototipo Recarga Fácil por Radio Frecuencia, RF^2, realizado como proyecto de fin de carrera de Ingeniería Eléctrica en la Universidad de la República entre marzo de 2010 y julio de 2011. El mismo consiste en un sistema embebido para recarga y consulta de tarjetas RFID, como las que se utilizan hoy día en el sistema de transporte metropolitano, y fue diseñado para operar de forma autónoma interactuando directamente con el usuario.

El hardware fue enteramente diseñado por el grupo de trabajo a excepción de la single board computer. Las herramientas de software utilizadas son open source, así como también las bibliotecas usadas para desarrollar la aplicación final. El diseño, la fabricación, y el armado del prototipo fue realizado en su totalidad en Uruguay.

Agradecimientos

En primer lugar queremos agradecer a nuestras familias y amigos. Agradecemos al grupo mina del INCO, a Leonardo Steinfeld, Nicolás Barabino, Francisco Lanzari, María Eugenia Corti, Santiago Reyes, Viterbo Rodríguez, Silvana Castro, Christian Gutierrez, Andrés Bergeret, Gonzalo Tabares, Klaus Rotzinger, Marcelo Fiori, Pablo Cancela, Ana y Claudia Rabino. Y a todos los que de alguna forma u otra colaboraron con nosotros.

A nuestras familias.

Prefacio

"El ciudadano Línea saca su billetera, extrae su tarjeta y la introduce en la máquina registradora; una serie de gestos automáticos. Unas mandíbulas de aluminio se cierran sobre ella, unos dientes de cobre buscan la clave magnética, y una lengua electrónica saborea la vida del ciudadano Línea. Lugar y fecha de nacimiento. Padres. Raza. Religión. Historial educativo, militar y de servicios civiles. Estado. Hijos. Ocupaciones, desde el comienzo hasta el presente. Asociaciones. Medidas físicas, huellas digitales, retínales, grupo sanguíneo. Grupo psíquico básico. Porcentaje de lealtad, índice de lealtad en función del tiempo hasta el momento del último análisis... ... El ciudadano Línea se encuentra en la ciudad donde, la noche anterior, dijo que estaría, así que no ha tenido que hacer una corrección. Los nuevos informes se añaden al historial del ciudadano Línea. Toda su vida regresa al banco de datos. Desaparece de la unidad exploradora y la unidad comparativa, para que éstas atiendan la próxima llegada. La máquina ha tragado y digerido otro día. Está satisfecha."

Sam Hall (1953), Poul Anderson

La narración anterior es parte de un cuento de ciencia ficción llamado "Sam Hall", escrita por Poul Anderson en 1953. En esta historia el autor describe un mundo donde cada persona tiene asignada una tarjeta conteniendo datos que la caracterizan, y puede ser controlado su accionar a través de una super computadora que almacena y procesa los datos de toda la humanidad. En nuestros días este cuento de ciencia ficción no está tan alejado de la realidad, las tarjetas "inteligentes" (smart cards) son cada vez más usadas en múltiples aplicaciones como ser, pasaporte electrónico, pago electrónico, sistemas de transporte, controles de acceso y sistemas de seguridad, entre otros. El siguiente proyecto se desarrolla con la intención de aprender las bases del mundo de las tarjetas "inteligentes" y que sirva como punto de partida para que otros entiendan su funcionamiento. Los autores no desean que se use el contenido de este documento con fines como los que se indicaban en la narrativa de ciencia ficción, muy por el contrario, el empleo de esta tecnología debe estar en favor de las personas y no en su contra.

Índice general

Índice de figuras

Índice de cuadros

Glosario

AFE – Artefacto Feo de Exhibir, dispositivo para recargar tarjetas RFID.

APDU – Application Protocol Data Unit, comando enviado desde la capa de aplicación a través de un lector a una tarjeta inteligente.

ASCII – American Standard Code for Information Interchange, Código Estadounidense Estándar para el Intercambio de Información.

ASIC – Application-Specific Integrated Circuit, circuito integrado para aplicaciones específicas

ASK – Amplitude Shift Key, modulación por amplitud de pulsos.

ATR – Answer To Reset, respuesta de una tarjeta inteligente luego de un reset.

baud rate – Usado de manera análoga a la tasa de bits por segundo.

CCID – Chip/Smart Card Interface Devices controlador genérico de dispositivos lectores de smart card con interfaz USB.

CL RC632 – Circuito integrado para lectores de tarjetas RFID de protocolo múltiple.

EMC – Electromagnetic Compatibility, conjunto de reglas dadas por la FCC para la compatibilidad electrónica de sistemas electrónicos.

ext3 – Sistema de archivos extendido, usado en sistemas operativos Linux.

FAT32 – Sistema de archivos FAT (File Allocation Table) de 32 bits, desarrollado para MS-DOS.

FCC – Federal Communications Commission, Comisión Federal de Comunicaciones de EEUU.

FIFO – First In First Out, consepto usado para indicar que el primer dato en ser leído es el primero en ser procesado y liberado.

fileSystem – Sistema de archivos manejado por el sistema operativo.

GDB – GNU Debugger, herramienta para depurar código fuente.

GPIO – General Purpose Input/Output puerto de entrada/salida de propósito general.

I/O – Input/Output puerto de entrada/salida.

I2C – Inter-Integrated Circuit, bus de comunicaciones serie para interconectar microcontroladores y/o circuitos integrados entre sí.

IDE – Integrated Development Environment, herramienta para el desarrollo de software.

IIE – Instituto de Ingeniería Eléctrica.

ISO – International Organization for Standardization, organización internacional para la estandarización.

ISR – Interrupt Service Routine, rutina de atención a una interrupción.

JTAG – Joint Test Action Group, puerto que permite entre otras cosas depurar la aplicación que se ejecuta en el sistema embebido.

LCD16x2 – Liquid Crystal Display, pantalla de dos líneas de 16 caracteres cada una.

LDO – Low Drop Output, salida de baja caída en reguladores de tensión.

lector mudo – Lector de tarjetas, que no cuenta con un ASIC entre la tarjeta y el microcontrolador.

Mifare – Tecnología de tarjetas inteligentes sin contacto (TISC) que cumplen con el estándar ISO14443.

Mifare Classic – Tarjetas inteligentes sin contacto, con memoria no volátil de 1K o 4K.

open-firmware – Software que permite el manejo directo del hardware donde se encuentra almacenado, cuyo código es abierto, permitiendo ser modificado y distribuído en forma libre.

open-hardware – Dispositivos de hardware cuyas especificaciones y diagramas esquemáticos son de acceso público.

open source – Software cuyo código es abierto, permitiendo ser modificado y distribuído en forma libre.

OpenPCD – Dispositivo lector/escritor de tarjetas RFID.

OTG – On The Go, clase de puerto USB que puede ser usado como Host o como Device.

PC – Personal Computer, computador personal.

PCB – Printed Circuited Board, placa de circuito impresa.

pcsclite – Biblioteca para el uso con smart cards.

PIC – Familia de microcontroladores tipo RISC (reduced instruction set computer) fabricados por Microchip Technology Inc.

Protocolo T=0 – Protocolo de datos orientado a byte.

Protocolo T=1 – Protocolo de datos orientado a bloques.

RF – Radio Frecuencia.

RF^2 – Recarga Fácil por Radio Frecuencia.

RS232 – Interfaz que designa una norma para el intercambio serie de datos binarios entre un DTE (Equipo terminal de datos) y un DCE (Data Communication Equipment, Equipo de Comunicación de datos).

RSTPD – Reset and Power Down, pin de reset y apagado del integrado CL RC632.

SBC – Single Board Computer.

SC – Smart Card, tarjeta inteligente.

SCUI – PCB que contiene un lector/escritor de tarjetas de contacto e interfaz de usuario.

SD – Secure Digital, es un formato de tarjeta de memoria.

SDK – Software Development Kit, herramienta para desarrollo de software.

SIMO – Slave In Master Out, entrada esclavo salida maestro en un puerto SPI.

SOC – System On a Chip.

SOMI – Slave Out Master In, salida esclavo entrada maestro en un puerto SPI.

SPI – Serial Peripheral Interface, bus de comunicación serie.

UART – Universal Asynchronous Receiver-Transmitter, puerto serie con transferencia de datos de forma asíncrona.

UID – Código único de identificación de tarjeta RFID.

USB – Universal Serial Bus.

USB4ALL – Interfaz USB genérica para comunicación con dispositivos electrónicos.

VLT – Voltage Level Translator, nombre del PCB que contiene circuitos integrados para el traslado de niveles de tensión.

Parte I

Introducción

Capítulo 1

Introducción

En el año 2010, se pone en marcha en Montevideo el Sistema de Transporte Metropolitano. El mismo propone el pago de viajes a través de una tarjeta que simplemente se acerca a un dispositivo para efectuarlo. Esa tarjeta logra comunicarse con el dispositivo en forma inalámbrica utilizando radiofrecuencia y se denomina tarjeta RFID.

El usuario del Sistema de Transporte Metropolitano puede recargar su tarjeta en los puntos de venta. Allí debe presentar su tarjeta e indicar el monto que desea acreditar, un operador recibe la tarjeta y la apoya sobre un dispositivo que se encuentra conectado a un PC de escritorio. Luego que la tarjeta es leída aparecen en pantalla los datos asociados a la misma, nombre del usuario, saldo disponible, etc., que están almacenados en un servidor centralizado en la Intendencia de Montevideo (IM). El operador utiliza un programa que por un lado registra la transacción en el servidor central de la IM y por otro actualiza el saldo en la tarjeta RFID.

Por razones de seguridad cada dispositivo de consulta/recarga posee una tarjeta de contactos denominada SAM (Security Access Module) que se utiliza para encriptar las transacciones entre el terminal y el servidor central, y por otro lado se encarga de generar las claves de acceso para la comunicación entre la tarjeta RFID y el terminal. Esta tarjeta es la encargada de dar seguridad en la transacción de modo que el sistema no sea vulnerable.

Si el usuario desea consultar el saldo disponible en la tarjeta, podrá observarlo en el boleto cuando haga uso de su tarjeta en el ómnibus, o a través de un operador en los

puntos de venta.

Se puede desprender de lo anterior, que para poder consultar y/o recargar una tarjeta, es necesario un sistema (ver figura 1.1) que pueda comunicarse con el servidor central de la IM, con las tarjetas RFID de los usuarios a efectos de leerlas y escribirlas, con la tarjeta de seguridad, SAM, y que cuente con algún tipo de interfaz de usuario.

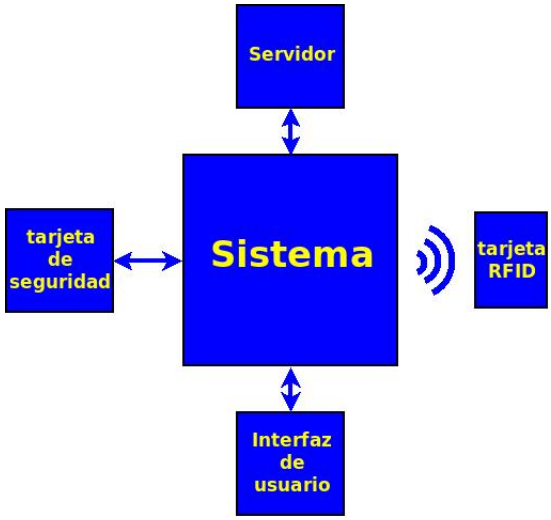

Figura 1.1: Bloques que conforman el sistema a diseñar

Este proyecto busca desarrollar un dispositivo autónomo, de bajo costo y mantenimiento, que permita realizar estas operaciones funcionando en línea con un servidor, de forma rápida, segura y autogestionada por parte del usuario, sin necesidad de personal, en diversos puntos de Montevideo.

La propuesta entonces es realizar un dispositivo, a partir del cual se puedan consultar y recargar tarjetas como las utilizadas en el Sistema de Transporte Metropolitano. En este sistema, el usuario que desee consultar el saldo actual no tiene más que acercar su tarjeta y esperar a que el sistema le indique el saldo disponible en la misma. Para poder acreditar saldo en su tarjeta debería antes efectuar el pago de dinero que desea acreditarle, a través de una red de pagos, mensajes de texto, web, o un mecanismo similar que se encuentra desacoplado del sistema implementado. La transacción anterior actualiza el servidor central con el saldo pendiente para que al acercar la tarjeta al dispositivo simplemente se acredite el saldo por el cual se pagó.

1.1. Objetivo general

El objetivo del proyecto es la fabricación de un prototipo de sistema embebido capaz de consultar y recargar tarjetas. Para ésto, como se mencionó en el punto 1.1, deberá lograr establecer comunicación con tarjetas como las utilizadas en el Sistema de Transporte Metropolitano (comunicación RFID a 13,56MHz), con tarjetas de contacto (módulo de seguridad SAM), con el usuario a través de una interfaz simple y con el servidor central.

Esto implica entonces la fabricación de dos lectores/escritores de tarjetas, uno para tarjetas RFID (sin contacto) y otro para tarjetas con contacto (SAM), una interfaz para el usuario capaz de informar el estado de la transacción mediante mensajes adecuados, y la utilización de un sistema basado en un microprocesador para controlar los periféricos y realizar las operaciones. Esto último implica además el desarrollo del software para que todo funcione adecuadamente.

1.2. Antecedentes

Existen antecedentes de todas las partes a diseñar.

El lector/escritor de tarjetas empleado en el Sistema de Transporte Metropolitano (IntegriSys iMFR) cumple la función de lectura y escritura de las tarjetas sin contacto y además contiene dentro un lector de tarjetas de contacto usado para la seguridad en las transacciones. Otro ejemplo de lector/escritor de ambos tipos de tarjetas, es el SCM SDI010.

Hay lectores/escritores exclusivamente de tarjetas de contacto como puede ser el Omnikey 3121, y lectores/escritores de tarjetas sin contacto como el ACR120.

Algo más completo (a nivel hardware), es un dispositivo lector/escritor RFID llamado OpenPCD [ref], de hardware abierto, fabricado en Alemania.

El único prototipo completo que se conoce como antecedente es el realizado por el grupo de electrónica de la IM llamado AFE (Artefacto Feo de Exhibir), que realiza lo

mismo que se propone pero con un enfoque de diseño diferente, ya que utiliza lectores/escritores de escritorio como los mencionados antes y no de diseño propio.

1.3. Alcance

Respecto al hardware, se fabricará un lector/escritor RFID (basado en el antecedente OpenPCD), un lector de tarjetas de contacto donde se insertará el módulo de seguridad (SAM), una interfaz de usuario donde se incluirá un display, leds y un indicador sonoro. Se estudiará la forma de conectar dichos periféricos a un sistema central capaz de controlar todas las funcionalidades.

Respecto al software, se desarrollará todo el software necesario para que el sistema funcione, haciendo lo posible para reutilizar código ya implementado para fines similares y lograr la mayor compatibilidad con lo ya existente.

1.4. Especificación funcional

El prototipo final deberá ser capaz de interactuar con tarjetas RFID a través del lector/escritor RFID, con una tarjeta de contacto SAM (seguridad) y con un servidor. Luego de los controles correspondientes en la tarjeta RFID, comenzará la interacción con el usuario mediante un display, leds e indicador sonoro, que será la interfaz de comunicación con el mismo. El display informará al usuario de las tareas que se estén realizando con mensajes cortos y descriptivos. Los tiempos de recarga y consulta deberán ser menores a un minuto.

1.5. Criterios de éxito

El proyecto será considerado exitoso si se logra construir un dispositivo de sistema embebido capaz de consultar y recargar tarjetas RFID, con tiempos de operación razonablemente cortos.

Parte II

Diseño

Capítulo 2

Funcionamiento del prototipo

2.1. Requerimientos

El principal requerimiento a cumplir es la interacción con tarjetas RFID (ver apéndice A), tanto para su lectura como escritura. La comunicación con tarjetas de contacto (ver apéndice A) es necesaria para la interacción con un módulo de seguridad que permita, la generación de las claves utilizadas para autenticarse con las tarjetas RFID, y una transacción segura con un servidor. En ambos casos es necesario cumplir con las normas y estándares adecuados (tarjetas RFID - ISO 14443 y tarjetas de contacto - ISO7816). Por último mantener informado al usuario de lo que sucede durante una transacción a través de una interfaz visual y sonora.

2.2. Descripción del prototipo

Este prototipo integra la lista de dispositivos que hoy en día se denominan sistemas embebidos. Su hardware está integrado por un sistema basado en un microprocesador que recibe el nombre de Single Board Computer (SBC), a la que se agrega un conversor de niveles que permite interconectarla con, un lector/escritor de tarjetas RFID a través de un puerto SPI, un lector de tarjetas de contacto a través de un puerto serial (UART), y la interfaz de usuario compuesta por un buzzer, tres leds (rojo, amarillo, verde) y un display conectado a través de puertos de entrada/salida de propósito general (GPIO).

En cuanto al software, está basado en bibliotecas de código abierto que permiten

desarrollar la aplicación que asegura el manejo del hardware y el funcionamiento de todo el sistema en conjunto.

2.3. Funcionamiento general del prototipo

Una vez que el prototipo RF^2 se encuentra operativo, el dispositivo despliega en el display el mensaje “Aproxime su tarjeta”, permaneciendo en dicho estado hasta que algún usuario acerque una tarjeta al lector/escritor RFID. En la primera transacción entre lector y tarjeta se obtiene el identificador único (UID) de ésta última, que será enviado al módulo de seguridad SAM (previa autenticación exitosa), para que a partir de éste, se generen las claves de acceso que permitan la lectura y escritura de la tarjeta RFID. Mientras se lleva a cabo la operación, se despliega en el display el mensaje, “No retire su tarjeta” a la vez que el led amarillo es encendido para indicar precaución ya que se están procesando datos. La siguiente acción a llevar a cabo es verificar que la tarjeta del usuario tenga saldo pendiente de acreditar, en caso afirmativo se indica al usuario el saldo a acreditar a través del display con el mensaje “Saldo a acreditar $...”. Si todo fue exitoso, se borra el saldo transferido de la lista de saldos pendientes a acreditar para que no se transfiera saldo indefinidas veces. A continuación se despliega en el display el nuevo monto almacenado en la tarjeta, “Su saldo es de $...”, se enciende el led verde y se emite un pitido mediante el buzzer en señal que la operación fue satisfactoria. Por último se muestran en el display los mensajes “Transacción finalizada”, “Gracias” y vuelve al inicio para comenzar un nuevo ciclo.

En caso que la tarjeta no tuviera saldo pendiente de acreditar, el prototipo RF^2 funciona en modo consulta y despliega en el display el saldo disponible en la tarjeta, “Su saldo es de $...”, encendiendo el led verde y emitiendo un pitido, seguido de los mensajes “Transacción finalizada”, “Gracias” y vuelve al inicio para comenzar un nuevo ciclo.

En caso de ocurrir un error durante alguno de los pasos anteriores, ya sea porque el usuario retiró la tarjeta en un momento inadecuado, o simplemente porque el prototipo RF^2 no logró leer o escribir la tarjeta en forma correcta, se enciende el led rojo, se emite un doble pitido mediante el buzzer, y el display muestra el mensaje “Error, vuelva a intentarlo”, acto seguido el ciclo vuelve a comenzar.

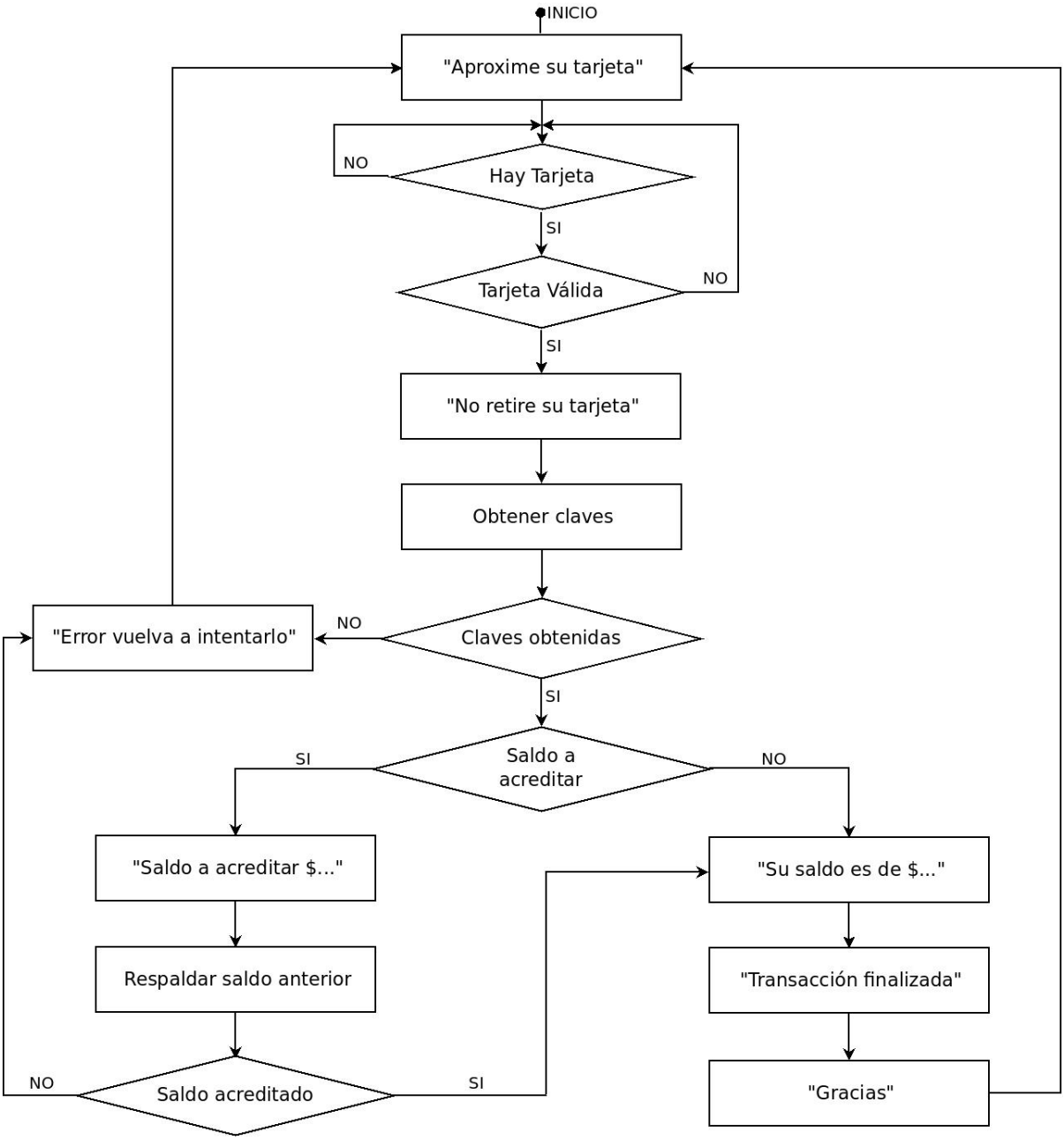

Figura 2.1: Diagrama de flujo

Capítulo 3

Hardware

3.1. Arquitecturas estudiadas

Se plantearon varias alternativas como posible solución. A medida que se encontraron limitantes o que no se cumplían los requerimientos exigidos, se fueron descartando dichas opciones.

A continuación se describen algunas de las arquitecturas consideradas:

1 - OpenPCD + lector de tarjetas de contacto + display + buzzer + leds

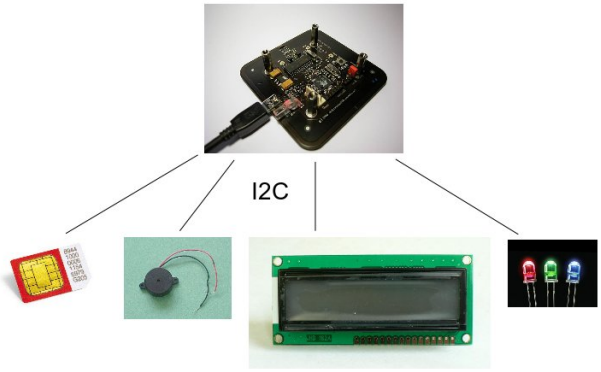

Figura 3.1: Solución considerada 1

Al dispositivo OpenPCD, se conecta el resto del hardware a través de su único puerto de entrada/salida disponible que es de tipo I2C.

2 - SBC + OpenPCD + microcontrolador + lector de tarjetas de contacto + display + buzzer + leds

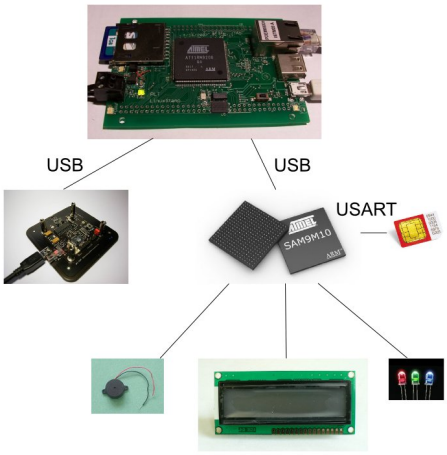

Figura 3.2: Solución considerada 2

Tanto el dispositivo OpenPCD como el microcontrolador se conectan directamente por USB a la SBC. El microcontrolador maneja el resto de los dispositivos (lector de tarjetas de contacto, display, buzzer y leds).

3 - SBC + OpenPCD + lector de tarjetas de contacto + display + buzzer + leds

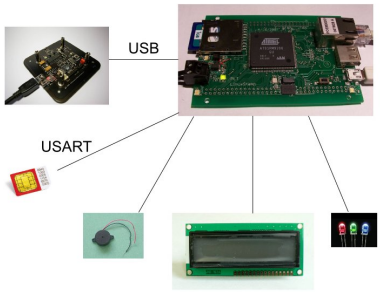

Figura 3.3: Solución considerada 3

El dispositivo OpenPCD se conecta por USB a la SBC. La SBC maneja los dispositivos (lector de tarjetas de contacto, display, buzzer y leds) a través de sus interfaces nativas.

4 - SBC + lector de tarjetas RFID + lector de tarjetas de contacto + display + buzzer + leds

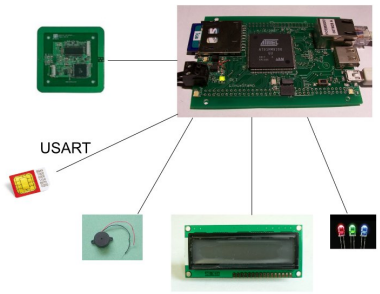

Figura 3.4: Solución considerada 4

Todos los periféricos se conectan a la SBC a través de sus interfaces nativas, esto incluye también el integrado CL RC632 de Philips [12] (ver hoja de datos en el apéndice D). Se debe diseñar la antena para propagar la señal RF hacia las tarjetas.

5 - microcontrolador + lector de tarjetas RFID + lector de tarjetas de contacto + display + buzzer + leds

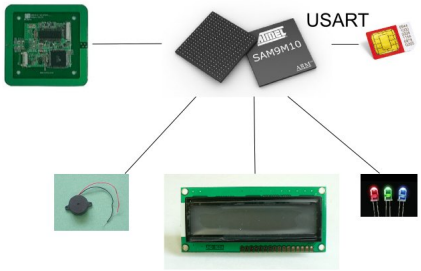

Figura 3.5: Solución considerada 5

Consta de un único PCB, que posee un microcontrolador como sistema central al cual se conectan el resto de los dispositivos. Dicho PCB tiene incorporada la antena para la propagación de RF.

3.2. Arquitectura seleccionada

En una primera instancia se pretendía utilizar únicamente el dispositivo OpenPCD (ver figura 3.1), ya que el mismo cuenta con un microcontrolador de la familia ARM, el AT91SAM7S128, una vez estudiado se llegó a la conclusión de que no permitía la instalación de un sistema operativo Linux, ya que el mismo precisa más de 4MB de RAM para poder hacer algo útil. Otra desventaja encontrada fue que sólo tiene un puerto I2C como forma de conectar periféricos.

Surgió entonces la necesidad de usar una SBC como dispositivo capaz de ejecutar un sistema operativo y las aplicaciones necesarias para que el dispositivo cumpla con los requerimientos exigidos. El dispositivo OpenPCD pasaría entonces a cumplir la función de lector/escritor de tarjetas RFID (ver figuras 3.2 y 3.3), conectado a la SBC a través de su puerto USB, mientras que para el resto de los periféricos se diseñaría un PCB que fuera capaz de ser conectado a la SBC a través de sus interfaces nativas. Esta arquitectura fue descartada por el incremento en el costo del proyecto.

Fue necesario entonces descartar el uso del dispositivo OpenPCD y dar lugar a un diseño propio del lector/escritor de tarjetas RFID (ver figura 3.4), utilizando para esto el integrado CL RC632 de Philips.

La última opción y la más ambiciosa, plantea el diseño completo de un PCB (ver figura 3.5) conteniendo un microcontrolador y memoria capaz de ejecutar un sistema operativo, los lectores de tarjetas, tanto de contacto como RFID, y el resto de los periféricos (display, leds, buzzer). Esta opción fue dejada de lado por entender que excedería los plazos de tiempo del proyecto.

Se pensó entonces en diseñar la arquitectura 4 indicada en la figura 3.4, SBC + lector de tarjetas RFID + lector de tarjeta de contacto + display + buzzer + leds, y dado que se cuenta con un OpenPCD, la opción 3 mostrada en la figura 3.3, SBC + OpenPCD + lector de tarjetas de contacto + display + buzzer + leds, se dejaría como arquitectura alternativa si no se alcanzaran buenos resultados con el lector/escritor de tarjetas RFID.

Luego de estudiar ventajas y desventajas de las arquitecturas planteadas, se eligió la indicada en el ítem 4 en la sección 4.1, que es la que más se adaptó a los requerimientos necesarios:

- SBC + lector de tarjetas RFID + lector de tarjeta de contacto + display + buzzer + leds

En la figura 3.6 se muestra un diagrama de bloques correspondiente a la arquitectura seleccionada:

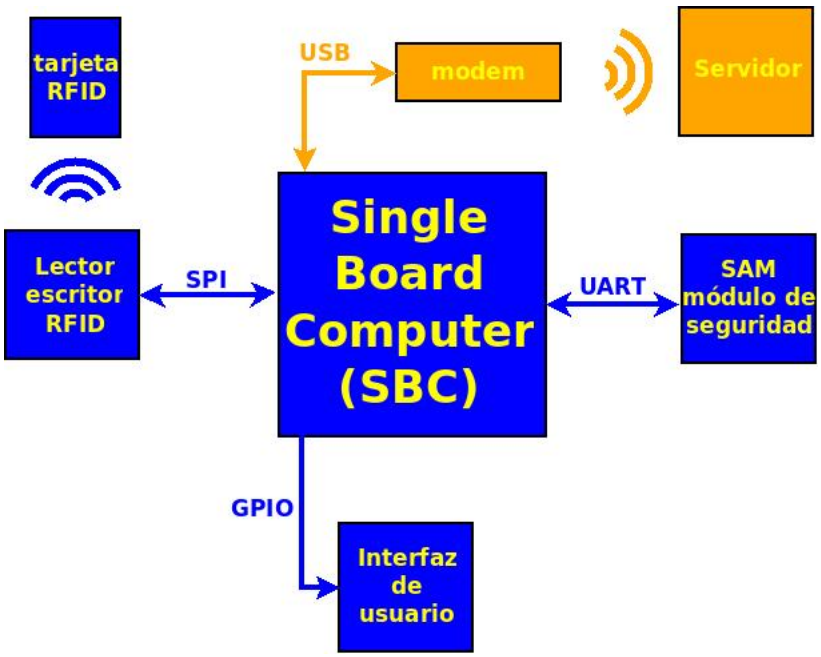

Figura 3.6: Diagrama de bloques de la arquitectura seleccionada

Los bloques oscuros serán implementados, no así los claros.

3.3. Elección de hardware

3.3.1. SBC

En primera instancia se confeccionó una lista con posibles candidatas de SBC disponibles en el mercado internacional, teniendo en cuenta factores como: precio, puertos

de E/S, memoria RAM, memoria Flash, puertos USB, soporte para GNU/Linux, entre otros. Se definieron una serie de requisitos mínimos necesarios para seleccionar de la lista la SBC que más se adecuara a la arquitectura definida. Para la comunicación con el resto de los módulos será necesario: una interfaz UART para el módulo de seguridad (SAM); una interfaz SPI para el módulo lector/escritor RFID (CL RC632 de Philips); 20 GPIO para display, leds, buzzer, otros; 1 USB host para una posible conexión de un modem 3G (intercambio de datos con un servidor). En cuanto a la memoria disponible, tomando como referencia el AFE, debe ser de 32Mb de RAM y 8Mb de flash para un funcionamiento aceptable. Es conveniente, pensando a futuro, que el procesador trabaje a una frecuencia no menor a 200MHz. Dado el presupuesto estimado para el proyecto, el precio no debe superar los 150 dólares en origen. Como requisito adicional se exigió que existiera un foro actualizado y soporte técnico que permitiera evacuar dudas.

Aplicados los requisitos mínimos a la lista previamente confeccionada de SBC candidatas, se optó por dos: GESBC-9G20 [29] y Hawkboard [30]. En cuanto a la primera opción, GESBC-9G20, los fabricantes no respondieron consultas, por tanto se descartó. Se optó entonces por la segunda opción, Hawkboard, puesto que respondieron a las consultas en tiempos razonables y se logró evacuar dudas desde el foro.

Luego de comprar dos Hawkboard, ambas resultaron defectuosas a nivel de hardware, después de varios meses de pruebas sin resultados y sin respuestas concretas por parte del proveedor y fabricante y con la intención de cumplir con los plazos del proyecto, se optó por utilizar una SBC (Beagleboard [31]) que se consiguió en préstamo por medio del INCO. Ésta SBC cumplió con los requisitos mínimos, aunque en ese momento tenía un costo del doble de la Hawkboard, teniéndose que diseñar un módulo hardware adicional. Finalmente, la SBC seleccionada para trabajar fue la Beagleboard.

Las características generales de la BeagleBoard son: cuenta con un procesador OMAP 3530 de 720MHz con arquitectura ARM. Posee memoria NAND-flash de 256Mb y memoria ROM de igual tamaño. Tiene una ranura adicional para extender la memoria a través de una memoria SD. Entre otras cosas cuenta con un puerto USB OTG, un puerto USB host, un bloque de expansión de 28 pines (con señales a 1,8 Volts), puerto JTAG, conector RS232, etc.

3.3.2. VLT - Conversor de Voltajes

Este módulo no fue tenido en cuenta en la primera etapa del diseño de la arquitectura hardware, sino que surge como necesidad debido al cambio de SBC. Como consecuencia de lo anterior se vio la ventaja de incorporar una placa que permite la conexión entre la SBC y el resto del hardware, el cual puede permanecer inalterado por más que no ocurra lo mismo con la SBC, ya que ésta puede cambiar de versión o dejar de fabricarse en un breve lapso de tiempo. El único elemento a cambiar sería entonces la placa VLT, que es más simple y barata de fabricar que las restantes partes. La placa de circuito impreso VLT consta básicamente de dos conectores, uno de ellos permite la conexión con la Beagleboard y el otro la conexión con el restante hardware el cual se encuentra intergrado en un PCB llamdo SCUI. Ambos conectores no se encuentran directamente interconectados entre sí a través de pistas, pues para el caso particular de Beagleboard fue necesario incorporar conversores de tensión que permitieran el traslado del nivel de tensión desde 1,8 Volts que usa esta SBC, a las tensiones con las que operan los periféricos, ya sea 3,3 o 5 Volts. El último elemento, no menos importante, es un regulador de tensión LDO que permite generar 3,3 Volts a partir de la fuente de tensión de 5 Volts de la propia Beagleboard.

3.3.3. SCUI - Lector de tarjetas de contacto e Interfaz de Usuario

El módulo SCUI puede dividirse en dos partes, una de ellas es un lector de tarjetas de contacto basadas en la norma ISO7816, y la otra es una simple interfaz para el usuario. El lector de tarjetas de contacto (smart cards), está compuesto por un conversor full duplex a half duplex el cual se encuentra conectado a uno de los puertos UART de la SBC a través del módulo VLT, que se describió en el punto anterior. Este conversor permite la transmisión de datos directamente entre la tarjeta y la SBC, sin necesidad de intercalar un ASIC para el manejo de tarjetas del tipo ISO7816. Cuenta también con un oscilador para alimentar la entrada de reloj de las tarjetas. La entrada de control (OE) del oscilador operada desde la SBC permite poner la salida de reloj en tercer estado, cosa muy útil a la hora de cumplir con la secuencia de inicialización de las tarjetas descritas en el estándar. El lector permite operar con tarjetas clase A (alimentadas a 5 Volts) y clase B (alimentadas a 3,3 Volts) haciendo uso de un jumper que permite intercambiar la tensión

de alimentación suministrada a la tarjeta. Se cuenta con un zócalo para insertar la tarjeta de contacto. Por otra parte, la intefaz de usuario está compuesta por tres leds (verde, amarillo y rojo), buzzer y un display LCD16x2 donde son desplegados los mensajes que indican al usuario la operación que se efectúa sobre su tarjeta Mifare. El último elemento a describir aquí es un conector receptáculo 5x2 (100mils) en el que se conecta el módulo lector/escritor RFID que opera con las tarjetas RFID Mifare.

3.3.4. Lector/Escritor RFID

Este módulo es el encargado de la comunicación con las tarjetas RFID que cumplen con la norma ISO14443. Consta básicamente de 4 secciones entre las que se encuentran: el integrado CL RC632; el filtro EMC, el circuito de adaptación de impedancia (matching); y el inductor de la antena. El ASIC CL RC632 permite, por un lado la comunicación digital con un microprocesador a través de su puerto de datos y por el otro lado la transmisión de datos hacia la antena que emitirá la señal RF para la comunicación con las tarjetas ISO14443. Lo que se llama propiamente antena RF está conformada por el circuito de adaptación de impedancia (matching) y por el inductor, que propaga el campo magnético para lograr el acoplamiento necesario entre lector y tarjeta, de aquí la sigla PCD (Proximity Coupling Device).

Los principios básicos de funcionamiento de la antena se detallan en el apéndice B.

3.4. Funcionamiento de módulos

3.4.1. SBC

La SBC está formada por un SOC y memoria suficiente para ejecutar un sistema operativo linux orientado a desarrollar sistemas embebidos. Sobre el sistema operativo se instalan los módulos y bibliotecas necesarias para hacer uso del hardware que contiene la SBC. En la aplicación se utilizará uno de sus puertos SPI para la comunicación con el lector/escritor de tarjetas RF, un puerto UART para la comunicación de datos con el lector de tarjetas de contacto y varias salidas GPIO para el control de la interfaz de usuario.

3.4.2. VLT - Conversor de Voltajes

El corazón de esta placa son los integrados TXB0108 [13] (ver hoja de datos en el apéndice D) que permiten la interconexión de dispositivos que operan en distintos niveles de tensión. Básicamente el integrado está constituído por dos puertos, puerto A y puerto B cada uno de 8 bits. El puerto A opera con la tensión de 1,8 Volts que permite ser conectado a la Beagleboard, el puerto B opera con la tensión de 3,3 Volts cuando se encuentra conectado al CL RC632, y de 5 Volts para los restantes periféricos. Cada I/O de un puerto es sensible a los flancos de subida o bajada, trasladando estos cambios a la I/O correspondiente del puerto opuesto. Este integrado posee también una entrada de control para poner los puertos en estado de alta impedancia. Una ventaja es que no poseen entrada de control de dirección de flujo de datos, de modo que se ahorran pines de control que no se tienen disponibles en la Beagleboard. En la figura 3.7 se puede observar como están constituídas cada una de las entradas/salidas del integrado. Otra pieza que compone esta placa es el regulador de tensión LDO implementado a partir del integrado LM1117 [15] (ver hoja de datos en el apéndice D), éste se utiliza para convertir la entrada de tensión de 5 Volts en una salida de tensión de 3,3 Volts y así poder alimentar el periférico correspondiente.

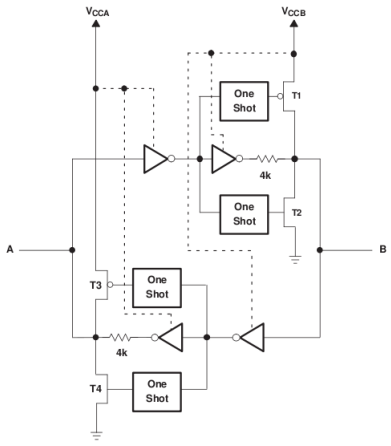

Figura 3.7: Arquitectura de una celda I/O del TXB0108

3.4.3. SCUI - Lector de tarjetas de contacto e Interfaz de Usuario

Lector de tarjetas de contacto ISO7816

Es un lector muy simple de implementar, su construcción se basa en un conversor full a half duplex construido a partir de un circuito transistorizado trabajando en zona de corte y saturación. Los transistores empleados son el NPN 2N3904 (ver hoja de datos en el apéndice D) y el PNP 2N3906 (ver hoja de datos en el apéndice D) los cuales fueron seleccionados en base a su rápida característica de conmutación que es del orden de algunas decenas de nanosegundos. Dada la característica del circuito, es posible recibir el eco de la transmisión de datos generados por la SBC. Un elemento fundamental que compone el circuito del lector es el oscilador de frecuencia 3,579545 Mhz, este valor no es antojadizo sino que permite generar la base de tiempo adecuada para la transmisión de datos entre la tarjeta y la SBC. Otras frecuencias de reloj fueron empleadas, como ser 4 Mhz y 5 Mhz, con resultados inciertos en la recepción de los datos, aún cuando sería posible usar estos valores según la referencia [18] para los parámetros obtenidos desde el ATR de la tarjeta. El circuito cuenta también con protección de descarga ESDA6V1W5 (ver hoja de datos en el apéndice D) para los contactos de la smart card.

Interfaz de usuario

El elemento a destacar es un display LCD16x2 que basa su funcionamiento en el controlador Hitachi HD44780 [14] (ver hoja de datos en el apéndice D). La transferencia de datos hacia el display se hace a través de un puerto con 4/8 bits de datos y 3 bits de control. Debido a que no se cuenta con la cantidad de pines disponibles en la Beagleboard para operar en el modo de 8 bits, se empleó en su lugar el modo 4 bits del display. El bit de control RS indica si el byte a enviar por el puerto de datos es una palabra de control o un caracter ASCII a ser almacenado en la memoria interna del display. El bit R/W por su parte indica si se efectuará una lectura o una escritura de la memoria interna del display. Por último en el bit E se indica mediante flanco de bajada que se ejecute la operación indicada con los anteriores dos bits de control, previo a este flanco las señales en el puerto de datos deben permanecer fijas. El display cuenta también con una entrada para calibrar el contraste del LCD, la calibración se realiza a partir de un divisor resistivo implementado con resistencias y un preset. El backlight del display es accionado desde uno de los pines de la SBC a partir de un circuito transistorizado que

opera en zona de corte/saturación. Los restantes elementos que componen la interfaz de usuario son leds y buzzer que son accionados directamente desde los pines del puerto de expansión de la SBC.

3.4.4. Lector/Escritor RFID

En el corazón del lector/escritor de tarjetas RFID, se encuentra el chip CL RC632 que forma parte de una familia de integrados empleados para la comunicación con tarjetas sin contacto, pertenecientes a la norma ISO14443 las cuales operan a la frecuencia 13,56 Mhz. El CL RC632 soporta todas las capas del esquema de comunicación que se establecen en la mencionada norma, incluyendo el algoritmo de seguridad (CRYPTO1) para autenticar las tarjetas Mifare Classic. En lo que sigue se describen algunas de las características principales del integrado.

Interfaz

Los comandos, bits de configuración y las banderas se acceden a través de la interfaz con un microprocesador. El puerto elegido para la comunicación desde la SBC es el SPI, aunque es posible la comunicación a través de su puerto paralelo.

Registros

La configuración del chip se lleva a cabo a partir de un mapa de registros de control que se encuentra dividido en 8 páginas con 8 registros cada una. La manera de alcanzar estos registros es mediante el intercambio de página, mecanismo que puede ser deshabilitado mediante escritura de un “1” en el bit 7 del registro 0 en la página 0, logrando direccionamiento plano. La función de cada uno de sus registros puede ser observada en la hoja de datos del integrado [12].

Memoria EEPROM

La memoria está dividida en 32 bloques con 16 bytes cada uno. El contenido de memoria EEPROM en los bloques 1 y 2 (dirección 10hex a 2Fhex) se utilizan para configurar los registros del CL RC632 durante la fase de inicialización, de forma automática. La configuración por defecto soporta la comunicación Mifare ISO 14443 A, aunque los usuarios pueden especificar la inicialización para I-Code1, ISO 15693 o ISO 14443 B, mediante los bloques de memoria 3 al 7. Se reservan 384 bytes para almacenar

las claves CRYPTO1 que son usadas para la autenticación con las tarjetas. El formato de una de estas claves puede verse en [12] y tiene una longitud de 12 bytes, por tanto es posible almacenar en memoria las 32 claves que posee una tarjeta.

Buffer FIFO

El integrado contiente un buffer FIFO de 64 bytes para flujo de datos con un microprocesador. La entrada y salida del buffer de datos está conectado con el registro FIFOData. Escribir en este registro almacena un byte en el buffer e incrementa el puntero de escritura del buffer. La lectura de este registro muestra el contenido del buffer e incrementa el puntero de lectura. La distancia entre el puntero de escritura y lectura se puede obtener mediante la lectura del registro FIFOLength, indicando así la cantidad de bytes que se llevan almacenados. Es posible observar y controlar el estado del buffer mediante varios registros, para evitar que se produzcan errores de comuncicación con el microprocesador.

Interrupciones

El CL RC632 indica ciertos eventos estableciendo el bit IRQ en el registro PrimaryStatus, y además, por la activación del pin IRQ. La señal en el pin IRQ se puede utilizar para interrumpir un micrprocesador. Las posibles fuentes de interrupción son:

- Timer, a través de su bandera TimerIRq
- Transmisor, coprocesador CRC y memoria E2PROM, a través de su bandera TxIRq
- Receptor, a través de su bandera RxIRq
- Registro de comando, a través de su bandera IdleIRq
- Buffer FIFO, a través de sus banderas HiAlertIRq y LoAlertIRq

El CL RC632 informa al microprocesador sobre el origen de una interrupción mediante el establecimiento del bit adecuado en el registro InterruptRq. La relevancia de cada bit de petición de interrupción como fuente de una interrupción puede ser enmascarada con el bit de habilitación de interrupciones en el registro InterruptEn. Si alguna bandera de solicitud de interrupción se establece en 1 (una solicitud de interrupción está pendiente) y la correspondiente bandera de habilitación de interrupción está en ”1”,

la bandera de estado IRq en el registro PrimaryStatus se establece en 1. Por otra parte diferentes fuentes de interrupción pueden estar activas al mismo tiempo. Por lo tanto, se hace un OR con todos los bits de solicitud de interrupción, el resultado se envía a la bandera IRq y se conecta al pin IRQ. Los bits de petición de interrupciones están seteados de forma automática por las máquinas de estado internas del CL RC632. Adicionalmente, el microprocesador tiene acceso para setearlos o borrarlos. Una implementación especial de los registros InterruptRQ y InterruptEn permiten el cambio de un único bit de estado sin tocar el resto.

Configuración del Pin IRQ: El nivel lógico de la bandera de estado IRq es visible por el pin IRQ. Además, la señal en el pin puede ser controlada por los siguientes bits del registro IRQPinConfig

- IRQInv: Si este bit es 0, la señal en el pin IRQ es igual al nivel lógico del bit IRq. Si es 1, la señal en el pin IRQ está invertida con respecto al bit IRq.
- IRQPushPull: Si este bit es 1, el pin IRQ tiene características de una salida estándar CMOS, de otra manera la salida es open drain y un resistor externo es necesario para alcanzar un nivel alto en este pin.

Para poder hacer uso de lo descrito anteriormente se previó y reservó una entrada en el conector de expansión de la Beagleboard (ver figura 4.1), sin embargo el software empleado no hace uso del mecanismo de interrupciones sino que opera mediante polling.

Transmisor, pines Tx1 y Tx2

La señal en Tx1 y Tx2 es la portadora, centrada en 13,56 Mhz, modulada ASK 100 % con los datos a transmitir. Estos pines son conectados directamente a la antena para propagar la señal RF hacia las tarjetas RFID. La distancia de operación alcanzada es de hasta 10cm de longitud, dependiendo de la geometría de la antena, así como también adaptación de impedancia lograda, entre otros (ver [4] y [5] incluídas en el apéndice D). Algunos registros del integrado permiten la configuración del transmisor, posibilitando entre otras cosas apagar la señal portadora en caso de ser necesario.

Conjunto de comandos

El CL RC632 opera como una máquina de estado capaz de interpretar y ejecutar un conjunto de comandos pre establecidos. La ejecución de uno de ellos es posible

escribiendo su código correspondiente en el "Registro de Comandos", si fuera necesario el pasaje de parámetros, éstos se colocarán en el buffer FIFO mencionado antes. Una lista detallada de comandos junto con los parámetros necesarios es mostrada en la hoja de datos, entre ellos se pueden destacar los siguientes: Authent, Transceive , LoadKey.

Antena RF

En lo que sigue se describen algunas de las partes que integran la antena RF que se conecta directamente a los pines Tx1 y Tx2 del integrado descrito antes.

Filtro EMC

La frecuencia de la portadora de la señal transmitida se centra en 13,56 Mhz, sin embargo se generan también armónicos de mayor frecuencia. Para cumplir con la regulación internacional EMC es que se agrega este filtro pasa bajos, cuya frecuencia de corte debe ubicarse en 14,4 Mhz, o sea 13,56 Mhz más 847,5 Khz para permitir el ancho de banda necesario que logre el baud rate requerido en la transmisión de los bits. En síntesis el filtro ayuda a mejorar la relación señal a ruido para la señal recibida y decrementa el sobretiro en los pulsos transmitidos mejorando la calidad de la señal transmitida. Los valores propuestos para los componentes de este filtro se encuentran en las notas de aplicación [4].

Matching

Por su parte el circuito de adaptación de impedancia permite que la antena resuene a la frecuencia deseada, en este caso 13,56 Mhz. Los valores de los elementos que conforman este circuito deben ser estimados y sintonizados a partir del diseño del inductor de la antena. El factor de calidad total de la antena debe ser tenido en cuenta para cumplir con los requerimientos establecidos en la norma ISO14443. El mecanismo para el cálculo de los elementos que foman este circuito se detallan en las notas de aplicación [4].

Inductor

El inductor de la antena es quien propaga el campo magnético para la transmisión de datos hacia las tarjetas. El diseño de la antena comienza a partir de este elemento. El cálculo detallado del valor del inductor se encuentra en las notas de aplicación [1], aunque su costo y tiempo en la práctica son considerables; una estimación del valor

de la inductancia puede verse en el apéndice B, en el que se deben tener en cuenta los siguientes elementos: geometría de la antena, ancho y espesor del conductor del PCB, longitud de una espira, número de vueltas, etc.

Receptor

El circuito receptor de la antena se encuentra bien detallado en las notas de aplicación [4] y no fue necesario efectuar ningún cambio para lograr buenos resultados en este diseño particular.

Capítulo 4

Documentos y esquemáticos del hardware

4.1. Herramientas de diseño

Las herramientas CAD utilizadas para el diseño del hardware son Kicad [34] y gEDA [35], ambas son de uso libre y open source. Para el caso del lector/escritor RFID se empleó la aplicación gEDA, ya que permite generar pistas con forma de arco de circunferencia, elemento necesario para formar el inductor de la antena. El resto del hardware se diseñó en Kicad, dado que tiene una interfaz gráfica más amigable que gEDA lo que la hace más sencilla de usar. Ambas aplicaciones tienen la ventaja de permitir agregar y editar componentes o módulos diseñados por el mismo usuario y permiten generar archivos en formato gerber necesarios para enviar al fabricante de circuitos impresos.

4.2. Esquemáticos y componentes

En lo que sigue se muestran una serie de cuadros y esquemáticos para lograr un mayor detalle del diseño de las distintas partes que conforman el hardware del prototipo RF^2. En el cuadro 4.1 se observa la distribución de pines asociada con el conector que une la SBC (Beagleboard), con el conversor de niveles de tensión, VLT. Por su parte el cuadro 4.2 muestra el orden de los pines en el conector que interconecta la placa de circuito impreso, VLT, con la placa de circuito impreso, SCUI, que contiene el resto del

hardware.

Función	Nombre	Nº de pin	Nº de pin	Nombre	Función
Fuente 1,8 Volts	1V8	1	2	5V	Fuente 5 Volts
Led Verde	GPIO	3	4	GPIO	RST_SC
Led Rojo	GPIO	5	6	UART_TX	UART_TX
XOE	GPIO	7	8	UART_RX	UART_RX
D7	GPIO	9	10	GPIO	Led Amarillo
SPI_CS	SPI_CS	11	12	GPIO	Buzzer
D5	GPIO	13	14	GPIO	BacK Light
E	GPIO	15	16	GPIO	D6
SPI_SOMI	SPI_SOMI	17	18	GPIO	D4
SPI_SIMO	SPI_SIMO	19	20	GPIO	RW
SPI_CLK	SPI_CLK	21	22	GPIO	RS
IRQ_RF	GPIO	23	24	GPIO	RST_RF
OE	REGEN	25	26	nRESET	N.C.
Referencia 0 Volts	GND	27	28	GND	Referencia 0 Volts

Cuadro 4.1: Conector 14x2 Beagleboard – VLT

Función	Nombre	Nº de pin	Nº de pin	Nombre	Función
Referencia 0 Volts	GND	1	2	N/C	RFU
Fuente 3,3 Volts	3V3	3	4	RST_RF	Reset RC632
RFU	N/C	5	6	SPI_SOMI	SPI para RC632
RFU	N/C	7	8	SPI_CLK	Reloj SPI
RFU	N/C	9	10	SPI_SIMO	SPI para RC632
RFU	N/C	11	12	SPI_CS	Chip Select RC632
Control LCD16x2	RS	13	14	IRQ_RF	Interrup RC632
Control LCD16x2	RW	15	16	BLK	BacK Light
Control LCD16x2	E	17	18	XOE	Reset oscilador
Dato LCD16x2	D4	19	20	BUZZ	Buzzer

Dato LCD16x2	D5	21	22	N/C	RFU
Dato LCD16x2	D6	23	24	N/C	RFU
Dato LCD16x2	D7	25	26	N/C	RFU
RFU	N/C	27	28	UART_RX	Smart card
Fuente 5 Volts	5V	29	30	UART_TX	Smart card
Referencia 0 Volts	GND	31	32	RST_SC	Reset Smart card
Led Rojo	LED_R	33	34	N/C	RFU
Led Amarillo	LED_A	35	36	N/C	RFU
Led Verde	LED_V	37	38	N/C	RFU
RFU	N/C	39	40	N/C	RFU

Cuadro 4.2: Conector 20x2 VLT - SCUI

4.2.1. SBC

Componente	**Descripción**
SBC	Beagleboard RevC4
Memoria SD	4GB SDHC Class 6 SD Card
Cable serial DB9 nulo	DB9F Null Modem (RS-232) (6-ft)
Cable conversor usb–serial	USB to DB9M RS-232 (PL-2302)
Cable USB	USB Mini-A to USB A Female, OTG
Cable USB	USB Mini-B Male to USB A Male
Fuente	5VDC/2,5A

Cuadro 4.3: SBC y lista de accesorios

4.2.2. VLT - Conversor de Voltajes

Componente	Descripción	Footprint	Valor
C1	Polarized Capacitor (Tantal)	6032[2312]	10uF, 25V
C2	Polarized Capacitor (Tantal)	6032[2312]	100uF, 6V3
U4	Regulador LM1117-3.3	SOT-223	3.3V, 800mA
U1, U2, U3	Voltage Level Translator	TSSOP20	-
P1	RECEPTACLE, 28WAY, 2ROW	SMD Pitch 2,54	28 pines
P2	RECEPTACLE, 40WAY, 2ROW	SMD Pitch 2,54	40 pines
P1b	HEADER, 28WAY, 2ROW	T H Pitch 2,54	28 pines
P2b	HEADER, 40WAY, 2ROW	T H Pitch 2,54	40 pines

Cuadro 4.4: Lista de componentes de la placa de circuito impreso VLT

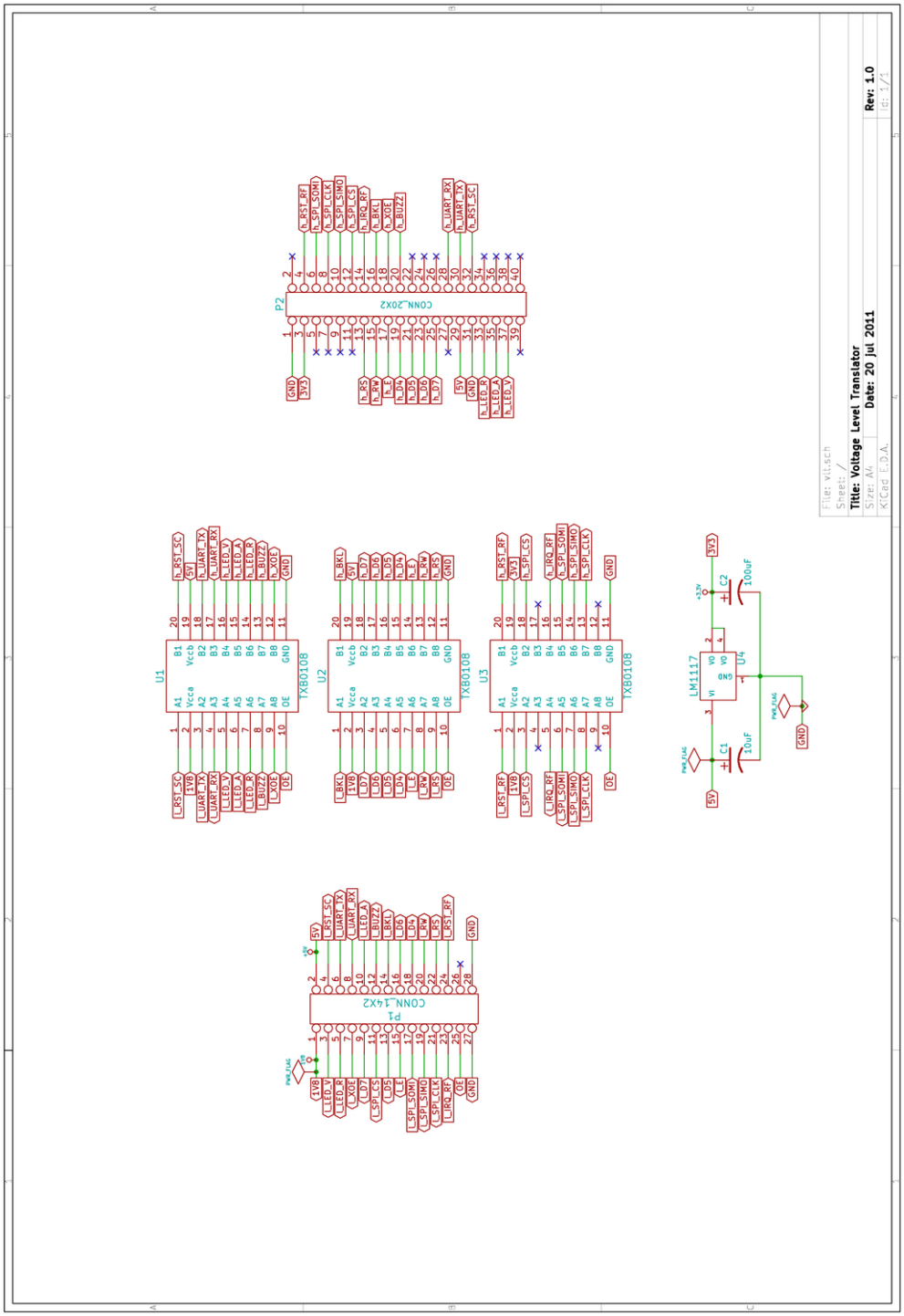

Figura 4.1: Esquemático de la placa VLT - Voltage Level Translator

4.2.3. SCUI - Lector de tarjetas de contacto e Interfaz de Usuario

Componente	Descripción	Footprint	Valor
R9	Resistor 100W 1/4W 1 %	3216[1206]	100W 1/4W 1 %
R10, R13	Resistor 100KW 1/4W 5 %	3216[1206]	100KW 1/4W 5 %
R11, R12, R14	Resistor 10KW 1/4W 5 %	3216[1206]	10KW 1/4W 5 %
Q2	TRANSISTOR, NPN, 300MHZ	SOT23	MMBT3904
Q3	TRANSISTOR, PNP, 250MHZ	SOT23	MMBT3906
J2	SIM socket (6 contacts)	SMD	-
JP3, JP4	HEADER, 1ROW, 3WAY	T H Pitch 2,54	3 pines
X1	Oscillator 3.579545MHz	SMD	3.579545 Mhz
ESD1	Anti ESD	SOT323	6V / 150W

Cuadro 4.5: Lista de componentes del lector de tarjetas de contacto, SC

Componente	Descripción	Footprint	Valor
R1	Resistor 4K7 1/10W 1 %	1608[0603]	4,7KW 1/10W 1 %
R2, R8	Resistor 3R3 1/10W 1 %	1608[0603]	3,3W 1/10W 1 %
R3, R4, R5	Resistor 680R 1/10W 1 %	1609[0603]	680W 1/10W 1 %
R6, R7	Resistor 10K 1/10W 1 %	1608[0603]	10KW 1/10W 1 %
RV1	Preset 15K 1/10W 25 %	SMD	15KW 1/10W 25 %
Q1	TRANSISTOR, NPN, 300MHZ	SOT23	MMBT3904
S1	LCD MODULE 16X2 CHARACTER	Pitch 2,54	-
CONN1	HEADER FEMALE 16POS.1”TIN	Through Hole	16 pines
CONN2	HEADER, 1ROW, 16WAY	T H Pitch 2,54	16 pines
LED1	Led green 5mm	Through Hole	1,9V, 2mA
LED2	Led red 5mm	Through Hole	1,9V, 2mA
LED3	Led yellow 5mm	Through Hole	2,4V, 2mA
BUZZ1	Buzzer	Through Hole	3 20Vdc, 3 16mA

Cuadro 4.6: Lista de componentes para la interfaz de usuario, LCD

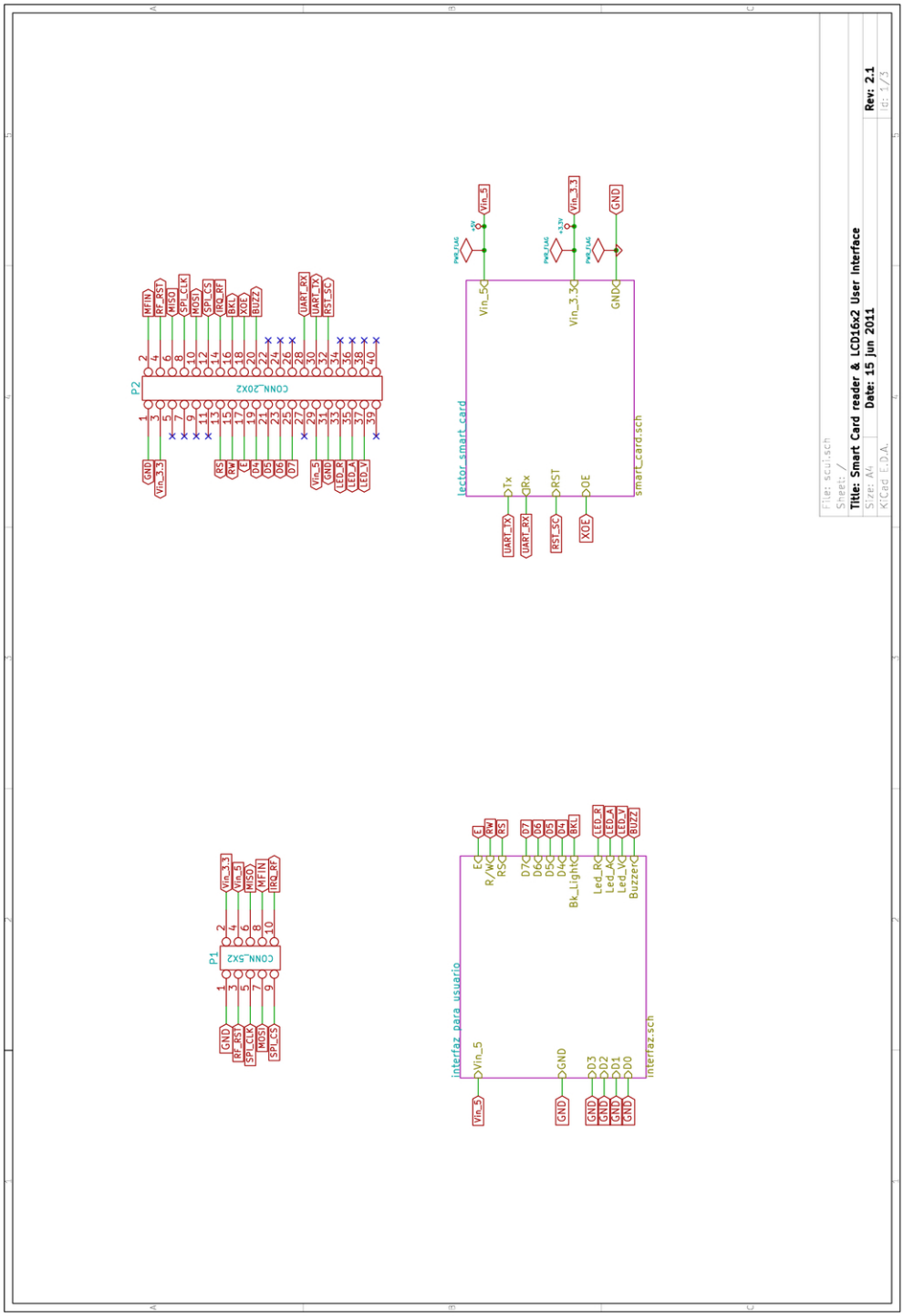

Figura 4.2: Esquemático de la placa SCUI

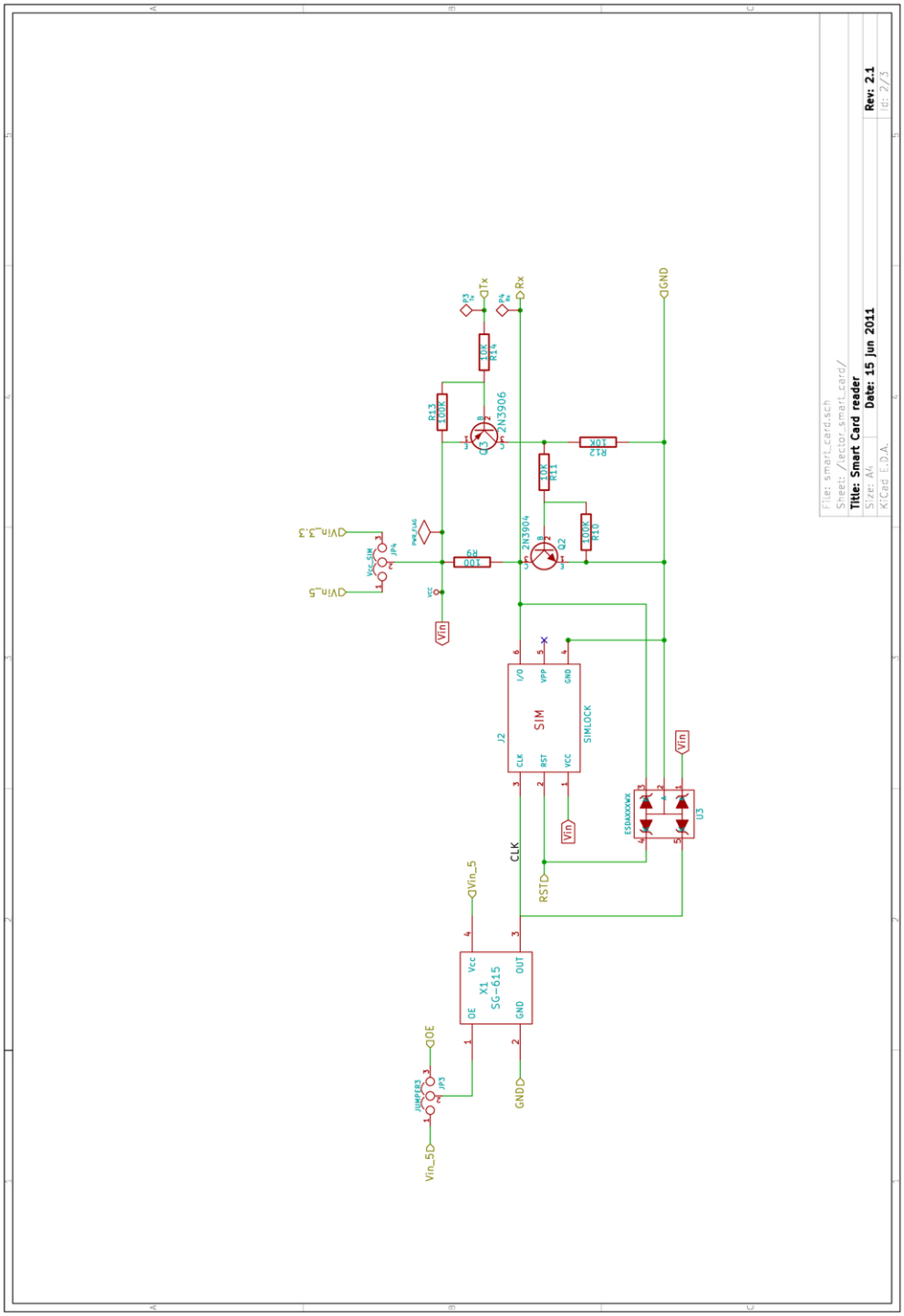

Figura 4.3: Esquemático del lector de tarjetas de contacto, incluido en la placa SCUI

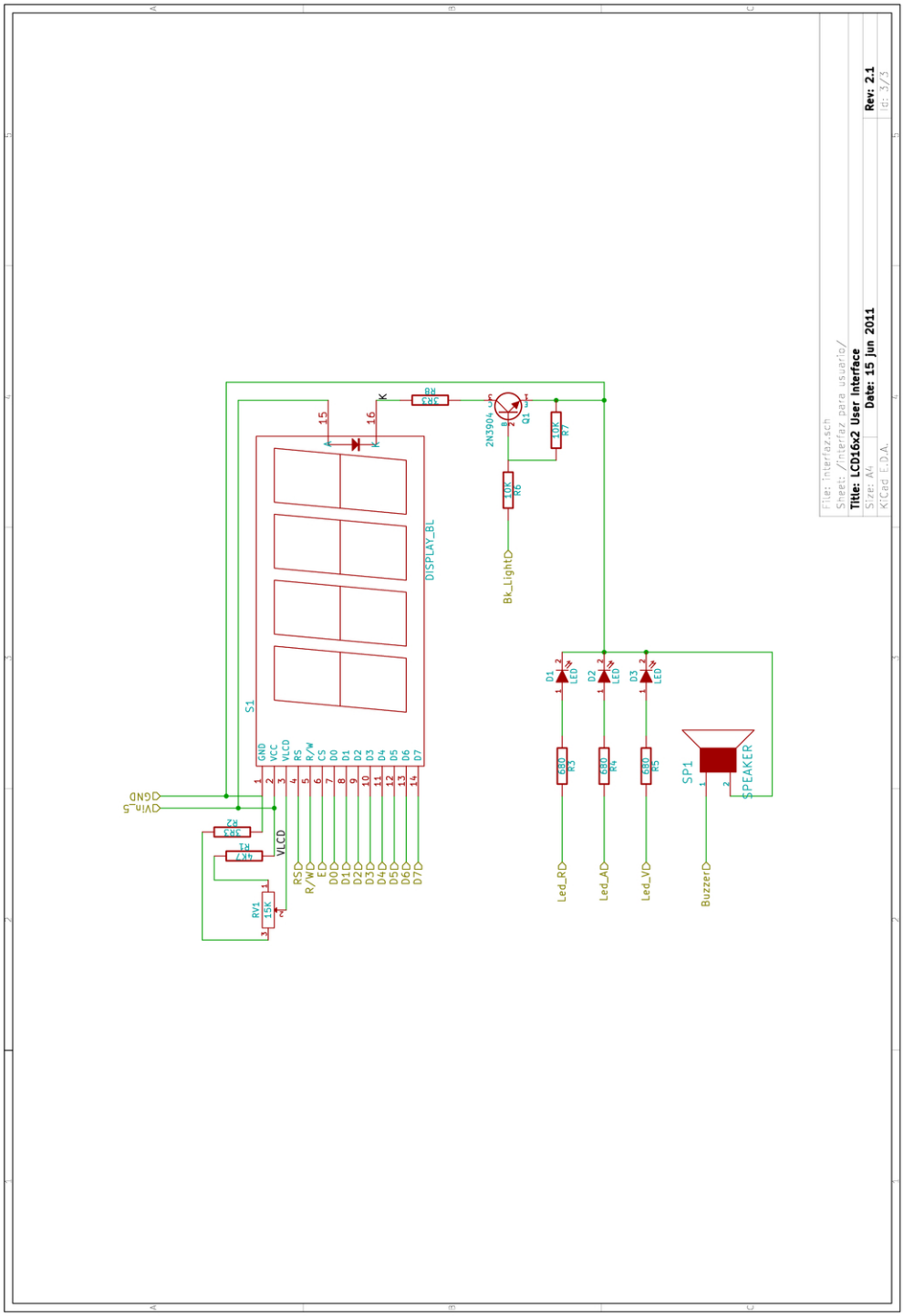

Figura 4.4: Esquemático de la interfaz de usuario, incluido en la placa SCUI

4.2.4. Lector/Escritor RFID

Componente	Descripción	Footprint	Valor
C1, C2	Capacitor	1608[0603]	10pF, Ceramic NPO, 2 %
C3, C4	Capacitor	1609[0603]	100pF, Ceramic NPO, 2 %
C5, C6, C7, C8	Capacitor	1608[0603]	NC
R1, R2	Resistor	1608[0603]	0W, 1/10W, 1 %

Cuadro 4.7: Lista de componentes de la antena RF, Inductor

+ Adaptación de impedancia

Componente	Descripción	Footprint	Valor
C10	Capacitor	1610[0603]	10pF, Ceramic NPO, 2 %
C1, C2	Capacitor	1608[0603]	15pF, Ceramic NPO, 5 %
C12, C13	Capacitor	1608[0603]	56pF, Ceramic NPO, 2 %
C14, C15	Capacitor	1608[0603]	68pF, Ceramic NPO, 1 %
C9	Capacitor	1609[0603]	100pF, Ceramic NPO, 2 %
C16	Capacitor	1608[0603]	1nF, Ceramic NPO, 10 %
C4, C5, C7, C8, C11, C17	Capacitor	1608[0603]	100nF, Ceramic X7R, 10 %
C3, C6, C18	Capacitor	1608[0603]	10uF, Ceramic X5R, 20 %
L1, L2, L3, L6	Inductor	2012[0805]	22nH, 700mA, 5 %
L4, L5	Inductor	3225[1210]	1uH, 400mA, 5 %
R3	Resistor	1608[0603]	50W, 1/10W 1 %
R2	Resistor	1608[0603]	820W, 1/10W 5 %
R1	Resistor	1608[0603]	2,2KW, 1/5W 1 %
U1	Reader ISO14443	SO32	CL RC632
U2	Crystal Oscillator, HC49 US SMD	49USMXL	13.56MHz, 10pF

U3	Operational Amplifier (up to 7.5V)	SOT23-5	OPA354
CONN1, CONN2	U.FL-R Connector	U.FL-R-SMT	-
J1	HEADER, 10WAY, 2ROW	T H Pitch 2,54	10 pines
J1b	RECEPTACLE, 10WAY, 2ROW	SMD Pitch 2,54	10 pines

Cuadro 4.8: Lista de componentes del lector/escritor RFID, sin la antena RF

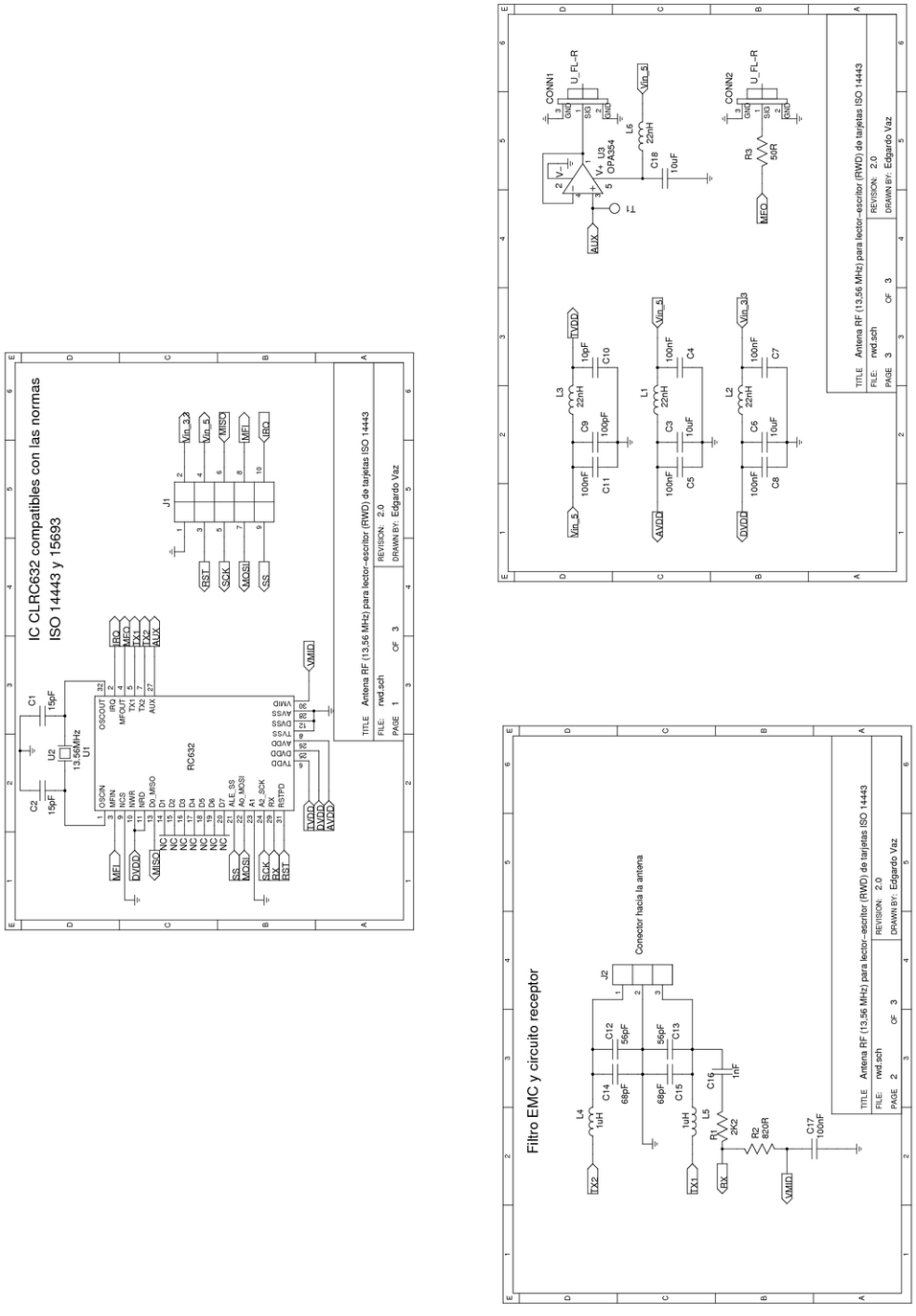

Figura 4.5: Esquemático del módulo digital del lector/escritor RFID

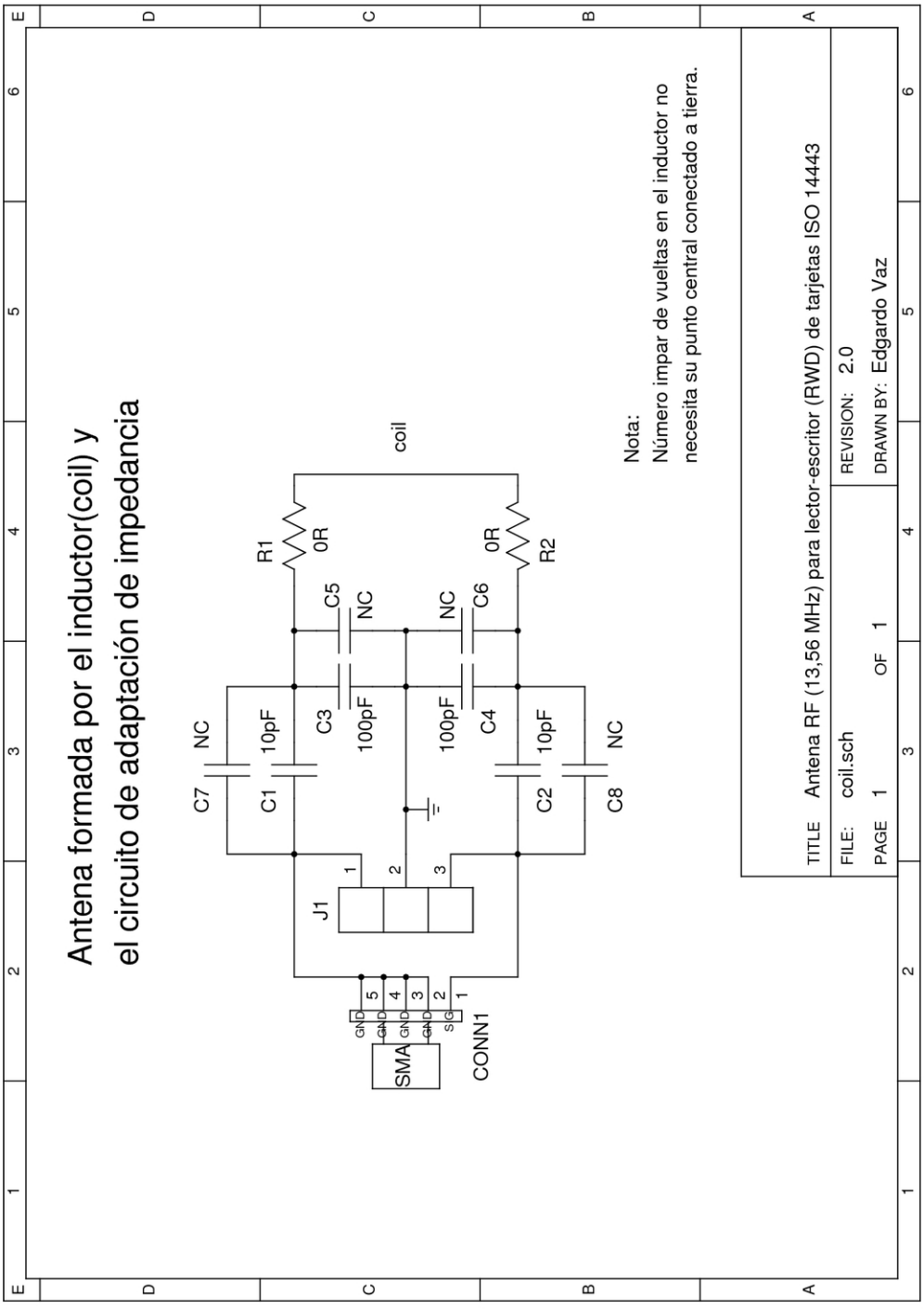

Figura 4.6: Esquemático de la antena RFID, Inductor + Adaptación de impedancia

Capítulo 5

Software

5.1. Introducción

Se debe destacar que todo el desarrollo de software se basó exclusivamente en herramientas de software libre. La distribución Linux elegida para el sistema embebido se llama Angström [37]. Esta distribución es muy usada en aplicaciones que usan una Beagleboard y cuenta con una gran cantidad de bibliotecas implementadas en lenguaje C, que permiten gran escalabilidad a la hora de incorporar nuevos periféricos en la aplicación.

5.2. Arquitectura de Software

5.2.1. Descripción

Un sistema Linux se compone de diferentes partes que interactúan entre sí, formando capas ordenadas con distintos grados de abstracción respecto al hardware. Ésto se puede apreciar en la figura 5.1 donde se muestra a grandes rasgos el sistema.

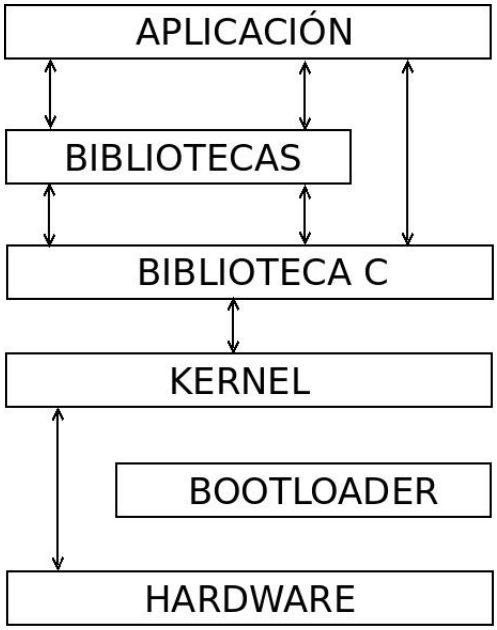

Figura 5.1: Sistema Linux

El bootloader es la parte del sistema más primitiva y su función es la de cargar el kernel en memoria RAM para su ejecución. En general el bootloader es una aplicación que se divide en dos etapas, la primera etapa es fuertemente dependiente del CPU con que cuenta la placa y su función es buscar en particiones activas para luego cargar en memoria RAM la segunda etapa del bootloader. Esta segunda etapa se encarga de descomprimir en memoria RAM la imagen comprimida del kernel para luego ser ejecutado y que éste tome el control del sistema. El kernel se encarga a grandes rasgos de habilitar interrupciones, configurar la memoria y montar un sistema de archivos primitivo que permite a su vez cargar los módulos necesarios para la interfaz con periféricos. Luego se monta el verdadero sistema de archivos (fileSystem). En este nuevo sistema de archivos es donde se instalarán diferentes programas y bibliotecas para la correcta ejecución de aplicaciones. En funcionamiento, toda la comunicación con periféricos se realiza a través del kernel que es la parte más cercana al hardware. Cada vez que se ejecuta una aplicación, ésta hace uso de las bibliotecas para poder comunicarse con el kernel, y éste se encarga de la comunicación con los periféricos. Las bibliotecas pueden ser nativas como es el caso de la biblioteca de lenguaje C, o desarrolladas para que determinada aplicación funcione correctamente.

Una referencia interesante para entender el proceso de arranque es [41].

5.2.2. Sistema Operativo

En el arranque, la Beagleboard tiene la posibilidad de buscar el bootloader y el kernel en NAND, o en dispositivos extraíbles tales como memorias USB o memorias SD. Para el sistema RF^2, se eligió un arranque a través de una memoria SD ya que es fácil de manipular.

En la figura 5.2 se puede ver como queda distribuída la memoria SD con las distintas partes que conforman el sistema operativo.

Figura 5.2: Memoria SD para funcionar en Beagleboard

En la memoria SD se pueden distinguir dos particiones, una en formato FAT32 y otra en formato ext3. La partición en FAT32 es llamada “de arranque” y es donde se encuentra el bootloader (MLO, u-boot.bin) y la imagen comprimida del kernel (uImage). La partición en ext3 es donde se encuentra el sistema de archivos (fileSystem).

El MLO es el equivalente al bootloader de la primera etapa, en general ya viene precargado en la memoria NAND de la Beagleboard. Es posible generarlo o incluso bajar una versión ya compilada desde la web de Angström [37]. Como característica principal tiene la capacidad de buscar el u-boot.bin en dispositivos extraíbles como memorias SD o USB.

El u-boot.bin es equivalente al bootloader de la segunda etapa. En el sistema RF^2 fue necesario generarlo ya que tiene la posibilidad de configurar el bloque de expansión de la Beagleboard.

El uImage es el kernel del sistema. Fue necesario generarlo ya que se debieron modificar sus fuentes para que queden habilitadas las interfaces de comunicación con los

dispositivos periféricos.

El fileSystem (también conocido como RootFileSystem) es el correspondiente a una distribución Linux llamada Angström. Se pueden llegar a precargar distintos programas y bibliotecas dependiendo de la forma en que se genere. Angström es una distribución Linux diseñada específicamente para sistemas embebidos desarrollados para SBCs como la usada para este prototipo. Ésto lo hace más eficiente que otros sistemas operativos para la aplicación desarrollada. La elección de esta distribución se debió a que es de los más recomendados y utilizados en la documentación y foros de Beagleboard [32].

5.3. Herramientas utilizadas en el desarrollo del sistema

5.3.1. Introducción

Para el desarrollo de sistemas existe una gran variedad de herramientas útiles, algunas de software libre y otras privativas. El hecho de tener tantas opciones disponibles, a pesar de ser una ventaja a veces dificulta la elección de las herramientas correctas.

Para la elección de las herramientas se tomó como primer criterio de decisión el hecho que sean libres así como las experiencias de otras personas que ya han transitado caminos comunes, consultando y participando en foros activos.

Cabe señalar que en el PC de desarrollo se utilizó Ubuntu 10.10.

A continuación se detallan las herramientas utilizadas para el desarrollo del sistema.

5.3.2. MLO, u-boot.bin y uImage

No fue necesario generar el MLO debido a su simpleza, puesto que el binario precompilado realiza bien su función.

El u-boot.bin y el uImage fueron generados con la herramienta de desarrollo y compilación OpenEmbedded-Bitbake [33] que es una fusión de dos herramientas: OpenEmbedded, herramienta para construcción y mantenimiento de distribuciones, y Bitbake, herramienta de compilación similar al Make [42] que automatiza la construcción de ejecutables entre otros. Esto es, OpenEmbedded utiliza Bitbake para su objetivo.

OpenEmbedded-Bitbake es una herramienta muy potente y difícil de aprender al principio. Luego de entendido su principio de funcionamiento se hace muy simple su uso, para lo que es necesario tener acceso a una buena conexión a internet debido a que realiza descargas de paquetes en forma habitual. Con esta herramienta también se pueden generar el MLO y el fileSystem, aunque se prefirió utilizar otras herramientas por sobre ésta. Su instalación, configuración, estructura y uso se pueden ver en el apéndice C.3.

5.3.3. FileSystem

Para la generación del fileSystem de Angström, se utilizó la herramienta web Narcissus [39] disponible en la página de Angström. Esta herramienta permite seleccionar entre diferentes dispositivos (entre los cuales está Beagleboard), los programas que se quieran instalar, el formato de la imágen seleccionada, e incluso se puede generar un kit de desarrollo (SDK) para el PC de desarrollo. Debido a la facildad de uso y a los buenos resultados obtenidos, se decidió utilizar esta opción por sobre la del fileSystem generado por la herramienta OpenEmbedded-Bitbake.

5.3.4. Croscompilación

Se llama croscompilar al proceso de compilación de una arquitectura a otra. En el caso del proyecto RF^2, se compilan paquetes en una arquitectura x86 (PC de desarrollo) para ser utilizados en una arquitectura ARM.

Para la croscompilación se utilizó el SDK generado por Narcissus y la herramienta Make para generar los archivos necesarios. La instalación, configuración y uso del SDK se encuentran en el apéndice C.2.

5.3.5. Depuración de código

Para la depuración, se utilizó la herramienta GDB del proyecto GNU. Al momento de compilar, es necesario agregar la opción -g para que la aplicación pueda ser depurada. Esta opción agrega información en el código de la aplicación. La interfaz del GDB es por consola, aunque existen algunos programas que utilizan GDB y además ofrecen una interfaz gráfica (DDD [43]).

Algunos de los comandos útiles y sus usos más comunes pueden encontrarse en el apéndice C.6.

Sin olvidar el hecho que la aplicación RF^2 está diseñada para una arquitectura distinta a la del PC de desarrollo, para el depurado de la aplicación existen dos alternativas.

La primera opción es lo que se podría llamar depuración local, esto es, instalar GDB en la Beagleboard y depurar la aplicación en ésta. Para saber lo que sucede es necesario acceder de forma remota a la Beagleboard desde el PC de desarrollo.

La segunda opción es la depuración remota. La depuración remota consiste en realizar la depuración de la aplicación desde el PC de desarrollo. Para esto, es necesario instalar GDBServer en la Beagleboard y tener instalado el GDB específico de la Beagleboard en el PC de desarrollo. Luego se establece una conexión, serial o ethernet, entre la Beagleboard y el PC de desarrollo. Por más detalles sobre la configuración referirse al apéndice C.7.

La primera opción no es posible para sistemas embebidos chicos en los cuales no se puede instalar GDB, aunque éste no es el caso de la Beagleboard. Se tienen más y mejores herramientas en el PC de desarrollo, por ejemplo programas con interfaz gráfica que ayudan a entender mejor lo que está pasando. Por algunas de estas razones, se prefiere el uso de la depuración remota.

Se utilizaron indistintamente tanto la primera opción como la segunda.

5.3.6. Bibliotecas

- pcsc-scan es una herramienta de uso por línea de comandos, que una vez ejecutado imprime el nombre del lector PC/SC conectado y en caso que exista una tarjeta de contacto insertada imprime también su ATR. Luego escanea el lector constantemente e indica si se retira o inserta una tarjeta. Esta aplicación fue utilizada para testear el lector de tarjetas de contacto.

- librfid-tool es una herramienta de uso por línea de comandos que da acceso de bajo nivel RFID utilizando los lectores soportados por la biblioteca para manejo de lectores/escritores RFID, librfid. Esta herramienta fue utilizada para testear lectores/escritores de tarjetas RFID.

A continuación se detallan la forma de llamar a la aplicación y algunas de las opciones soportadas. Para la instalación y uso referirse al apéndice C.5

5.3.7. Otras herramientas

subversion y Git, para control de versiones.

Geany para edición y compilación de código.

minicom para comunicación serial.

strace es un comando linux que lista las llamadas al sistema, útil para la depuración.

5.4. Desarrollo

5.4.1. MLO

Como se mencionó anteriormente, no fue necesario generar el MLO debido a que la Beagleboard viene con uno pre-instalado, y en caso de que no funcione en forma correcta se puede descargar desde un repositorio de Angström [38]. No fue necesario realizar ningún cambio en el archivo ya que realiza su función correctamente.

5.4.2. Multiplexado de pines

El microprocesador OMAP3530 tiene muchos pines con distintas interfaces entre las que se cuentan puertos UART, SPI, GPIO, etc., pero no todos son accecibles desde la Beagleboard. Para poder acceder a algunos de estos puertos del microprocesador, existe en la placa de la Beagleboard un bloque de expansión de 28 pines.

Por defecto, en el bloque de expansión no se encuentran las señales que se quieren. Esto lleva a que se tenga que modificar el estado inicial de los pines. Existen dos formas de modificar los pines de modo de tener las señales que se precisan. Una de ellas es modificar el bootloader y la otra es modificar el kernel. Esto implica cambios en los archivos fuentes y posterior compilación que genere los nuevos binarios u-boot o uImage.

Para la modificación de las señales disponibles en el bloque de expansión se decidió modificar el u-boot ya que la modificación por u-boot es más intuitiva y por experiencia se sabe que lo que más se actualiza y/o modifica es el kernel.

5.4.3. u-boot

Como se mencionó anteriormente en el u-boot se realizó la configuración de los pines del bloque de expansión de la Beagleboard. Cada pin del bloque de expansión tiene varias funcionalidades asociadas, y la configuración de la funcionalidad depende de un multiplexado modificable a nivel de software. Esto es, dependiendo del "modo de pin" elegido, la función que se obtiene en dicho pin. Para que los cambios hechos en el u-boot tengan el efecto esperado al arrancar el sistema, es necesario que en la configuración del kernel esté la opción CONFIG_OMAP_MUX=no, lo que imposibilita al kernel a realizar este mismo cambio. Esta opción no está activada por defecto para versiones de kernel 2.6.32 y posteriores.

Antes que pueda ser modificado el estado de los pines del bloque de expansión, es necesario obtener los archivos fuente con los cuales se genera el archivo binario u-boot.

Nota: Puede que cuando se realiza la instalación de OpenEmbedded-Bitbake, se genere un directorio relacionado con u-boot en /stuff/build/tmp/work/beagleboard-angstrom-linux-gnueabi/, si esto es así, no es necesario volver a obtener los fuentes.

Se configura el bitbake para poder utilizarlo:

```
$ export BBPATH=/stuff/build:/stuff/openembedded
$ export PATH=/stuff/bitbake/bin:$PATH
```

Se obtienen los fuentes:

```
$ cd /stuff/build
$ bitbake -f -c clean -b ../openembedded/recipes/u-boot/u-boot_git.bb
$ bitbake -f -c compile -b ../openembedded/recipes/u-boot/u-boot_git.bb
```

Los fuentes se encuentran en
/stuff/build/tmp/work/beagleboard-angstrom-linux-gnueabi/u-boot.../git/.

En los fuentes del u-boot dentro de board/ti/beagle/ se encuentra el archivo beagle.h que es donde se establece la configuración de los pines del bloque de expansión de la Beagleboard.

Si se abre este archivo se ven líneas del estilo:

```
MUX_VAL(CP(MCBSP3_DX), (IEN | PTD | DIS | M4)) /*GPIO_140*/\
```

MUX_VAL indica que se va a modificar el valor de multiplexado de lo que está entre paréntesis.

CP(MCBSP3_DX) es el Control_PadConf, esto es el registro del microprocesador asociado con el pin a modificar.

(IEN | PTD | DIS | M4) esta es la configuración del pin en cuestión:

La opción IEN (input enable) hace que el pin sea bidireccional.

La opción PTD y PTU, indica si el pin tiene un pull down o pull up respectivamente.

La opción DIS y EN, indica si se deshabilitan o no las opciones PTD y PTU respectivamente.

La opción M4 es el modo seleccionado para el pin. Para la Beagleboard existen cuatro modos: M1, M2, M3 y M4.

GPIO_140 es el nombre del pin (solo es un comentario).

El Control_PadConf es un registro de 32bits el cual controla el estado de dos pines, esto es, la parte baja del registro controla un pin y la parte alta controla otro.

Analizando el "manual de referencia Beagleboard del usuario" ("Expansion connector signals" – tabla 20), y "manual técnico de referencia OMAP35x" ("SCM functional description" – capítulo 7.4.4); ambos adjuntos en el apéndice D; y agregando las opciones que interesan para los pines, se confeccionó la siguiente tabla:

CONTROL_PADCONF_	**EQUIV. BEAGLE.H**	**PIN EXPANSIÓN**	**SEÑAL EN EL PIN**	**MODO**
MMC2_DAT6[31:16]	MMC2_DAT7	3	GPIO_139	4
UART2_CTS[15:0]	UART2_CTS	4	GPIO_144	4
MMC2_DAT6[15:0]	MMC2_DAT6	5	GPIO_138	4

UART2_TX[15:0]	UART2_TX	6	UART2_TX	0
MMC2_DAT4[31:16]	MMC2_DAT5	7	GPIO_137	4
McBSP3_CLKX[31:16]	McBSP3_FSX	8	UART2_RX	1
MMC2_DAT4[15:0]	MMC2_DAT4	9	GPIO_136	4
UART2_CTS[31:16]	UART2_RTS	10	GPIO_145	4
MMC2_DAT2[31:16]	MMC2_DAT3	11	McSPI3_CS0	1
McBSP1_DX[15:0]	McBSP1_DX	12	GPIO_158	4
MMC2_DAT2[15:0]	MMC2_DAT2	13	GPIO_134	4
McBSP1_CLKX[15:0]	McBSP1_CLKX	14	GPIO_162	4
MMC2_DAT0[31:16]	MMC2_DAT1	15	GPIO_133	4
McBSP_CLKS[31:16]	McBSP1_FSX	16	GPIO_161	4
MMC2_DAT0[15:0]	MMC2_DAT0	17	McSPI3_SOMI	1
McBSP1_DX[31:16]	McBSP1_DR	18	GPIO_159	4
MMC2_CLK[31:16]	MMC2_CMD	19	McSPI3_SIMO	1
McBSP1_CLKR[15:0]	McBSP1_CLKR	20	GPIO_156	4
MMC2_CLK[15:0]	MMC2_CLK	21	McSPI3_CLK	1
McBSP1_CLKR[31:16]	McBSP1_FSR	22	GPIO_157	4
I2C2_SDA[15:0]	I2C2_SDA	23	GPIO_183	4
I2C1_SDA[31:16]	I2C2_SCL	24	GPIO_168	4

Cuadro 5.1: Modo de pines en bloque de expansión

Al modificar el archivo beagle.h hay que tener mucho cuidado ya que al sustituir los valores no se deben repetir pines ni registros, no deben haber incoherencias, un registro por cada pin y un pin por cada registro. Dentro del archivo hay un macro definido MUX_BEAGLE_C(), donde se deben realizar las modificaciones ya que este macro está asociado con el modelo de Beagleboard utilizado (C4).
En una primera instancia se sustituyeron los valores del cuadro 5.1 buscando los equivalentes del PadConf en el archivo beagle.h.

```
\#define MUX_BEAGLE_C() \
MUX_VAL(CP(MCBSP3_DX),   (IEN  | PTD | DIS | M4))/*GPIO_140*/\
```

```
MUX_VAL(CP(MCBSP3_DR),    (IEN  | PTD | DIS | M4))/*GPIO_142*/\
MUX_VAL(CP(MCBSP3_CLKX), (IEN  | PTD | DIS | M4))/*GPIO_141*/\
MUX_VAL(CP(MCBSP3_FSX),   (IEN  | PTD | DIS | M1))/*UART2_RX*/\
MUX_VAL(CP(UART2_TX),     (IDIS | PTD | DIS | M0))/*UART2_TX*/\
MUX_VAL(CP(MMC2_DAT7),    (IEN  | PTD | EN  | M4))/*GPIO_139*/\
MUX_VAL(CP(UART2_CTS),    (IEN  | PTD | DIS | M4))/*GPIO_144*/\
MUX_VAL(CP(MMC2_DAT6),    (IEN  | PTD | EN  | M4))/*GPIO_138*/\
MUX_VAL(CP(MMC2_DAT5),    (IEN  | PTD | EN  | M4))/*GPIO_137*/\
MUX_VAL(CP(MMC2_DAT4),    (IEN  | PTD | EN  | M4))/*GPIO_136*/\
MUX_VAL(CP(UART2_RTS),    (IEN  | PTD | EN  | M4))/*GPIO_145*/\
MUX_VAL(CP(MCBSP1_DX),    (IEN  | PTD | EN  | M4))/*GPIO_158*/\
MUX_VAL(CP(MMC2_DAT2),    (IEN  | PTD | EN  | M4))/*GPIO_134*/\
MUX_VAL(CP(MCBSP1_CLKX), (IEN  | PTD | EN  | M4))/*GPIO_162*/\
MUX_VAL(CP(MMC2_DAT1),    (IEN  | PTU | EN  | M4))/*GPIO_133*/\
MUX_VAL(CP(MCBSP1_FSX),   (IEN  | PTD | EN  | M4))/*GPIO_161*/\
MUX_VAL(CP(MCBSP1_DR),    (IEN  | PTD | EN  | M4))/*GPIO_159*/\
MUX_VAL(CP(MCBSP1_CLKR), (IEN  | PTD | EN  | M4))/*GPIO_156*/\
MUX_VAL(CP(MCBSP1_FSR),   (IEN  | PTD | EN  | M4))/*GPIO_157*/\
MUX_VAL(CP(I2C2_SDA),     (IEN  | PTD | EN  | M4))/*GPIO_183*/\
MUX_VAL(CP(I2C2_SCL),     (IEN  | PTU | EN  | M4))/*GPIO_168*/\
MUX_VAL(CP(MMC2_DAT3),    (IEN  | PTD | EN  | M1))/*SPI3_CS0*/\
MUX_VAL(CP(MMC2_DAT0),    (IEN  | PTU | EN  | M1))/*SPI3_SOMI*/\
MUX_VAL(CP(MMC2_CMD),     (IEN  | PTU | DIS | M1))/*SPI3_SIMO*/\
MUX_VAL(CP(MMC2_CLK),     (IEN  | PTU | DIS | M1))/*SPI3_CLK*/
```

Luego es necesario compilar para obtener el u-boot.bin.

$ cd /stuff/build

$ bitbake -f -c compile -b ../openembedded/recipes/u-boot/u-boot_git.bb

$ bitbake -f -c deploy -b ../openembedded/recipes/u-boot/u-boot_git.bb

Nota: Cada vez que se introduzca un nuevo cambio, no es necesario ejecutar el comando con la opción clean (lo que implica volver a bajar los fuentes), solo basta con volver a compilar.

El archivo generado (u-boot.bin) se encuentra en

/stuff/build/tmp/deploy/glibc/images/beagleboard/ aunque con su nombre seguido de un número identificatorio, el cual debe ser borrado para poder mantener el nombre necesario (u-boot.bin).

Pese a que en la literatura y foros, se plantea lo contrario, no fue posible establecer los atributos “valor” y “dirección” de los pines GPIO mediante la modificación planteada (una posible solución puede verse en el apéndice C.4). Lo que sí cambia efectivamente es el modo del pin, permitiendo obtener las interfaces adecuadas en el bloque de expansión.

5.4.4. uImage

La versión del kernel elegida fue la 2.6.32 ya que en el momento del desarrollo era la versión más estable. Aunque también se hicieron pruebas con las versiones 2.6.29 y 2.6.37.
Durante el arranque del sistema, el kernel carga los módulos y controladores necesarios para el funcionamiento del hardware que forma parte del sistema embebido. También se montan las interfaces para poder interactuar con los distintos dispositivos a ser conectados a la Beagleboard como lo son: SPI, GPIO, UART, etc. Las interfaces SPI y UART se encuentran bajo el directorio /dev en el sistema de archivos, y GPIO bajo el directorio /sys/class/gpio. En algunos casos, en /dev no aparecen algunas de las interfaces configuradas, lo que lleva a modificar los fuentes del kernel para que esto así suceda. Este fue el caso de la interfaz SPI que no quedó mapeada en /dev pese a que había sido configurada en los fuentes del u-boot. También hubo problemas con los atributos “valor” y “dirección” de los GPIO, como se mencionó anteriormente. Adicionalmente hacía falta un módulo para simular una conexión ethernet sobre una interfaz USB para establecer una conexión entre la Beagleboard y un PC como si fuera un enlace de red. Todo esto llevó a que se tuvieran que modificar los archivos fuente del kernel.

A continuación se detallan los pasos a seguir para la modificación de los fuentes del kernel:

Comando necesario para el desarrollo del uImage:

```
$ bitbake virtual/kernel -c [comando]
```

virtual/kernel: refiere a que se quiere generar un kernel.

Entre los comandos:

clean: borra el contenido del directorio work. Borra todos los cambios hechos en la configuración del uImage fuente y parches agregados.

patch: genera los archivos de configuración y el fuente del uImage. Además le aplica los parches.

menuconfig: abre el editor de la configuración del kernel.

compile: compila todo.

deploy: genera los archivos referidos en este caso al uImage (.config, módulos, uImage) y los guarda en el directorio /stuff/build/tmp/deploy/glibc/images/beagleboard/.

Nota: Es necesario que todos estos comandos sean ejecutados en el orden adecuado para que todo funcione correctamente.

Primero se configura bitbake para poder utilizarlo:

```
$ export BBPATH=/stuff/build:/stuff/openembedded
$ export PATH=/stuff/bitbake/bin:$PATH
```

Se comienza el desarrollo:

```
$ cd /stuff/build
$ bitbake virtual/kernel -c clean
$ bitbake virtual/kernel -c patch
$ bitbake virtual/kernel -c menuconfig
```

Luego de ejecutar este comando se abre el editor de la configuración del kernel (ver figura 5.3). Este editor indica qué módulos cargar y cuáles no. En este caso un cambio de configuración es necesario para el buen funcionamiento de la interfaz SPI y de la conexión USB-ethernet con la Beagleboard.

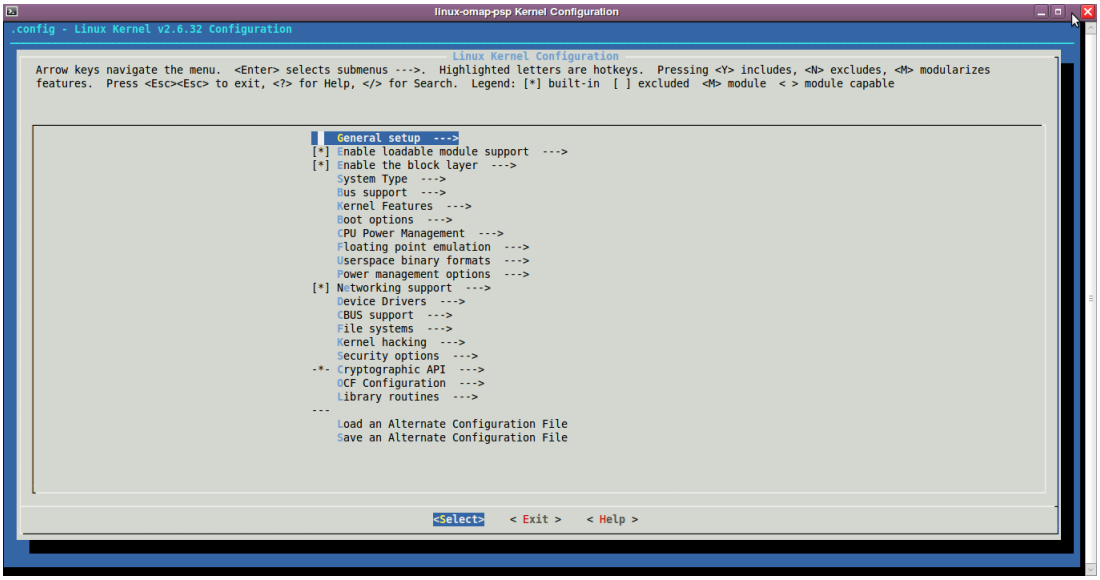

Figura 5.3: Editor de configuración del kernel

Para configurar la interfaz SPI, se debe configurar como sigue:

Device Drivers – SPI Support=y y luego como en la figura 5.4.

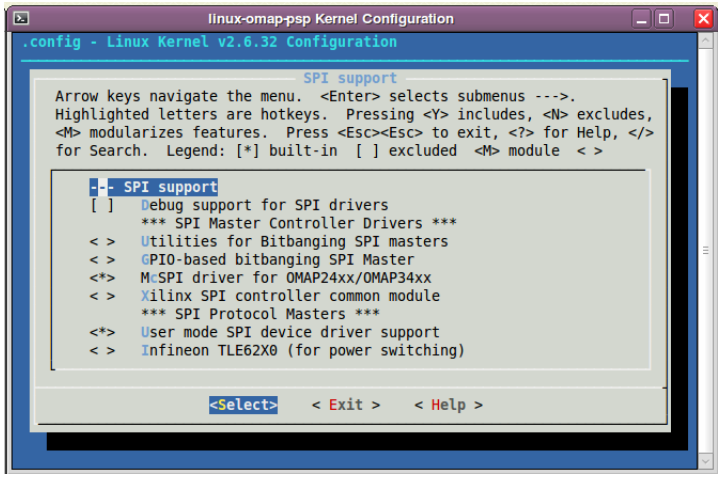

Figura 5.4: Configuración SPI

Para poder establecer la conexión por USB con la Beagleboard, se debe configurar como sigue:

Device Drivers – USB Support=y – USB Gadget Support=y y luego como en la

figura 5.5.

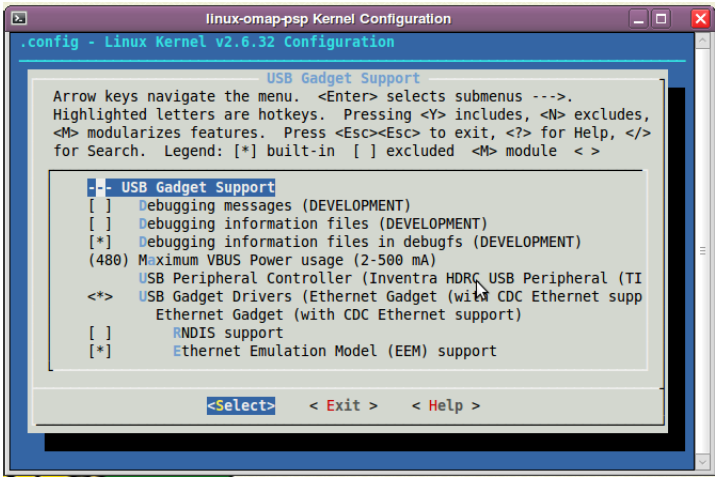

Figura 5.5: Configuración USB Gadget

Luego, es necesario modificar el archivo board_omap3beagle.c que se encuentra en /stuff/build/tmp/work/beagleboard-angstrom-linux-gnueabi/linux-omap-.../git/arch/arm/mach-omap2/. En este archivo está toda la inicialización de las interfaces. Los detalles de los cambios introducidos en este archivo se pueden observar en el apéndice C.4.

Se compila y genera el archivo uImage:

$ bitbake virtual/kernel -c compile

$ bitbake virtual/kernel -c deploy

Dentro de /stuff/build/tmp/deploy/glibc/images/beagleboard/ se encuentra el archivo uImage generado.

Nota: Respecto al nombre del archivo, al igual que con el caso del u-boot.bin el nombre que aparece es un nombre más largo y necesita ser renombrado a uImage para que se pueda ejecutar correctamente.

5.4.5. FileSystem

Como se mencionó anteriormente, el fileSystem se generó a partir de la herramienta web Narcissus.

En el fileSystem es donde se encuentran los paquetes y programas ya instalados. Cuanto más programas se instalen más grande será en tamaño el fileSystem.

Para utilizar la herramienta Narcissus se debe acceder a la web de la herramienta [39].

A continuación se detallan las diferentes características y opciones que se eligieron para crear un fileSystem a medida para la Beagleboard y un SDK para el PC de desarrollo:

Select Machine: Beagleboard.

Image Name: el nombre que se le quiera dar.

Complexity: complejidad, se eligió "advanced" ya que la opción "simple" no brinda libertad de configuración.

Release: versión, aquí hay varias opciones disponibles, se eligió "unstable" ya que es la más estable de las disponibles. La primera opción disponible es la más estable.

Base System: aquí se elige el soporte de drivers y paquetes que se pretenden. "bare bones" es la opción con menos soporte y "extended" es la de mayor soporte. Cuanto más soporte, más pesado se hace el fileSystem. Una opción interesante es la opción "regular", y es la que se eligió.

/dev manager: esto es el manejador de /dev, se recomienda "udev".

Type of Image: formato en el que se quiere descargar el fileSystem. Se eligió "tar.gz" ya que es la opción más versátil.

Software manifest: se genera un archivo en la web con todos los paquetes que se instalaron en detalle.

SDK type: esta opción permite generar un SDK para el PC de desarrollo compatible con el fileSystem generado. Esto es sumamente útil por ejemplo para croscompilar. Aquí se eligió la opción "Full SDK".

User environment section: aquí se indica que tipo de sistema operativo se quiere, básicamente se tienen dos opciones; una es un fileSystem sin interfaz gráfica y las otra con entorno gráfico. Se eligió la opción “console” (sin entorno gráfico) ya que la aplicación no exige entorno gráfico.

Luego se permite seleccionar programas que se desean instalar. Las aplicaciones elegidas son: nano editor (editor de texto) ya que hace las cosas más fáciles que el programa vi, GDB y GDBServer necesarios para la depuración de la aplicación, y toolchain para tener herramientas de compilación nativas en la Beagleboard.

Cuando todo fue seleccionado, se da un click en “build me” (demora un poco). Cuando el proceso termina, se generan dos archivos comprimidos, un archivo con el nombre elegido para la imagen en un formato .tar.gz y el SDK para el PC de desarrollo en formato .tar.bz2.

5.4.6. Beagleboard

Preparación de la memoria SD

Con los archivos MLO, u-boot.bin, uImage y fileSystem generados, se procede a la preparación de la memoria SD para correr por primera vez en la Beagleboard. En el apéndice C.8 se describen los pasos necesarios para el correcto funcionamiento del sistema.

Configuración de los parámetros de arranque

Existe una forma de saber lo que sucede en el arranque del sistema conectando la Beagleboard a un PC a través de un cable con conversor USB-SERIAL y utilizando la interfaz serial de la Beagleboard (/dev/ttyS2) para la comunicación con el PC.

Para saber sobre la configuración en el PC para conexión serial con la Beagleboard ver apéndice C.9.
Primer arranque de la Beagleboard

Para evitar problemas, se recomienda alimentar la Beagleboard con el transformador correspondiente.

Antes de arrancar se abre minicom por consola:

$ minicom

Se coloca la SD en la Beagleboard, se enchufa, y comienza el arranque del sistema.

Para que el sistema arranque correctamente con los archivos generados previamente, se debe mantener presionado el botón "user" de la Beagleboard ya que en caso contrario en el arranque se busca el u-boot.bin primero en NAND y si ya existe una versión preinstalada es ésta la que se va a cargar. Al apretar el botón "user" (no más de tres segundos) se logra un cambio en el orden de arranque lo que hace que se cargue primero el u-boot.bin de la memoria SD.

Luego se debe ver una cuenta regresiva en pantalla. Si se apreta alguna tecla del teclado esta cuenta regresiva se detiene y se accede a la configuración de los parámetros de arranque del sistema (similar a una BIOS de un PC de escritorio). Para ver una lista de todas los parámetros del arranque, se debe ejecutar el comando printenv. En el apéndice C.10 puede verse el listado que devuelve la ejecución de dicho comando.

Entre los parámetros a modificar están los que indican como acceder a la SD (bootargs) y los comandos de arranque (bootcmd) donde se indica en que lugar encontrar el uImage. Para un arranque correcto a través de la SD, los valores de los parámetros anteriores deben ser los siguientes:

```
bootargs= 'console=ttyS2,115200n8 root=/dev/mmcblk0p2
rw rootwait'
bootcmd 'mmc init;fatload mmc 0 80300000 uImage;
bootm 80300000'
```

En general esto ya viene seteado correctamente. Si no es así, van a haber errores al arrancar el sistema. Una solución es setear los valores de estos atributos mediante el comando setenv:

```
OMAP3 beagleboard.org # setenv bootargs 'console=ttyS2,115200n8
root=/dev/mmcblk0p2 rw rootwait'

OMAP3 beagleboard.org # setenv bootcmd 'mmc init;fatload mmc
0 80300000 uImage;bootm 80300000'

OMAP3 beagleboard.org # saveenv
```

Nota: saveenv guarda los valores seteados en NAND.

También se puede modificar el tiempo de la cuenta regresiva (bootdelay) mencionada anteriormente para lograr un mejor tiempo de arranque.

```
OMAP3 beagleboard.org # setenv bootdelay 1

OMAP3 beagleboard.org # saveenv
```

Luego de aplicados los cambios en los parámetros de arranque, se reinicia el sistema.

```
OMAP3 beagleboard.org # reset
```

Un ejemplo de arranque del sistema puede verse en el apéndice C.11.

Copia de MLO y u-boot.bin en NAND

Recordando que se debe dejar apretado el botón "user" de la Beagleboard para lograr que se cargue el u-boot.bin de la memoria SD y no el de la NAND (por defecto), y dado que esto no es conveniente para un sistema que pretende ser autónomo, se decidió copiar los archivos MLO y u-boot.bin a la memoria NAND. Esta operación es muy delicada y se deben seguir los pasos que se mencionan a continuación al pie de la letra, ya que cualquier error puede dejar inservible a la Beagleboard. Antes de hacer esto es conveniente tener la versión definitiva de los archivos a copiar ya que la NAND es una memoria delicada.

Se accede a la ventana de cambios en los parámetros de arranque y se realiza lo siguiente:

```
OMAP3 beagleboard.org # mmc init
mmc1 is available
OMAP3 beagleboard.org # fatload mmc 0:1 80000000 MLO
reading MLO
19908 bytes read
```

Primero se inicializa el subsistema mmc para la memoria SD y luego se copia de la partición 1 de la mmc 0 el archivo MLO en la dirección 80000000.

Para escribir en la NAND se debe habilitar la siguiente funcionalidad.

```
OMAP3 beagleboard.org # nandecc hw
HW ECC selected
```

Se borran las direcciones de memoria en donde se quiere escribir.

```
OMAP3 beagleboard.org # nand erase 0 80000
NAND erase: device 0 offset 0x0, size 0x80000
Erasing at 0x60000 -- 100% complete.
OK
```

Se copia en NAND el MLO.

```
OMAP3 beagleboard.org # nand write 80000000 0 80000
NAND write: device 0 offset 0x0, size 0x80000
524288 bytes written: OK
```

Para copiar el u-boot se procede d ela misma manera.

```
OMAP3 beagleboard.org # mmc init
mmc1 is available
OMAP3 beagleboard.org # fatload mmc 0:1 80000000 u-boot.bin
reading u-boot.bin
188600 bytes read
OMAP3 beagleboard.org # nandecc sw
SW ECC selected
OMAP3 beagleboard.org # nand erase 80000 160000
NAND erase: device 0 offset 0x80000, size 0x160000
Erasing at 0x1c0000 -- 100% complete.
OK
OMAP3 beagleboard.org # nand write 80000000 80000 160000
NAND write: device 0 offset 0x80000, size 0x160000
1441792 bytes written: OK
```

Beagleboard en la red

Es de interés conectar la Beagleboard a través de una interfaz ethernet con el PC de desarrollo para poder intercambiar archivos e incluso conectar la Beagleboard a internet. Para lograr lo anterior se agregaron los módulos necesarios durante el desarrollo del uImage. Para la emulación de la conexión por red se utiliza el USB-OTG de la Beagleboard.

Conexión con beagle a través del USB-OTG

Tener una conexión por red con la BeagleBoard permite el uso de comandos como ssh para acceso remoto y el envío de archivos mediante el comando scp. Si se conecta la Beagleboard con el PC de desarrollo, se puede ver en el PC la aparición de una conexión de red mediante la interfaz usb0. Se debe configurar esta interfaz como sigue:

$ ifconfig usb0 172.16.1.17 netmask 255.255.255.0

La única condición que debe tener la ip es que no sea parte de la subred a la que pertenece el PC desarrollo (en general 192.168.x.x) ya que esto lleva a errores de comunicación y por ejemplo, no permite la salida de la Beagleboard a internet.

Para que se establezca la conexión es necesaria la ejecución de algunos comandos en la Beagleboard, por ejemplo interesa una ip fija para identificarla en la red. Para que la conexión se realice cada vez que se inicia el sistema, se creó un script (ethusb) que se corre al inicio y ejecuta los comandos antes mencionados. A continuación se muestra el script ethusb:

```
#!/bin/sh
echo Ya se puede conectar...
ifconfig usb0 172.16.1.14 netmask 255.255.255.0
route add default gw 172.16.1.17
echo nameserver 192.168.1.2 > /etc/resolv.conf
```

Se configura la ip que va a tener la Beagleboard. La puerta de enlace debe ser la ip del PC de desarrollo y el nameserver debe ser la dirección de la ip de salida a internet que tiene la PC de desarrollo.

Para que el script se ejecute cada vez que arranca el sistema, se debe hacer lo siguiente:

$ cp ethusb /etc/init.d

$ cd /etc/rc5.d

$ ln -s ../init.d/ethusb S99ethusb

El S99 quiere decir que lo que se ejecuta es un script, y que es el último en ejecutarse.

Ya no va a ser necesaria una configuración cada vez que se inicie el sistema. Hay que tener el USB desconectado durante el inicio del sistema, en caso contrario al arrancar se presenta un kernel panic.

Para la conexión de la Beagleboard a internet, ver apéndice C.12.

Para el uso del gestor de paquetes (opkg), ver apéndice C.1.

5.4.7. Bibliotecas

Software para el manejo de GPIO

Este módulo de software fue realizado basándose en la interfaz genérica SYSFS para el control de GPIOs [44] [45], esto permite que el código pueda ser portado a cualquier otro sistema que use Linux y que disponga de este tipo de hardware.

El módulo de software para el uso de los puertos de propósito general, GPIO, en principio puede resultar poco importante a simple vista, pero esta porción de código es usada por el resto de los módulos que conforman la aplicación completa del prototipo RF^2. Este módulo cuenta básicamente con una estructura que permite almacenar el estado de cada puerto, una macro y 4 funciones que se datallan a continuación. La primera de las funciones se llama config_gpio_pin() y permite exportar desde el espacio kernel al espacio usuario las funcionalidades necesarias para hacer uso del puerto que se indica como argumento. Al momento en que se exporta, se indica la dirección, o sea si será un puerto de entrada o salida, a través de un parámetro que es pasado a la función. La función que permite leer el valor actual de un puerto se llama read_gpio_pin(), es necesario pasarle como argumento el indicador del puerto del cual queremos conocer su valor. El valor del puerto es guardado en la estructura que almacena el estado de cada puerto para posteriores consultas, sin tener que volver a llamar a dicha función. Las últimas dos funciones son contrapuestas, set_gpio_pin() y clear_gpio_pin(), éstas permiten poner el valor de un puerto específico en el valor lógico “1” o “0” respectivamente. Previo a establecer o borrar el valor del puerto ambas funciones verifican que la dirección del mismo sea de salida (como mecanismo de seguridad no es posible cambiar el valor de un puerto de entrada). Por su parte la macro reset_status_gpio() permite borrar el estado de un puerto que ya no esté en uso.

Software para comunicación SAM

Hoy en día la mayoría de los lectores de tarjetas de contacto tienen una interfaz USB para ser conectado en un PC en aplicaciones de escritorio. Para el uso de este tipo de lectores sobre Linux existe un controlador genérico llamado CCID. Sin embargo las

tarjetas de contacto no poseen un puerto USB sino un puerto serie para establecer la comunicación con algún dispositivo, es por esto que en la nueva generación de lectores siempre hay un ASIC para lograr la interacción, por un lado con la tarjeta de contacto y por el otro la comunicación con el PC. Como fue descrito en la sección de hardware, el lector de tarjetas de contacto tiene una interfaz serial pura para la transferencia de datos con las tarjetas. En base al diseño hardware elegido, las capas de software sobre las que se decidió trabajar son las que se detallan en la figura 5.6.

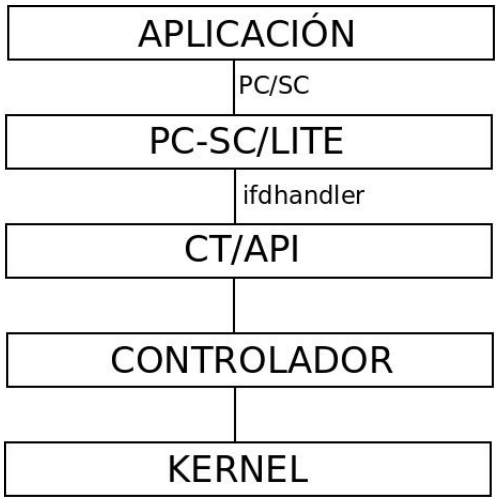

Figura 5.6: Capas de software de trabajo

En una primera etapa y para simplificar el desarrollo y la depuración del software, las capas empleadas fueron las que se muestran en la figura 5.7.

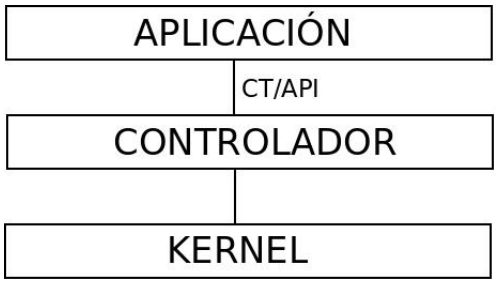

Figura 5.7: Capas de software en una primera etapa

A continuación se describen las capas asociadas a la figura 5.6.

Controlador:

El kernel es el encargado de manipular directamente los registros del puerto serial, las interrupciones que desde éste se generan y la ISR para atender las interrupciones. La implementación del controlador del lector de tarjetas se basó en el controlador serial de Linux a través de su estructura "termios" [40]. Esta estructura permite configurar todos los parámetros necesarios para la comunicación serial como ser, baud rate, cantidad de bits por byte, bit de paridad, bit de parada entre otros. Las funciones read y write permiten la lectura y escritura de los bytes de datos que son recibidos y transmitidos por el puerto serial.

Dentro del proyecto MUSCLE/PCSCLite existe código ejemplo de controladores de lectores de tarjetas de contacto que son compatibles con la biblioteca, si bien el desarrollo se basó en alguno de ellos, como el hecho de usar el mismo nombre de funciones, su contenido fue desarrollado específicamente para controlar el hardware específico del prototipo RF^2.

CT/API (Card Terminal / Application Programming Interface):

Por encima del controlador serial se encuentra CT/API [20], una interfaz definida por varias empresas (entre las que se incluye Telekom Alemania) en la década de los noventa, que permite encapsular el controlador específico de cada lector de tarjetas, de manera que la aplicación final no se vea afectada al cambiar un lector por otro. Esta interfaz de programación está formada tan solo por 3 funciones, CT_init, CT_data y CT_close, que permiten la inicialización del lector, la transferencia de datos entre host/lector o host/-tarjeta (host se refiere a la SBC o PC donde se encuentra conectado el lector de tarjetas de contacto) directamente y el cierre de la comunicación.
CT_init se encarga del pasaje de parámetros a la capa del controlador, para la configuración del puerto de comunicación entre el host y el lector de tarjetas. Los parámetros en uso aquí son: la tasa de transferencia de datos, el número de bits por cada byte, el tipo de paridad empleado y el puerto serie a ser utilizado.
CT_data es la función encargada de transferir comandos y datos hacia y desde la tarjeta o hacia y desde el lector (en caso que el mismo esté formado por un ASIC o microprocesador). La manera de diferenciar desde donde es enviado el dato, es a través de un parámetro pasado a esta función, y de forma análoga se determina el destino del mensaje. El protocolo usado para la transferencia de datos es T=0, orientado a bytes y del cual

pueden conocerse más detalles en [18].
CT_close es la contracara de CT_init, se encarga de cerrar la comunicación con el lector. Lo que hace básicamente es liberar el handle (puntero) asociado al puerto serial.

Las funciones de apertura y cierre de la comunicación no necesitan ser modificadas, son similares en todos los controladores de lectores seriales compatibles con la biblioteca. Por otro lado la función CT_data debió ser sustancialmente modificada para lograr el intercambio de datos a partir del protocolo T=0.

Para el caso en que los comandos y/o datos estén dirigidos hacia el lector, existe otra especificación, llamada CT/BCS [21] (Card Terminal / Basic Command Set), donde se encuentran definidos una serie de comandos básicos para el manejo del lector. Estos comandos se nombran a continuación y se da una breve descripción.

RESET CT permite reiniciar el lector o tarjeta (en el caso de ser un lector mudo); de manera opcional puede devolver el ATR.

REQUEST ICC tiene como objeto devolver el ATR de la tarjeta una vez que la misma se encuentra ubicada en el zócalo del lector.

GET STATUS es empleado para conocer información sobre el lector o si la tarjeta está insertada y eléctricamente conectada en el lector.

EJECT ICC genera la desactivación eléctrica de la tarjeta.

IFDHandler:

El siguiente componente en este stack de capas es ifdhandler [22], no es otra cosa que un conjunto de funciones formando una API, empleada por pcsclite para encapsular el manejo del hardware de lectores cuyos fabricantes quieran cumplir con las especificaciones PC/SC [62]. Una ventaja importante de esta API es que le permite a pcsclite operar tanto con lectores de puerto serial como con lectores de puerto USB. Esta capa de software podría usarse directamente sobre el controlador del lector, prescindiendo de CT/API, aunque se decidió mantenerla por motivos de simplicidad ya que sólo es necesario sustituir la capa de aplicación por las restantes capas superiores como se indica en las figuras anteriores.

Las funciones que contiene esta capa de software fueron modificadas para que contengan a su vez las funciones de CT/API, y de esta manera hacer más fácil de integrar el controlador del lector con la biblioteca PCSCLite.

En lo que sigue se enumeran algunas de las funciones de esta API y se describen brevemente. Por más detalles ver el manual ifdhandler [22].

IFDHCreateChannel establece el canal de comunicación con el lector. Para conseguirlo usa un parámetro llamado Channel, que indica cual es el puerto serial a usar, por ejemplo para el caso de Linux /dev/ttySx (x es el número que corresponda).

IFDHCloseChannel implementa la acción opuesta a la función anterior, cerrando el canal de comunicación con el lector de tarjetas.

IFDHGetCapabilities permite obtener las capacidades específicas del lector o de la tarjeta insertada en el mismo.

IFDHPowerICC se encarga del control de las señales de alimentación y reset que el lector suministra a la tarjeta. Desempeña tres acciones posibles, encendido, reset y apagado de la tarjeta.

IFDHTransmitToICC se encarga de la transferencia de datos con la tarjeta a través de alguno de los protocolos disponibles, como ser T=0 o T=1.

IFDHICCPresence retorna el estado de la tarjeta insertada en el zócalo del lector.

PCSCLite:

Por arriba de ifdhandler se encuentra la biblioteca pcsclite, ésta contiente todas las funciones necesarias para establecer la comunicación con un lector y la tarjeta conectada a éste último. Para el uso del controlador encapsulado por ifdhandler desde pcsclite es necesario seguir los pasos de configuración detallados en el apéndice C.13.

Aplicación final:

Por arriba de todas las capas descritas antes, se encuentra la aplicación del prototipo que hace uso de las funciones suministradas por pcsclite y donde se encuentran definidos los comandos APDU específicos con los que opera la tarjeta de contacto. Por razones de seguridad no se nos permite difundir la lista de comandos que son usados para la comunicación con el módulo SAM.

Software RFID

librfid es una biblioteca de software libre para manejo de lectores/escritores RFID. Implementa el stack de protocolos del lado del dispositivo lector/escritor ISO 14443A, ISO 14443B, ISO 15693, Mifare Ultralight y Mifare Classic.

Entre los lectores soportados están, OpenPCD y algunos modelos Omnikey, éstos con interfaz de conexión USB. Además tiene soporte para cualquier otro lector con comunicación directa con el CL RC632 mediante la interfaz SPI y es por esta razón que se tuvo en cuenta.

El manejo de librfid es de bajo nivel y se comunica directamente con el kernel utilizando el módulo spidev. Librfid-tool implementa funciones de más alto nivel que hacen uso de las funciones de bajo nivel de la librfid. La estrucrura de software comentada se muestra en la figura 5.8.

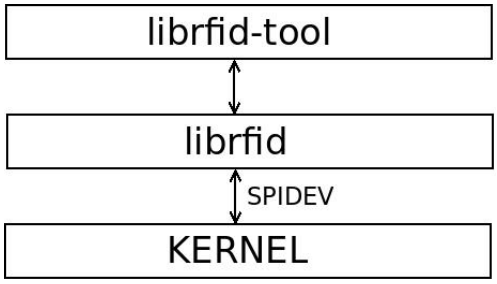

Figura 5.8: Capa de software RFID

A continuación se detallan los cambios introducidos en la librfid para el correcto funcionamiento del lector/escritor RFID utilizando librfid-tool.

Se analizó el código principal de la aplicación librfid-tool y se vio que dentro del directorio utils en el archivo common.c se encuentra la función reader_init la cual busca un lector/escritor entre los soportados. Esta función no tenía implementada una búsqueda para dispositivos conectados por SPI. Por lo tanto se tuvieron que agregar las siguientes líneas a la función para que el funcionamiento fuera posible:

```
rh = rfid_reader_open("/dev/spidev3.0", RFID_READER_SPIDEV);
if (!rh) {
    fprintf(stderr, "No SPIDEV found\n");
    return -1;
}
```

Este cambio permitió detectar la interfaz SPI (spidev3.0). Algo a tener en cuenta, es que como en Linux las interfaces están asociadas a un archivo, el hecho de abrir el archivo no implica que haya nada conectado en esa interfaz. Por esta razón, la búsqueda de un lector/escritor conectado por interfaz SPI se realiza en última instancia.

Otro cambio fundamental es en la frecuencia de reloj del puerto SPI. Se incrementa a 10MHz, ya que con la frecuencia establecida por defecto en la librfid (1MHz) el lector/escritor RFID no funciona correctamente.

Toda la configuración de la comunicación por SPI se encuentra en rfid_reader_spidev.c que está en el directorio fuente src.

La función que se modificó es spidev_open y el cambio se muestra a continuación:

```
tmp = 10e6; /* 10 MHz */
if (ioctl(spidev_fd, SPI_IOC_WR_MAX_SPEED_HZ, &tmp) < 0)
    goto out_rath;
```

Se implementaron las funciones adecuadas para el manejo del pin RST_RF, el cual puede encender y apagar el lector/escritor RFID diseñado.

Después de estudiadas las funciones que provee la herramienta librfid-tool, se estudió la posibilidad de su uso para la aplicación RF^2.

Aunque sus funciones son muy útiles, la gran mayoría no sirven completamente para la aplicación RF^2 debido a que fueron definidas para otros propósitos.

Es de interés, que la herramienta librfid-tool siga manteniendo sus funcionalidades y pueda convivir con la aplicación RF^2, por lo que no se modificó el contenido de ninguna función de la herramienta. En el caso que alguna función fuera mayormente utilizable, se procedió a crear una nueva con los cambios necesarios.

Se implementaron la mayoría de las funciones, logrando compatibilidad con la biblioteca librfid. Entre las funciones creadas están las asociadas con la tarjeta RFID: autenticación según el tipo de clave, búsqueda de tarjetas próximas al lector, lectura y escritura de bloques de memoria, obtención del UID, etc.

5.4.8. Aplicación final

Para el desarrollo de la aplicación RF^2 se decidió trabajar sobre los fuentes de la herramienta librfid-tool, ya que maneja varias funciones de utilidad y es de ayuda a la

hora de compilar para el armado de una aplicación completa. Se mantuvieron todas las opciones de la herramienta ya que son muy útiles y pueden ayudar en un futuro para establecer orígenes de fallas. No se modificó ninguna función de la aplicación original y cuando fue necesaria alguna modificación, se procedió a implementar una nueva.

En la figura 5.9 se detallan las capas de software del sistema RF^2.

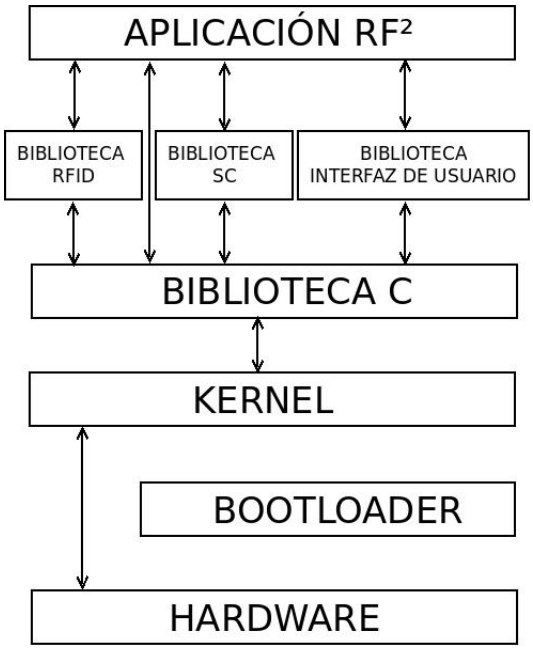

Figura 5.9: Capas de software del sistema RF^2

Antes de seguir fue necesario entender el funcionamiento de las reglas de compilación creadas para la aplicación librfid-tool sobre librfid. En el directorio raíz, se encuentran los siguientes archivos importantes para el desarrollo del sistema: autogen.sh, configure.in, configure, Makefile.am, Makefile.flags.am, Makefile.in, Makefile.

A continuación se describe a grandes rasgos la utilidad de cada uno de estos archivos:

configure.in es el archivo de configuración con el cual se crea el archivo configure.

Makefile.am es el archivo que establece las reglas de compilación (orden, dependencias, etc) y es con el cual se crea el archivo Makefile.in. Diferentes Makefile.am se encuentran en los subdirectorios de la librfid, por lo que se crea un Makefile.in dentro de cada subdirectorio.

Makefile.flags.am es un archivo con banderas establecidas para el sistema. Este archivo está incluido en todos los Makefile.am de la aplicación.

autogen.sh es un script que se encarga de crear los archivos configure y Makefile.in, cada uno de ellos creados a partir de los archivos correspondientes antes mencionados.

Al ejecutar el archivo configure, éste establece la configuración que necesita el Makefile.in para conocer las reglas de compilación que se van a usar y algunos parámetros extra. Luego de este proceso se generan los Makefile.

El archivo Makefile es el que conoce las reglas de compilación, las dependencias y las opciones elegidas para el desarrollo. Sabe incluso donde se encuentran los demás Makefile para ejecutarlos cuando sea necesario.

Cada subdirectorio tiene reglas locales de construcción de sus objetos y a su vez se ayudan mutuamente para lograr una aplicación más completa. Desde el Makefile principal se controla que todo funcione correctamente. Para saber más sobre reglas de compilación puede visitarse [42].

La aplicación RF^2 utiliza otras bibliotecas además de librfid, además estas bibliotecas interactúan entre sí, lo que lleva a que aparezcan nuevas dependencias. Por lo tanto, fue necesario modificar las reglas de compilación.

Se agregó en el raíz de la librfid el directorio rf2 en el cual se encuentran subdirectorios con los fuentes necesarios para la comunicación con el display, leds, buzzer, tarjeta de contacto y otros utilitarios, formando la estructura que se muestra en la figura 5.10.

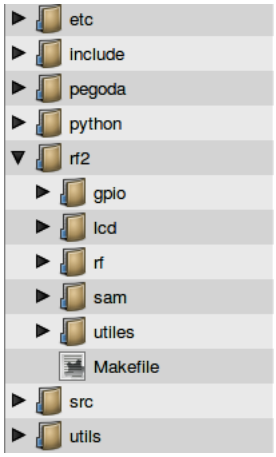

Figura 5.10: Estructura de árbol de aplicación RF^2

gpio incluye fuentes para el manejo de los GPIO.

lcd incluye fuentes para el manejo del display.

rf incluye fuentes con funciones extra de utilidad para el manejo del lector/escritor RFID diseñado.

sam incluye los fuentes para la comunicación con la tarjeta de contacto.

utiles incluye fuentes con funciones útiles de la aplicación.

Dentro de cada uno de estos directorios se encuentra un archivo Makefile con las reglas de compilación correspondientes a cada uno.

Las modificaciones a los archivos de construcción de la aplicación para que contemplen el agregado del directorio rf2, como la creación de nuevas reglas para la construcción dentro de éste, se detallan a continuación.

Antes que nada se decidió modificar los archivos Makefile.am para que sepan de la existencia del nuevo directorio. Se modifican estos archivos debido a que nunca son borrados. Por ejemplo, si los cambios se hacen sobre los Makefile.in, puede pasar que en algún momento se regeneren borrando los cambios que se hicieron. Además, luego de entender la estructura y funcionamiento de estos archivos, es más fácil incluir una modificación en Makefile.am que en un Makefile.in o Makefile directamente.

Dentro del directorio utiles, se creó el archivo Variables_Make que establece el valor de algunas variables específicas de la aplicación RF^2 como ser CC_arm que hace

referencia al gcc asociado con la herramienta de croscompilación para ARM.

Makefile.flags.am: Aquí se indicó que se agregue rf2 a la ruta de búsqueda de archivos encabezados.

```
INCLUDES = \$(all_includes) -I$(top_srcdir)/include
-I\$(top_srcdir)/rf2
```

Makefile.am en el raíz:

```
include rf2/utiles/Variables_Make
```

Con esto se aseguró que el resto de los subdirectorios pertenecientes a la librfid, sepan los valores de las variables específicas de la aplicación RF^2.

Luego se incluye el directorio rf2 a la aplicación, teniendo en cuenta que solo se va a incluir cuando se use un lector/escritor RFID con iterfaz SPI.

```
if ENABLE_SPIDEV
SUBDIRS += rf2
endif
```

Makefile.am en utils: Este es el cambio más dificil de entender y surge de la observación del Makefile.in generado cada vez (prueba y error). Como la aplicación RF^2 se basa en la modificación de los fuentes de la herramienta librfid-tool, se deben agregar todas las dependencias con archivos del nuevo subdirectorio rf2.

```
librfid_tool_SOURCES = librfid-tool.c librfid-tool.h
common.c common.h ../rf2/gpio/gpio.c ../rf2/gpio/gpio.h
../rf2/rf/rc632_utils.c ../rf2/rf/rc632_utils.h
../rf2/lcd/lcd16x2.c ../rf2/lcd/lcd16x2.h ../rf2/sam/sam.c
../rf2/sam/sam.h ../rf2/sam/sam_util.c ../rf2/sam/sam_util.h
../rf2/utiles/utiles.c ../rf2/utiles/utiles.h
```

Cada .h y .c se traduce en un .o al crear el Makefile.in.

Otro cambio necesario, ya que si no se incluye provoca errores de compilación, es el siguiente.

```
mifare_tool_SOURCES = mifare-tool.c common.c
../rf2/gpio/gpio.c ../rf2/rf/rc632_utils.c
../rf2/lcd/lcd16x2.c ../rf2/sam/sam.c
../rf2/sam/sam_util.c ../rf2/utiles/utiles.c
```

mifare-tool es otra herramienta de la librfid.

Luego se creó dentro del directorio rf2 un archivo Makefile que es el que ordena la construcción de todos los objetivos dentro de cada subdirectorio.

```
include utiles/Variables_Make

all: gpio/gpio.o rf/rc632_utils.o lcd/lcd16x2.o
sam/sam.o utiles/utiles.o

gpio/gpio.o:
$(MAKE) -C gpio

rf/rc632_utils.o:
$(MAKE) -C rf

lcd/lcd16x2.o:
$(MAKE) -C lcd

sam/sam.o:
$(MAKE) -C sam

utiles/utiles.o:
$(MAKE) -C utiles

install:

clean:
rm -f *.o
$(MAKE) -C gpio clean
$(MAKE) -C rf clean
$(MAKE) -C lcd clean
$(MAKE) -C sam clean
$(MAKE) -C utiles clean

distclean: clean
```

En este caso $(MAKE) -C “directorio” indica que la regla de construcción del objetivo se encuentra en “directorio”.

Se implementó una función de inicialización, la cual inicializa todos los periféricos al arrancar la aplicación RF^2.

Se agregó una nueva opción (n) en el main de la aplicación librfid-tool que llama a la función principal(). Ésta, es la función principal de la aplicación RF^2, la cual se ejecuta en loop. De este modo no se modifica el main original de librfid-tool (solo se agrega la opción n) y se dejan las opciones por defecto. Se hizo uso de algunas funciones ya escritas y se crearon otras. En este punto no se modificó ninguna función de la aplicación original y cuando fue necesaria alguna modificación, se procedió a implementar una nueva función con las modificaciones previstas. Luego viene la etapa de croscompilación de la aplicación para ser probada en la SBC.

Croscompilación de la aplicación final: Para guardar el resultado de la croscompilación se creó un directorio work en el home del usuario.

$ ./autogen.sh

Este paso es necesario siempre que se modifiquen los Makefile.am, en otro caso no. Se recuerda que este script genera los archivos Makefile.in y configure.

$ ./configure −−enable-spidev −−host=arm-angstrom-linux-gnueabi −−prefix=/home/proyecto/work

Se configura el sistema para lectores/escritores RFID con interfaz SPI, se indica la arquitectura de la SBC para la cual se compila y se indica el directorio donde se instala la aplicación. Luego de hecho esto ya no es necesario volver a ejecutarlo ya que los Makefile quedan creados con esta configuración.

$ make clean && make -j5 && make install

Se construye la aplicación.

En el directorio work se encuentran cuatro nuevos directorios creados:

bin: incluye los binarios construidos.
include: directorio con los .h necesarios para correr la aplicación.
lib: la biblioteca en sí.
share: nada de utilidad.

El paso siguiente es el de copiar estos directorios en el sistema de archivos de la SBC.

Copia de archivos a la SBC:

Para realizar la copia conviene comprimir el resultado de la croscompilación y luego enviarlo a la SBC.

```
$ tar -czf rf2.tar.gz bin include lib share
```

Luego de enviar el archivo comprimido, es necesaria la copia de estos directorios en el sistema de archivos de la SBC.

Instalación en la SBC:

```
$ tar -xf rf2.tar.gz -C /usr
```

Esto descomprime los directorios creados anteriormente, dentro del directorio /usr.

Se ejecuta la aplicación RF^2:

```
$ librfid-tool -n
```

Parte III

Ensayos

Capítulo 6

Ensayos

6.1. SBC

Las Hawkboard fabricadas entre el 1º de agosto y el 20 de octubre de 2010 fueron vendidas en el mercado con un error a nivel de hardware que no había sido constatado por el fabricante y que no fue reconocido por éste hasta el mes de noviembre. La solución al problema fue liberada en la fecha 20 de diciembre de 2010 y constaba de sustituir en el circuito, los ferrites FB12 y FB13 por un puente de soldadura de estaño (el uso de jumper 0R fue probado sin obtener buenos resultados). Mayores detalles de la solución pueden encontrarse en el documento Hawkboard Press Release Solution que se adjunta en D. El inconveniente mencionado antes evitaba que el sistema operativo Linux iniciara correctamente, generándose un mensaje de "kernel panic" indicando que el sistema operativo no podía ser ejecutado. Esto evitó que se pudieran probar las partes de hardware y software que se tenían desarrolladas hasta ese entonces, teniendo que recurrirse a mecanismos alternativos como el uso de un microprocesador rabbit para efectuar pruebas sobre el lector/escritor RFID.

Pruebas sobre las interfaces en la Beagleboard

Testeo de GPIO: Para el testeo de los GPIO, se compiló y probó el archivo led.c (ver apéndice C.15) el cual cambia el valor del pin 13 del bloque de expansión cada un segundo. Si se coloca un led entre este pin y la tierra, se puede ver como el led prende y apaga.

Testeo de UART: Para el testeo de la interfaz serial UART, se compiló y probó el programa uart.c (ver apéndice C.16) que envía una serie de caracteres por uart_tx y luego lee por uart_rx. Para verificar el correcto funcionamiento se debe cortocircuitar uart_tx con uart_rx.

Testeo de SPI: Para el testeo del SPI se consiguió un ejecutable (spidev_test) que hace algo parecido a lo que realiza el archivo de testeo de la uart. En este caso se debe ejecutar el archivo con los parámetros correspondientes ./spidev_test -D /dev/spidev3.0 (spidev3.0 porque se está usando spi3 con cs0). Cuando se ejecuta, deben aparecer en pantalla varias filas con "FF". Si se cortocircuita SIMO y SOMI algunas filas deberían cambiar (no "FF") con lo que queda verificado el buen funcionamiento del SPI.

6.2. VLT - Conversor de Voltajes

No existieron problemas en este módulo, y dadas las características del circuito no hay demasiados puntos de falla. Si fuera necesario verificar los valores de tensión en el regulador de tensión, la tensión de entrada puede ser medida desde el conector CONN_14x2 y la de salida desde el conector CONN_20x2, ver figura 4.1. Un detalle a tener en cuenta a la hora de medir los valores de tensión de las señales que pasan a través de los conversores de nivel, cuando las mismas se encuentren en estado ocioso (estáticas), es que no debe hacerse con multímetros de mala calidad, o se obtendrán valores incorrectos durante la medición. Se recomienda para una correcta medición el empleo de osciloscopio con puntas x10. Como se mencionó antes no se tuvieron inconvenientes con este módulo, pero generó conflictos en el circuito conversor full a half duplex del lector de tarjetas de contacto que serán detallados más adelante.

6.3. SCUI - Lector de tarjetas de contacto e Interfaz de usuario

Lector de tarjetas de contacto ISO7816

Las primeras pruebas realizadas sobre el lector de tarjetas de contacto se efectuaron sobre una placa de circuito impreso de fabricación propia, conectándose el lector direc-

tamente sobre el conector de expansión de la Beagleboard. La intención de esta prueba era más que nada la de probar el circuito conversor full a half duplex, transmitiendo una serie de bytes por el canal Tx y recibiendo el eco mediante el canal Rx, cotejando que los bytes recibidos coincidieran con los transmitidos. El primer problema encontrado estuvo asociado a una falla en uno de los transistores, el PNP 3906, que debió ser sustituido por encontrarse defectuoso. El software usado para efectuar las pruebas sobre el hardware se basa en un controlador serial desarrollado por el grupo mina del INCO, el cual fue mínimamente modificado ya que uno de los parámetros, CSIZE, en la configuración del puerto afectaba el número de bits que conforman un byte recibido. La línea de código que hacía referencia a este parámetro fue comentada ya que modificaba el valor del parámetro csN, con N=5 en lugar de N=8 (donde N es el número de bits que forman el byte). El cambio anterior permitió que los bytes recibidos en el canal Rx coinicidieran con los transmitidos en Tx, validando en una primera instancia el hardware conversor full a half duplex del lector de tarjetas. El siguiente paso fue intercalar entre la Beagleboard y el lector de tarjetas de contacto, el conversor de niveles (VLT) para realizar las mismas pruebas que se datallaron antes, aunque en este caso los resultados no fueron alentadores ya que los bytes recibidos no coincidían con los transmitidos. Todo indicaba que el conversor de nivel afectaba el conversor full a half duplex. Luego de algunas pruebas más sobre el circuito, sin cambios favorables, se decidió consultar al foro de Texas Instruments (fabricante del integrado TXB0108). Desde el soporte técnico solicitaron se les enviara una imagen capturada con osciloscopio de las señales en el puerto serial para observar la forma de los pulsos. En la figura 6.1 debajo se puede ver la deformación de los pulsos en la señal Rx (canal 1 del osciloscopio) cuando el circuito contaba con un valor de 500 Ohms en la resistencia R9 (ver figura 4.3); la solución encontrada fue disminuir el valor de R9 y no aumentarlo como se había intentado anteriormente sin beneficio alguno.

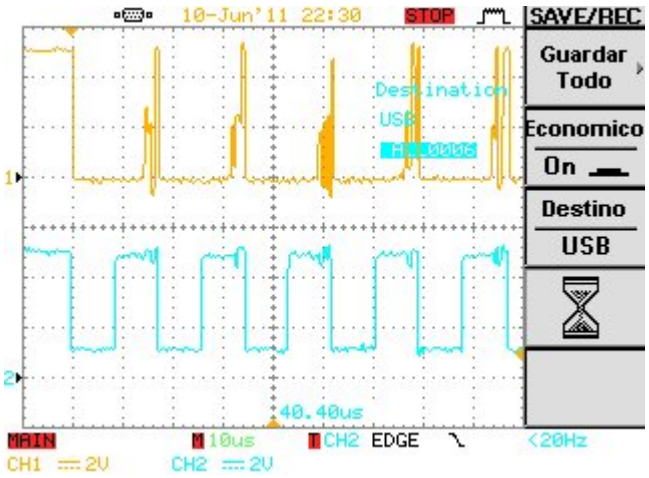

Figura 6.1: Señal en canal Rx para un valor de resistencia R9 de 500Ω

Al usar valores entre 90 Ohms y 180 Ohms para la resistencia R9, la forma de los pulsos recibidos en Rx (canal 1) fueron la copia de los pulsos transmitidos en Tx (canal 2), como puede verse en la figura 6.2 para un valor de 90 Ohms.

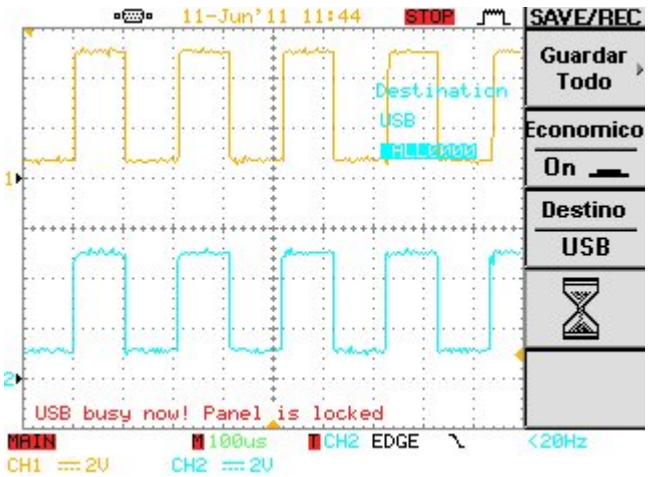

Figura 6.2: Señal en canal Rx para un valor de resistencia R9 de 90Ω

Aquí puede verse el hilo de discusión en el foro:
http://e2e.ti.com/support/interface/etc_interface/f/391/t/114719.aspx.

Una vez superados los obstáculos anteriores fue posible probar el circuito completo del lector, incluyendo la tarjeta de contacto en su zócalo correspondiente. El software

usado en tal fin se basa en un controlador serial, implementado por David Corcoran (uno de los desarrolladores de pcsclite), el cual debió ser modificado para usarse en el lector de tarjetas de contacto del prototipo RF^2. Una de las mayores dificultades encontradas en esta etapa fue el hallar los parámetros adecuados de inicialización del puerto serial, que debe cumplir con las opciones 8E2 (8 bits por byte, bit de paridad par, y dos bits de parada) para operar con las tarjetas de contacto compatibles con la norma ISO7816; sin embargo las opciones adecuadas elegidas en la configuración del puerto serial fueron 8E1. Adicionalmente al problema de encontrar las opciones correctas mencionadas antes, los bytes de datos recibidos como el ATR de la tarjeta no coincidían en su totalidad con los valores esperados (leídos con un lector Omnikey 3121 y la herramienta pcsc_scan de pcsclite), sólo algunos bytes y algunos nibbles bajos eran correctos. Esta diferencia estuvo asociada a la frecuencia usada para alimentar la señal de reloj de la tarjeta de contacto; se usaron frecuencias de 4 Mhz y 5 Mhz que si bien podrían usarse según se indica en [18] para los parámetros especificados en el ATR de las tarjetas empleadas, estos valores no fueron adecuados según lo ya comentado, teniendo que usar en su lugar un oscilador de frecuencia 3,579545 Mhz, valor que no se conseguió cuando se realizó la primer compra de componentes. Una dificultad adicional tuvo que ser sorteada en este módulo de hardware, el diseño del PCB que se envió a fabricar tenía un error, las pistas de datos de Rx y Tx estaban intercambiadas. El diseño tuvo que ser corregido y se envió a fabricar un nuevo PCB.

Interfaz de usuario

No existieron mayores inconvenientes con la interfaz para el usuario, sí fue necesaria la corrección en el valor de una resistencia en el circuito que calibra el contraste del LCD, ya que los caracteres se observaban muy tenues.

Al momento de probar el display imprimía caracteres extraños, salvo cuando se enviaban mensajes conteniendo una única palabra. Se probó cambiando los mensajes a desplegar en el mismo, y el problema persistía, pero se llegó a la conclusión de que era provocado por los espacios (“ ”) puesto que cuando se envió un mensaje omitiéndolos fue deplegado en forma correcta. Luego simplemente se modificó el código fuente, para que cada vez que recibiera un caracter espacio, enviara al display el código ASCII correspondiente solucionando el problema.

Cuando se comenzaron a imprimir los saldos de las tarjetas, volvió a imprimir caracteres extraños, esta vez el problema eran los caracteres “0”. La solución más rápida encontrada fue imprimir “O” cada vez que llegara un caracter “0”, por lo que se modificó el código para que así sea.

6.4. Lector/Escritor RFID

Al comenzar la fase de pruebas sobre el lector/escritor RFID, la SBC que había sido elegida no estaba operativa por razones que ya se explicaron, y la substituta no podía ser conectada directamente al lector por incompatibilidad en los niveles de tensión que manejan sus correspondientes interfaces. Como mecanismo alternativo se usó un módulo Rabbit, RCM4300, que permitió desarrollar una pequeña aplicación de software para hacer ciertas pruebas que validaran el diseño del hardware que se tenía hasta el momento. El software implementado se basó en el estudio de la hoja de datos del integrado CL RC632 , y en la biblioteca librfid [28].

Una vez conectado el lector/escritor RFID al conector de expansión del microcontrolador, el paso siguiente fue configurar el puerto SPI, para esto se contó con una biblioteca provista por Dynamic C (IDE para los microcontroladores Rabbit) que permite la configuración del puerto y posee funciones para la transmisión y recepción de datos. Contar con 4 modos posibles para la configuración de este puerto del microcontrolador dificultó la tarea; fue necesario emplear un osciloscopio para observar cual de éstas se adecuaba a la forma de señal que se indica en la hoja de datos del integrado CL RC632. Una vez seleccionado el modo correcto se puedo obtener una lectura válida del valor por defecto con que cuentan los registros de página 0 del CL RC632 luego de una inicialización, no así la de registros que se encuentran en otras páginas (ver 3.4.4 Registros), para alcanzar los demás registros fue necesario implementar una función que configurase el integrado para obtener direccionamiento plano de los registros y no por páginas. Validada la comunicación microcontrolador - CL RC632, lo siguiente fue implementar un conjunto de funciones que permitieran la lectura/escritura de registros de control, el buffer de datos, la memoria EEPROM, el establecimiento y borrado de bits de configuración, así como también se escribieron las funciones para generar el formato adecuado de las claves de autenticación y su posterior almacenamiento en memoria. Cuando se

culminaron las pruebas sobre el módulo digital del CL RC632, lo próximo fue poner en funcionamiento el transmisor para establecer una comunicación con las tarjetas RFID, esta etapa fue la más compleja y que se prolongó por mayor tiempo, pues se cayó en la disyuntiva si la imposibilidad en la comunicación por RF se debía a un error cometido a nivel de hardware o de software. Por un lado se podía pensar que al seguir las pautas de diseño y los cálculos indicados en [4] el error no debía ser de hardware, pero a medida que los cambios a nivel de software no generaban resultados favorables la balanza se inclinó hacia el lado del hardware. Cuando no se tiene al alcance el instrumental adecuado, un elemento que fue de mucha utilidad a la hora de comprobar la existencia de campo magnético fue un detector de campo magnético fabricado a partir de un alambre de cobre al que se le dio forma de bobina y se le soldó un led en sus extremos (ver figura 6.3), esta simple herramienta indica la presencia de campo magnético generado por la antena al encender su led cuando se encuentra en las proximidades de la antena; no olvidemos que debajo de la complejidad que puede llegar a tener este tipo de lector/escritor RFID se encuentran los principios básicos de la ley de Faraday.

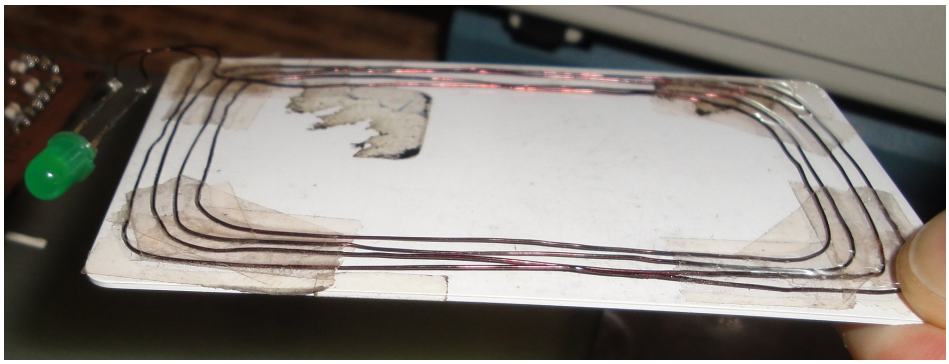

Figura 6.3: Detector de campo magnético casero

Posteriores pruebas usando un osciloscopio en el que una de sus puntas de prueba formaba una espira en lazo cerrado con su línea de tierra, permitió llegar a la misma conclusión que con el detector mencionado antes, no se producía campo magnético en el entorno próximo al inductor de la antena. Una manera nada elegante de resolver el problema anterior fue desconectar el inductor fabricado en el PCB y colocar en su lugar una bobina de 3 espiras y aproximadamente 10cm de diámetro, hecha a partir de alambre de cobre con aislante incluido, la fabricación de esta bobina se basó en una similar que

se puede encontrar en el lector/escritor Proxmark [36] (ver apéndice D). Este cambio permitió que la antena resonara a la frecuencia adecuada propagando la señal portadora a 13,56 Mhz, siendo posible observar ésta en el osciloscopio además de poder visualizar la forma de los pulsos que se generan en la transmisión de datos desde la antena hacia una tarjeta. Ya sin la incertidumbre a nivel de hardware, era necesario continuar con pruebas a nivel de software para lograr implementar el algoritmo de anticolisión que permitiera obtener el UID de cada tarjeta que se aproximara al lector/escritor, así como también su autenticación y su posterior lectura y escritura. Mientras se desarrollaban las funciones necesarias se avanzaba en paralelo en el diseño y la fabricación del conversor de niveles (VLT) que permitiera conectar el lector/escritor RFID directamente sobre la Beagleboard para hacer uso de la biblioteca librfid que ya contaba con las funciones necesarias y simplificaría todo el desarrollo desde cero. Una vez lista la placa VLT, se conectó el lector/escritor RFID directamente sobre la Beagleboard y comenzaron las pruebas de software con la bibilioteca antes mencionada. Las primeras pruebas fueron infructíferas, decidiendo continuar en paralelo con las pruebas de software también sobre el microcontrolador Rabbit. Mientras se continuaba con las pruebas en software, el hardware también debía ser modificado ya que la solución alcanzada anteriormente no era definitiva sino transitoria; se resolvió entonces diseñar un nuevo lector/escritor con su módulo digital separado de la antena, esto permitiría realizar mediciones sobre ésta última haciendo uso de un analizador de red, este tipo de instrumental es sumamente útil y hasta imprescindible cuando se diseña en RF. Las mediciones se realizaron con el instrumento: ”Vector Network Analyzers”de ROHDE&SCHWARZ (LXI Class C conformant) con que cuenta en préstamo el IIE, por medio del mismo fue posible observar el comportamiento de la impedancia de la antena a medida que se varía la frecuencia de trabajo. Para el circuito usado en el primer diseño de la antena, la frecuencia de resonancia se presentaba aproximadamente a 18 Mhz, es posible observar este detalle en la figura 6.4. Esta frecuencia se encuentra lejos de la frecuencia de trabajo de 13,56 Mhz y explica el porque no funcionaba el primer diseño.

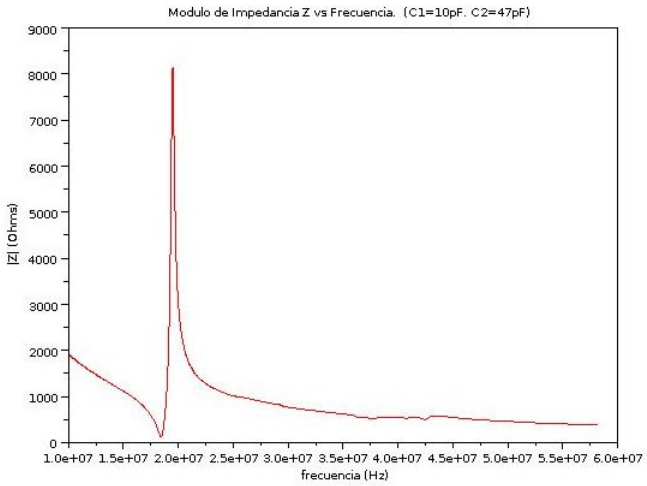

Figura 6.4: Módulo de impedancia z vs. frecuencia (C_1 = 10pF, C_2 = 47pF)

Posteriores modificaciones en el circuito de adaptación de impedancia, incrementando o disminuyendo el valor de los capacitores según el valor de impedancia obtenida, como se indica en [4], permitieron centrar la frecuencia de resonancia en 13,56 Mhz como puede observarse en la figura 6.5. Se debe hacer notar que el uso de las ecuaciones para hallar los valores de los capacitores que se encuentran en el [4] no son válidas, causando que se incurriera en error en la primer versión de la antena fabricada.

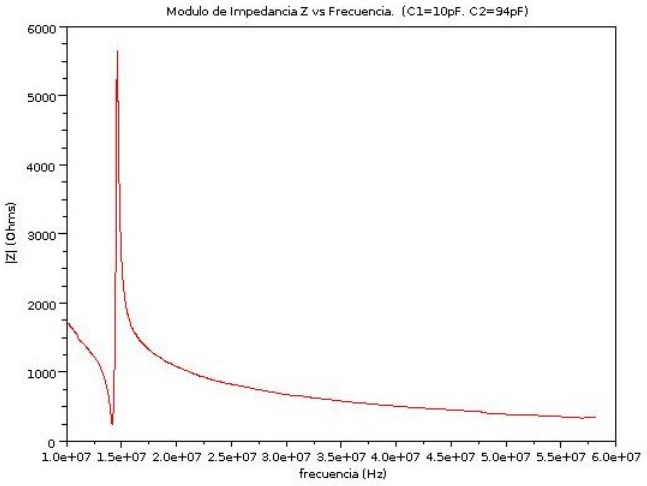

Figura 6.5: Módulo de impedancia z vs. frecuencia (C_1 = 10pF, C_2 = 94pF)

Las nuevas ecuaciones usadas se encuentran en B, y los valores de los capacitores que se obtienen a partir de éstas se encuentran próximos a los obtenidos por el método práctico haciendo uso del analizador de red. Restaba sólo incorporar el módulo digital a la antena para tener un lector/escritor RFID que permitiera continuar las pruebas de software. Las pruebas sobre el microcontrolador Rabbit continuaron mientras no se obtenían buenos resultados usando la biblioteca librfid sobre la Beagleboard, fue posible entonces la lectura del UID que posee cada tarjeta obtenido a partir del algoritmo de anticolisión, aunque no fue posible la lectura y/o escritura de las mismas por no lograr una adecuada autenticación. Las pruebas sobre la biblioteca librfid no arrojaron buenos resultados en una primera instancia, se tuvo que hacer algunas modificaciones para obtener el funcionamiento adecuado con el hardware fabricado. En primera instancia esta biblioteca sólo inicializa el lector OpenPCD, sin tener en cuenta si la croscompilación fue efectuada con opciones para manejo del puerto SPI, incorporar la inicialización del nuevo lector/escritor en esta biblioteca fue el primer paso para hacer uso adecuado del hardware. La siguiente dificultad que se debió enfrentar fue el haber pasado por alto que el pin RSTPD del integrado CL RC632 se encontraba en el estado lógico "1" produciendo un reset permanente, esto impedía que el integrado respondiera a las instrucciones enviadas desde la aplicación. Resuelto lo anterior comenzaron las pruebas con la lectura de tarjetas Mifare; la lectura de un bloque de una tarjeta no ofrecía mayores inconvenientes, pero el intento de lectura completa de una tarjeta traía aparejado que no se puedieran leer sectores completos de la misma. Luego de varias pruebas en la configuración de tiempos para la transmisión de datos en el canal RF, todas sin buenos resultados, se probó otra opción relacionada con la tasa de transferencia de datos usada por librfid en el canal SPI, que se establece por defecto en el valor 1Mbit/s. Este valor no era adecuado para lograr una lectura completa de los 64 bloques de una tarjeta que equivalen a 1024 bytes, teniéndose que incrementar a una tasa de 10Mbit/s para lograr una lectura completa de una tarjeta en forma válida. Modificaciones adicionales que no revistieron mayores dificultades fueron realizadas sobre librfid con el fin de hacer uso de las tarjetas Mifare que se usan en sistema de transporte, las que cuentan con claves de autenticación que no son las que vienen por defecto cuando las tarjetas están vírgenes.

Parte IV

Gestión de proyecto

Capítulo 7

Compras

La compra de componentes electrónicos se efectuó casi exclusivamente en empresas de Estados Unidos, entre las que figuran Newark, Digikey y Special Computing. Esto se debió a la dificultad que existe en plaza para conseguir los insumos necesarios para la fabricación del prototipo RF^2.

Entre los pocos elementos comprados en plaza se encuentran las placas de cirucito impreso que fueron fabricadas por la empresa Eneka.

En lo que sigue se mustran las listas de componentes separadas por el módulo de hardware al que pertenecen. Los precios que se detallan son en origen, en dólares americanos y no incluyen impuestos, excepto los PCB que incluyen IVA.

7.1. SBC

Componente	Descripción	Cantidad	Precio x1	Total
SBC	Beagleboard RevC4	1	125	125
Memoria SD	4GB SDHC Class 6 SD Card	1	15	15
Cable serial DB9 nulo	DB9F Null Modem (RS-232) (6-ft)	1	4	4
Cable conversor usb–serial	USB to DB9M RS-232 (PL-2302)	1	10	10
Cable USB	USB Mini-A to USB A Female, OTG	1	9	9
Cable USB	USB Mini-B Male to USB A Male	1	5	5
Fuente	5VDC/2,5A	1	10	10
				178

Cuadro 7.1: Single board computer y lista de accesorios.

7.2. PCBs

Componente	Descripción	Cantidad	Precio x1	Total
VLT	Interfaz entre SBC y SCUI	1	28	28
SCUI	Interfaz de usuario y lector de tarjetas ISO7816	1	53	53
RWD RFID	Lector/Escritor de tarjetas RFID ISO14443	1	64	64
				145

Cuadro 7.2: Lista de placas de circuito impreso.

7.3. VLT

Componente	Descripción	Footprint	Valor	Cantidad	Precio x1	Total
C1	Polarized Capacitor (Tantal)	6032[2312]	10uF, 25V	1	1,09	1,09
C2	Polarized Capacitor (Tantal)	6032[2312]	100uF, 6V3	1	1,16	1,16
U4	Regulador LM1117-3.3	SOT-223	3.3V, 800mA	1	1,1	1,1
U1, U2, U3	Voltage Level Translator	TSSOP20	-	3	2,24	6,72
P1	RECEPTACLE, 28WAY, 2ROW	SMD Pitch 2,54	28 pines	1	4,19	4,19
P2	RECEPTACLE, 40WAY, 2ROW	SMD Pitch 2,54	40 pines	1	4,36	4,36
P1b	HEADER, 28WAY, 2ROW	T H Pitch 2,54	28 pines	1	2	2
P2b	HEADER, 40WAY, 2ROW	T H Pitch 2,54	40 pines	1	1,94	1,94
						22,56

Cuadro 7.3: Lista de componentes de la placa de circuito impreso VLT.

7.4. SCUI

Componente	Descripción	Footprint	Valor	Cantidad	Precio x1	Total
R9	Resistor 100W 1/4W 1 %	3216[1206]	100W 1/4W 1 %	1	0,07	0,07
R10, R13	Resistor 100KW 1/4W 5 %	3216[1206]	100KW 1/4W 5 %	2	0,09	0,18
R11, R12, R14	Resistor 10KW 1/4W 5 %	3216[1206]	10KW 1/4W 5 %	3	0,08	0,24
Q2	TRANSISTOR, NPN, 300MHZ	SOT23	MMBT3904	1	0,125	0,125
Q3	TRANSISTOR, PNP, 250MHZ	SOT23	MMBT3906	1	0,18	0,18
J2	SIM socket (6 contacts)	SMD	-	1	1,25	1,25
JP3, JP4	HEADER, 1ROW, 3WAY	T H Pitch 2,54	3 pines	2	0,11	0,22
X1	Oscillator 3.579545MHz	SMD	3.579545 Mhz	1	5,25	5,25
ESD1	Anti ESD	SOT323	6V / 150W	1	0,45	0,45
						7,965

Cuadro 7.4: Lista de componentes del lector de tarjetas de contacto, SC.

Componente	Descripción	Footprint	Valor	Cantidad	Precio x1	Total
R1	Resistor 4K7 1/10W 1 %	1608[0603]	4,7KW 1/10W 1 %	1	0,05	0,05

R2, R8	Resistor 3R3 1/10W 1 %	1608[0603]	3,3W 1/10W 1 %	2	0,09	0,18
R3, R4, R5	Resistor 680R 1/10W 1 %	1609[0603]	680W 1/10W 1 %	3	0,05	0,15
R6, R7	Resistor 10K 1/10W 1 %	1608[0603]	10KW 1/10W 1 %	2	0,05	0,1
RV1	Preset 15K 1/10W 25 %	SMD	15KW 1/10W 25 %	1	0,71	0,71
Q1	TRANSISTOR, NPN, 300MHZ	SOT23	MMBT3904	1	0,125	0,125
S1	LCD MODULE 16X2 CHARAC-TER	Pitch 2,54	-	1	10,85	10,85
CONN1	HEADER FEMALE 16POS.1”TIN	Through Hole	16 pines	1	1,25	1,25
CONN2	HEADER, 1ROW, 16WAY	T H Pitch 2,54	16 pines	1	0,155	0,155
LED1	Led green 5mm	Through Hole	1,9V, 2mA	1	0,11	0,11
LED2	Led red 5mm	Through Hole	1,9V, 2mA	1	0,1	0,1
LED3	Led yellow 5mm	Through Hole	2,4V, 2mA	1	0,13	0,13
BUZZ1	Buzzer	Through Hole	3 20Vdc, 3 16mA	1	5,31	5,31
						19,17

Cuadro 7.5: Lista de componentes para la interfaz de usuario, LCD.

7.5. Lector/Escritor RFID

Componente	Descripción	Footprint	Valor	Cantidad	Precio x1	Total
C1, C2	Capacitor	1608[0603]	10pF, Ceramic NPO, 2 %	2	0,135	0,27
C3, C4	Capacitor	1609[0603]	100pF, Ceramic NPO, 2 %	2	0,194	0,388
C5, C6, C7, C8	Capacitor	1608[0603]	NC	4	-	0
R1, R2	Resistor	1608[0603]	0W, 1/10W, 1 %	2	0,015	0,03
						0,688

Cuadro 7.6: Lista de componentes del lector/escritor RFID.

Antena.

Componente	Descripción	Footprint	Valor	Cantidad	Precio x1	Total
C10	Capacitor	1610[0603]	10pF, Ceramic NPO, 2 %	1	0,135	0,135
C1, C2	Capacitor	1608[0603]	15pF, Ceramic NPO, 5 %	2	0,03	0,06
C12, C13	Capacitor	1608[0603]	56pF, Ceramic NPO, 2 %	2	0,194	0,388
C14, C15	Capacitor	1608[0603]	68pF, Ceramic NPO, 1 %	2	0,197	0,394
C9	Capacitor	1609[0603]	100pF, Ceramic NPO, 2 %	1	0,194	0,194
C16	Capacitor	1608[0603]	1nF, Ceramic NPO, 10 %	1	0,08	0,08

C4, C5, C7, C8, C11, C17	Capacitor	1608[0603]	100nF, Ceramic X7R, 10 %	6	0,074	0,444
C3, C6, C18	Capacitor	1608[0603]	10uF, Ceramic X5R, 20 %	3	0,195	0,585
L1, L2, L3, L6	Inductor	2012[0805]	22nH, 700mA, 5 %	4	0,454	1,816
L4, L5	Inductor	3225[1210]	1uH, 400mA, 5 %	2	0,29	0,58
R3	Resistor	1608[0603]	50W, 1/10W 1 %	1	0,268	0,268
R2	Resistor	1608[0603]	820W, 1/10W 5 %	1	0,027	0,027
R1	Resistor	1608[0603]	2,2KW, 1/5W 1 %	1	0,08	0,08
U1	Reader ISO14443	SO32	CL RC632	1	14,22	14,22
U2	Crystal Oscillator, HC49 US SMD	49USMXL	13.56MHz, 10pF	1	0,98	0,98
U3	Operational Amplifier (up to 7.5V)	SOT23-5	OPA354	1	2,8	2,8

CONN1, CONN2	U.FL-R Connector	U.FL-R-SMT	-	2	1,76	3,52
J1	HEADER, 10WAY, 2ROW	T H Pitch 2,54	10 pines	1	0,389	0,389
J1b	RECEPTACLE, 10WAY, 2ROW	SMD Pitch 2,54	10 pines	1	2,27	2,27
						29,23

Cuadro 7.7: Lista de componentes del lector/escritor RFID. Módulo digital + filtro EMC.

Capítulo 8

Tiempos

Cuando se planteó el proyecto, los objetivos eran similares pero el enfoque de solución era otro. Se tenía pensado utilizar, en vez de la SBC, un dispositivo (OpenPCD) como corazón del sistema. Luego de estudiarlo fue descartado y se estudiaron una serie de posibles soluciones, detalladas en 3.2, hasta dar con la solución desarrollada. Esto provocó un cambio en la planificación inicial. Para el primer hito ya se pensó en la posibilidad de replanificar sacrificando la comunicación con servidor, pero se dejó en suspenso dependiendo un poco de la respuesta que se tuviera del "cliente" (IM), que hasta el momento había sido de tiempos muy lentos para la planificación planteada y los plazos manejados. Pesaba además el grado de avance que se pudiera lograr a partir del nuevo planteo. El heho de realizar un lector/escritor RFID fue un gran desafío que se decidió tomar, a riesgo de que el proyecto se hiciera muy cuesta arriba en tiempos puesto que agregó varias tareas a la planificación pero como se mencionó quedó intacta en cuanto a alcance.

Para el segundo hito, aún el lector/escritor RFID no funcionaba y los tiempos se agotaban, ya se tenía un atraso de unos 2 meses dados los problemas (mencionados en el capítulo ensayos) con la SBC seleccionada en principio (Hawkboard) pero se propuso esperar por una replanificación, viendo que el hecho de solucionar el problema del lector/escritor RFID se transformó en el camino crítico y podía suceder de un momento a otro. Se definió un tiempo aproximado de un mes como espera razonable antes de plantear replanificar. Pasado ese tiempo el lector/escritor RFID estaba funcionando, pero los plazos ya se habían hecho muy cortos, por lo que se decidió pedir una prórroga

y replanificar para redondear lo mejor posible el trabajo. El pedido de prórroga se vio justificado por los dos meses perdidos con la primer SBC (mencionado en ensayos), y en la replanificación finalmente se sacrificó la comunicación con el servidor, que desde el primer hito se veía como la parte más firme a descartar de la planificación original.

Finalmente los tiempos se agotaron sin que se llegara a completar la comunicación con el lector de tarjetas de contacto.

Parte V

Conclusiones

Capítulo 9

Conclusiones

9.1. Conclusión final

El mundo de la tecnología RFID está poco explorado en nuestro país, éste tal vez sea el primer proyecto que incluye el diseño y fabricación de una antena RFID capaz de operar con tarjetas sin contacto en la banda de frecuencia de 13,56 Mhz.

Las posibilidades de aplicación son muy bastas, entre las que se pueden nombrar, sistemas de transporte, pasaporte electrónico, identificación civil, tarjetas de crédito y/o débito, salud, telefonía, control de acceso, entre otras.

Nuestro aporte en este campo es tan solo una primera aproximación y aún queda mucho por hacer al respecto. Sobre este proyecto en particular es necesario mejorar varios aspectos antes de pasar de la fase de prototipo a la de producción.

Repasando en particular los criterios de éxito, el proyecto ha resultado muy bueno porque se logró construir un dispositivo capaz de consultar y recargar tarjetas RFID tal como se quería. Si bien no se logró llevar a cabo de la forma específica en que se planteó solucionarlo, se logró el objetivo planteado sorteando todos los problemas que se presentaron. De modo que se hizo lo necesario para suplir la solución planteada con soluciones alternativas que hicieran funcionar el prototipo RF^2. En particular, el módulo de seguridad SAM no realizó su papel por dos simples razones. La primera que realmente no se necesitó conexión a un servidor porque se descartó en la última replanificación, por lo tanto el SAM no tuvo que encargarse de hacer segura la transacción, pero la función del servidor debió ser suplida. Lo que se hizo a través de un simple archivo que

contiene los montos que se desea acreditar a cada una de las tarjetas RFID y luego de acreditarlo se escribe un cero para no incurrir en el error de acreditarlo varias veces. La segunda que al no lograr comunicación entre la SBC y el lector de tarjetas de contacto, la SAM no necesitó devolver las claves de las tarjetas RFID sino que se insertaron directamente en el código para su uso dependiendo de la tarjeta RFID utilizada.

Además, si bien se diseñó enfocado a lo que es el actual sistema de transporte, con unos pocos cambios en el sotfware se puede emplear para variadas funcionalidades, como puede ser el acceso de personal a una empresa, el control de socios en un club deportivo, etc..

En suma, el equipo está conforme con el trabajo realizado.

9.2. Ventajas y desventajas

Entre las ventajas que se pueden hallar en este dispositivo es que el lector/escritor de tarjetas RFID es un diseño realizado en PCB de dos capas que lo hace más sencillo y económico que el diseño del lector/escritor OpenPCD, y al igual que este último es compatible con la biblioteca open source conocida como librfid. Por su lado, en el lector de tarjetas de contacto se debe destacar su simplicidad, ya que no cuenta con ningún tipo de hardware específico (ASIC) que cumpla con el estandar ISO7816, sólo es necesario tener disponible un puerto serial (UART) y un par de puertos de entrada/salida de propósito general para lo que tiene que ver con el manejo del oscilador y el reset para la tarjeta de contacto. Esto lo hace portable a cualquier SBC que cuente con los puertos detallados anteriormente.

9.3. Enseñanzas y aprendizajes

Los conceptos aprendidos sobre reglas de diseño de lectores/escritores de tarjetas sin contacto compatibles con la norma ISO14443, son muy importantes para llevar a cabo mejoras y modificaciones futuras. Las dificultades afrontadas durante la fase inicial, generadas por errores en las ecuaciones suministradas por una de las notas de aplicación usada y la falta de soporte técnico por parte del fabricante (por no ser quien consultó una empresa), permitió ver la necesidad del uso de instrumentos de medición adecuados para

validar el diseño y comparar con los resultados teóricos. También dejó como enseñanza el hecho de ver con ojo más crítico y no creer tanto en las publicaciones de otros a la hora de diseñar.

El manejo de herramientas de diseño CAD, fue de mucha utilidad para luego llevar adelante las fabricación de las placas de circuito impreso, se tuvo que aprender a utilizar el programa gEDA ya que KiCad no permite hacer pistas curvas y eran necesarias para el diseño del lector/escritor RFID.

A nivel de software se dejó de ser un simple usuario del sistema operativo Linux, adquiriendo conocimientos para cross compilar su Kernel e incorporando modificaciones y parches que permitieran poner en funcionamiento el hardware que conforma el prototipo. Se debió entender el funcionamiento de buena parte de las bibliotecas usadas, para modificarlas y que fuera posible su uso en la aplicación final.

En cuanto al cliente, se entendió que tiene tiempos distintos y finalmente no se logró tenerlo como tal. Entre otras cosas, el contacto encargado de ser el cliente por parte de la IM no estaba dispuesto a oficiar como tal por entender que en un proyecto de fin de carrera no podía exigir nada porque es algo didáctico. Se tuvo más bien el respaldo en cuanto a los préstamos de tarjetas RFID y SAM. Se aprendió que los mecanismos de compra en la IM son muy complejos, no se puede hacer un proyecto apoyado por la IM durante un cambio de dirección porque las prioridades cambian y puede no ser bueno. Además se aprendió que las compras se deben realizar en el momento necesario, y no cuando se encuentren los medios disponibles, puesto que pueden no encontrarse nunca.

9.4. Mejoras y trabajos a futuro

Entre las cosas que se deben investigar es que el blindaje incluido en la primer antena fabricada cumpla con las regulaciones de compatibilidad electromagnética, EMC, definidas por la EN de Europa o por la FCC de Estados Unidos, a tales efectos sería necesario contar con el equipamiento adecuado para efectuar las mediciones necesarias. Tal vez la disminución del tamaño del PCB de la antena o la incorporación de ésta al resto del hardware puedan ser vistas como mejoras desde el punto de vista de disminución de costos. En cuanto al lector de tarjetas de contacto su incorporación a la lista de lectores soportados por la biblioteca pcsclite quedó inconclusa; si bien el hardwa-

re está operativo y es posible enviar comandos APDU al módulo SAM de seguridad a través de una pequeña aplicación, sería importante alcanzar el objetivo planteado al comenzar el proyecto. Un aspecto importante a destacar es que no se detallan los comandos APDU empleados, ya que por motivos de seguridad los mismos no pueden ser revelados.

La comunicación por red a través de un modem 3G con un servidor central tampoco fue alcanzada. La alternativa encontrada fue tener en forma local un archivo que contenga los saldos pendientes para acreditar en las tarjetas de los usuarios. Una vez asignado el saldo a la tarjeta correspondiente se elimina de la lista para que no se re asigne saldo ya acreditado.

Algo que no se tiene en cuenta muchas veces es qué tan simple puede resultar el armado de las partes del dispositivo, en este caso se pensó en un método simple y rápido donde no se emplearan cables para los conexionados. El ensamblado entre sí de las placas de hardware que conforman el prototipo es una tarea simple que puede ser hecha por un niño, como si armara su mecano. Las distintas partes encajan una encima de otra y son aseguradas a través de un separador con tuerca, no existiendo posibilidad que sean conectadas al revés. En cuanto a diseño industrial faltaría diseñar y fabricar una carcasa acorde, que permita el anclaje rápido y seguro de las distintas partes de hardware, sin que el material del cual esté hecha interfiera con la propagación de radio frecuencia, esto descarta la posibilidad de usar metal como elemento a emplear.

Parte VI

Anexos

Apéndice A

Tarjetas inteligentes (Smart Cards)

Una tarjeta inteligente (smart card), o tarjeta con circuito integrado (ICC, de su sigla en inglés), es cualquier tarjeta del tamaño de un bolsillo con circuitos integrados que permiten la ejecución de cierta lógica programada. Aunque existe un diverso rango de aplicaciones, hay dos categorías principales de ICC. Las tarjetas de memoria contienen sólo componentes de memoria no volátil y posiblemente alguna lógica de seguridad. Las tarjetas microprocesadoras contienen memoria y microprocesador. La percepción estándar de una smart card es una tarjeta microprocesadora de las dimensiones de una tarjeta de crédito (o más pequeña, como por ejemplo, tarjetas SIM para GSM) con varias propiedades especiales (ej. un procesador criptográfico seguro, sistema de archivos seguro, características legibles por humanos) y es capaz de proveer servicios de seguridad (ej. confidencialidad de la información en la memoria). Las tarjetas no contienen baterías; la energía es suministrada por los lectores de tarjetas.

A.1. Clasificaciones

A.1.1. Tipos de tarjetas según su capacidad

Según las capacidades de su chip, las tarjetas más habituales son:

- Memoria: tarjetas que únicamente son un contenedor de ficheros pero que no albergan aplicaciones ejecutables. Por ejemplo, MIFARE. Éstas se usan general-

mente en aplicaciones de identificación y control de acceso sin altos requisitos de seguridad.

- Microprocesadas: tarjetas con una estructura análoga a la de una computadora (procesador, memoria volátil, memoria persistente). Éstas albergan ficheros y aplicaciones y suelen usarse para identificación y pago con monederos electrónicos.

- Criptográficas: tarjetas microprocesadas avanzadas en las que hay módulos hardware para la ejecución de algoritmos usados en cifrados y firmas digitales. En estas tarjetas se puede almacenar de forma segura un certificado digital (y su clave privada) y firmar documentos o autenticarse con la tarjeta sin que el certificado salga de la misma, ya que es el procesador de la propia tarjeta el que realiza la firma.

A.1.2. Tipos de tarjetas según la estructura de su sistema operativo

- Tarjetas de memoria. Tarjetas que únicamente son un contenedor de datos pero que no albergan aplicaciones ejecutables. Disponen de un sistema operativo limitado con una serie de comandos básicos de lectura y escritura de las distintas secciones de memoria y pueden tener capacidades de seguridad para proteger el acceso a determinadas zonas de memoria.

- Basadas en sistemas de ficheros, aplicaciones y comandos. Estas tarjetas disponen del equivalente a un sistema de ficheros compatible con el estándar ISO/IEC 7816 parte 4 y un sistema operativo en el que se incrustan una o más aplicaciones (durante el proceso de fabricación) que exponen una serie de comandos que se pueden invocar a través de API de programación.

- Java Card. Tarjeta capaz de ejecutar mini-aplicaciones Java. En este tipo de tarjetas el sistema operativo es una pequeña máquina virtual Java (JVM) y en ellas se pueden cargar dinámicamente aplicaciones desarrolladas específicamente para este entorno.

A.1.3. Tipos de tarjetas según el formato (tamaño)

En el estándar ISO/IEC 7816 parte 1 se definen los siguientes tamaños para tarjetas inteligentes:

- ID 000: el de las tarjetas SIM usadas para teléfonos móviles GSM. También acostumbran a tener este formato las tarjetas SAM (Security Access Module) utilizadas para la autenticación criptográfica mutua de tarjeta y terminal.
- ID 00: un tamaño intermedio poco utilizado comercialmente.
- ID 1: el más habitual, tamaño tarjeta de crédito.
- ID 1/000: permite remover la tarjeta ID 000 desde la tarjeta ID 1 sin herramientas de corte.

A.1.4. Tipos de tarjetas según la interfaz

Tarjeta inteligente de contacto

Estas tarjetas disponen de contactos metálicos visibles y debidamente estandarizados (parte 2 de la ISO/IEC 7816). Estas tarjetas, por tanto, deben ser insertadas en una ranura de un lector para poder operar con ellas. A través de estos contactos el lector alimenta eléctricamente a la tarjeta y transmite los datos oportunos para operar con ella conforme al estándar.

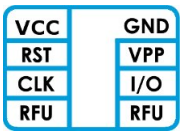

Figura A.1: Tarjeta de contacto

La serie de estándares ISO/IEC 7816 e ISO/IEC 7810 definen:

- La forma física (parte 1).
- La posición de las formas de los conectores eléctricos (parte 2).

- Las características eléctricas (parte 3).
- Los protocolos de comunicación (parte 3).
- El formato de los comandos (ADPU's) enviados a la tarjeta y las respuestas retornadas por la misma (parte 3).
- La dureza de la tarjeta.
- La funcionalidad.

Tarjetas Inteligentes sin Contacto

El segundo tipo es la tarjeta inteligente sin contacto, RFID, en el cual el chip se comunica con el lector de tarjetas mediante acoplamiento magnético a una tasa de transferencia de 106 a 848 Kbits/s. El estándar de comunicación de tarjetas inteligentes sin contacto es el ISO/IEC 14443. Define dos tipos de tarjetas sin contacto (A y B), permitidos para distancias de comunicación de hasta 10cm. Las más abundantes son las tarjetas de la familia MIFARE de Philips, las cuales representan a la ISO/IEC 14443-A. Las tarjetas inteligentes sin contacto son una evolución de la tecnología usada desde hace años por los RFID (identificación por radio frecuencia), añadiéndoles dispositivos que los chip RFID no suelen incluir, como memoria de escritura o microcontroladores.

Tarjetas híbridas y duales

Una tarjeta híbrida es una tarjeta sin contacto (contactless) a la cual se le agrega un segundo chip de contacto. Ambos chips pueden ser chips microprocesadores o simples chips de memoria. El chip sin contacto es generalmente usado en aplicaciones que requieren transacciones rápidas. Por ejemplo el transporte, mientras que el chip de contacto es generalmente utilizado en aplicaciones que requieren de alta seguridad como las bancarias.

Seguridad

La seguridad es una de las propiedades más importantes de las tarjetas inteligentes y se aplica a múltiples niveles y con distintos mecanismos. Cada fichero lleva asociadas unas condiciones de acceso y deben ser satisfechas antes de ejecutar un comando sobre ese fichero.

En el momento de personalización de la tarjeta (durante su fabricación) se puede indicar que mecanismos de seguridad se aplican a los ficheros. Normalmente se definirán:

- Ficheros de acceso libre.
- Ficheros protegidos por claves: Pueden definirse varias claves con distintos propósitos. Normalmente se definen claves para proteger la escritura de algunos ficheros y claves específicas para los comandos de consumo y carga de las aplicaciones de monedero electrónico. De ese modo la aplicación que intente ejecutar comandos sobre ficheros protegidos tendrá que negociar previamente con la tarjeta la clave oportuna.
- Ficheros protegidos por PIN: El PIN es un número secreto que va almacenado en un fichero protegido y que es solicitado al usuario para acceder a este tipo de ficheros protegidos. Cuando el usuario lo introduce y el programa se lo pasa a la operación que va a abrir el fichero en cuestión, el sistema valida que el PIN sea correcto para dar acceso al fichero.

La negociación de claves se realiza habitualmente apoyándose en un Módulo SAM, que no deja de ser otra tarjeta inteligente en formato ID-000 alojada en un lector interno propio dentro de la carcasa del lector principal o del TPV(Terminal de Punto de Venta) y que contiene aplicaciones criptográficas que permiten negociar las claves oportunas con la tarjeta inteligente del usuario. Operando de este modo se está autenticando el lector, la tarjeta y el módulo SAM involucrados en cada operación.

Programación de aplicaciones para los sistemas en los que se utiliza la tarjeta

Existen varias API de programación estandarizadas para comunicarse con los lectores de tarjetas inteligentes desde un computador. Las principales son:

- PC/SC (Personal Computer/Smart Card). El proyecto MUSCLE proporciona una implementación casi completa de esta especificación para los sistemas operativos GNU Linux-UNIX.

- OCF (OpenCard Framework), especificado por el grupo de empresas OpenCard. Este entorno intenta proporcionar un diseño orientado a objetos fácilmente extensible y modular. El consorcio OpenCard publica el API y proporciona una implementación de referencia en Java. Existe un adaptador para que OCF trabaje sobre PC/SC.

En ambos casos, el modelo de programación que utilizan las tarjetas inteligentes está basado en protocolos de petición-respuesta. La tarjeta (su software) expone una serie de comandos que pueden ser invocados. Estos comandos interactúan con los ficheros que subyacen a cada aplicación de la tarjeta y proporcionan un resultado. Desde el terminal se invocan estos comandos a través de cualquiera de las API antes descritas componiendo un APDU (Application Protocol Data Unit - comandos con parámetros) que son enviados a la tarjeta para que ésta responda.

A.2. ISO 14443

ISO 14443 es un estándar internacional relacionado con las tarjetas de identificación electrónicas, en especial las smart cards, gestionado conjuntamente por la Organización Internacional de Normalización (ISO) y la Comisión Electrotécnica Internacional (IEC). Este estándar define una tarjeta de proximidad utilizada para identificación y pagos que por lo general utiliza el formato de tarjeta de crédito definida por ISO 7816 - ID 1 (aunque otros formatos son posibles). El sistema RFID utiliza un lector con un microcontrolador o ASIC y una antena que opera a 13,56MHz (frecuencia RFID). El lector mantiene a su alrededor un campo electromagnético de modo que al acercarse una tarjeta al campo, ésta se alimenta eléctricamente de esta energía inducida y puede establecerse la comunicación lector-tarjeta.

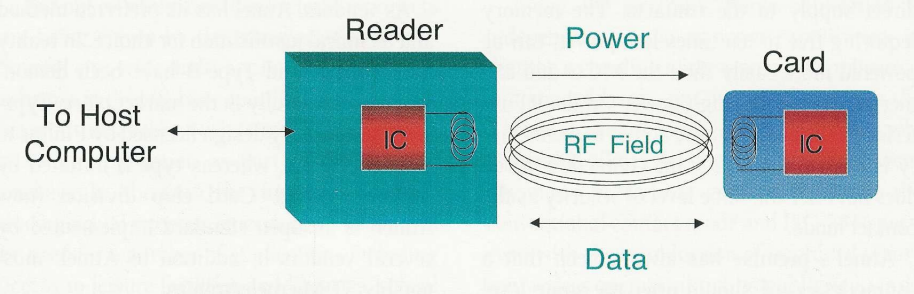

Figura A.2: Acoplamiento entre lector y tarjeta

El estándar ISO 14443 consta de cuatro partes y se describen dos tipos de tarjetas: tipo A y tipo B. Las principales diferencias entre estos tipos está en los métodos de modulación, codificación de los planes (parte2) y el protocolo de inicialización de los procedimientos (parte3). Las tarjetas de ambos tipos (A y B) utilizan el mismo protocolo de alto nivel (llamado T=CL) que se describe en la parte4. El protocolo T=CL especifica los bloques de datos y los mecanismos de intercambio:

1. Bloque de datos de encadenamiento.
2. Tiempo de espera de extensión.
3. Múltiple activación.

Las tarjetas Mifare cumplen con las partes 1, 2 y 3 de tipo A de la especificación ISO/IEC 14443.

A.3. Mifare

Mifare es la tecnología de smart card sin contacto más ampliamente usada en el mundo. Es equivalente a las 3 primeras partes de la norma ISO 14443 Tipo A. La distancia típica de lectura es de hasta 10 cm, depende de la potencia del lector y factores del entorno, existiendo lectores de mayor y menor alcance. La tecnología Mifare es económica y rápida, razón por la cual es la más usada a nivel mundial hoy día.

A.3.1. Operación

Las tarjetas Mifare son tarjetas de memoria protegida. Están divididas en sectores que a su vez son subdivididos en bloques y poseen mecanismos de seguridad para el control de acceso. Su capacidad de cómputo no permite realizar operaciones criptográficas o de autenticación mutua de alto nivel, estando principalmente destinadas a monederos electrónicos simples, control de acceso, tarjetas de identidad corporativas, tarjetas de transporte urbano o para ticketing. Cada sector se divide en cuatro bloques, de los cuales tres pueden contener información del usuario, y el cuarto, llamado trailer, contiene elementos de seguridad. La información es almacenada sin un formato pre establecido, y se puede modificar con comandos simples de lectura y escritura. Mifare provee un formato especial llamado "bloque de valor"(value block); los bloques que tienen información guardada en este formato se comportan de una forma diferente, incluyendo operaciones de incremento y descuento. Los sectores utilizan dos claves de acceso llamadas Á'y 'B'. Estas claves se almacenan en el cuarto bloque junto con los permisos de acceso a cada uno de los tres bloques que son parte del mismo sector. Estos permisos pueden ser: lectura, escritura, descuento o incremento (para bloques de valor).

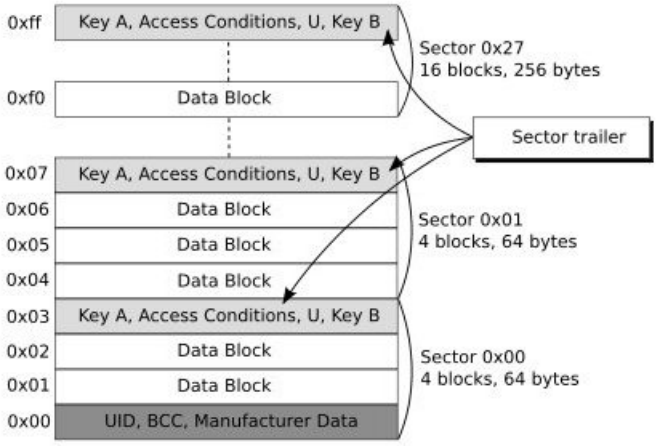

Figura A.3: Mifare Classic de 4K

Una vez que se acerca la tarjeta a un lector, ésta se activa e inicia un proceso de intercambio con el lector para establecer una comunicación cifrada. Este proceso es igual con todas las tarjetas y está diseñado para proveer protección del canal(evitando que se

espíe), y no para autenticar la tarjeta o el lector. Previo a establecer un canal cifrado la tarjeta envía un código de identificación, mediante el algoritmo de anticolisión, que usualmente es el número de serie de la tarjeta, aunque la norma ISO 14443 indica que este número puede ser aleatorio. Con este número de identificación el lector está en condiciones de realizar cualquier operación en la tarjeta, previa autenticación con las claves de acceso en los respectivos sectores. Se debe destacar que un sistema con claves diversificadas facilita el fortalecimiento de la seguridad, apoyada por una base de datos que pueda monitorear los aumentos de los saldos y demás estrategias operativas y finalmente la autenticación remota por SAM, no todos los sistemas poseen esto.

Variantes

- Mifare Classic. Son fundamentalmente de los dispositivos de almacenamiento de memoria. Existen tarjetas de 1Kb y de 4Kb. La Mifare Standard de 1KB ofrece unos 768 bytes de almacenamiento de datos, dividida en 16 sectores. La Mifare Standard de 4k ofrece 3 KB dividido en 64 sectores.

- Mifare Ultralight. Es semejante a la classic, pero sólo tiene 512 bits de memoria (es decir 64 octetos), sin seguridad. Esta tarjeta es muy barata así que se utiliza a menudo de forma desechable.

- Mifare T = CL. Bajo esta denominación se encuadran las tarjetas Mifare ProX y SmartMX. Son tarjetas con microprocesador que incorporan un sistema operativo de tarjeta (Card Operating System - COS) y aplicaciones desarrolladas específicamente para ser ejecutadas en la tarjeta. Estas tarjetas son capaces de ejecutar operaciones complejas de forma rápida y segura, igual que las tarjetas con contactos ISO 7816.

- Mifare DESFire. Esta tarjeta es una versión especial de Philips SmartMX. Se vende con un software de propósito general incorporado (el sistema operativo DESFire), que ofrece más o menos las mismas funciones que Mifare Standard (4kB de almacenamiento de datos dividido en 16 bloques), pero con una mayor flexibilidad, una mayor seguridad (triple DES), y con mayor rapidez (protocolo T=CL).

- Mifare DESFire EV1. Es la primera evolución de Mifare DESFire, compatible con la versión anterior, pero aún más segura, alcanzando la certificación EAL 4.

Apéndice B

Lector/Escritor RFID

B.1. Reglas y Parámetros de Diseño de una Antena RF

Pasos para el diseño de la antena RF:

A) Diseñar el inductor, medir su inductancia L y resistencia R (o factor de calidad Q).

B) Calcular los capacitores para el circuito resonante, que forman junto con el inductor.

C) Sintonizar el circuito resonante junto con el filtro pasa bajo a la impedancia requerida.

D) Conectar el circuito resonante a la salida del integrado(TX1 y TX2), verificar la corriente ITVDD y si es necesario sintonizar los componentes para un desempeño óptimo.

E) Verificar y ajustar el factor de calidad Q.

F) Verificar y ajustar el circuito receptor.

B.1.1. Diseño del inductor

Se recomienda usar antenas cuyo inductor tenga forma circular o cuadrado. El valor exacto del inductor es difícil de calcular pero puede ser aproximado por la siguiente

ecuación:

$$L_1[nH] = 2 \cdot l_1 \cdot (ln(\frac{l_1}{D_1}) - K) \cdot N_1^{1,8}$$

donde:

l_1 ..... Longitud de una vuelta del conductor (en cm).

D_1 ..... Diámetro del conductor o ancho del conductor del PCB.

K ..... Factor de forma ($K = 1,07$ antena circular y $K = 1,47$ antena cuadrada).

N_1 ..... Número de vueltas.

La antena debe ser simétrica, donde el punto central puede estar conectado a GND. Si esto es así, se sugiere mantener este punto lo más cercano posible al conector de la antena. El radio de la antena deberá ser $r > 5cm$ para una sola vuelta de cada uno de los inductores simétricos L_a y L_b, o $r < 5cm$ para dos vueltas. El blindaje de campo eléctrico debe ser conectado a GND.

Los valores de inductancia y resistencia del inductor, si se siguen las reglas de diseño, se encuentran entre:

$L = 300nH...2\mu H$

$R_{coil} = 0,5\Omega...5\Omega$

Resistor externo

En serie con el inductor se agrega un resistor externo. El valor del mismo se encuentra mediante las ecuaciones:

$$Q = \frac{wL}{R_{coil}} \rightarrow R_{coil} = \frac{wL}{Q}$$

y $R = 2 \cdot R_s + R_{coil} = R_{Sa} + R_{Sb} + R_{coil}$, donde R es la resistencia total, R_{Sa} y R_{Sb} son cada uno de lo

Definiendo Q entre 20 y 30, se pueden hallar los resistores externos:

$R_{Sa} = R_{Sb} = \frac{1}{2} \cdot (R - R_{coil}) = \frac{wL}{2Q} - \frac{R_{coil}}{2}$ con $w = 2\pi \cdot 13,56MHz$

Dejando de lado la influencia de todos los otros componentes en el factor Q, este cálculo sólo da una estimación del valor de R_S, pero esta estimación es necesaria para el cálculo de los capacitores del circuito de resonancia.

En muchas aplicaciones prácticas es posible observar que se prescinde del valor de resistencia externa R_S, teniendo en cuenta sólo el valor de resistencia del inductor R_{coil}.

B.1.2. Capacitores del circuito resonante

Gracias a la simertría del circuito es posible simplificar los calculos operando sólo con la mitad del circuito, por tanto los valores del inductor L, la resistencia R (incluyendo el resistor externo) y el valor de impedancia requerida de la antena Z_{ant}, empleados en los siguientes cálculos son la mitad del valor correspondiente a la totalidad del circuito. Aplicando entonces la suma de impedancias a la mitad del cirucito y sabiendo que el resultado tiene que ser real e igual a Z_{ant}, es posible hallar los valores del capacitor paralelo C_2 y el capacitor serie C_1 mediante las siguientes igualdades:

$$C_{2a} = C_{2b} = \frac{L}{\omega^2 L^2 + R^2} - \frac{R}{(\omega^2 L^2 + R^2)\omega \sqrt{\frac{Z_a}{(\frac{\omega^2 L^2 + R^2}{R} - Z_a)}}}$$

$$C_{1a} = C_{1b} = \frac{1}{\omega \sqrt{Z_a \cdot (\frac{\omega^2 L^2 + R^2}{R} - Z_a)}}$$

$$Z_{ant} = 250\Omega$$

El valor de $Z = 2 \cdot Z_{ant} = 500\Omega$ podría ser incrementado hasta $Z = 2 \cdot Z_{ant} = 800\Omega$ para incrementar la potencia de salida, pero el límite de corriente de salida desde el integrado no debe ser excedido.

B.1.3. Sintonizar el circuito resonante

El circuito todo (incluyendo el filtro pasa bajos) tiene que ser adaptado a una impedancia de aproximadamente 40 Ω entre TX1 y TX2 (500 Ω si no tenemos en cuenta el filtro). Donde los valores propuestos para los componentes del filtro son:

$L_0 = 1\mu H$ (e.g. TDK NL322522T-1R0J)
$C_{01} = 68pF$ (Ceramic NP0, tolerance $\leq \pm 2\,\%$)
$C_{02} = 56pF$ (Ceramic NP0, tolerance $\leq \pm 2\,\%$)

Con estos valores, la frecuencia de resonancia del filtro se encuentra centrada en $14,4MHz (13,56MHz + 847,5KHz)$. Esto mejora la performance en dos formas:

a) Incrementa la relación señal a ruido de la señal recibida.

b) Decrementa el sobretiro de los pulsos transmitidos, mejorando la calidad de la señal transmitida.

El procedimiento para sintonizar el circuito, es el siguiente:
Los materiales necesarios son:

- Generador de señales ($13,56MHz$).
- Osciloscopio con puntas de prueba.
- Resistor de referencia (40Ω).

Se debe conectar las puntas del osciloscopio a la salida del generador y en paralelo un resistor de referencia de 40Ω (500Ω en caso de sintonizar el circuito sin conectar el filtro pasa bajos).

Calibración

Se genera una señal sinusoidal de frecuencia $13,56MHz$ y de amplitud entre $2V$ y $5V$. El osciloscopio se configura para observar las figuras de Lissajous, con la escala del eje X dos veces la del eje Y. Se calibra el capacitor, C_{cal}, de la punta de prueba del osciloscopio hasta que la figura de Lissajous sea un segmento de recta, inclinado 45^{o}.

Sintonizado

Luego de la calibración se sustituye el resistor de referencia por la antena y se sintoniza la misma variando los capacitores C_1 y C_2, hasta que se obtenga una figura como la obtenida en el caso anterior. En ese momento la antena se encuentra sintonizada.

En caso de contar con un analizador de redes, el método anterior puede ser evitado, ya que es posible sintonizar el circuito buscando que la impedancia en el diagrama de Smith se ubique sobre el eje real al alcanzar la frecuencia de trabajo, en este caso $13,56Mhz$.

B.1.4. Valor de ITVDD

El integrado de la familia Micore, entrega a la salida una señal cuadrada, con valor de pico a pico $U_{TxAC} = 2,5V_{pp}$ centrada en el valor de continua $U_{TxDC} = 2,5V$, con una frecuencia $f_0 = 13,56MHz$ y un máximo de salida de corriente:

$$I_{TVDD} \leq 150mA$$

Esto significa que la salida TX oscila entre $0V$ y $5V$. TX1 y TX2 usualmente están desfasados 180^{o}, dependiendo de la configuración del bit 3 (TX2Inv) del registro Tx-Control (ver hoja de datos del integrado RC632).

B.1.5. Factor de calidad Q

El factor de calidad Q está directamente asociado con la forma de los pulsos modulados, ésto puede ser usado para verificar el valor del factor.

Un osciloscopio con al menos $50MHz$ de ancho de banda puede ser usado para observar la forma de los pulsos; donde los canales son conectados de la siguiente forma:

CH1: La punta de prueba conectada en este canal forma un loop con su línea de tierra para generar el acoplamiento necesario al estar próxima a la antena.

CH2: Este canal es conectado a la salida del pin 4 (MFout) de integrado y es usado como canal de disparo.

El registro MFoutSelect (26h) es configurado con los valores:

"2" para que la señal sea modulada con código Miller.

"3" para flujo de datos serial (sin código Miller).

Ver hoja de datos del integrado RC632 por más detalles.

Es recomendado verificar que la forma de los pulsos cumpla con lo establecido en la norma ISO14443. La figura debajo muestra como son estos pulsos. Para garantizar que la antena se encuentre bien sintonizada y el factor Q sea el correcto, debe verificarse que:

i. La señal caiga debajo del $5\,\%$ de su valor máximo (sin tener en cuenta el sobretiro).

ii. El tiempo $t2$ debe estar limitado entre: $0,7\mu s < t2 < 1,4\mu s$.

Si $t2 < 0,7\mu s$, el factor Q es muy alto (mayor que 35), por lo tanto la resistencia externa R_{ext} debe ser incrementada.

Si $t2 > 1,4\mu s$ el factor Q es muy bajo, la distancia de operación no será cumplida y por lo tanto R_{ext} debe ser decrementada.

La tabla siguiente muestra la duración de los pulsos en μs de acuerdo a la norma ISO 14443.

Pulses length	**t1**	**t2 min**	**t3 max**	**t4 max**
T1 MAX	3.0	0.7	1.0	0.4
T1 MIN	2.0	0.7	1.0	0.4

Cuadro B.1: Duración de los pulsos en μs - ISO 14443

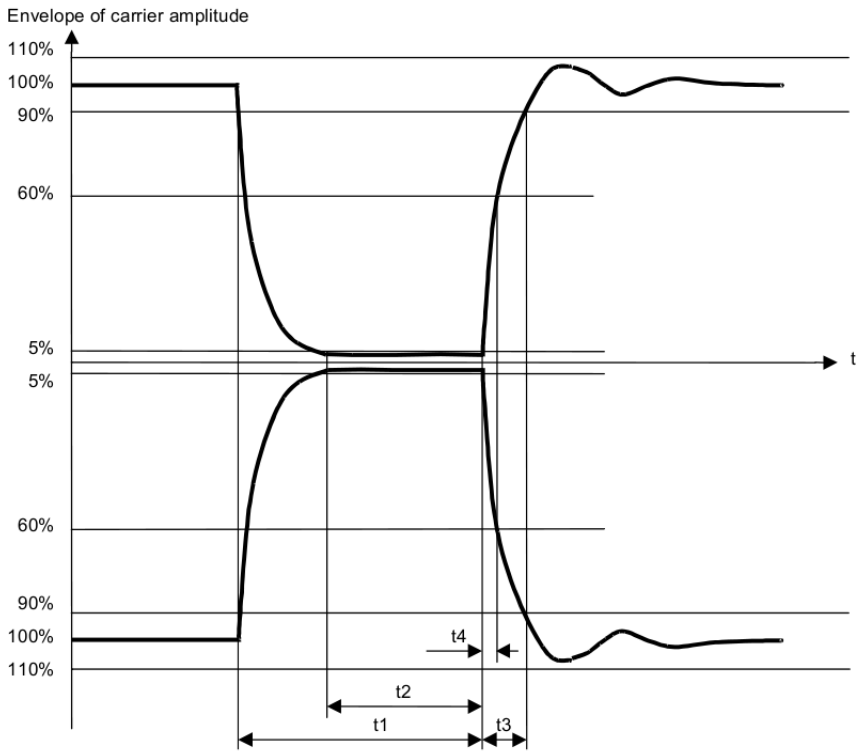

Figura B.1: Forma de pulso acorde a la norma ISO 14443

B.1.6. Circuito receptor

Cuando ya se han tomado en cuenta todos los cuidados en el diseño del transmisor, el circuito receptor debe ser conectado y ajustado. Los valores de los componentes sugeridos en el circuito receptor son los siguientes:

$C_3 = 1nF$ (Ceramic NP0, tolerance $\leq \pm 10\,\%$)
$C_4 = 100nF$ (Ceramic X7R, tolerance $\leq \pm 10\,\%$)
$R_1 = 470\Omega ... 4{,}7k\Omega$
$R_2 = 820\Omega$

Dos reglas deben ser tenidas en cuenta para este circuito:

i. El nivel de tensión de continua, DC, en la entrada Rx tiene que ser mantenido a V_{mid} (por eso es necesario R_2 y C_4, ver figura debajo).

ii. El nivel de tensión de alterna, AC, en la entrada Rx debe ser mantenido entre los siguientes límites: $1,5V_{pp} < V_{Rx} < 3,0V_{pp}$.

Si $V_{Rx} > 3,0V_{pp}$, R_1 debe ser incrementada.

Si $V_{Rx} < 1,5V_{pp}$, R_1 debe ser decrementada.

El voltaje a la entrada Rx debe ser verificado con y sin presencia de una tarjeta entre los límites máximo y mínimo de distancia de operación.

El valor límite $V_{Rx} = 3,0V_{pp}$ no debe ser excedido, un valor mayor puede causar fallos en la recepción.

Otros puntos a tener en cuenta

PCB

La parte más crítica de todo el circuito analógico es el directamente conectado al integrado, o sea el filtro pasa bajos y la conexión de TVDD a la fuente de alimentación. Entonces, por un lado un filtro puede ser usado para la conexión a la fuente de alimentación. Por otro lado el diseño del filtro a la salida del integrado, formado por L_0 y C_0,

debe ser considerado con mucho cuidado. *El área y la distancia del filtro al integrado deben ser mantenidas lo más pequeñas posibles. Es recomendado además un plano de tierra.*

La figura siguiente muestra un esquemático del diseño de una antena:

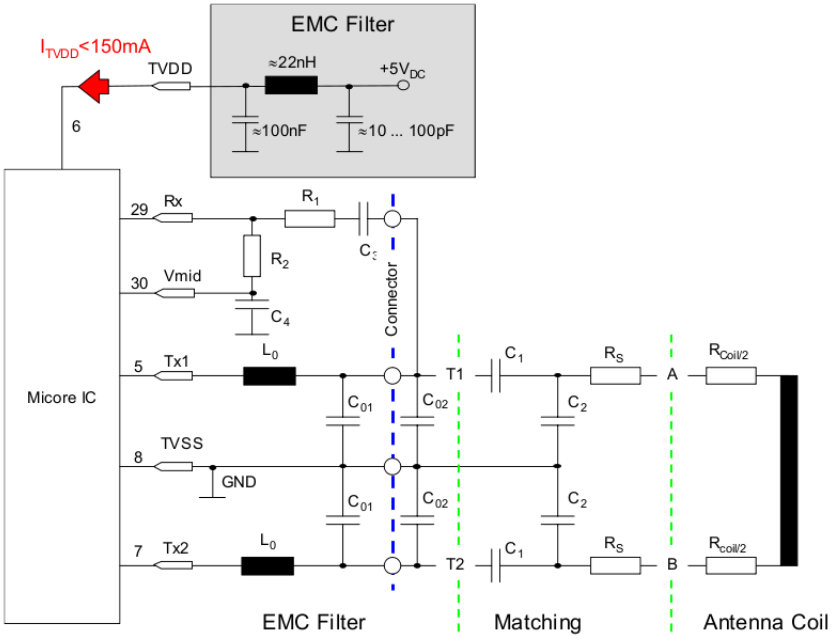

Figura B.2: Esquema de una antena, identificando sus principales secciones

Filtro de entrada de alimentación

Aunque no sería necesario, un filtro puede ser conectado a la entrada TVDD para mejorar los siguientes puntos:

a) suprimir ruido llegado desde la fuente de alimentación.

b) suprimir armónicos provenientes desde el transmisor.

Filtros idénticos pueden ser ubicados en las entradas AVDD y DVDD.

Blindaje

El blindaje eléctrico absorbe el campo eléctrico generado por la antena. Para construir un blindaje, es recomendable usar un PCB de al menos 4 capas, donde el loop del blindaje se encuentra en las 2 capas externas. Este loop no debe ser cerrado y debe estar conectado en su punto central al sistema de tierra mediante una vía. Los extremos de la bobina deben ser ruteados próximos entre sí para evitar inductancias adicionales. La figura siguiente da una idea de como debe ser un blindaje:

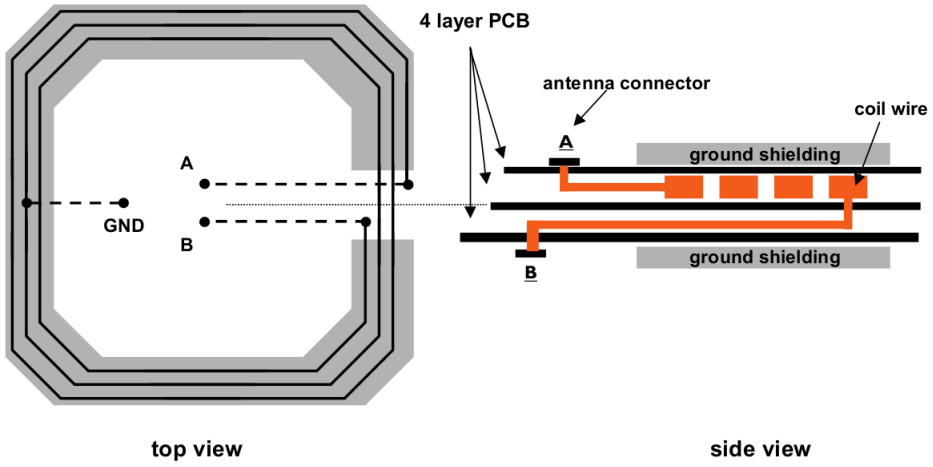

Figura B.3: Blindaje de una antena en un diseño de 4 capas

Apéndice C

Software

C.1. Gestor de paquetes opkg

Se puede configurar el gestor de paquetes de dos maneras para que encuentre los repositorios desde donde descargar los paquetes para su instalación.

La primera de ellas es la más prolija y consiste en los siguientes pasos:

Se crean los archivos xxx-feed.conf (xxx hace referencia a los subdirectorios dentro del repositorio) que contienen la ruta a los repositorios:

$ echo src/gz angstrom-feed http://www.angstrom-distribution.org/feeds/unstable/ipk/glibc/armv7a/base > /etc/opkg/angstrom-feed.conf

$ echo src/gz perl-feed http://www.angstrom-distribution.org/feeds/unstable/ipk/glibc/armv7a/perl/ > /etc/opkg/perl-feed.conf

$ echo src/gz sdk-feed http://www.angstrom-distribution.org/feeds/unstable/ipk/glibc/sdk/ > /etc/opkg/sdk-feed.conf

$ echo src/gz python-feed http://www.angstrom-distribution.org/feeds/unstable/ipk/glibc/armv7a/python/ > /etc/opkg/pyton-feed.conf

$ echo src/gz debug-feed http://www.angstrom-distribution.org/feeds/unstable/ipk/glibc/armv7a/debug/ > /etc/opkg/debug-feed.conf

$ echo src/gz beagleboard-feed http://www.angstrom-distribution.org/feeds/unstable/ipk/glibc/armv7a/machine/beagleboard/ > /etc/opkg/beagleboard-feed.conf

$ echo src/gz noarch-feed http://www.angstrom-distribution.org/feeds/unstable/ipk/glibc/all/ > /etc/opkg/noarch-feed.conf

Nota: /gz indica que los paquetes están comprimidos con extensión gz.

Se debe actualizar la lista de paquetes disponibles:

$ opkg update

El comando anterior crea los archivos /var/lib/opkg/xxx-feed que contienen la lista completa de paquetes en el repositorio.

Nota: Se puede usar el comando opkg list | wc -l para conocer la lista de paquetes.

La segunda opción es modificar directamente el archivo /etc/opkg/opkg.conf a continuación de la línea:

src <src-name> <source-url>

Se agrega lo siguiente:

```
src angstrom-feed http://www.angstrom-distribution.org/
feeds/unstable/ipk/glibc/armv7a/base/

src perl-feed http://www.angstrom-distribution.org/feeds/
unstable/ipk/glibc/armv7a/perl/

src sdk-feed http://www.angstrom-distribution.org/feeds/
unstable/ipk/glibc/sdk/

src python-feed http://www.angstrom-distribution.org/feeds/
unstable/ipk/glibc/armv7a/python/

src debug-feed http://www.angstrom-distribution.org/feeds/
unstable/ipk/glibc/armv7a/debug/

src beagleboard-feed http://www.angstrom-distribution.org/
feeds/unstable/ipk/glibc/armv7a/machine/beagleboard/ src
noarch-feed http://www.angstrom-distribution.org/feeds/
unstable/ipk/glibc/all/
```

Luego se actualiza la lista de paquetes:

$ opkg update

Una vez que se agregan los repositorios es posible instalar paquetes mediante el comando:

$ opkg install <nombre del paquete>

El nombre del paquete puede buscarse en el sitio web que fue pasado como dirección del repositorio en la configuración anterior.

C.2. Instalación, configuración y uso de SDK

Como fue comentado antes, el SDK es generado a partir de la herramienta web Narcissus. Para la instalación se debe tener el archivo (SDK.tar.bz2, por ejemplo) generado.

En el PC de desarrollo se debe hacer lo que sigue:

Se realiza la instalación con el usuario root y luego se explica como proceder con cualquier otro usuario.

$ sudo su

Se descomprime el SDK en /

$ sudo tar -xf SDK.tar.bz2 -C /

Se debe haber copiado el directorio angstrom en /usr/local

Luego, se habilita el uso del gestor de paquetes (opkg) y se agraga en el croscompilador al PATH del sistema. Se debe establecer esta configuración con cada usuario que quiera utilizar la herramienta.

$. /usr/local/angstrom/arm/environment-setup

Lo que se tiene es un sistema Angström configurado en el PC de desarrollo con el que se puede actualizar la lista de repositorios, bajar paquetes, compilarlos para la arquitectura deseada, etc.

Actualización de repositorios:

$ opkg-target update

Instalación de paquetes:

$ opkg-target install <paquete>

Actualización de paquetes:

$ opkg-target upgrade

El uso de opkg-target es solo permitido para el usuario root por un tema de permisos, aunque el uso del croscompilador es permitido para cualquier usuario siempre y cuando haga lo siguiente cada vez que vaya a utilizar la herramienta:

$. /usr/local/angstrom/arm/environment-setup

Para referirse a las herramientas de la arquitectura para la cual se generan los binarios se utiliza el prefijo arm-angstrom-linux-gnueabi-.

Por más detalles referirse a [46].

C.3. OpenEmbedded-Bitbake

Para la instalación, configuración y ejecución de esta herramienta, se deben seguir los pasos en detalle. Se debe tener una buena conexión a internet para la descarga de fuentes, espacio libre en disco duro de no menos de 10GB, buen procesador y paciencia ya que los desarrollos completos demoran varias horas.

Primero se instalan los paquetes previos necesarios.

Paquetes importantes:

$ sudo apt-get install sed wget cvs subversion git-core \
coreutils unzip texi2html texinfo docbook-utils \
gawk python-pysqlite2 diffstat help2man make gcc build-essential g++ \

desktop-file-utils chrpath

Paquetes secundarios (aceleran los procesos):

$ sudo apt-get install libxml2-utils xmlto python-psyco apr

Chequear que /bin/sh no tiene un enlace simbólico a “dash”:

$ ls -l /bin/sh

Debe estar enlazado a “bash”. Si no lo está:

$ sudo dpkg-reconfigure dash

Aquí seleccionamos NO instalar “dash” como /bin/sh

Compilador para aumentar la velocidad de bitbake:

$ sudo apt-get install python-psyco

Para versiones de Ubuntu mayores o iguales a la 10.04:

$ sudo su
$ echo 128 > /proc/sys/vm/mmap_min_addr
$ exit
$ sudo sysctl -w vm.mmap_min_addr=128

Instalación:

El directorio base seleccionado para el desarrollo con OpenEmbedded es stuff (puede ser otro), y se ubicó en /.

Se crea la estructura de directorios:

$ sudo mkdir -p /stuff/build/conf
$ cd /stuff/

Bitbake es la herramienta de construcción que utiliza OpenEmbedded. Está escrita en Python, por lo que no es necesario compilarlo para que funcione.

$ sudo wget http://download.berlios.de/bitbake/bitbake-1.10.2.tar.gz

Nota: esta es la última versión disponible al momento. Para saber cual es la última versión disponible, entrar a: http://download.berlios.de/bitbake/.

Para las nuevas versiones es necesario tener instalado Python 2.6 o posterior.

Luego se descomprime el tar.gz bajado:

```
$ sudo tar -xzf bitbake-1.10.2.tar.gz
$ ls
```

Se debe ver un directorio llamado en este caso bitbake-1.10.2 (se puede renombrar a: “bitbake” por comodidad).

Para obtener OpenEmbedded es necesario tener instalado git.

Ahora se hace lo siguiente:

```
$ cd /stuff
$ sudo git clone git://git.openembedded.org/openembedded
```

Nota: git clone es para hacer un checkout del repositorio.
Otro repositorio es http://repo.or.cz/r/openembedded.git

Demora mucho. Al finalizar se crea un directorio con el nombre openembedded.

Se recomienda actualizar OpenEmbedded una vez al día:

```
$ cd /stuff/openembedded
$ sudo git pull
```

Configuración:

Se modifica la configuración local, donde se debe indicar todo lo que se quiera crear.

```
$ cd /stuff/
$ sudo cp openembedded/conf/local.conf.sample build/conf/local.conf
$ sudo vi build/conf/local.conf
```

Nota: en lugar de vi, se pueden usar nano o incluso gedit. Estos son editores de texto ordenados por complejidad.

En este punto hay que tener cuidado debido a que existen muchas variables a editar y se necesita mucho conocimiento para poder cambiarlas.

La mínima cantidad de variables a editar para un desarrollo correcto son las siguientes:

BBFILES = "/stuff/openembedded/recipes/*/*.bb"
MACHINE = "beagleboard"
DISTRO = "angstrom-2008.1"
PARALLEL_MAKE = "-j 5"
INHERIT += "rm_work"

Nota: Los parámetros detallados antes ya existen en el archivo de configuración, algunos deben ser descomentados y/o editados ya que lo único que cambia es el valor asignado.

BBFILES: indica que archivos son considerados durante el desarrollo.

MACHINE: el nombre asociado a la SBC que se esté usando. Para saber el nombre asociado a la SBC se puede ver el contenido del directorio /stuff/openembedded/conf/machine.

DISTRO: qué versión de la distribución se quiere instalar. Aquí se eligió la última versión estable de Angström, aunque si se pone "angstrom-2010.x", se obtiene una versión más nueva (no asegura estabilidad). La versión no solo afecta a la distribución sino que la versión del kernel generado va a depender de esta versión. Por ejemplo, con 2008.1 se obtiene un kernel 2.6.32 y con 2010.x se obtiene un kernel 2.6.37. Para saber cuales son las versiones disponibles al momento se puede ver el contenido del directorio /stuff/openembedded/conf/distro.

PARALLEL_MAKE: indica cuantas operaciones simultáneas puede realizar el procesador de la pc de desarrollo. En general el valor se puede calcular como sigue: (cantidad de procesadores)x2 +1. En este caso 5 equivale a una cpu del tipo core2duo.

INHERIT += "rm_work": esta opción elimina los fuentes después de haber construido los paquetes. Esto hace que el tamaño del desarrollo en disco, no supere los 10GB. Si esta opción no se elige, se deben tener por lo menos 40GB de espacio libre en disco duro. También se puede decir que si se elige esta opción algunos fuentes deberán ser bajados nuevamente en cada desarrollo, lo que lo puede hacer más lento.

Conviene leer todo el archivo para tener una idea básica de lo que hace. Si en algún lugar se quiere hacer referencia al "home" del usuario, la ruta se debe escribir completa (no se puede ∼).

La última línea debe ser borrada (esto es para asegurarse de que se leyó todo).

Ejecución:

Siempre antes de empezar a desarrollar se debe ejecutar lo siguiente:

```
$ export BBPATH=/stuff/build:/stuff/openembedded
$ export PATH=/stuff/bitbake/bin:$PATH
```

Comenzando el desarrollo:

```
$ cd /stuff/build
$ bitbake console-image
```

Nota: Al ejecutarlo por primera vez, demora varias horas.

Si aparecen errores como "Please set persistent and cache" o "no se puede acceder al directorio tmp" esto se soluciona dando permisos al directorio stuff:

```
$ sudo su
$ chmod -R 777 /stuff
```

El comando bitbake console-image baja todos los fuentes y genera todos los binarios necesarios para la ejecución de la distribución Angström en modo consola. Otra opción, si solo queremos un kernel, es el comando bitbake virtual/kernel, pero para entender bien el funcionamiento del bitbake, la primera vez se recomienda bitbake console-image.

Luego de ejecutado el comando, se crea toda la estructura de directorios.

A continuación se detallan los más importantes:

/stuff/build/tmp/deploy/glibc/images/beagleboard/
Aquí se guardan los archivos generados.

/stuff/build/tmp/work/beagleboard-angstrom-linux-gnueabi/
Aquí se encuentran los directorios con los fuentes del kernel y el u-boot.

C.4. uImage

Se detallan las modificaciones en el archivo board_omap3beagle.c necesarios para la inicialización correcta de las interfaces SPI y GPIO. Como ya se comentó este archivo está ubicado en /stuff/build/tmp/work/beagleboard-angstrom-linux-gnueabi/linux-omap-.../git/arch/arm/mach-omap2/ y es el encargado de toda la inicialización de las interfaces del sistema.

En el caso de la interfaz SPI se vio que no se lograba un mapeo de la interfaz en /dev, lo que no permitía el acceso a ésta a nivel de usuario. En el caso de la interfaz GPIO se vio que para los pines GPIO no basta con los cambios realizados en el u-boot para establecer su dirección y valor al iniciar el sistema. A continuación se plantea una posible solución para ambos casos.

Cambios asociados al SPI:

Se creó una estructura (spi_board_info beagle_mcspi_board_info) que contempla todas las posibilidades de interfaz SPI en la Beagleboard y agrega información sobre éstas. En la estrucutra se pueden ver tres formas de representar al SPI: spi3.0, spi3.1 y spi4.0. El microprocesador de la Beagleboard tiene 4 interfaces SPI disponibles de las cuales la 3 y la 4 son accesibles desde el bloque de expansión de la Beagleboard. Además la interfaz spi3 se puede encontrar en dos modalidades 3.0 o 3.1 dependiendo de si se utiliza el CS0 o el CS1 como chip select de la interfaz. La interfaz spi4 solo puede utilizar el CS0. Es por esto que en la estructura se definen spi3.0, spi3.1 y spi4.0. Dentro de cada interfaz definida se agrega información sobre la interfaz, como ser el nombre (modalias), la máxima velocidad de transferencia/recepción de datos (max_speed_hz), número de bus (bus_num), chip select (chip_select) y modo del spi (SPI_MODE_1).

```
static struct spi_board_info beagle_mcspi_board_info[] = {
    /* spi 3.0 */
    {
        .modalias = "spidev",
        .max_speed_hz = 48000000, /* 48 Mbps */
        .bus_num = 3,
        .chip_select = 0,
        .mode = SPI_MODE_1,
    },
```

```
    /* spi 3.1 */
    {
        .modalias = "spidev",
        .max_speed_hz = 48000000, /* 48 Mbps */
        .bus_num = 3,
        .chip_select = 1,
        .mode = SPI_MODE_1,
    },
    /* spi 4.0 */
    {
        .modalias = "spidev",
        .max_speed_hz = 48000000, /* 48 Mbps */
        .bus_num = 4,
        .chip_select = 0,
        .mode = SPI_MODE_1,
    },
};
```

Luego de definidas las interfaces fue creada una función (omap3_beagle_init_spi_rf2) que las inicializara.

```
static void __init omap3_beagle_init_spi_rf2(void)
{
    printk(KERN_INFO "Usando SPI\n");
    /* hook the spi ports to the spidev driver */
    spi_register_board_info(beagle_mcspi_board_info,
    ARRAY_SIZE(beagle_mcspi_board_info));
}
```

Para que la función de inicialización de la interfaz SPI pueda ejecutarse, se debe hacer referencia a ésta en la función general de inicialización de interfaces asociada a la Beagleboard (omap3_beagle_init).

```
omap3_beagle_init_spi_rf2();
```

Con estos cambios se logró que las interfaces SPI sean accesibles en el espacio de usuario bajo /dev y mapeadas como spidev3.0, spidev3.1, spidev4.0.

Cambios asociados al GPIO:

Se creó una función (gpio_config_rf2) que dado un número asociado con el pin GPIO, establece su dirección y valor.

```
static void gpio_config_rf2(unsigned gpio, int direction,
int value) {
  /* Tell the kernel, we want to use the GPIO*/
  if (gpio_request(gpio, "gpio\n") != 0) {
    printk(KERN_ALERT "Unable to request GPIO %d\n",
    gpio);
  }
  else {
    /* Now tell the kernel that GPIO is an (in-out)put
    and should be set to value (only as output) */
    switch (direction) {
      case 0: if (gpio_direction_output(gpio, value) != 0) {
                printk(KERN_ALERT "Unable to set GPIO
                direction for GPIO %d\n", gpio);
              }
              else {
                /* enable direction on userspace */
                if (gpio_export(gpio, 1) != 0){
                  printk(KERN_ALERT "Unable to set GPIO
                  export for GPIO %d\n", gpio);
                }
              }
              break;
      case 1: if (gpio_direction_input(gpio) != 0) {
                printk(KERN_ALERT "Unable to set GPIO
                direction for GPIO %d\n", gpio);
              }
              else {
                /* enable direction on userspace */
                if (gpio_export(gpio, 1) != 0) {
                  printk(KERN_ALERT "Unable to set GPIO
                  export for GPIO %d\n", gpio);
                }
              }
              break;
      default: break;
    }
  }
}
```

Se creó la función de inicialización (gpio_rf2) que establece la configuración de todos los pines GPIO del sistema RF^2.

```
static void __init gpio_rf2(void)
{
    printk(KERN_ALERT "Configurando GPIO RF2...\n");
    gpio_config_rf2(133, 0, 1); /*E*/
    gpio_config_rf2(134, 0, 0); /*D5*/
    gpio_config_rf2(136, 0, 0); /*D7*/
    gpio_config_rf2(137, 0, 0); /*XOE*/
    gpio_config_rf2(138, 0, 0); /*led rojo*/
    gpio_config_rf2(139, 0, 0); /*led_verde*/
    gpio_config_rf2(144, 0, 0); /*RST_SC*/
    gpio_config_rf2(145, 0, 0); /*led_amarillo*/
    gpio_config_rf2(156, 0, 0); /*RW*/
    gpio_config_rf2(157, 0, 0); /*RS*/
    gpio_config_rf2(158, 0, 0); /*Buzzer*/
    gpio_config_rf2(159, 0, 0); /*D4*/
    gpio_config_rf2(161, 0, 0); /*D6*/
    gpio_config_rf2(162, 0, 0); /*Backlight*/
    gpio_config_rf2(168, 0, 1); /*RST_RF*/
    gpio_config_rf2(183, 1, 0); /*IRQ_RF*/
}
```

Luego, como en el caso de la interfaz SPI, se debe hacer referencia a la función anterior en la función de inicialización del sistema (omap3_beagle_init).

```
gpio_rf2();
```

Con estos cambios se logró una cambio en la configuración de los pines GPIO, según la dirección y el valor que les corresponde para el buen funcionamiento del sistema RF^2.

C.5. Instalación y configuración de librfid-tool

A continiuación se detalla la instalación y configuración de librfid para su uso con OpenPCD y con el lector/escritor RFID diseñado.

Para comenzar se descarga la biblioteca:

$ svn checkout https://svn.gnumonks.org/trunk/librfid/

Dentro del directorio raíz, se encuentran una serie de directorios con los fuentes. Los más importantes se detallan a continuación:

utils: en este directorio de encuentran las funciones asociadas con la herramienta librfid-tool. Se destaca librfid-tool.c donde se encuentra la función main de la aplicación librfid-tool.

src: en este directorio se encuentran todos los fuentes de la biblioteca librfid.

include: en este directorio se encuentran todos los encabezados de las funciones de la biblioteca librfid.

OpenPCD

Para comenzar se intentó la comunicación desde el PC para luego pasar a la Beagleboard, ya que existe mucha documentación para el primer caso [REF]. En los dos casos fue necesario compilar la biblioteca para utilizar en la arquitectura elegida.

OpenPCD en PC:

Como primer paso, se conecta el OpenPCD al PC. Para saber si el dispositivo es detectado por la PC, es necesario lo siguiente:

```
$ lsusb
```

Aquí debe aparecer (entre otros dispositivos conectados) un dispositivo con un identificador ID 16c0:076b (vendor:product).

Nota: En caso de que este dispositivo no aparezca, se debe usar un HUB con alimentación externa.

Compilando librfid:

Para compilar la biblioteca son necesarios los siguientes paquetes: libtool, libusb-dev, libcurl-dev (libcurl4gnutls-dev instalado en este caso).

Se debe ir hasta el directorio donde se encuentra la biblioteca descargada (librfid por ejemplo) y escribir lo siguiente:

$ cd librfid
$ ./configure
$ make
$ sudo make install

OpenPCD en la Beagleboard:

Antes que nada se debe conectar el OpenPCD a la Beagleboard y verificar que es detectado al igual que en la instalación en el PC:

$ lsusb

Se tuvo que utilizar un HUB con alimentación externa ya que la Beagleboard no logró ver al OpenPCD.

La compilación se realiza directamente en la Beagleboard.

Al igual que en el caso de la compilación para el uso en un PC, se debieron obtener los paquetes libtool, libusb-dev, libcurl-dev. Para instalar los paquetes se utilizó el siguiente comando:

$ opkg install “paquete”

Se copia la biblioteca (sin compilar) a la Beagleboard y luego:

$ cd librfid
$ ./configure
$ make
$ make install

Lector-escritor RFID en la Beagleboard:

El lector/escritor RFID se conecta a la Beagleboard a través de una interfaz SPI. Para habilitar las funcionalidades de la librfid asociadas con la comunicación SPI, es necesario indicar al configurar, que se quiere utilizar esta interfaz a través de la opción −−enable-spidev. La compilación se realiza en la Beagleboard:

$ cd librfid

```
$ ./configure −−enable-spidev
$ make
$ make install
```

Con esto se logró compilar la librfid con opciones para utilizar la interfaz SPI

A continuación se explica el uso de librfid-tool:

librfid-tool -[opción]

Dentro de las opciones:

s: realiza una búsqueda de tarjetas RFID hasta encontrar una.

S: loop infinito con la opción “s”, muestra información sobre la tarjeta RFID encontrada en cada paso.

p: especifica el protocolo RFID a utilizar, entre las opciones se encuentran “tcl”, “mifare-classic” y “mifare-ultralight”.

l: especifica el protocolo de capa 2 a utilizar, entre las opciones se encuentran “ISO14443a”, “ISO14443b” y “ISO15693”.

h: ayuda.

Ejemplos de uso recomendados:

```
$ librfid-tool -p mifare-classic
```

Si encuentra una tarjeta con protocolo Mifare-classic, devuelve la lectura completa de los bloques de memoria de la tarjeta.

```
$ librfid-tool -S
```

Devuelve el UID y el protocolo soportado por la tarjeta.

C.6. Depuración de código

A continuación se detallan algunos comandos útiles para el depurado de código con GDB:

breakpoint: para colocar un breakpoint. En general se lo llama seguido del nombre de una función de la aplicación.

print: seguido del nombre de una variable, muestra el contenido de la variable durante el proceso de depuración. Si la variable es local a alguna función, el valor de la variable se pierde al salir de la función.

next o “n”: sirve para ir línea a línea en modalidad step-over (sin entrar a las funciones).

step o “s”: sirve para ir línea a línea en modalidad step-into (entrando a las funciones).

backtrace o “bt”: despliega el stack de llamadas a funciones, sirve para saber por donde se pasó y donde se está.

C.7. Depuración remota

A continuación se detalla la configuración de depuración remota con conexión ethernet.

En la Beagleboard:

$ gdbserver localhost: <puerto> <ejecutable Aplicación> <argumentos>

En el pc de desarrollo:

$ gdb

$ target remote <ip_Beagleboard>:<puerto>

<puerto>: número del puerto por el que se conectan. El puerto no debe estar ocupado por otra aplicación. Por ejemplo un número > 2000 funciona correctamente.

<ejecutable Aplicación> <argumentos>: es el nombre de la aplicación que se quiere depurar y si la aplicación necesita algún argumento para ejecutarse correctamente también se deben agregar.

<ip_Beagleboard>: es la ip de la Beagleboard.

C.8. Preparación de la memoria SD

Se van a necesitar dos particiones, una FAT32 de más de 32MB y una ext3 (el resto del espacio), la primer partición (FAT32) debe ser booteable. Hay dos maneras de hacerlo, una es usando el GParted (modo gráfico) y la otra es por consola.

C.8.1. Formateo de la memoria SD

Formateo utilizando GParted

Se instala el GParted:

$ sudo apt-get install gparted

Se crean las particiones y se les da formato. Se da click derecho sobre la partición FAT32, gestionar opciones y se marca boot. también se puede dar un nombre a cada partición.

Formateo manual de la memoria SD

El siguiente procedimiento se realiza por consola.

Antes que nada se debe saber dónde está la SD (memory card). Para saberlo se ejecuta en consola:

$ dmesg

Se obtiene en parte lo siguiente:

```
sd 3:0:0:0: Attached scsi generic sg1 type 0
sd 3:0:0:0: [sdb] 7729152 512-byte logical blocks:
(3.95 GB/3.68 GiB)
sd 3:0:0:0: [sdb] Write Protect is off
sd 3:0:0:0: [sdb] Mode Sense: 03 00 00 00
sd 3:0:0:0: [sdb] Assuming drive cache: write through
sd 3:0:0:0: [sdb] Assuming drive cache: write through
sdb: sdb1
sd 3:0:0:0: [sdb] Assuming drive cache: write through
sd 3:0:0:0: [sdb] Attached SCSI removable disk
```

En este caso la SD está en /dev/sdb

Se borran las particiones:

$ sudo fdisk /dev/sdb

```
Command (m for help): o
Building a new DOS disklabel. Changes will remain in
memory only, until you decide to write them. After
that, of course, the previous content won't be
recoverable.
Warning: invalid flag 0x0000 of partition table 4
will be corrected by w(rite)
```

Información de la memoria:

```
Command (m for help): p
Disk /dev/sdb: 3957 MB, 3957325824 bytes
....
```

Recordar la cantidad de bytes.

Se entra en Expert mode:

```
Command (m for help): x
```

Se quiere setear la geometría de la siguiente forma: 255 heads, 63 sectors, y se calcula el número de cilindros requeridos por la tarjeta:

C=trunk(B/255/63/512), donde C:cilindros, B:número de bytes de la tarjeta (anotado previamente) En este caso: C=trunk(481,117)=481

```
Expert command (m for help): h
Number of heads (1-256, default 4): 255
Expert command (m for help): s
Number of sectors (1-63, default 62): 63
Warning: setting sector offset for DOS compatiblity
Expert command (m for help): c
Number of cylinders (1-1048576, default 1011): 481
```

Se va a crear la partición FAT32 y a marcarla como booteable:

```
Expert command (m for help): r
Command (m for help): n
Command action
e extended
p primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-481, default 1): (Enter)
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-481,
default 481): +50
//son como 400MB
Command (m for help): t
Selected partition 1
Hex code (type L to list codes): c
Changed system type of partition 1 to c (W95 FAT32
(LBA))
Command (m for help): a
Partition number (1-4): 1
```

Se crea la segunda partición:

```
Command (m for help): n
Command action
e extended
p primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (52-481, default 52): (Enter)
Using default value 52
Last cylinder or +size or +sizeM or +sizeK (52-481,
default 481):(Enter)
Using default value 481
```

Se imprime para ver como va todo:

```
Command (m for help): p
Disk /dev/sdb: 3957 MB, 3957325824 bytes
255 heads, 63 sectors/track, 481 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Device Boot
/dev/sdb1 *
Start
1
```

```
End
Blocks Id System
51
409626 c W95 FAT32 (LBA)
/dev/sdb2
52
481
3453975 83 Linux
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
WARNING: Re-reading the partition table failed with
error 16: Device or resource busy. The kernel still uses
the old table. The new table will be used at the next reboot.
WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for
additional information.
Syncing disks.
```

Se desmontan las particiones creadas:

$ umount /dev/sdb1

$ umount /dev/sdb2

Puede que diga que ya no está montado, en ese caso se sigue igualmente.

Luego hay que formatear las particiones:

$ sudo mkfs.msdos -F 32 /dev/sdb1 -n nombre

```
mkfs.msdos 3.0.7 (24 Dec 2009)
```

$ sudo mkfs.ext3 /dev/sdb2 -L otroNombre

```
mke2fs 1.41.11 (14-Mar-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
216000 inodes, 863493 blocks
43174 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=402653184
```

```
27 block groups
32768 blocks per group, 32768 fragments per group
8000 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

Si dice que no encuentra el dispositivo, se saca la SD y se vuelve a colocar.

C.8.2. Copia de archivos a la SD

A partir de aquí se supone que la partición FAT32 tiene el nombre boot y la partición ext3 tiene el nombre rootFS y que los archivos generados se encuentran en el directorio beagleboard ubicado en el home del usuario (∼/beagleboard es su ubicación).

Se realiza por consola. Los primeros tres archivos van a la partición FAT32 y el fileSystem (sin descomprimir) va a la partición ext3.

$ cd ∼/beagleboard

$ sudo cp MLO /media/boot

Nota: El MLO debe ser el primer archivo a copiar.

$ sudo cp u-boot.bin uImage /media/boot

$ cp fileSystem.tar.gz /media/rootFS

Se descomprime el fileSystem en la partición ext3.

$ cd /media/rootFS

$ sudo tar -xvzf fileSystem.tar.gz

Esto demora un rato.

$ sudo rm -f fileSystem.tar.gz

Se borra el archivo original.

Se desmonta la memoria SD y queda lista para ser probada.

$ sync
$ umount /media/boot
$ umount /media/rootFS

C.9. Configuración en el PC para conexión serial con la Beagleboard

Para la comunicación con la Beagleboard se utilizó el programa minicom el cual se maneja desde consola (si se quiere uno con ambiente gráfico, se recomienda cutecom). Para configurar minicom se accede a la consola y se escribe lo siguiente:

$ sudo minicom -s

Se accede a "Configuración de la puerta serial" y ahí se debe configurar como sigue:
A - Dispositivo Serial: /dev/ttyUSB0 (si no se encuentra nada, probar con ttyUSB1)
B - Localización del Archivo de Bloqueo: /var/lock
C - Programa de Acceso:
D - Programa de Salida:
E - Bps/Paridad/Bits: 115200 8N1
F - Control de Flujo por Hardware: No
G - Control de Flujo por Software: No

Luego, para guardar las opciones elegidas se marca la opción "Salvar configuración como dfl".

C.10. printenv

```
OMAP3 beagleboard.org # printenv
bootdelay=3
baudrate=115200
loadaddr=0x80200000
rdaddr=0x81600000
```

```
usbtty=cdc_acm
console=ttyS2,115200n8
optargs=
bootscr=boot.scr
camera=lbcm3m1
vram=12M
dvimode=640x480MR-16@60
defaultdisplay=dvi
mmcdev=1
mmcroot=/dev/mmcblk0p2 rw
mmcrootfstype=ext3 rootwait
nandroot=/dev/mtdblock4 rw
nandrootfstype=jffs2
ramroot=/dev/ram0 rw
ramrootfstype=ext2
mmcargs=setenv bootargs console=${console} ${optargs}
mpurate=${mpurate} buddy=${buddy} camera=${camera}
vram=${vram} omapfb}
nandargs=setenv bootargs console=${console} ${optargs}
mpurate=${mpurate} buddy=${buddy} camera=${camera}
vram=${vram} omapf}
loadbootscript=fatload mmc ${mmcdev} ${loadaddr} ${bootscr}
ramargs=setenv bootargs console=${console} ${optargs}
mpurate=${mpurate} buddy=${buddy} camera=${camera}
vram=${vram} omapfb}
loadramdisk=fatload mmc ${mmcdev} ${rdaddr} ramdisk.gz
bootscript=echo Running bootscript from mmc ...;
source ${loadaddr}
loaduimage=fatload mmc ${mmcdev} ${loadaddr} uImage
mmcboot=echo Booting from mmc ...; run mmcargs;
bootm ${loadaddr}
nandboot=echo Booting from nand ...; run nandargs;
nand read ${loadaddr} 280000 400000; bootm ${loadaddr}
ramboot=echo Booting from ramdisk ...; run ramargs;
bootm ${loadaddr}
dieid#=096800040000000040365fa1301901a
mtdids=nand0=nand
stdin=serial
stdout=serial
stderr=serial
bootcmd=mmc init;fatload mmc 0 80300000 uImage;bootm 80300000
bootargs=console=ttyS2,115200n8 root=/dev/mmcblk0p2 rw rootwait
buddy=unknown
beaglerev=Cx
```

```
mpurate=720
```

C.11. Ejemplo de arranque del sistema

```
U-Boot 2009.11-rc1-00601-g3aa4b51 (Jan 05 2010 - 20:56:38)
OMAP3530-GP ES3.1, CPU-OPP2 L3-165MHz
OMAP3 Beagle board + LPDDR/NAND
I2C: ready
DRAM: 256 MB
NAND: 256 MiB
In: serial
Out: serial
Err: serial
Board revision C4
Die ID #0968000400000000040365fa1301901a

Hit any key to stop autoboot: 0
mmc1 is available
reading uImage
3194180 bytes read
## Booting kernel from Legacy Image at 80300000 ...
Image Name: Angström/2.6.32/beagleboard
Image Type: ARM Linux Kernel Image (uncompressed)
Data Size: 3194116 Bytes = 3 MB
Load Address: 80008000
Entry Point: 80008000
Verifying Checksum ... OK
Loading Kernel Image ... OK
OK
Starting kernel ...
...
...
...

The Angström Distribution beagleboard ttyS2
Angström 2009.X-test-20100104 beagleboard ttyS2
beagleboard login:
```

Aquí el usuario es root, sin contraseña.

C.12. Conexión de Beagleboard a internet

En el PC de desarrollo se debe configurar lo que sigue:

$ sudo gedit /etc/sysctl.conf

Se debe descomentar la línea que dice net.ipv4.ip_forward=1 para habilitar el "forwarding" de paquetes.

$ sudo sysctl -p

$ sudo cat /proc/sys/net/ipv4/ip_forward

Lo anterior es para verificar que los cambios fueron hechos de manera correcta.

$ sudo iptables −−table nat −−append POSTROUTING −−out-interface eth0 -j MASQUERADE

$ sudo iptables −−append FORWARD −−in-interface usb0 -j ACCEPT

Cuando se refiere a −−out-interface, se refiere a la interfaz del PC por la que se accede a internet.

C.13. Instalación de PCSC-Lite, CCID y pcsc-tools, y agregado de lector serial

A continuación se describe el proceso de instalación en la Beagleboard, el procedimiento para un PC común es el mismo.

Primero se bajan los fuentes asociados a la PCSC-Lite [63], CCID [63] y pcsc-tools [64]. Luego se envían a la Beagleboard.

Paquetes necesarios en la Beagleboard: libusb-1.0-dev.

$ opkg install libusb-1.0-dev

PCSC-Lite:

Para la instalación de la PCSC-Lite se usó la biblioteca libusb, aunque se puede hacer con el uso de la biblioteca libudev. Se supone que los fuentes se encuentran en el directorio pcsclite.

$ cd pcsclite

$ ./configure −−disable-libudev LIBUSB_CFLAGS=-I/usr/include/libusb-1.0 LIBUSB_LIBS=-L/usr/lib -lusb-1.0”−−enable-ipcdir=/var/run/pcscd −−enable-usbdropdir=/usr/drivers −−enable-confdir=/etc/reader.conf.d −−prefix=/usr

LIBUDEV_CFLAGS: Dirección donde se encuentran los .h asociados con libusb.

LIBUDEV_LIBS: Dirección donde está libusb.

−−enable-ipcdir: Dirección donde se guardan archivos relacionados con la comunicación con el demonio de la pcsclite (pcscd).

−−enable-usbdropdir: Dirección donde se guardan drivers usb (asociado con la instalación del ccid).

−−enable-confdir: Dirección donde se guardan archivos relacionados con la configuración serial.

−−prefix: Dirección a partir de la cual se instalan los archivos necesarios para el correcto funcionamiento de la biblioteca.

$ make

$ make install

CCID:

Se supone que los fuentes se encuentran en el directorio ccid.

$ cd ccid

$ ./configure LIBUSB_CFLAGS=-I/usr/include/libusb-1.0 LIBUSB_LIBS=L/usr/lib -lusb-1.0”PCSC_CFLAGS=-I/usr/include/PCSC/ PCSC_LIBS=L/usr/lib -lpcsclite”−−enable-usbdropdir=/usr/drivers −−prefix=/usr

$ make

$ make install

PCSC-TOOLS:

Se supone que los fuentes se encuentran en el directorio pcsc-tools.

$ make

$ make install

Pasos para agregar el lector de tarjetas serial, en la biblioteca pcsclite:

Se crea el directorio para drivers de lectores (es posible que ya haya sido creado por CCID al ser instalado).

$ mkdir /usr/drivers/

Dentro de este directorio se ubica la biblioteca dinámica del driver para el lector de tarjetas.

$ scp rf2_sc.so root@172.16.1.14:/usr/drivers

Luego es necesario crear el archivo de configuración donde se encuentran los parámetros del lector serial. Es posible conocer el directorio donde debe ubicarse el archivo de configuración ejecutando el comando:

$ pcscd -v

Se crea el archivo:

$ mkdir /etc/reader.conf.d/

$ touch /etc/reader.conf.d/reader.conf

Se edita el archivo reader.conf agregando las siguientes líneas:

```
########################################
# Configuration file for pcsc-lite
# Reader Proyecto-RF2 FING-UDELAR

FRIENDLYNAME  rf2_sc
DEVICENAME    /dev/ttyS1
LIBPATH       /usr/drivers/rf2_sc.so
CHANNELID     1
########################################
```

CHANNELID indica el número de puerto donde se encuentra el dispositivo:

```
1 ---> /dev/pcsc/1
2 ---> /dev/pcsc/2
3 ---> /dev/pcsc/3
```

El puerto serial se encuentra realmente en /dev/ttySi, por tanto se debe realizar un enlace simbólico /dev/pcsc/i —> /dev/ttySi . Para efectuar lo anterior se crea el directorio pcsc bajo dev:

$ mkdir /dev/pcsc

$ cd /dev/pcsc

$ ln -s /dev/ttySi i

donde i es el número de puerto serial que eusado por el lector de tarjetas.

Se reinicia el demonio pcscd y se ejecuta la herramienta pcsc_scan para que muestre el dispositivo:

```
...
Scanning present readers...
0: rf2_sc 00 00

...
Reader 0: rf2_sc 00 00
```

C.14. Pruebas sobre las interfaces

C.15. led.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
FILE *fp;

int main(int argc, char** argv)
{
  printf("\n**********************************\n"
  "* Welcome to PIN Blink program *\n"
  "* ....blinking pin 13 on expansion port *\n"
  "* ....rate of 1 Hz.......... *\n"
  "**********************************\n");
  //create a variable to store whether we are
  //sending a '1' or a '0'
  char set_value[4];
```

```
//Integer to keep track of whether we want on or off
int toggle = 0;
//Using sysfs we need to write "134"
//to /sys/class/gpio/export
//This will create the folder /sys/class/gpio/gpio134
if ((fp = fopen("/sys/class/gpio/export", "ab")) == NULL)
{
  printf("Cannot open export file.\n");
  exit(1);
}
//Set pointer to begining of the file
rewind(fp);
//Write our value of "134" to the file
strcpy(set_value,"134");
fwrite(&set_value, sizeof(char), 3, fp);
fclose(fp);
printf("...export file accessed, new pin now accessible\n");
//SET DIRECTION
//Open the LED's sysfs file in binary for
//reading and writing, store file pointer in fp
if ((fp = fopen("/sys/class/gpio/gpio134/direction", "rb+"))
== NULL)
{
  printf("Cannot open direction file.\n");
  exit(1);
}
//Set pointer to begining of the file
rewind(fp);
//Write our value of "out" to the file
strcpy(set_value,"out");
fwrite(&set_value, sizeof(char), 3, fp);
fclose(fp);
printf("...direction set to output\n");
//SET VALUE
//Open the LED's sysfs file in binary for
//reading and writing, store file pointer in fp
if ((fp = fopen("/sys/class/gpio/gpio134/value", "rb+"))
== NULL)
{
  printf("Cannot open value file.\n");
  exit(1);
}
//Set pointer to begining of the file
rewind(fp);
```

```
  //Write our value of "1" to the file
  strcpy(set_value,"1");
  fwrite(&set_value, sizeof(char), 1, fp);
  fclose(fp);
  printf("...value set to 1...\n");
  //Run an infinite loop - will require
  //Ctrl-C to exit this program
  while(1)
  {
    //Set it so we know the starting value
    //in case something above doesn't leave it as 1
    strcpy(set_value,"1");
    if ((fp = fopen("/sys/class/gpio/gpio134/value", "rb+"))
    == NULL)
    {
      printf("Cannot open value file.\n");
      exit(1);
    }
    toggle = !toggle;
    if(toggle)
    {
      //Set pointer to begining of the file
      rewind(fp);
      //Write our value of "1" to the file
      strcpy(set_value,"1");
      fwrite(&set_value, sizeof(char), 1, fp);
      fclose(fp);
      printf("...value set to 1...\n");
    }
    else
    {
      //Set pointer to begining of the file
      rewind(fp);
      //Write our value of "0" to the file
      strcpy(set_value,"0");
      fwrite(&set_value, sizeof(char), 1, fp);
      fclose(fp);
      printf("...value set to 0...\n");
    }
    //Pause for one second
    sleep(1);
  }
  return 0;
}
```

C.16. uart.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

#define BAUDRATE B9600
#define SERIALPORT "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1
#define BUFFWRITE 10
#define BUFFREAD 255

int fd; //descriptor del archivo asociado con tty
struct termios oldtio,newtio;

//inicialización
int tty_init (void)
{
  //abrimos el puerto ttyS1
  fd = open(SERIALPORT, O_RDWR | O_NOCTTY );
  if(fd<0)
  return 0; //hubo errores
  else
  {
    printf("Puerto %s, abierto. fd: %d\n", SERIALPORT, fd);
    /* save current port settings */
    tcgetattr(fd,&oldtio);
    //bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
    newtio.c_iflag = IGNPAR;
    newtio.c_oflag = 0;
    /* set input mode (non-canonical, no echo,...) */
    newtio.c_lflag = 0;
    /* inter-character timer unused */
    newtio.c_cc[VTIME] = 0;
    /* bloqueo la lectura hasta que llegue 1 caracter */
    newtio.c_cc[VMIN] = 1;
    tcflush(fd, TCIFLUSH);
    tcsetattr(fd,TCSANOW,&newtio);
    return 1; //no hubo errores
```

```
  }
}

//envio de un caracter por UART
void escribir (char s[10])
{
  write(fd, s, BUFFWRITE); //escribe el string en UART-Tx
  printf("dato enviado: %s\n", s);
}

//lectura de caracter por UART
char leer (void)
{
  char r[BUFFREAD];
  if(read(fd, r, BUFFREAD) == -1)
  return 0; //No hay datos o se incluyen ceros
  else
  return r[0]; //no hubo errores, se retorna el valor leido
}

int main (void)
{
  //cantidad de caracteres leidos por UART
  int UART_get_return;
  //indica si se pudo abrir el puerto o no
  int UART_init_return;
  char buff[BUFFREAD];
  //inicializacion de la UART
  UART_init_return = tty_init();

  //no se pudo abrir el puerto
  if (UART_init_return == 0)
  printf("no se pudo abrir el puerto ttyS1.\n");
  else //se pudo abrir el puerto
  {
    char str[10]={'a','s','d','f','g','h','j','k','l','t'};
    escribir(str);
    UART_get_return = read(fd,buff,BUFFREAD);

    // 0 = no se encontraron datos en Rx
    if (UART_get_return == 0)
    printf("no hay datos para leer\n");
    else //1 = dato encontrado
    {
```

```
      buff[UART_get_return]=0;
      printf("dato leido: %s:%d\n", buff,UART_get_return);
    }
  }
  tcsetattr(fd,TCSANOW,&oldtio);
}
```

Apéndice D

Hojas de datos

A continuación se adjuntan algunas de las hojas de datos a las que se hace referencia en el texto. Se encuentran ordenadas de acuerdo al siguiente listado, respetando el orden en el cual se mencionan:

- MIFARE® and I Code CL RC632 Multiple protocol contactless reader IC.
- 8-BIT BIDIRECTIONAL VOLTAGE-LEVEL TRANSLATOR.
- LM1117/LM1117I 800mA Low-Dropout Linear Regulator.
- Transistor 2N3904.
- Transistor 2N3906.
- ESDA6V1W5.
- HD44780U (LCD-II), Dot Matrix Liquid Crystal Display Controller/Driver.
- LCM-S01602DSF-A - hoja de datos del LCD16x2.
- Micore Reader IC Family; Directly Matched Antenna Design.
- Mifare®(14443A) 13,56 MHz RFID Proximity Antennas.
- BeagleBoard System Reference Manual Rev C4, Revision 0.0.
- OMAP 35x Applications Processor Technical Reference Manual.

- Hawkboard Press Release Solution.
- Build a cheap 13.56MHz MIFARE antenna for the Proxmark.

Parte VII

Bibliografía

Bibliografía

[1] Antenna Circuit Design for RFID Applications, AN710.

[2] HF Antenna Design Notes, Literature Number: 11-08-26-003.

[3] HF Antenna Cook Book, Lit. Number 11-08-26-001.

[4] Micore Reader IC Family; Directly Matched Antenna Design, Rev. 2.05, 10 May 2006.

[5] Mifare®(14443A) 13,56 MHz RFID Proximity Antennas, Application Note.

[6] MIFARE Type Identification Procedure; AN10833 Rev. 3.1 — 07 July 2009.

[7] MIFARE ISO/IEC 14443 PICC Selection; AN10834 Rev. 3.0 — 26 June 2009.

[8] MIFARE Classic Commands; AN10856 Rev. 1.0 — 31 July 2009.

[9] MIFARE and handling of UIDs; AN10927 Rev. 2.0 — 01 September 2010.

[10] Mainstream contactless smart card IC for fast and easy; MF1S5009 Rev. 3 — 27 July 2010.

[11] MIFARE Standard Card IC MF1 IC S50 Functional Specification; Revision 5.1 May 2001.

[12] MIFARE® and I Code CL RC632 Multiple protocol contactless reader IC.

[13] 8-BIT BIDIRECTIONAL VOLTAGE-LEVEL TRANSLATOR.

[14] HD44780U (LCD-II), Dot Matrix Liquid Crystal Display Controller/Driver, Rev. 0.0 1998.

[15] LM1117/LM1117I 800mA Low-Dropout Linear Regulator.

[16] LCM-S01602DSF/A - hoja de datos del LCD16x2.

[17] BeagleBoard System Reference Manual Rev C4, Revision 0.0.

[18] Smart Card Handbook, Third Edition, Wolfgang Rankl and Wolfgang Effing, 2003, ISBN: 0-470-85668-8.

[19] RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification, Second Edition, Klaus Finkenzeller, 2003, ISBN: 0-470-84402-7.

[20] Application Independent CardTerminal Application Programming Interface for ICC Applications; Deutsche Telekom AG / PZ Telesec, GMD - Forschungszentrum Informationstechnik GmbH, TÜV Informationstechnik GmbH, TELETRUST Deutschland e.V., 30.10.1996.

[21] Application independent CardTerminal Basic Command Set for ICC applications; TeleTrusT Deutschland e.V., GMD Forschungszentrum Informationstechnik GmbH, Version 0.9, 28.07.1995.

[22] IFDHandler 3.0 MUSCLE PC/SC IFD Driver API; David Corcoran & Ludovic Rousseau, July 28, 2004.

[23] Diseño y desarrollo de circuitos impresos con KiCad, Miguel Pareja Aparicio, 2010, ISBN: 978-84-937769-1-6.

[24] Norma ISO/IEC 7816.

[25] Norma ISO/IEC 14443.

[26] OpenPCD, http://www.openpcd.org/, agosto 2011.

[27] USB4ALL, http://www.fing.edu.uy/inco/grupos/mina/pGrado/pgusb/main.php, agosto 2011.

[28] librfid, http://openmrtd.org/projects/librfid/, agosto 2011.

[29] GES9G20, http://www.glomationinc.com/product_9G20.html, agosto 2011.

[30] Hawkboard, http://www.hawkboard.org/, agosto 2011.

[31] Beagleboard, http://beagleboard.org/, agosto 2011.

[32] Foro de Beagleboard, https://groups.google.com/forum/#!forum/beagleboard, agosto 2011.

[33] Openembedded-Bitbake, http://wiki.openembedded.net/index.php/BitBake_%28dev%29, agosto 2011.

[34] KiCad, http://www.lis.inpg.fr/realise_au_lis/kicad/, agosto 2011.

[35] gEDA, http://www.gpleda.org/index.html, agosto 2011.

[36] Proxmark, http://www.proxmark.org/proxmark, agosto 2011.

[37] Angström, http://www.angstrom-distribution.org/, agosto 2011.

[38] MLO, http://www.angstrom-distribution.org/demo/beagleboard/MLO/, agosto 2011.

[39] Narcissus, http://narcissus.angstrom-distribution.org/, agosto 2011.

[40] termios, http://pubs.opengroup.org/onlinepubs/007908799/xsh/termios.h.html, agosto 2011.

[41] Proceso de booteo de linux, http://www.ibm.com/developerworks/linux/library/l-linuxboot/index.html, agosto 2011.

[42] Make, http://es.wikipedia.org/wiki/Make, agosto 2011.

[43] DDD, http://www.gnu.org/s/ddd/, agosto 2011.

[44] GPIO, http://www.avrfreaks.net/wiki/index.php/Documentation:Linux/GPIO#Interfaces_explained, agosto 2011.

[45] Kernel GPIO, http://kernel.org/doc/Documentation/gpio.txt, agosto 2011.

[46] Configuración SDK, http://www.electronsonradio.com/2011/04/intro-to-basic-cross-compiling-for-the-beagleboard/, agosto 2011.

[47] http://ladyada.net/make/simreader/

[48] http://www.freeinfosociety.com/electronics/schemview.php?id=1995

[49] http://www.gooze.eu/howto/smartcard-quickstarter-guide

[50] http://linux.die.net/man/3/dlsym

[51] http://www.mkssoftware.com/docs/man3/select.3.asp

[52] http://tldp.org/HOWTO/Program-Library-HOWTO/dl-libraries.html

[53] http://www.easysw.com/ mike/serial/serial.html

[54] http://linux.die.net/man/3/fd_set

[55] http://linux.die.net/man/2/select_tut

[56] http://www.iearobotics.com/wiki/index.php?title=Tutorial:Puerto_serie_en_Linux

[57] http://es.tldp.org/COMO-INSFLUG/COMOs/Programacion-Serie-Como/Programacion-Serie-Como-3.html

[58] http://www.rastersoft.com/articulo/pserie.html

[59] http://www.faqs.org/docs/Linux-mini/IO-Port-Programming.html

[60] http://tldp.org/HOWTO/IO-Port-Programming-2.html

[61] PCSC-Lite, http://pcsclite.alioth.debian.org/

[62] PCSC Workgroup, http://www.pcscworkgroup.com/

[63] Descargas PCSC-Lite, https://alioth.debian.org/frs/?group_id=30105

[64] http://ludovic.rousseau.free.fr/softwares/pcsc-tools/

[65] http://ludovicrousseau.blogspot.com/2011/04/libccid-and-usb-selective-suspend.html

[66] http://readthedocs.org/docs/pvdevtools_doc/en/latest/index.html

[67] http://www.springcard.com/support/pcsclite.html