UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE

"DESARROLLO DE MÓDEM CELULAR USB MULTICOMPAÑÍA APLICADO A LA TRANSFERENCIA DE ARCHIVOS DE TEXTO"

MANUEL PATRICIO COHEN SCHEIHING

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO
MENCIÓN COMPUTADORES

PROFESOR GUÍA: **AGUSTÍN GONZÁLEZ VALENZUELA**
PROFESOR CORREFERENTE: **TOMÁS ARREDONDO VIDAL**

NOVIEMBRE - 2013

A mi mamá quién siempre ha estado a mi lado para apoyarme y levantarme en los peores momentos.

A mi papá que me ha traspasado su amor por la ciencia e ingeniería desde muy pequeño.

A Electrosoft Ingeniería por las grandes oportunidades que me ha entregado en cuanto al aprendizaje de la ingeniería electrónica y de aprender la gran diferencia que existe entre la teoría y la práctica.

A mi abuelita Mima que me ha entregado su amor incondicional desde los primeros días de mi vida. Si hubiera más personas como ella, el mundo sería un lugar muy distinto.

A mi Tata que siempre compartió su amor y sabiduría conmigo durante mi niñez y que hoy lo recuerdo con mucho cariño.

Índice Contenidos

Índice de figuras

Índice de tablas

Desarrollo de módem celular USB multicompañía aplicado a la transferencia de archivos de texto

Memoria para optar al título de Ingeniero Civil Electrónico
Mención Computadores

Manuel Patricio Cohen Scheihing

Profesor Guía: Agustín González V.
Profesor Correferente: Tomás Arredondo V.

Valparaíso, Noviembre 2013

Resumen

El objetivo de este trabajo, es implementar un enlace inalámbrico entre dos computadores remotos. Uno de los computadores (servidor) estará conectado mediante un módem convencional a la red telefónica alambrada (POTS), mientras que el otro computador (cliente) estará conectado a la red celular mediante un módem USB especializado diseñado durante el desarrollo de este trabajo.

El módem USB especializado debe contar con la característica de poder conectarse a la red celular mediante uno o más proveedores de servicio, es por ello, que debe poseer la capacidad de tener dos o más tarjetas SIM. La finalidad de esta característica es otorgar una mayor seguridad en cuanto la conexión, en caso de que algún proveedor de servicio no esté disponible en alguna zona geográfica.

Se realizaron escenarios de pruebas para evaluar el funcionamiento del módem USB especializado diseñado, donde se obtuvieron resultados exitosos. Para esto fue necesario implementar dos programas, uno corriendo en el computador cliente que controla al módem USB especializado y otro en el servidor que controla al módem convencional. Ambos programas además implementan el protocolo de comunicaciones entre cliente y servidor. Al existir ausencia de cobertura se verifica la conmutación de tarjeta SIM sin problemas y se vuelve a establecer la conectividad con el servidor. Dentro de los escenarios de prueba se simula una falla en la comunicación durante la transmisión de archivos de texto. Una vez que esta falla es superada, se restablece la comunicación y se retoma la transmisión desde el último paquete recibido con éxito.

Palabras Claves: GSM, SIM, POTS, USB, MCU, MSP430. AT, C, C#

Development of an USB multi-network cellular modem for text file transmission

Manuel Patricio Cohen Scheihing
November 2013

Abstract

The goal of this work is to implement a wireless link between two remote computers. One of the computers (server) will be connected through a standard modem to the wired telephone network (POTS), while the other computer (client) will be connected to the cellular network through a custom USB modem designed during the development of this work.

The custom USB modem must have the capability of connecting to the cellular network through one or more service providers, for that reason, it must allow two or more SIMCARDS to be installed on the device. The purpose of this feature is to provide more reliability for the connection in the case that a service provider is not available in some geographical zones.

Test scenarios were successfully carried out for evaluating the operation of the designed custom USB modem. In order to perform the tests it was necessary to implement two software applications, one running in the client computer that controls the custom USB modem and the other one running in the server that controls the conventional modem. Both software applications also implement the communication protocol between client and server. In one of the test scenarios a communication failure was simulated during the transmission of a text file. Once this failure mode finished, the communication was restored and the transmission was resumed from the last packet successfully received.

CAPÍTULO 1

1. Introducción

Este tema de memoria surge con el requerimiento de mejorar el sistema actual de transmisión de datos que se utiliza en las elecciones en Chile, ya que el procedimiento utilizado es muy precario, propenso a errores y poco eficiente. En las localidades remotas, que se encuentran bastante lejos del centro del país, es frecuente encontrar lugares donde no existe disponibilidad de líneas telefónicas lo que hace muy tedioso el proceso de envío de resultados de los sufragios a la capital.

1.1 Motivación, descripción del problema y objetivos

Cuando se realizan las elecciones en Chile, es necesario enviar los datos de cada local de votación hacia un servidor central para realizar el recuento de votos. Actualmente esto se realiza instalando una línea física en cada local de votación, donde un digitador ingresa el conteo de votos en minutas que son enviadas a la central vía módem o vía fax. Los profesionales que trabajan en el proyecto elecciones propusieron la idea de utilizar conectividad celular en vez de línea física para reducir el esfuerzo y costo de instalación y hacer más eficiente el proceso. Pero debido a la falta de cobertura celular en algunas áreas, es necesario que la solución cuente con dos o más proveedores de servicio de manera de garantizar conectividad en las áreas donde se encuentran los locales de votación.

Se ha planteado en repetidas ocasiones la posibilidad de utilizar como medio de transmisión el Internet para transmitir los datos de conteo de votos hacia la central, tal como se realiza en otros países. Sin embargo, se ha decidido por mantener como medio de transmisión el canal de telefonía, permitiendo su extensión a la comunicación celular para sectores sin telefonía fija. Las razones para no utilizar Internet todavía, son las medidas de seguridad adicionales que implica el adoptar este nuevo medio de transmisión, ya que por el hecho de ser una red pública existe una probabilidad de ataques y robos de información bastante mayor al que existe mediante una comunicación telefónica dedicada. Por otro lado, el incorporar en la plataforma de comunicaciones sistemas de seguridad de alto desempeño tiene impactos económicos y en tiempo bastante importantes que hacen poco interesante por el momento adoptar dicha forma de conectividad.

Para este trabajo de título, el problema a resolver es diseñar una plataforma hardware/software que permita transmitir datos en forma de archivos de texto desde un local de votación (cliente) hacia el servidor donde se realiza la recolección de los datos. El

computador que se encuentra en el local de votación transmitirá mediante la red celular los datos al servidor que se encuentra conectado a una línea telefónica física en la central de acopio de datos. Para maximizar la probabilidad de conectividad con el servidor, es necesario que el dispositivo que realiza la transmisión, que estará instalado en el computador del cliente, pueda utilizar dos tarjetas SIM, las que corresponderán a proveedores de servicio diferentes (ej., Movistar y Entel). Si por algún motivo la cobertura de uno de los proveedores se viese comprometida, se conmutará a la otra tarjeta SIM con la finalidad de contar con cobertura para resumir la transmisión de datos hacia el servidor.

El diseño de esta plataforma contempla tres etapas, el diseño del hardware de un módem USB especializado, el diseño del firmware del mismo y por último el diseño del software que se encuentra en el lado cliente tanto como en el lado servidor.

Según restricciones impuestas por el equipo de trabajo del proyecto elecciones, es imperativo que el módem USB especializado a desarrollar se instale dentro de los gabinetes de los computadores clientes que estarán en los locales de votación, esto es para evitar eventuales robos que pudiesen impedir el desarrollo del proceso eleccionario. Por otro lado, el acceso a las tarjetas SIM debe ser solamente dentro del interior del equipo, en otras palabras, no debe poseer ningún tipo de conector al cual pueda tenerse acceso desde fuera del gabinete. Sin embargo, la tarea de cambiar las tarjetas SIM por parte de los operadores tiene que ser lo más simple posible una vez que se tenga acceso al interior del computador que alberga el dispositivo de comunicación celular.

En este trabajo de título se desarrollará un prototipo para la plataforma hardware/ software mencionada. Este prototipo es con la finalidad de evaluar el desempeño del sistema y no contempla en sí el diseño definitivo que será utilizado en el proyecto elecciones. El hardware definitivo que sería desarrollado en una etapa posterior a este trabajo de memoria, diferiría en cuando a forma solamente y no a funcionalidad. El software en cambio, tanto del computador cliente como del servidor, es sólo una demostración para evaluar la plataforma y no es en sí el software definitivo a utilizar en el proyecto elecciones, ya que éste sería desarrollado por otros ingenieros.

Los objetivos que se deben verificar con la plataforma hardware/software a desarrollar son:

1. Detección del módem USB especializado por parte del sistema operativo Microsoft Windows en el computador cliente.

2. Detección de las tarjetas SIM instaladas en el módem USB especializado mediante el software desarrollado para el computador del cliente.

3. Búsqueda de tarjeta SIM con cobertura.

4. Conexión del cliente con el servidor.

5. Transmisión de un archivo de texto desde el cliente hacia el servidor.

6. Reanudación de la transmisión desde el último paquete recibido con éxito mediante la conmutación de la tarjeta SIM ante una desconexión o ausencia de cobertura.

1.2 Tecnología de conteo de votos utilizada en otros países

En países desarrollados como EE.UU, se utilizan diversas maneras de registrar los votos realizados. Los medios para votar más importantes son el voto manual y el voto electrónico de registro directo "DRE" [1]. Sin embargo, en algunos países ya se está incorporando la forma de votar mediante Internet para militares que se encuentran en el extranjero y quieren poder ejercer su derecho cívico [2].

El voto manual corresponde al sistema tradicional empleado en casi todo el mundo, donde el votante realiza una marca ya sea con un lápiz o mediante una perforación en un papel o cartón por el candidato deseado. Una vez realizada la selección, el votante introduce el voto en una urna la cual después es abierta para realizar la cuenta del total de votos en su interior. El conteo se puede realizar en forma manual, o mediante el uso de tecnología de escaneo óptico si las marcas son realizadas con lápiz o mediante la tecnología de escaneo electromecánico para aquellas realizadas con una perforación.

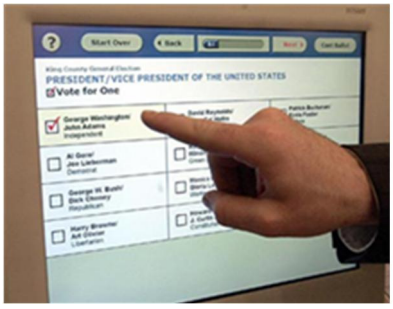


Fig. 1.1. Sistema de voto electrónico de registro directo "DRE"

El sistema de voto electrónico de registro directo "DRE" [3] que se puede apreciar en la fig. 1.1, es el sistema electrónico más utilizado en EE.UU donde alrededor del 25% de los votantes del país lo utilizan durante las elecciones [4]. Para votar mediante este sistema, el votante se acerca a una máquina que posee una pantalla sensible al tacto y mediante un dedo toca el área de la pantalla donde se encuentra la opción para un candidato en particular.

Los votos emitidos son almacenados en una memoria interna en el mismo dispositivo [3]. Estos datos pueden ser transmitidos en forma individual como también los totales por candidato a una localidad central donde se realizan los recuentos totales del país.

Las transmisiones realizadas por los equipos "DRE" pueden realizarse por distintos medios, como por ejemplo, líneas telefónicas fijas, redes LAN, Internet, etc. [5]

Tal como se señala en [6], en algunos países como Filipinas, se utilizan enlaces satelitales a Internet para comunicar áreas remotas no accesibles mediante la red alambrada y así poder enviar los datos de votación al servidor central.

Para el caso de los países desarrollados como EE.UU que tienen una excelente infraestructura en conectividad, la comunicación con áreas remotas no es un problema. Por otro lado la utilización de Internet como medio de transmisión de información para información altamente sensible como es el caso de los resultados de votación, tiene sus riesgos en cuanto a seguridad, lo que necesariamente implica el desarrollo de protocolos de comunicación muy seguros que utilicen avanzadas técnicas de encriptación.

Los datos enviados por cada sistema "DRE" son enviados a una central para realizar los conteos de votos finales para que próximamente sean informados al país.

1.3 Formas de transmisión de información desde los locales de votación

En Chile, en cada local de votación existen varias mesas, cuyo número, depende principalmente del territorio donde se encuentre el local. En aquellas localidades más centrales o donde existe una mayor cantidad de votantes, generalmente existe disponibilidad de línea física por lo que no se presenta problema alguno en el envío de datos al servidor. Cada cuarenta mesas se asigna un computador, por lo que en aquellos lugares como el Estadio Nacional donde hay cientos de ellas, generalmente hay varios computadores en el recinto. Las minutas de votaciones van siendo pasadas a un digitador que se encarga de ingresar los datos al computador al cual la mesa está asignada. La forma de transmitir está basada en el criterio 5 minutas/5 minutos que quiere decir que si se juntan 5 minutas antes de 5 minutos éstas son enviadas inmediatamente. En caso contrario si se superan 5 minutos desde la última transmisión se envía inmediatamente lo digitado hasta ese momento sin importar si se han completado 5 minutas o no.

En aquellas localidades remotas donde el número de votantes es reducido generalmente no existe disponibilidad de líneas físicas por lo que una vez cerradas las mesas, se llevan las minutas hacia el pueblo o ciudad más cercana donde se envían por fax a alguna localidad cercana donde existan digitadores. Luego estos digitadores mediante el uso de un computador conectado a una línea telefónica física envían los datos de los sufragios al servidor central.

1.4 Problemas asociados con los métodos actuales de transmisión

Para el caso de aquellos locales de votación que se encuentran en ciudades grandes donde existen una gran cantidad de votantes la transmisión de datos no es un problema. Sin embargo, aquellas localidades remotas que no cuentan con líneas de telefonía fija, que en Chile son bastantes, tienen el importante problema de transportar las minutas a la localidad más cercana que tenga fax para poder realizar el envío a los digitadores. Esto conlleva un retraso importante sobre todo en el momento que se encuentra el país cuando hay una elección en curso. Además el hecho de mover las minutas del local de votación implica un riesgo ya que eventualmente podrían extraviarse.

1.5 Comunicación celular y disponibilidad de cobertura

La escasez de líneas físicas en las zonas remotas se debe principalmente a la compleja geografía del país que hace muy difícil la instalación para las compañías telefónicas. Lo mismo sucede para el caso de las comunicaciones celulares. La presencia de numerosos sectores montañosos en el país hace difícil la comunicación entre sectores extremos ya que los enlaces entre antenas son interrumpidos por la geografía. Es por ello que para tener a Chile completamente conectado es necesario un esfuerzo importante por parte de las compañías telefónicas. Sin embargo, las localidades remotas al poseer una cantidad de usuarios muy reducidos no es un gran negocio para las compañías por lo que esas localidades tienen acceso a una cobertura celular restringida. Dependiendo del sector hay compañías que tienen una mayor intensidad de señal que otras, es por ello, que el módem USB especializado debe poseer la capacidad de comunicarse por más de una vía o compañía, que en otras palabras, se traduce a que el dispositivo permita alojar más de una tarjeta SIM.

Es importante tener en cuenta que a medida del transcurso de los años, la telefonía celular ha tenido un crecimiento muy importante que ha permitido que supere ampliamente en número de abonados de telefonía fija. Según indicadores provistos por la Subsecretaría de Transportes y Telecomunicaciones o Subtel el número de abonados móviles es de aproximadamente 24 millones [7] mientras que para el caso de los

abonados de líneas físicas es aproximadamente 3 millones [8]. Es por ello que es una buena idea pretender realizar las comunicaciones por la red celular ya que debido al número de usuarios su cobertura es mucho más amplia que en el caso de las líneas físicas convencionales.

1.6 Alternativas de solución disponibles en el mercado

Es natural hacerse la pregunta de si existe actualmente alguna solución en el mercado que permita resolver el problema planteado de comunicación entre el cliente y el servidor de una manera rápida y fácil sin tener que recurrir al diseño de un hardware a medida. La parte crítica de este trabajo es el módem USB especializado, por lo que en esta sección se analizarán las distintas alternativas de solución disponibles que puedan adaptarse a las necesidades del hardware a instalar en el computador cliente. Se efectuará una evaluación económica para una partida de 1.000 unidades para las mejores dos de las alternativas de solución presentadas en esta sección.

1.6.1 Teléfonos celulares utilizados como módem

Hoy en día, casi cualquier celular en el mercado es capaz de ser utilizado como módem. Por ello es posible pensar en utilizar uno de ellos para que cumpla la función del módem USB especializado.

La gran mayoría de los teléfonos celulares de hoy ofrecen una interfaz de comunicación para ser conectada a un computador. Las interfaces principalmente son USB por lo que requieren un cable mini USB estándar para su conexión. Sin embargo, cabe destacar que existen también algunas interfaces no universales que requieren de un adaptador especial para la conexión a un PC mediante USB.

Para que el teléfono celular pueda ser utilizado como módem se debe instalar un controlador suministrado por el fabricante que permita al sistema operativo su reconocimiento como un módem. Esto es válido también si se quiere que el sistema operativo lo reconozca como algún otro dispositivo (p. ej. cámara). Cabe destacar, que en algunos casos, cuando se quiere acceder al teléfono en modo de almacenamiento masivo no es necesaria la instalación de un driver especial.

Por otro lado, la gran mayoría de los fabricantes de equipos celulares de hoy en día, pretenden que la conexión del aparato a un computador sea realizada de la forma más transparente posible. Esto implica que se pueden utilizar los recursos del teléfono directamente desde el sistema operativo sin tener que utilizar software propietario. Por ejemplo al conectar el equipo al PC podemos revisar los archivos que contiene tal

como se acceden a los archivos de un disco duro. En otras palabras, el dispositivo es visto como un disco más del computador. Lo mismo sucede cuando es utilizado como módem, ya que bastará utilizar cualquier programa tipo terminal de comunicaciones (p. ej. Hyperterminal) para poder utilizarlo como tal.

Sin embargo, debido a que los teléfonos celulares son dispositivos multi-función o "composite device" según la jerga USB, es requerido que se especifique el modo de conexión hacia el computador. En otras palabras se requiere especificar bajo qué clase de dispositivo se requiere que se reconozca el equipo. En el caso del bus USB existen varias clases como por ejemplo [9]:

Tabla 1.1. Algunas de las clases USB a la que pueden pertenecer los dispositivos

Nombre de la clase	Descripción
"Mass storage device class (MSC)"	Utilizada principalmente para todo aquel dispositivo de almacenamiento masivo que se quiere que sea reconocido por el sistema operativo como una unidad de disco adicional. Los discos duros externos y "pendrives" son ejemplos de hardware que va asociada a esta clase.
"Human interface device (HID)"	Utilizada para dispositivos como teclados, "mice" (ratones), punteros, entre otros.
"Communications device class (CDC)"	Utilizada principalmente para módems.
"Video device class"	Utilizada para cámaras WEB, etc.

Debido a lo explicado anteriormente, algunos celulares al ser conectados a un puerto USB de un PC, solicitan al usuario a través de sus pantallas que seleccione el modo en que debe operar con el computador. En el caso de los teléfonos Nokia de la serie 5200 [10] y 5300 [11], mostrados en la fig. 1.2, pueden funcionar como sistema de almacenamiento masivo o como módem y la selección debe ser ingresada por el teclado del equipo celular.

Fig. 1.2. Teléfonos Celulares Nokia 5200 y 5300

Para este proyecto es fundamental que el módem USB especializado sea una PCB interna, es decir que se encuentre dentro del gabinete del computador como cualquier otro hardware. Esta es una restricción de seguridad impuesta por el proyecto elecciones, por lo que basada en ella, los teléfonos de este

tipo quedan automáticamente descartados ya que no se tendrá acceso al teclado. Es claro que estos equipos no están concebidos para ser utilizados como un dispositivo interno en un computador.

Otra desventaja clara es que estos equipos no poseen la cualidad de tener tecnología multi tarjeta SIM para la extensión de cobertura.

Fig. 1.3. Teléfonos "smartphones" con tecnología dual sim.

En el mercado existen "smartphones" que son capaces de contar con dos tarjetas SIM tales como los equipos Nokia [12] mostrados en la fig. 1.3. La conmutación de la tarjeta SIM se puede hacer en forma automática cuando la que se encuentra en servicio pierde cobertura o cuando el usuario lo desee, mediante una aplicación especial instalada en el equipo, la cual es necesaria ser operada mediante el teclado. Para estos equipos al igual que los Nokia de la serie 5200/5300 el modo de operación USB debe seleccionarse mediante el teclado lo que claramente viola la restricción de que el equipo debe ser interno.

Ahora en el caso que si se pudiesen controlar desde un computador sin intervención del usuario mediante el teclado, existe el grave problema de cobertura que se produciría si el equipo celular estuviese dentro de un gabinete metálico. Esto se debe a que el gabinete actúa como una jaula de Faraday, impidiendo así la recepción y transmisión de ondas electromagnéticas desde y hacia el teléfono. La tecnología celular actual funciona con frecuencias de 850, 900 y 1900MHz por lo que para estas frecuencias los gabinetes metálicos se comportan como una jaula de Faraday muy efectiva. Por otro lado, la potencia RF emitida desde el celular que es importante, puede afectar el funcionamiento de la electrónica dentro del computador, produciendo graves problemas de compatibilidad electromagnética y una mala estabilidad del computador que lo aloja. Por estas razones se descarta desde ya la utilización de teléfonos celulares convencionales para este proyecto.

Las ventajas y desventajas para la alternativa 1 se resumen a continuación:

Ventajas:

- No es necesario diseñar hardware ni firmware.
- Solución disponible en el mercado.
- Existen algunos teléfonos celulares con capacidad para 2 tarjetas SIM.

Desventajas:

- Violan la restricción de que debe ser un módem interno, ya que no puede ser instalado dentro del gabinete del computador que será cliente. Esto se debe a que es necesario el ingreso de ciertos parámetros mediante el teclado lo cual no es posible cuando está dentro del gabinete.
- El hecho de tener que acceder al teclado para poder ser utilizado como módem, permite que se tenga acceso a las tarjetas SIM lo que viola la restricción de seguridad, ya que éstas quedarían disponibles desde el exterior.
- No todos los celulares poseen tecnología multi tarjeta SIM.
- Cobertura celular gravemente afectada debido a que funcionaría dentro de un gabinete metálico.
- Interferencia con los demás circuitos que conforman el computador podrían producir graves problemas de compatibilidad electromagnética y fallas de funcionamiento.

1.6.2 Módulos celulares

También en el mercado existen módems GSM/GPRS compatibles con bus PCI y USB, cualidades que permiten considerarlos como eventuales alternativas para el módem USB especializado.

Los módulos celulares PCI están diseñados para ser instalados en el interior de un computador por lo cual podría ser una solución para el problema planteado. Sin embargo la gran desventaja es que no existe un módem de este tipo con tecnología multi tarjeta SIM.

Por otro lado, para el caso de los módulos USB así como también para los demás dispositivos USB están pensados para ser conectados en forma externa por lo que quedarían descartados por violar la restricción de seguridad. La única excepción es el módem provisto por Digicom que se menciona más abajo, sin embargo no soporta tecnología multi tarjeta SIM.

Las alternativas disponibles en el mercado que más se aproximan a la solución requerida, son provistas por la compañía lituana Teltonika y la compañía italiana Digicom. Dentro de los productos que comercializan se encuentran los módems Teltonika PCI/G10 [13] y USB/H7.2 [14] mostrados en las figuras 1.4 y 1.5 respectivamente. Por

su lado Digicom ofrece el producto 3G Modem USB Internal [15] mostrado en la figura 1.6.

Fig. 1.4. Teltonika Modem PCI/G10.

Fig. 1.5. Teltonika Modem USB/H7.2.

Fig. 1.6. Digicom 3G Modem USB Internal

En las figuras 1.4 y 1.6 que corresponden a los módulos celulares internos, se puede apreciar claramente que la tarjeta SIM puede ser extraída desde fuera del gabinete. Esto se debe a que exponen un conector eyector accesible hacia fuera lo cual es una desventaja respecto a la seguridad. Dentro de los módulos celulares presentados, el más apropiado dentro de esta alternativa sería el 3G Modem USB Internal de Digicom [15]. Por lo que para poder dar solución al problema mediante su utilización habría que pensar en 2 módems de este tipo para poder implementar la característica de poder alojar 2 tarjetas SIM. El costo de cada módem incluyendo la antena es de 192 euros lo que equivale a aprox. 254 USD [16]. Para una cantidad de 1.000 computadores clientes, se necesitarían 2.000 módems. Asumiendo un descuento por volumen de un 15% del valor FOB, el precio por unidad quedaría en 215,9 USD. El costo por el transporte del contenedor más los honorarios del agente de aduanas se asumen en 10.000 USD como peor caso.

Fig. 1.7. Vista posterior de un gabinete de computador donde se muestra en destacado en rojo la zona donde las tarjetas SIMs de ambos módems estarían expuestas

Para no violar la condición de seguridad en cuanto a que las tarjetas SIM estén expuestas desde fuera, se deberá para esta alternativa, incorporar algún tipo de tapa metálica por fuera del gabinete del computador que permita bloquear el acceso a las tarjetas SIM. Para ello, se podría pensar en que la tapa metálica estuviese adosada al gabinete por fuera. Si bien esta solución permitiría que el cambio de tarjeta SIM fuese fácil, la gran variedad de gabinetes distintos, hace inviable esta idea. Esto

hace necesaria la confección de una tapa metálica exclusiva para cada módem.

En la fig. 1.7 se puede apreciar la vista de un gabinete convencional de un computador donde se han destacado en color rojo las ranuras donde estarían los conectores eyectores de ambos módems USB. La tapa metálica a confeccionar debería ser aproximadamente del tamaño de una ranura y deberá anclarse al módem una vez ya instalada la tarjeta SIM en el conector. Luego el conjunto comprendido por el módem más la tarjeta SIM y la tapa metálica serían instalados dentro del gabinete del computador. La desventaja evidente que se puede apreciar es la necesidad de desinstalar el módem del interior del computador y a su vez extraer la tapa metálica para poder tener acceso a la tarjeta SIM. Para esto los mantenedores de los computadores necesariamente deberán apagarlos previamente a cambiar la tarjeta SIM. El costo unitario estimado de la fabricación de la tapa metálica para cantidades de 2.000 unidades en la industria nacional es de aproximadamente 1 USD.

La tabla 1.2 muestra un resumen de los costos asociados para esta alternativa considerando dotar a 1.000 computadores clientes con la conectividad requerida. Se incluye el costo del transporte en contenedor, seguro, agente de aduanas y arancel ad valorem. De esta tabla se desprende que el costo neto puesto en Chile de cada par de módems es de 477,462 USD en cantidades de 1.000 unidades. Como se requieren dos módems por cada computador cliente, ya que cada uno permite alojar solamente una tarjeta SIM, el costo total de la solución para 1.000 computadores clientes (2.000 módems en total) asciende a 477.463 USD.

Tabla 1.2. Costo neto unitario y total para dotar a 1.000 computadores clientes con la conectividad requerida

Ítem	Cantidad (unidades)	Costo Unitario (USD)	Costo Total (USD)
Digicom 3G Modem USB Internal (FOB Europa). El costo unitario indicado es para 2 módems (215,9 USD x 2)	1.000	431,8	431.800
Seguro (2% del FOB)	-	8,636	8.636
Transporte en contenedor a Chile más honorarios agente de aduanas	-	10	10.000
Total CIF	**-**	**450,436**	**450.436**
Total CIF + Arancel Ad Valorem (6 % del CIF)	-	477,462	477.463
	Total	**477,462**	**477.463**

A modo de resumen para la alternativa 2, sus ventajas y desventajas se muestran a continuación:

Ventajas:

- No es necesario diseñar hardware ni firmware.
- Módem disponible en el mercado.
- Los módulos disponibles como tarjeta de circuito impreso están diseñados para ser instalados dentro de un gabinete.
- Conector para antena accesible desde fuera del gabinete.
- Reconocidos automáticamente como módem por el sistema operativo una vez instalados.

Desventajas:

- Necesidad de confeccionar una placa metálica adosada al módem con la finalidad de impedir el acceso a la tarjeta SIM.
- Se requieren 2 módems, ya que no existen módulos internos con más de una tarjeta SIM. Esto implica un mayor costo.
- Acceso a la tarjeta SIM una vez apagado el computador, desinstalado el módem de la ranura y extraída la placa metálica.

1.6.3 Diseño e implementación de un módem USB especializado

La tercera y última alternativa corresponde al diseño hardware y software de un módem USB especializado. Este módem tiene la ventaja de incorporar todos los requisitos planteados por el proyecto elecciones, aspecto que no se cumple al utilizar soluciones disponibles en el mercado.

Otra ventaja que tiene este dispositivo a desarrollar es que no será detectado como un módem convencional por el sistema operativo, sino como un hardware propietario mediante la carga de un driver especializado. Esto permite hacer que el sistema sea más seguro por el hecho de utilizar una aplicación propietaria para ser utilizado. De esta manera el protocolo de comunicación entre el módem y el computador cliente puede ser implementado a bajo nivel en el microcontrolador.

Para la evaluación económica del proyecto, se realiza una estimación de los costos de desarrollo del hardware y firmware del módem USB especializado. Para ello se da

una descripción preliminar breve de los elementos que constituirían el módem, ya que la ingeniería en detalle no está disponible durante esta etapa de evaluación.

En sí el módem USB especializado se compone por un módulo celular GSM, un microcontrolador de la serie MSP430, un multiplexor de dos tarjetas SIM y una fuente de switching. Los costos FOB de estas componentes se pueden apreciar en la tabla 1.3. Los valores para la fuente de switching conformada por el controlador LM25576 son basados en la actual experiencia del autor por el recurrente uso de dicho bloque funcional en otros proyectos. Las componentes varias que generalmente son condensadores, resistores y otros elementos pasivos como ferritas y conectores se valorizan en 10 USD por módem. Finalmente el total FOB por concepto de componentes es de aproximadamente 54 USD.

Para la instalación de estas componentes se requiere la fabricación de una PCB y a su vez realizar el proceso de ensamblado o montaje de la misma. La experiencia del autor en la empresa Electrosoft Ingeniería ha demostrado que en el 90% de los diseños, la manufactura de PCBs es aproximadamente el 20% del costo de las componentes cuando se fabrican volúmenes grandes (típicamente arriba de 100 unidades). También es posible corroborar que se cumple que el servicio de ensamblado tiene un costo aproximado de un 15% del costo de la PCB más sus componentes.

Tabla 1.3. Costo FOB de las componentes que componen el módem USB especializado

Ítem	Cantidad (unidades)	Costo Unitario por módem (USD)	Costo Total (USD)
Módulo Celular SIM900	1.000	25 [17]	25.000
2 Sockets Tarjetas SIM	1.000	1,52 [18]	1.520
uC MSP430F5436	1.000	5,45 [19]	5.450
Multiplexor CD4053	1.000	0,16275 [20]	162,75
Fuente switching (LM25576 más componentes)	1.000	12	12.000
Set componentes varias	1.000	10	10.000
TOTAL COMPONENTES	**-**	**54,133**	**54.133**

Tabla 1.4. Costo FOB de las PCBs incluyendo servicio de ensamblado

Ítem	Cantidad (unidades)	Costo Unitario por módem (USD)	Costo Total (USD)
PCB sin componentes (aprox. 10% de componentes)	1.000	5,4133	5.414
Servicio de ensamblado (aprox. 15% de PCB + componentes)	1.000	8,12	8.120
TOTAL PCB + ENSAMBLADO	**-**	**13,53**	**13.534**

Además, para cada módem se requieren accesorios como una antena GSM más un cable coaxial con conector que será conectado directamente al módulo celular. En la tabla 1.5 se detallan los costos de estos accesorios.

Tabla 1.5. Costo FOB de los accesorios

Ítem	Cantidad (unidades)	Costo Unitario por módem (USD)	Costo Total (USD)
Antena GSM ángulo recto	1.000	5,67 [21]	5.670
Cable coaxial SMA	1.000	3,828 [22]	3.828
TOTAL PCB + ENSAMBLADO	**-**	**9,498**	**9.498**

Mediante las tablas 1.3, 1.4 y 1.5 se calcula el costo FOB de cada módem USB especializado. Este costo corresponde a la suma de los totales de las tres tablas mencionadas el cual es 77,161 USD. Se asume un margen de seguridad de un 20% que permite amortiguar los cambios que puedan surgir durante el desarrollo, por lo que el costo FOB final por unidad es aproximadamente de 92,6 USD.

Mediante la información del costo FOB de cada módem USB especializado, se construye la tabla 1.6 que determina el costo neto por unidad puesta en Chile. Basado en la experiencia del autor, se asume un costo de transporte más agente de aduana de 5.000 USD.

Tabla 1.6. Costo neto unitario y total para una partida de 1.000 unidades del módem USB especializado puesto en Chile

Ítem	Cantidad (unidades)	Costo Unitario (USD)	Costo Total (USD)
PCB Ensamblada más accesorios y margen de seguridad	1.000	92,6	92.600
Seguro (2% del FOB)	-	1,852	1.852
Transporte en contenedor a Chile más honorarios agente de aduanas	-	5	5.000
Total CIF	**-**	**99,452**	**99.452**
Arancel Ad Valorem (6 % del CIF)	-	5,967	5.967
Horas de ingeniería de desarrollo no recurrente NRE (costo fijo)		**15.923**	**15.923**
	Total	16.028	121.342

Para implementar la solución se requieren dos meses-ingeniero. Asumiendo el costo de una hora-ingeniero en una unidad de fomento y su vez considerando que se trabaja 45 horas por semana y 4 semanas por mes, el costo total por concepto de diseño es 360 UF. Asumiendo el valor de la unidad de fomento en 23.000 pesos chilenos y el valor del dólar en 520 pesos chilenos, el costo ingeniero en dólares americanos asciende a aproximadamente 15.923 USD. Este es un costo fijo que en otras palabras no depende del número de unidades producidas por lo que debe incluirse una sola vez. En la jerga de ingeniería es llamado NRE (Non-Recurring Engineering). El costo final para 1.000 computadores clientes es de 121.342 USD, lo que implica que el costo de dotar a un computador con la conectividad requerida es de 121,342 USD.

Ventajas:

- Posee tecnología multi tarjeta SIM. Permite alojar 2.
- Instalación interna lo que implica que el acceso a la tarjeta SIM será limitado.
- Conexión interna al bus USB.
- Costo muy apropiado, permitirá amplios márgenes de utilidad.
- Conector para antena accesible desde fuera del gabinete.
- No reconocido como un módem convencional, requiere aplicación propietaria para operar.
- Protocolo de comunicaciones con módem USB especializado diseñado a medida a bajo nivel en el microcontrolador.

Desventajas:

- Diseño completo de hardware y firmware para el módem USB especializado además del software para el computador cliente.
- Desarrollo e implementación requieren un periodo de tiempo mayor que el tomar una solución ya disponible en el mercado.

1.7 Solución propuesta

Tal como se presentó en las subsecciones anteriores, las alternativas disponibles en el mercado que podrían hacer de hardware de comunicaciones para el computador cliente no pueden cumplir con todos los requerimientos impuestos por el proyecto elecciones en cuanto a seguridad y facilidad de uso. Estos requerimientos se resumen en la tabla 1.7.

Por otro lado, de las tres alternativas presentadas la primera tuvo que ser descartada por el hecho de tener que instalar el teléfono celular dentro del gabinete. El instalarlo dentro del gabinete produciría interferencias importantes en los circuitos del computador generando un eventual problema de estabilidad. Además, el teléfono celular una vez dentro del gabinete metálico ve reducida la intensidad de señal recibida en forma importante.

Tabla 1.7. Requerimientos que deben cumplirse por el hardware de comunicaciones a ser instalado en el computador cliente

Requerimiento	Descripción
1) Instalación Interna	El hardware debe ser instalado dentro del gabinete del computador cliente
2) Más una tarjeta SIM	Debe poseer la cualidad de poder instalar más de una tarjeta SIM. Para este trabajo de título deben alojarse 2.
3) Acceso interno a tarjetas SIM	El acceso a las tarjetas SIM debe solamente dentro del gabinete, es decir no puede exponer el módem ningún conector hacia fuera.

La segunda alternativa es posible si se utilizan dos módems USB internos. Se requieren dos por computador cliente ya que ninguno soporta multi-tarjeta SIM. Esto produce un costo para 1.000 computadores clientes de aproximadamente 477.463 USD. En otras palabras por computador cliente los costos solamente del hardware ascienden a 477,4 USD.

La tercera alternativa sin duda es la más atractiva ya que cumpliría con todos los requisitos mostrados en la tabla 1.7, por el hecho de ser un sistema diseñado a medida. Además el costo por computador cliente según la tabla 1.6 es de aproximadamente 121,4 USD para cantidades de 1.000 unidades. El costo total para esta alternativa es aproximadamente un 25% del costo de la segunda, lo cual permitirá obtener un margen de utilidades aceptable. Todas estas razones hacen que la tercera alternativa sea la más apropiada y por ello se selecciona.

En las subsecciones siguientes se pretende mostrar la estructura completa de la plataforma de comunicaciones entre cliente y servidor que se abordará durante este trabajo de título. El bloque más importante de la plataforma de comunicaciones es el módem USB especializado cuyo diseño en detalle se presentará en los capítulos 2, 3, Anexo 1 y 2.

1.7.1 Esquema de implementación de la plataforma de comunicaciones

Para una mejor comprensión se definen dos computadores para la conexión. El primero, el **PC**_A, el cual corresponde al computador cliente y que se encuentra en el

local de votación. El segundo, definido como **PC_B**, corresponde al computador que realiza las funciones de servidor de acopio de datos.

En un puerto USB del PC_A se conectará el módem USB especializado que permite la comunicación mediante la red celular, en otras palabras, transforma un puerto USB en un puerto de comunicación inalámbrica mediante la red celular. Por otro lado, el PC_B estará conectado mediante un módem PCI convencional a la red de telefonía fija (POTS) tal como se muestra en la figura 1.8.

Para comunicarse ambos computadores, el PC_A disca el número de teléfono fijo en el cual se encuentra el módem del PC_B. En el caso de que no haya cobertura con la tarjeta SIM 1 (p.ej. perteneciente a Entel) el sistema conmutará a la tarjeta SIM 2 (p.ej. de Movistar). De esa manera podemos tener mayor probabilidad en lo que respecta a la conectividad de ambos computadores.

En el caso de que la conexión se caiga por algún motivo, se cambiará de tarjeta SIM y se restablecerá la comunicación con el PC_B. Una vez restablecida la comunicación se resume la transmisión de datos desde el último paquete transmitido con éxito.

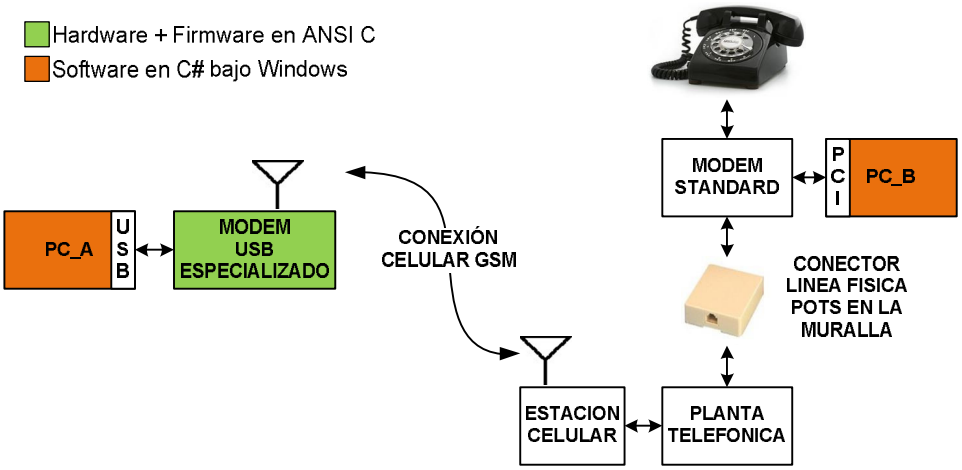


Fig. 1.8. Esquema básico de conexión entre cliente y servidor

La figura 1.8 muestra un diagrama donde se pueden apreciar los distintos bloques que participan en la conexión. Para este trabajo de memoria el desarrollo se centrará en los bloques destacados de color verde y naranja, siendo el de color verde el tema principal a desarrollar. El significado de cada color puede verse en la esquina superior izquierda de la misma figura. El bloque correspondiente al módem USB especializado se encuentra destacado en verde para indicar que tanto el hardware como el firmware serán diseñados en el transcurso de esta memoria. Los bloques que corres-

ponden a los computadores PC_A y PC_B se encuentran destacados en naranja para indicar que se diseñará el software de alto nivel solamente. La finalidad del software a implementar en ambos computadores es permitir establecer la comunicación y administrar la transmisión y recepción de datos para así poder evaluar el hardware y firmware del módem USB especializado. El lenguaje a utilizar para estos dos bloques es Microsoft Visual C#.

CAPÍTULO 2

2. Diseño hardware del módem USB especializado

Este capítulo tiene por objetivo mostrar la estructura del módem USB especializado a diseñar. Además se presentarán las características de las distintas componentes que lo componen. Los diagramas esquemáticos y aspectos del layout de la PCB se muestran en el Anexo 2.

2.1 Estructura del circuito a construir

En el capítulo anterior se presentó el esquema básico de conexionado entre ambos computadores. En esta sección se realizará una descripción en bloques del hardware de comunicaciones en el PC_A correspondiente al módem USB especializado.

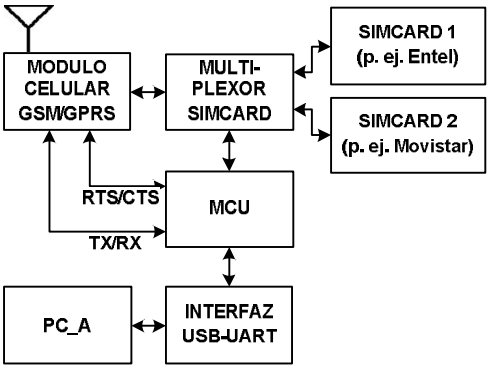


Fig. 2.1. Esquema básico del módem USB especializado y su conexión con el computador cliente

El hardware a diseñar se aprecia en el diagrama de la figura 2.1 el cual consta de varios componentes fundamentales que cumplen una función específica.

El módulo celular GSM/GPRS contiene toda la infraestructura hardware/software necesaria para establecer comunicaciones mediante las normas que rigen a los teléfonos celulares. Poseen generalmente una o más UARTs para comunicación con algún otro dispositivo, como por ejemplo, un microcontrolador.

La interfaz USB/UART es necesaria para poder conectar el microcontrolador al PC_A. Genera un puerto COM VIRTUAL [23], que básicamente es una UART a partir de un puerto USB convencional. La señalización tiene niveles entre 0V y 3.3V por lo que no es del tipo RS232.

El multiplexor de tarjetas SIM consiste en un circuito integrado que conmuta varias señales en forma simultánea. Será utilizado para conmutar las señales de una tarjeta SIM a otra. Se han realizado pruebas satisfactorias con el MUX/DEMUX de 2 canales CD4053B [24].

Las tarjetas SIM o SIMCARDs 1 y 2, son proporcionadas por las compañías telefónicas y contienen la identidad del subscriptor (IMSI), información de autentificación y cifrado, información temporal relacionada con la red celular, listas de servicios para el usuarios, un número de identificación personal (PIN) y un número único que identifica a la tarjeta SIM llamado ICCID. Además puede contener datos de contactos que se hayan guardado desde algún equipo celular. [25]

El MCU o Microcontrolador es un computador embebido que contiene el software necesario para controlar el módulo celular y al multiplexor de tarjetas SIM. Para discar sobre el módulo, el PC_A envía los datos al microcontrolador. Luego el microcontrolador envía los comandos y datos adecuados al módulo celular para realizar la comunicación.

En las siguientes secciones se entrega una descripción más detallada de cada uno de los bloques funcionales que componen el hardware a desarrollar.

2.2 Módulo celular SIM900

Dentro de los módulos celulares a utilizar, se selecciona el SIM900TEC [26] de la empresa SIMCOM debido a su bajo precio y a que ya fue utilizado por el autor en proyectos anteriores.

Fig. 2.2. SIM900TEC

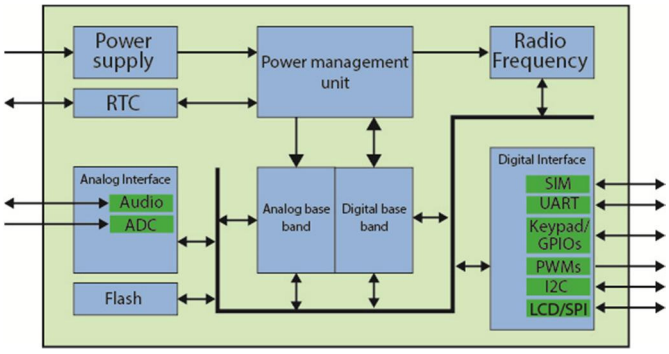


Fig. 2.3. Diagrama funcional del módulo SIM900TEC

El SIM900TEC, mostrado en la figura 2.2, es un módulo celular "cuatribanda", ya que puede funcionar con las 4 bandas utilizadas alrededor del mundo las cuales corresponden a 850, 900, 1800 y 1900 [MHz]. En Chile se utiliza la banda de 1900[MHz] [27]. Su datasheet puede ser encontrado en [28].

Además, el módulo tiene varios periféricos embebidos como por ejemplo: las interfaces para tarjeta SIM y de audio analógico, reloj de tiempo real (RTC), generador PWM, GPIOs y conversor análogo digital (ADC) entre otros. Respecto a los periféricos

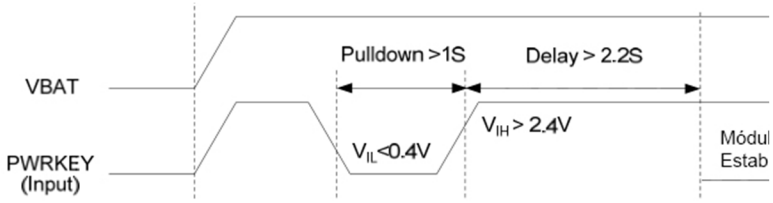


Fig. 2.4. Diagrama de tiempo de encendido del SIM900

de comunicaciones el módulo soporta los estándares más comunes los cuales corresponden a UART, I2C y SPI. En este proyecto se utilizará el periférico UART del SIM900TEC, destinado a la comunicación con el microcontrolador.

El módulo para ser encendido requiere que se le envíe un pulso activo bajo en el pin PWRKEY el cual debe tener un ancho mayor a 1[s]. Una vez finalizado el pulso, después de un tiempo de 2.2 [s] el módulo se encuentra estable y listo para operar. El diagrama de tiempo de encendido se puede apreciar en la figura 2.4.

La forma para producir el pulso puede ser mediante un pulsador en caso que el encendido se realice en forma manual, o mediante un circuito recomendado por el fabricante que emplea un transistor BJT NPN. El transistor es operado por una puerta GPIO del microcontrolador. El esquema del circuito se muestra en la figura 2.5.

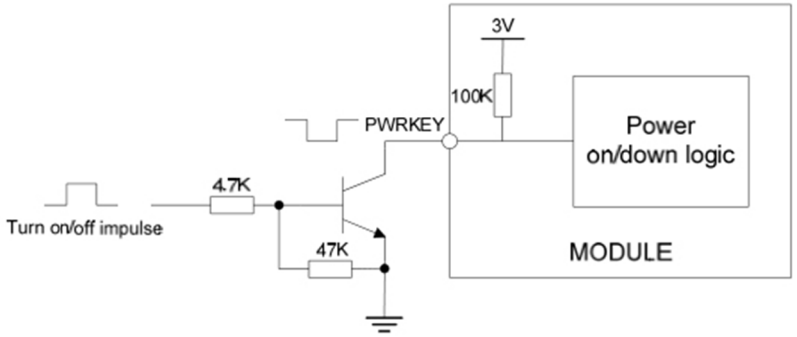


Fig. 2.5. Circuito de encendido del SIM900

Es importante destacar que la fuente de alimentación del módulo debe poder entregar hasta un máximo de 2 amperes en forma transitoria tal como se muestra en la figura 2.6, donde el voltaje no caiga más de 300[mV]. Estos consumos transitorios tienen cantos de subida abruptos y se producen cuando el módulo está transmitiendo. Es por ello, que para el correcto funcionamiento del módulo se tomen estas consideraciones tanto en el diseño de la fuente como en el layout de la PCB. Una forma de mejorar la respuesta de la fuente es agregar condensadores de bypass muy cerca de módulo. En el Anexo 1, se comentan algunos conceptos relacionados con este tema.

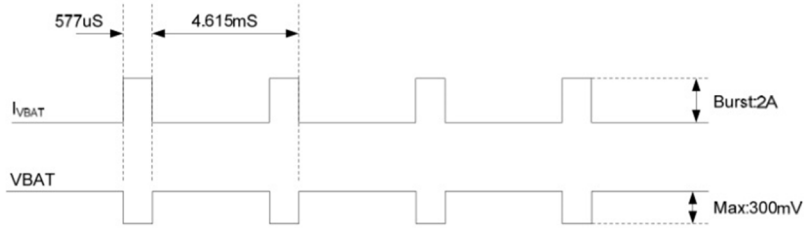


Fig. 2.6. Corrientes transitorias requeridas por el módulo.

2.3 Sistema de conmutación de tarjetas SIM

El SIM900 posee una interfaz para solamente una tarjeta SIM, por lo que será necesario implementar un multiplexor para poder alojar dos. Se utiliza en principio la idea presentada en [29], la cual es modificada para esta aplicación. El circuito correspondiente al conmutador se muestra en la figura 2.7.

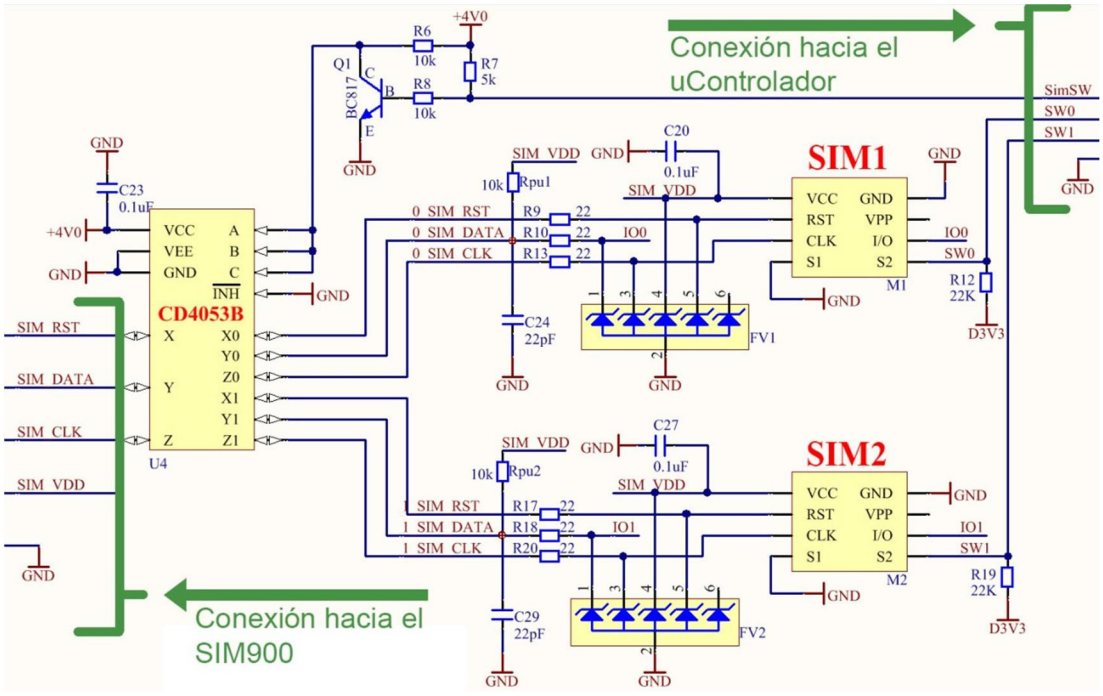


Fig. 2.7. Circuito conmutador de tarjetas SIM

El integrado CD4053B [24] es un multiplexor/demultiplexor de dos canales 0 y 1 con tres entradas cada uno. Los canales 0 y 1 están compuestos por las señales X0, Y0, Z0 y X1, Y1, Z1 respectivamente. El canal común está compuesto por las señales X, Y, Z. El mux/demux se controla por las entradas A, B, C y INH de acuerdo a la tabla de verdad que se muestra en la tabla 2.1. La señal INH tiene por finalidad desactivar el ca-

Tabla 2.1 Tabla de verdad del CD4053

INH	C	B	A	Z, Y, X
0	0	0	0	Z0, Y0, X0
0	0	0	1	Z0, Y0, X1
0	0	1	0	Z0, Y1, X0
0	0	1	1	Z0, Y1, X1
0	1	0	0	Z1, Y0, X0
0	1	0	1	Z1, Y0, X1
0	1	1	0	Z1, Y1, X0
0	1	1	1	Z1, Y1, X1
1	d	d	d	NADA

nal común por lo que se conecta a GND en el circuito de la figura 2.7 para tenerlo permanentemente activado.

Para que el circuito conmute de tarjeta SIM es necesario conmutar las señales que provienen del SIM900 hacia la SIM1 o hacia la SIM2. Esto implica que los únicos casos que interesan para el multiplexor son los que se encuentran destacados en verde en la tabla 2.1, por ello es que se unen las entradas de control A, B y C. El control del CD4053 se realiza por el microcontrolador a través de la señal **SimSW** que opera el transistor Q1. Este transistor cumple la función de un conversor de nivel ya que los umbrales lógicos del microcontrolador no son iguales a los utilizados en el circuito multiplexor.

A su vez este circuito permite la detección de qué tarjeta SIM se encuentra instalada. Para su detección se utilizan "sockets" que contienen un "switch" de presencia mecánico. Cuando se inserta una tarjeta en el "socket" SIM1 el pin S1 y S2 del mismo se cortocircuitan a través de la tarjeta, llevando a **SW0** a GND, lo que produce que el microcontrolador detecte la inserción. El resistor de "pull-up" permite que cuando no haya tarjeta instalada en el "socket" SIM1, el microcontrolador pueda leer un valor estable en la señal **SW0**. Lo mismo es válido para el caso del "socket" SIM2.

2.4 Controlador USB FT232R

Para poder establecer una comunicación USB con el computador cliente, se utiliza el controlador USB FT232R de FTDI [30].

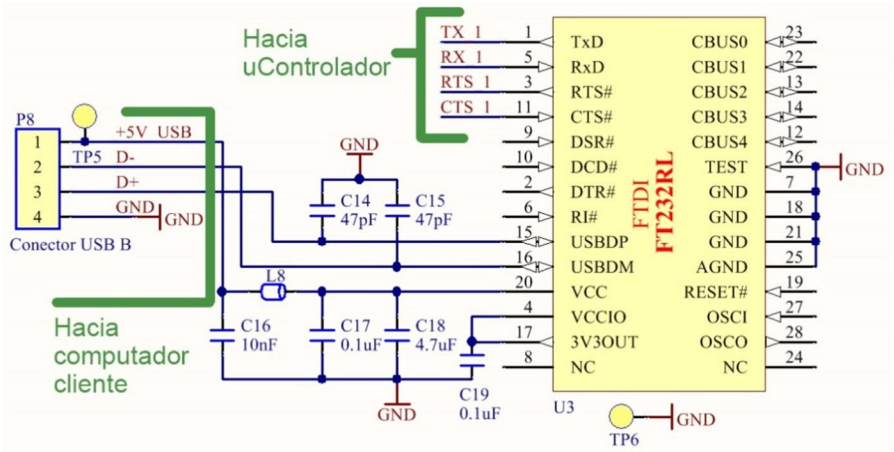


Fig. 2.8. Circuito del controlador USB FT232R

Este chip por un lado posee una UART la cual va conectada al microcontrolador y por el otro lado termina en un conector USB para ser conectado en el computador cliente. El circuito implementado se muestra en la figura 2.8.

El chip FT232R posee hardware y firmware que permite manejar completamente el protocolo USB sin tener que implementar el protocolo en el microcontrolador. Debido a esta razón es un chip ampliamente utilizado en la industria para dotar a los sistemas embebidos de conectividad USB en forma simple.

2.5 Microcontrolador MSP430

Existen una variedad bastante grande de microcontroladores en el mercado que son apropiados para este proyecto. Sin embargo se elige el microcontrolador [31] de 18[MHz] de Texas Instruments debido a la experiencia que el autor posee con este dispositivo. La figura 2.9 muestra un digrama funcional que contiene todos lo perifericos integrados en el chip.

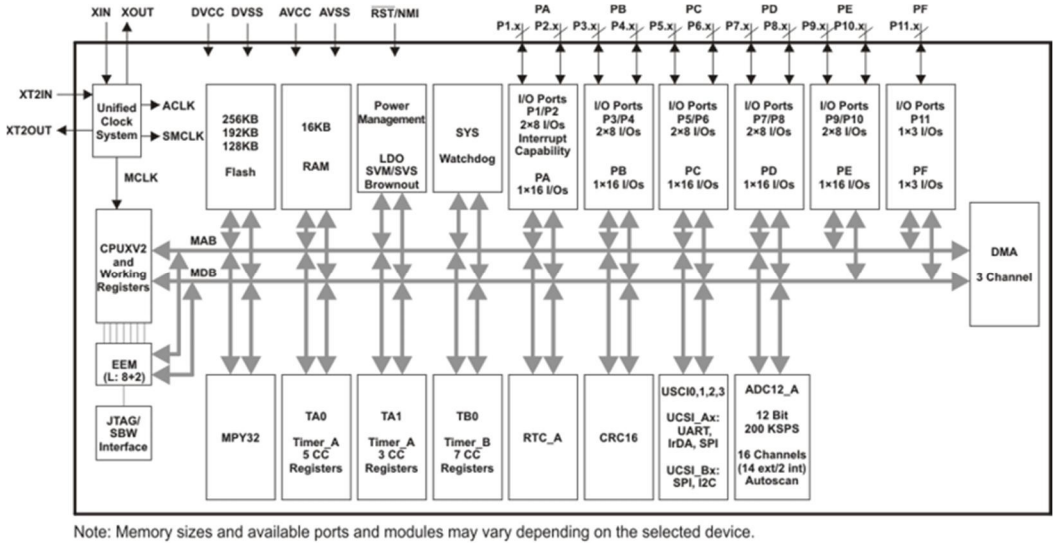


Fig. 2.9. Diagrama funcional del MSP430F5436

El microcontrolador está constituido por una CPU de 16bits, posee 192KBytes de flash y 16KBytes de RAM. Está dotado con 11 puertos GPIO donde los puertos 1 y 2 aceptan interrupciones. Tiene 3 timers, TA0, TA1 y TA2 con 5, 3 y 7 módulos de captura/comparación respectivamente. Además posee un multiplicador por hardware de 32bits, un conversor análogo-digital de 12 bits, un reloj de tiempo real, un bloque para efectuar comprobaciones de redundancia cíclica y 4 USCIs (0,1,2 y 3) (Universal Serial Communication Interface).

Este periférico de comunicaciones tiene la capacidad de funcionar en forma asincrónica en modo UART, o en forma sincrónica, en modo SPI o I2C.

Para este diseño se utilizarán 2 USCIs en modo UART. Donde la USCI 0 estará conectada al puerto de comunicaciones del módulo celular SIM900, mientras que la USCI 1 estará conectada al controlador USB FT232R tal como se muestra en la figura 2.10.

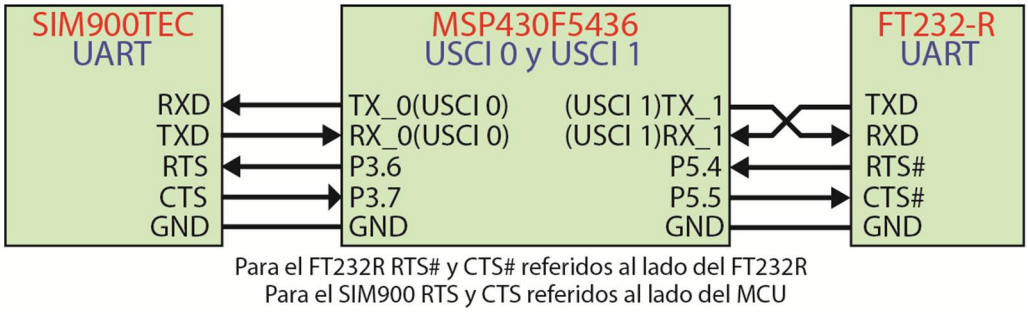


Fig. 2.10. Esquema de conexión entre el MSP430F5436, SIM900TEC y FT232R

Es importante no confundir la dirección de las señales ya que los fabricantes suelen definir los nombres de éstas respecto al microcontrolador o respecto al chip, lo cual puede generar confusión. Es por ello que la figura 2.10 incluye flechas en las conexiones para comprender fácilmente las direcciones de las señales.

Las señales RX y TX requeridas para la conexión hacia el SIM900 y el FT232R se obtienen de los periféricos de comunicaciones USCI 0 y USCI 1 respectivamente.

Las señales RTS y CTS se realizan en el microcontrolador mediante puertos GPIO convencionales. Estas señales conforman el control de flujo por hardware cuya finalidad es garantizar que el microcontrolador, el módulo SIM900 o el controlador FT232R estén siempre listos para recibir información con la finalidad que no se pierdan datos.

2.6 Módulo de alimentación

Para alimentar la electrónica se utiliza por motivos de eficiencia una fuente switching de 4[V] de topología Buck basada en el controlador LM25576 [32]. La razón de su utilización, es para dar versatilidad en cuanto a la alimentación del circuito ya sea del bus USB o directamente de la fuente de alimentación del computador mediante 9[V] o 12[V].

Una fuente de switching a diferencia de las lineales tradicionales, permite mantener la eficiencia en forma casi independiente del voltaje de entrada. En algunos momentos, tal como se mostró en la figura 2.6, el SIM900 tiene requerimientos de

corriente de 2[A] por lo que la fuente debe estar preparada para estos consumos. La energía que puede suministrar el bus USB depende principalmente de los elementos que estén conectados al él por lo que podría ser insuficiente para las condiciones de alto consumo. El circuito de la fuente switching implementado se muestra en la figura 2.11. El diseño se realizó mediante la plataforma online WEBENCH Power Designer® de Texas Instruments [33].

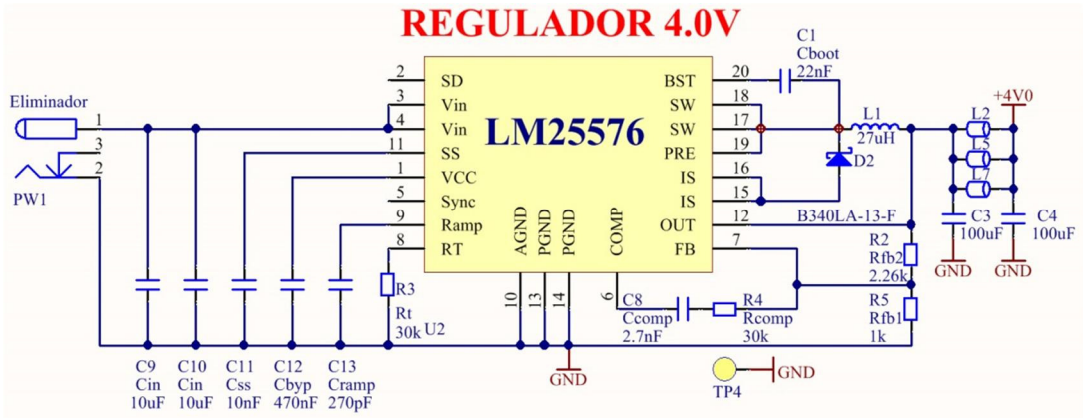


Fig. 2.11. Regulador switching de 4[V] de topología Buck basado en el LM25576 de TI

La fuente switching de 4[V] está orientada a alimentar principalmente al módulo SIM900. Para alimentar al microcontrolador MSP430 es necesario incorporar una etapa de regulación de 3.3[V] conectada a la salida de la fuente switching de 4[V]. Como el MSP430 requiere muy poca corriente (típicamente 100[mA]) y además la diferencia de voltaje entre los 4[V] de entrada y 3.3[V] de salida son solamente 0.7[V] , un regulador lineal tradicional es suficiente para esta aplicación. El circuito del regulador de 3.3[V] se muestra en la figura 2.12.

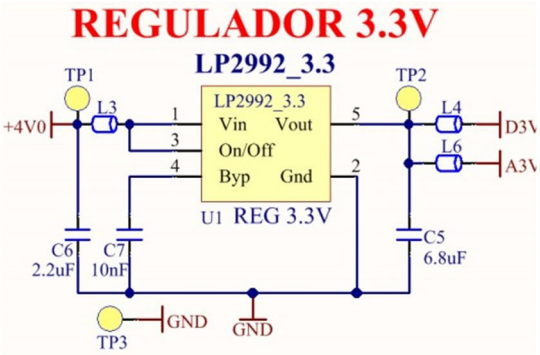


Fig. 2.12. Regulador lineal de 3.3[V]

Es natural hacerse la pregunta si el utilizar un diodo en serie con el regulador de 4[V] es una solución apropiada para generar los 3.3[V] necesarios. Lamentablemente esto no es posible para todas las condiciones de funcionamiento del microcontrolador ya que cuando este se encuentra funcionando en muy bajo consumo, la corriente requerida por él no es suficiente para que el diodo presente el voltaje de 0.7[V] requerido. El voltaje presentado en estos casos de bajo consumo es menor que 0.7[V] produciendo así un voltaje ma-

yor que 3.3[V]. Si el voltaje de alimentación llegara a superar los 3.6[V] el microcontrolador podría dañarse y poner en peligro la estabilidad del diseño.

CAPÍTULO 3

3. Diseño de firmware del módem USB especializado

El hardware del módem USB especializado está gobernado por un microcontrolador MSP430 el cual requiere ser programado para implementar las funciones requeridas por el proyecto elecciones. En este capítulo se pretende mostrar la estructura del firmware implementado en el microcontrolador.

Los comandos que llegan desde el computador cliente mediante el bus USB son interpretados por el MSP430 donde este último controla el módulo SIM900.

El firmware del microcontrolador debe implementar las siguientes funcionalidades:

- Comunicación con el computador cliente mediante el controlador USB FT232R.
- Búsqueda de tarjetas SIM instaladas.
- Conectarse a la red celular mediante el módulo SIM900 por primera vez a través de la SIM2 en caso de encontrarse instalada. En caso contrario, se comienza por la SIM1.
- En caso de perder la cobertura con la red celular se debe conmutar a la otra tarjeta SIM en caso de que exista una instalada. De no existir otra instalada, vuelve a intentar con la misma tarjeta SIM hasta encontrar cobertura nuevamente.
- Discar un número de teléfono por instrucción del computador cliente y esperar respuesta del módem remoto correspondiente al servidor de acopio de datos.
- Cuando la conexión haya sido establecida entre ambos computadores, debe notificarse a la aplicación del computador cliente para que esta comience a transferir los datos del archivo de texto.

El firmware del microcontrolador MSP430 consta de varios programas los cuales serán explicados en las siguientes secciones mediante pseudocódigo y diagramas de máquinas de estado. Los diagramas presentados son representaciones simplificadas del firmware que permiten mostrar su funcionalidad solamente sin entrar en detalles que se desvían del foco principal.

3.1 Estructura del firmware

En general el firmware está constituido por una sección de medio nivel y una de bajo nivel. En la sección de medio nivel, se encuentran todas las rutinas que permiten dar la funcionalidad requerida para el módem USB especializado sin entrar en detalles de configuración del hardware a nivel de bit y de programas que realizan operaciones elementales. En la sección de bajo nivel se encuentran todas aquellas rutinas requeridas para inicializar el hardware, como por ejemplo, mapeo de puertos, configuración de periféricos mediante registros e interrupciones (ISRs). También se incluyen las rutinas que se encargan en separar los mensajes llegados en los búferes de recepción de ambas UARTs, conectadas al módulo SIM900 y controlador FT232R respectivamente. En esta sección de bajo nivel se encuentran las capas de abstracciones que permiten exponer al software de nivel medio a interfaces que permitan manejar el hardware sin tener que comprender los detalles internos. La estructura del firmware se puede apreciar en la figura 3.1, donde los bloques con prefijo "HAL" (Hardware Abstraction Layer) indican un bloque de abstracción de hardware.

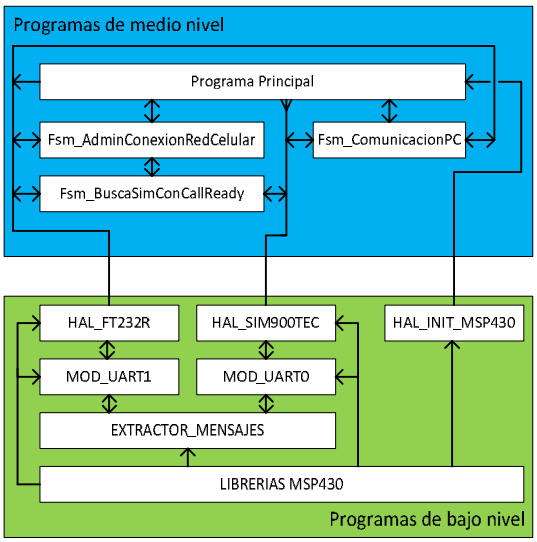


Fig. 3.1. Estructura del firmware. Se muestran los diferentes bloques software que lo conforman.

Como se puede apreciar en la figura 3.1, los programas existentes de bajo nivel son los que permiten controlar el hardware, que en este caso serían el módulo celular SIM900 y el controlador USB FT232R. Los programas de nivel medio solamente tienen acceso a los bloques con prefijo "HAL" de la sección de bajo nivel.

Cada bloque de la figura 3.1 corresponde a un programa ".c" donde cada uno cuenta con su archivo cabecera ".h" correspondiente.

El módulo SIM900 tiene una estructura especial para enviar y recibir mensajes de control. Lo mismo sucede para el caso del FT232R que estará conectado al computador cliente, donde también existe una estructura definida para el envío de mensajes con el microcontrolador. La extracción de mensajes para ambos módulos es completamente manejada por la sección de bajo nivel sin intervención de los programas de nivel medio.

En las secciones siguientes se describirá con mayor detalle la estructura de cada bloque de la sección de nivel medio, utilizando un enfoque de lo general a lo particular. Para ello se comienza con la descripción del programa principal para luego pasar al detalle de las máquinas de estados de medio nivel pero de nivel más bajo. Por motivos de espacio los programas que conforman la sección de bajo nivel que manejan el hardware se tratarán en el Anexo 2.

En la tabla 3.1 se describe brevemente la funcionalidad de cada programa de la sección de medio nivel mostrada en la figura 3.1.

Tabla 3.1. Descripción de las funciones principales de cada bloque de la sección de medio nivel.

Nombre función	Descripción
Principal()	Realiza inicializaciones generales y que contiene el bucle perpetuo de ejecución del programa.
Fsm_AdminConexionCelular()	Detecta tarjetas SIM instaladas y llama a rutinas implementadas dentro de Fsm_BuscaSimConCallReady.c. Además informa el estado de la sesión en la red celular al computador cliente.
Fsm_BuscaSimConCallReady()	Se encarga de buscar dentro de las tarjetas SIM instaladas una que tenga cobertura.
Fsm_ComunicacionPC()	Establece protocolo de comunicación con el computador cliente.

3.2 Estructura de las máquinas de estado

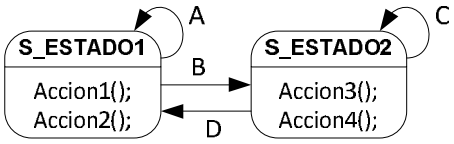
Cuando el programa tiene el prefijo "Fsm", existe en su interior una función que corresponde a la máquina de estado que lleva el mismo nombre. "Fsm" es un acrónimo para "Finite State Machine" o máquina de estados finitos en español.

Para el caso particular del programa Fsm_BuscaSimConCallReady.c, aparte de contener la función ***Fsm_BuscaSimConCallReady()***, contiene otras adicionales que permiten realizar ciertas operaciones utilizadas por esta misma FSM en sí y por ***Fsm_AdminConexionCelular().*** Dentro de estas funciones se encuentran: ***DevuelveSimsInstaladas()*** y ***SwapSimsEjecucionAnterior()*** para encuestar las tarjetas SIM instaladas y seleccionar la próxima a utilizar respectivamente. La última función ***CheckearCoberturaConSimActual()***, tal como lo señala su nombre es para comprobar que exista sesión activa con la red celular o equivalentemente que exista cobertura.

Estas tres funciones serán utilizadas para las explicaciones en este capítulo tomando en consideración sus retornos solamente por motivos de espacio.

Cada vez que una máquina de estado es invocada, éstas ejecutan un estado solamente y retornan con la finalidad de no dejar tomado el MSP430 y poder así atender otras labores. En otras palabras, son procesos con atomicidad a nivel de cada estado.

La razón por la cual se definen como funciones máquinas de estado, es debido a que su diseño está basado en el empleo de diagramas de este tipo. El utilizar esta clase de diagramas para algunos procesos es una manera muy útil y cómoda de visualizar los distintos estados que puede asumir el programa en función de su estado actual y transiciones. Además la estructura software utilizada permite una rápida traducción del algoritmo desarrollado en el diagrama de estados a código. A modo de ejemplo, considérese una máquina de 2 estados, para la cual su pseudocódigo y diagrama asociado se encuentran en la figura 3.2.




```
typedef enum
{
  S_ESTADO1,
  S_ESTADO2
} EstadoFsmEjemplo;

void Fsm_Ejemplo(void)
{//Ini
    static EstadoFsmEjemplo estadoActualFsmEjemplo = S_ESTADO1;
    switch(estadoActualFsmSesionRedCelular)
    {
    case S_ESTADO1:
        Accion1();
        Accion2();
        if (Condicion==A)
        {
            estadoActualFsmEjemplo = S_ESTADO1;
            AccionDuranteTransicion1();
        }
        else if (Condicion==B)
             {
                estadoActualFsmEjemplo = S_ESTADO2;
                AccionDuranteTransicion2();
             }
        break;

    case S_ESTADO2:
        Accion3();
        Accion4();
        if (Condicion==C)
        {
            estadoActualFsmEjemplo = S_ESTADO2;
            AccionDuranteTransicion3();
        }
        else if (Condicion==D)
             {
                estadoActualFsmEjemplo = S_ESTADO1;
                AccionDuranteTransicion4();
             }
        break;
    }
}//fin
```

Fig. 3.2. Correspondencia directa entre pseudocódigo y diagrama para el ejemplo de máquina de 2 estados

El diagrama de estados permite ver el algoritmo en un nivel que es fácil para su compresión, sin embargo, en él no se muestra el mismo nivel de detalle que la representación mediante pseudocódigo. Un ejemplo son las acciones durante las transiciones que no son representadas en el diagrama, pero sí en el pseudocódigo. En el pseudocódigo de la figura 3.2, estas acciones se representan por las funciones definidas por ***AccionDuranteTransicionX()*** donde X es un índice. En el programa implementado en lenguaje C, estas acciones corresponden principalmente a funciones que permiten enviar al computador cliente a través del módulo FT232R mensajes que representan los estados en que se encuentran las máquinas de estado que conforman al firmware.

Cada vez que se ejecuta un estado, la función retorna. El estado actual en que está la máquina queda almacenado en una variable estática dentro de la función, la cual será utilizada la próxima vez que la función sea invocada.

La estructura presentada en la figura 3.2 es utilizada para todas las máquinas de estado implementadas en el firmware del MSP430.

3.3 Programa principal

Al comenzar el programa principal, se ejecutan las líneas de código mostradas en la figura 3.3. Estas líneas llaman a las rutinas de inicialización del MSP430 que permiten configurar el reloj de 18[MHz], ambas UARTs y la habilitación de interrupciones. Además se lanza el programa extractor y eliminador de mensajes simbolizado por EXTRACTOR_MENSAJES en la figura 3.1, el cual queda corriendo de trasfondo.

```
LanzaACLK_MCLK_SMCLK();          //Inicializa clocks del MSP430
InitUart0Sim900();               //Inicializa la UART0 para el SIM900
InitUart1Ft232r();               //Inicializa la UART1 para el FT232R
InitExtractorMensajes ();        //Inicializa mantenedor de arreglos de mensajes

_EINT();                         //Activamos las interrupciones globales

EnviarStringPC("Estamos OK");    //Envío hacia el PC mediante el FT232R
while(!llegoComandoDePartidaDesdePC)
{
_NOP();                  //Se espera hasta que llegue comando de partida
}
```

Fig. 3.3. Primeras líneas de código del programa principal

Una vez realizadas las inicializaciones, se le envía el mensaje "Estamos OK" al computador cliente a través del controlador FT232R, con el cual el MSP430 se reporta. Luego queda en espera hasta que el computador cliente le envía un mensaje de partida. Una vez llegado este mensaje se continúa con la ejecución.

Una vez llegado el mensaje de partida, se entra en un bucle perpetuo, el cual se muestra en la figura 3.4.

Dentro de este bucle se llama a la máquina de estados ***Fsm_ComunicacionPc().*** Esta función detecta ciertas instrucciones provenientes del computador cliente como por ejemplo el discado de un número o colgado de línea entre otras. Si el computador cliente envía un comando de discado y se logra establecer conexión con el equipo remoto, ***Fsm_ComunicacionPc()*** devuelve en la variable **respuesta** un "1" indicando que la conexión se encuentra en modo datos. En caso contrario devuelve un "0" ya que se encuentra en modo comando. Modo comando significa que cuando se escribe un "string" en la UART donde se encuentra conectado el módulo SIM900, este último

asume que es un comando para el cual envía de vuelta una respuesta. Modo datos significa que el "string" enviado a la UART es pasado directamente al módem remoto (modo transparente) con la excepción del comando utilizado para colgar la comunicación el cual es interpretado por el microcontrolador para retornar a modo comando. Si se encuentra el SIM900 en modo comando se llama a la máquina de estados ***Fsm_AdminConexionRedCelular(),*** la cual tiene por finalidad intentar iniciar sesión en la red celular mediante llamados a ***Fsm_BuscaSimConCallReady()***, y su vez informar al computador cliente del estado en que se encuentra dicha sesión en forma periódica. Cuando hay una sesión activa se dice que existe cobertura.

```
while(1)
{
  /*la variable "respuesta" contiene el resultado de la ejecución de la
    máquina de estado LayerComunicacionPcFsm. Si resultado==1 implica que
    el SIM900 entro en modo data, en caso contrario en modo comando. */
  respuesta = Fsm_ComunicacionPc();
  if (respuesta==0)
  {
    if (pasePorModoComm==0)
    {
      EnviarStringPC("- - - MODO COMANDO - - -");
      pasePorModoComm=1;
      pasePorModoData=0;
    }
     //Intenta iniciar sesión en la red celular
    Fsm_AdminConexionRedCelular();
  }
  else
  {
    if (pasePorModoData==0)
    {
      EnviarStringPC("- - - MODO DATA - - -");
      pasePorModoData=1;
      pasePorModoComm=0;
    }
  _NOP();
  }
}
```

Fig. 3.4. Líneas de código restantes del programa principal

Las variables **pasePorModoComm** y **pasePorModoData** se utilizan solamente para enviar una sola vez un mensaje al computador cliente que represente el estado en que se encuentra el módem USB especializado. En otras palabras, el mensaje se envía solo en las transiciones cuando el módulo pasa de modo comando a modo data y viceversa. Esto se realiza para evitar que se envíe este mensaje en forma repetitiva.

En las secciones siguientes 3.4 y 3.6 de este capítulo se estudiará el funcionamiento de las máquinas de estado ***Fsm_AdminConexionRedCelular()*** y ***Fsm_ComunicacionPc()*** respectivamente.

3.4 FSM de conexión a la red celular *(Fsm_AdminConexionRedCelular())*

Esta máquina de estados es la función que tiene por objetivo intentar iniciar sesión en la red celular e informar al computador cliente del estado de la cobertura. La existencia de cobertura se detecta con la llegada de un "string" desde el módulo SIM900 señalando "CALL READY".

```
typedef enum
{
  S_DETECTAR_SIMS_INSTALADAS,
  S_INTENTAR_OBTENER_CALL_READY,
  S_RED_CON_CALL_READY
} EstadoFsmSesionRedCelular;
```

Fig. 3.5. Definición del tipo de dato EstadoFsmSesionRedCelular

Al tratarse de una máquina de estados, su estructura está de acuerdo con la que se presenta en la sección 3.2. Los estados que puede asumir se pueden apreciar en la definición del tipo **enum EstadoFsmSesionRedCelular** que se muestra en la figura 3.5.

En la figura 3.6 se muestra la estructura general del programa en cuestión, el código que va dentro de cada estado se irá mostrando más adelante en esta sección.

La primera línea de código define la variable que contendrá el valor del estado actual en que se encuentra la FSM, el cual al comienzo corresponde al estado S_DETECTAR_SIMS_INSTALADAS. Luego viene el código para cada estado donde se incluyen sus transiciones.

```
void Fsm_AdminConexionRedCelular(void)
{
  /* La primera linea de codigo
     se divide en 3 por motivos
     de espacio */
  static EstadoFsmSesionRedCelular  \
    estadoActualFsmSesionRedCelular \
      = S_DETECTAR_SIMS_INSTALADAS;

  switch(estadoActualFsmSesionRedCelular)
  {
    case S_DETECTAR_SIMS_INSTALADAS:
      /* Aca va el codigo a ejecutar
         para este estado */
      break;

    case S_INTENTAR_OBTENER_CALL_READY:
      /* Aca va el codigo a ejecutar
         para este estado */
      break;

    case S_RED_CON_CALL_READY:
      /* Aca va el codigo a ejecutar
         para este estado */
      break;
    }
}
```

Fig. 3.6. Pseudocódigo de la estructura de *Fsm_AdminConexionRedCelular()*

El primer estado tiene por objetivo detectar las tarjetas SIM instaladas en los sockets. El código se muestra en la figura 3.7 y comienza con la inicialización de la variable **statusCoberturaRedCelular**. Esta variable puede asumir 3 valores; "0", "1" y "2" que significan "Esperando respuesta", "Sin cobertura" y "Con cobertura" respectivamente. Se inicializa en "0" para indicar que no hay cobertura aún, ya que se está recién en el estado de detección de tarjetas SIMs instaladas. Luego se hace un llamado a la función ***DevuelveSimsInstaladas()*** que en **simsInstaladasEnSockets** deja un número que identifica las tarjetas SIM

que han sido instaladas. A su vez escribe las variables globales **simInsertadaEnSocketSim1** y **simInsertadaEnSocketSim2** que serán utilizadas por la máquina de estados ***Fsm_BuscaSimConCallReady()***. La rutina ***DevuelveSimsInstaladas()*** se encuentra en Fsm_BuscaSimConCallReady.c, que será explicado en la sección 3.5 y está representado por el bloque con el mismo nombre en la figura 3.1. Los valores que retorna se detallan en la tabla 3.2.

Mientras el valor devuelto sea "0", implica que no hay tarjetas SIM instaladas, y por ende se permanece en el estado inicial. En los 3 últimos casos el próximo estado será el mismo, S_INTENTAR_OBTENER_CALL_READY, con la diferencia que el mensaje que se envía al computador cliente mediante USB es distinto dependiendo de las tarjetas SIM que estén instaladas.

El mensaje se envía durante las transiciones utilizando la función ***EnviarStringPC_SiCambiaronSIMs()*** que solamente transmite el "string" al cliente la primera vez que se detecta un cambio en las tarjetas SIM instaladas (por. ej. justo después de extraer o insertar una SIM). Esto se hace para evitar que se esté enviando el mismo mensaje inútilmente en forma permanente cada vez que la función es llamada por el programa principal.

```
case S_DETECTAR_SIMS_INSTALADAS:
  statusCoberturaRedCelular=0;
  simsInstaladasEnSockets = DevuelveSimsInstaladas();
  EnviarStringPC_SiCambiaronSIMs("ESTADO: Detectando SIMs instaladas...");
  if      (simsInstaladasEnSockets==0)   // Transición A
  {
    estadoActualFsmSesionRedCelular=S_DETECTAR_SIMS_INSTALADAS;
    EnviarStringPC_SiCambiaronSIMs("Ninguna SIM instalada");
  }
  else if (simsInstaladasEnSockets==1)   // Transición B
  {
    estadoActualFsmSesionRedCelular=S_INTENTAR_OBTENER_CALL_READY;
    EnviarStringPC_SiCambiaronSIMs("Solo SIM M1 instalada");
  }
  else if (simsInstaladasEnSockets==2)   // Transición C
  {
    estadoActualFsmSesionRedCelular=S_INTENTAR_OBTENER_CALL_READY;
    EnviarStringPC_SiCambiaronSIMs("Solo SIM M2 instalada");
  }
  else if (simsInstaladasEnSockets==3)   // Transición D
  {
    estadoActualFsmSesionRedCelular=S_INTENTAR_OBTENER_CALL_READY;
    EnviarStringPC_SiCambiaronSIMs("SIM M1 y SIM M2 instaladas");
  }
  break;
```

Fig. 3.7. Pseudocódigo del estado S_DETECTAR_SIMS_INSTALADAS

Tabla 3.2. Valores devueltos por la función ***DevuelveSimsInstaladas()*** dependiendo de qué tarjetas SIM que estén instaladas.

Valor	Explicación del valor devuelto
0	Ninguna tarjeta SIM instalada
1	Solamente tarjeta SIM M1
2	Solamente tarjeta SIM M2
3	Ambas tarjetas SIM, M1 y M2 instaladas

El código para el segundo estado S_INTENTAR_OBTENER_CALL_READY se muestra en la figura 3.8.

Cuando el módulo SIM900 inicia sesión en una red celular a través de una tarjeta SIM en particular, éste le envía al microcontrolador un mensaje que contiene la cadena "CALL READY". Una vez llegado este mensaje, el módulo está preparado para discar un número telefónico. La detección de la cadena "CALL READY" es realizada por la máquina de estado ***Fsm_BuscaSimConCallReady()***.

En este estado, se utiliza una lógica similar a la del estado anterior, donde se envía un mensaje al computador cliente en las transiciones para reportar su estado. Este mensaje se envía solo cuando el valor de **statusIntentoObtenerCallReady** ha cambiado respecto a la ejecución anterior. El valor de esta variable se obtiene de la función ***Fsm_BuscarSimConCallReady()*** que corresponde a la máquina de estados implementada en el programa Fsm_BuscaSimConCallReady.c mostrada en los bloques de la figura 3.1. Los valores que devuelve esta función se muestran en la tabla 3.3 y sus detalles de funcionamiento se detallan en la sección 3.5.

```
case S_INTENTAR_OBTENER_CALL_READY:
  EnviarStringPC_SiCambioEstadoFsmBuscaSim("ESTADO: Intentando obtener CallReady...");
  statusIntentoObtenerCallReady = Fsm_BuscaSimConCallReady();
  if      (statusIntentoObtenerCallReady==0)   // Transición E
  { //Intentando
    EnviarStringPC_SiCambioEstadoFsmBuscaSim("Intentando obtener CallReady...");
    estadoActualFsmSesionRedCelular=S_DETECTAR_SIMS_INSTALADAS;
  }
  else if (statusIntentoObtenerCallReady==1)   // Transición F
  {
    //Superamos intentos cambiando de Tarjeta SIM
    //OBS: Se parte en dos lineas por espacio
    EnviarStringPC_SiCambioEstadoFsmBuscaSim \
        ("Se superaron los K_MAX_INTENTOS de obtener Call Ready. Cambiando de SIM...");
    estadoActualFsmSesionRedCelular=S_INTENTAR_OBTENER_CALL_READY;
  }
  else if (statusIntentoObtenerCallReady==2)   // Transición G
  { //Encontramos SIM M1 ARRIBA con CallReady
    EnviarStringPC_SiCambioEstadoFsmBuscaSim("Encontramos SIM M1 ARRIBA con CallReady");
    estadoActualFsmSesionRedCelular=S_RED_CON_CALL_READY;
  }
  else if (statusIntentoObtenerCallReady==3)   // Transición H
  { //Encontramos SIM M2 ABAJO  con CallReady
    EnviarStringPC_SiCambioEstadoFsmBuscaSim("Encontramos SIM M2 ABAJO con CallReady");
    estadoActualFsmSesionRedCelular=S_RED_CON_CALL_READY;
  }
  break;
```

Fig. 3.8. Pseudocódigo del estado S_INTENTAR_OBTENER_CALL_READY

Tabla 3.3. Valores devueltos por la función ***Fsm_BuscaSimConCallReady()*** dependiendo del estado en que se encuentra la búsqueda de SIM con cobertura.

Valor	Explicación del valor devuelto
0	Ninguna tarjeta SIM instalada
1	Se Superaron las K_MAX_CONMUTACIONES_SIM y no se encontró CALL READY con ninguna tarjeta SIM
2	Se encontró CALL READY con la tarjeta SIM M1
3	Se encontró CALL READY con la tarjeta SIM M2

El caso cuando el valor vale "0" es solamente para depuración y no se puede dar en la realidad ya que habría sido detectado por el código del estado anterior.

Cuando el valor devuelto por la función es "1" significa que no ha sido posible obtener "CALL READY" con la tarjeta SIM de turno dentro de un número de intentos definido por la constante K_MAX_CONMUTACIONES_SIM. El valor de esta constante permite hacer intentos hasta aproximadamente por 30 segundos. Para este caso es necesario volver al estado inicial donde se detectan las tarjetas SIM instaladas para volver a repetir el proceso.

Los casos para los valores "2" o "3" representan que fue posible iniciar sesión en la red celular con la SIM M1 o SIM M2 respectivamente. Para este caso el próximo estado será S_RED_CON_CALL_READY.

El código para el tercer y último estado S_RED_CON_CALL_READY se muestra en la figura 3.9.

La máquina de estados de conexión a la red celular se encuentra en este estado cuando el módulo SIM900 ya ha iniciado sesión con la red celular GSM mediante una de las tarjetas SIM seleccionadas por la máquina de estados implementada en Fsm_BuscaSimConCallReady.c.

```
case S_RED_CON_CALL_READY:
EnviarStringPC_SiCambioCobertura("Verificando Cobertura...");
statusCoberturaRedCelular = CheckearCoberturaConSimActual();
if      (statusCoberturaRedCelular==0)    // Transición I
{ //Esperando respuesta
  EnviarStringPC("Esperando");
  _NOP();
}
else if (statusCoberturaRedCelular==2)    // Transición J
{//Se cayo la cobertura
  EnviarStringPC("Se cayo la cobertura. Reiniciando...");
  estadoActualFsmSesionRedCelular = S_DETECTAR_SIMS_INSTALADAS;
}
else if (statusCoberturaRedCelular==1)    // Transición K
{//Seguimos cobertura con alguna SIM
  estadoActualFsmSesionRedCelular=S_RED_CON_CALL_READY;
  EnviarStringPC("Con cobertura");
}
break;
```

Fig. 3.9. Pseudocódigo del estado S_RED_CON_CALL_READY

El objetivo que tiene este estado es verificar que la sesión en la red celular se mantiene en el tiempo. De existir eventualmente un problema de cobertura, esa condición se detecta en este estado. La detección se hace al invocar a la función ***CheckearCoberturaConSimActual(),*** rutina que se encuentra dentro del programa Fsm_BuscaSimConCallReady.c que se muestra en la figura 3.1 y cuyos valores de retorno se encuentran definidos en la tabla 3.4.

Tabla 3.4. Valores devueltos por la función ***CheckearCoberturaConSimActual()*** dependiendo del estado de la cobertura.

Valor	Explicación del valor devuelto
0	No hay respuesta al comando de verificación de cobertura aún
1	Hay cobertura
2	No hay cobertura

Para el caso del valor "0", es necesario esperar, ya que el módulo celular aún no retorna la respuesta ante el comando de verificación de cobertura. Este comando tiene una duración variable que puede tomar en el peor de los casos varios segundos. Mientras se presente esta condición se permanece en este estado. Durante la espera se informa al computador cliente el proceso de espera mediante la función de bajo nivel ***EnviarStringPC()***. Existe un "timeout" interno en la función que cuando se cumple produce que ella retorne el valor "2".

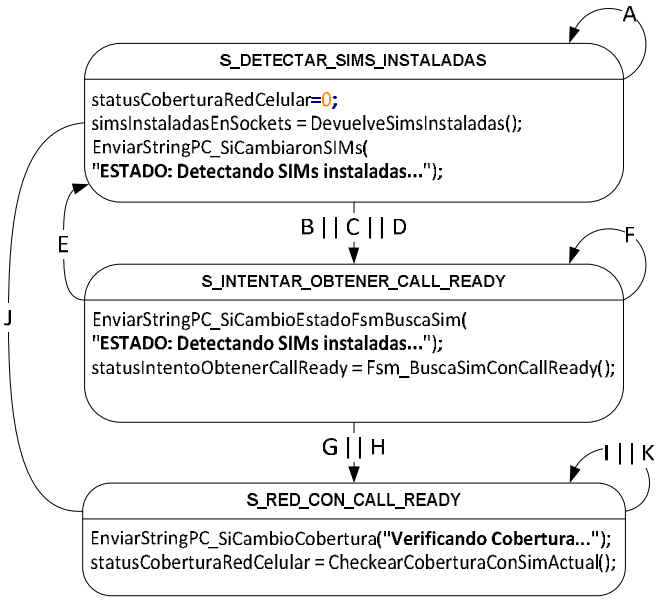


Fig. 3.10. Diagrama de estados simplificado para ***Fsm_AdminConexionRedCelular().***

Para el caso en que la función retorna "1", significa que el módulo continúa con cobertura. Esta respuesta se envía en forma permanente al computador cliente para saber que es posible discar. Mientras se presente esta condición se permanece en este estado.

Finalmente el caso en que la función retorna "2", significa que la cobertura fue interrumpida y es necesario volver al estado inicial donde se detectan las tarjetas SIM instaladas para volver a repetir el proceso.

En la figura 3.10 se puede apreciar un diagrama de estados simplificado para esta FSM. Las transiciones mostradas en el diagrama son obtenidas de los comentarios de cada estado mostrados en las figura 3.7, 3.8 y 3.9.

3.5 FSM de búsqueda de tarjetas SIM *(Fsm_ BuscaSimConCallReady())*

Esta máquina de estado tiene por finalidad buscar dentro de las tarjetas SIM que se encuentran instaladas una que contenga cobertura lo que significa que el SIM900 envíe el mensaje "CALL READY" al MSP430.

Dentro del mismo programa Fsm_BuscaSimConCallReady.c existe una rutina que es utilizada por la misma máquina de estados ***Fsm_BuscaSimConCallReady().*** Esta rutina es ***VerificaSiEjecucionActualEsPrimeraVez()***.

También dentro de las rutinas se encuentran ***CheckearCoberturaConSimActual()*** y ***DevuelveSimsInstaladas()*** que son utilizadas directamente por la máquina de estados ***Fsm_AdminConexionRedCelular()***, tal como se presentó en la sección anterior de este capítulo. La implementación de estas rutinas es muy simple y no requiere explicación ya que para la compresión de esta máquina de estados solamente es necesario tener claro el resultado que éstas retornan. De todas maneras en caso que el lector lo considere necesario, su implementación puede revisarse en el disco CD adjunto para este trabajo [35].

```
typedef enum
{
  S_INICIO,
  S_ESPERANDO_CALL_READY_SIMCARD_M1_ARRIBA,
  S_ESPERANDO_CALL_READY_SIMCARD_M2_ABAJO
} EstadoFsmBuscaSim;
```

Fig. 3.11. Definición del tipo de dato EstadoFsmSesionRedCelular

Si la cobertura es interrumpida con la tarjeta SIM de turno, por ejemplo SIM 1, se selecciona la otra tarjeta SIM para la próxima ejecución vez mediante ***CheckearCoberturaConSimActual()*** que es invocada desde la máquina de estados ***Fsm_AdminConexionRedCelular()***. Luego una vez realizada la selección de la próxima tarjeta SIM, desde la misma función ***Fsm_AdminConexionRedCelular()*** se llama a ***Fsm_BuscaSimConCallReady()*** para intentar obtener "CALL READY". Si la nueva tarjeta de turno SIM 2 no responde o no tiene cobertura por algún motivo, ***Fsm_BuscaSimConCallReady()*** se encarga de conmutarla a la otra hasta que se vuelva

a encontrar cobertura. Una vez que se obtiene cobertura y ésta se mantiene, ***Fsm_BuscaSimConCallReady()*** no se vuelve a llamar.

La tarjeta SIM a utilizar para obtener cobertura se encuentra en la variable global **nSimSeleccionadaEjecucionProxima** que puede tomar los valores 1 y 2 para las tarjetas SIM 1 y 2 respectivamente. Es importante tener en consideración que la rutina que se encarga en seleccionar la próxima tarjeta SIM a utilizar cuando se pierde la sesión de red es ***CheckearCoberturaConSimActual()*** mediante la modificación de **nSimSeleccionadaEjecucionProxima**. Una vez que la cobertura se asume como perdida ***Fsm_BuscaSimConCallReady()*** se encarga de las nuevas conmutaciones hasta volver a reestablecer la sesión en la red celular.

Los estados que puede asumir esta rutina se muestran en la definición del tipo **enum EstadoFsmBuscaSim** que se muestra en la figura 3.11.

```
char Fsm_BuscarSimConCallReady(void)
{//1i
  /* La primera linea de codigo
     se divide en 3 por motivos
     de espacio */
  static EstadoFsmBuscaSim \
    estadoActualFsmBuscaSimConCallReady \
      = S_INICIO;

  //Contador de intentos para obtener
  //CallReady
  static int  intento=0;

  //Flag indicador de CallReady
  static char flagCallReadyOK=0;

  //Resultado de la función. Puede asumir
  //valores entre 0 a 4
  static char buscaSimConCallReadyStatus=0;

  VerificaSiEjecucionActualEsPrimeraVez();

  //Se encuesta la lista de mensajes para
  //saber si llego "CALL READY"
  flagCallReadyOK = VerificarCallReady();

  switch (estadoActualFsmBuscaSimConCallReady)
  {//3i
    case S_INICIO:
      break;

    case S_ESPERANDO_CALL_READY_SIMCARD_M1_ARRIBA:
      break;

    case S_ESPERANDO_CALL_READY_SIMCARD_M2_ABAJO:
      break;
  }//3i
  intento++;
  return buscaSimConCallReadyStatus;
}//1f
```

Fig. 3.12. Pseudocódigo de la estructura de ***Fsm_BuscaSimConCallReady()***

En la figura 3.12 se puede apreciar la estructura general de esta máquina de estados, donde el código que va dentro de cada estado se irá mostrando en forma paulatina en esta misma sección. Las primeras líneas de código definen 4 variables.

La primera variable de tipo **EstadoFsmBuscaSim** contiene el valor del estado actual en que se encuentra la FSM, el cual al comienzo corresponde al estado S_INICIO.

La segunda variable es **intento**, la cual es un contador que se incrementa en "1" cada vez que se realiza un intento en obtener "CALL READY" desde el SIM900, lo cual es realizado llamando a ***Fsm_BuscaSimConCallReady()***. Cuando se supera el valor definido en la constante

K_MAX_INTENTOS_OBTENER_CALL_READY, se produce un "timeout" donde la FSM retorna "0". La idea es poder temporizar los intentos en obtener sesión en la red. La condición de "timeout" se produce cuando se supera un tiempo de aproximadamente 30 segundos.

La tercera variable que se define es **flagCallReadyOK**, que tal como lo dice su nombre, es un "flag" que representa si se encontró o no "CALL READY". En caso de haberlo encontrado retorna "1" en caso contrario "0".

La cuarta y última variable es **buscaSimConCallReadyStatus** que corresponde al valor que retornará la FSM dependiendo del estado en que se encuentre. Los valores que puede retornar está función van del "0" al "3", tal como se muestra en la tabla 3.3 de la sección anterior.

Una vez definidas todas las variables se procede a verificar si la ejecución del programa es por primera vez. Por primera vez se define que el módem estuvo apagado un tiempo previo a su encendido y ejecución del programa, tal que los condensadores de bypass del microcontrolador MSP430 se descargaron, y por ende no pudieron conservar los valores de los datos almacenados en la RAM. Para realizar esta verificación se llama a la función ***VerificaSiEjecucionActualEsPrimeraVez()*** que al igual que la máquina de estados explicada en esta sección es parte de Fsm_BuscaSimConCallReady.c. La implementación de esta función se muestra en la figura 3.13.

```
void VerificaSiEjecucionActualEsPrimeraVez(void)
{
  if (nFlagPrimeraEjecucion==0x12345678)
  { // No es la primera vez que se ejecuta
    // el programa. La SIM inicial que hay
    // que seleccionar esta en la variable
    // nSimSeleccionadaEjecucionProxima
    _NOP();
  }
  else
  { // Es la primera vez que se energiza
    // el uC, partimos con la SIMCARD 2
    // Escribimos flag de comienzo
    nFlagPrimeraEjecucion=0x12345678;
    nSimSeleccionadaEjecucionProxima = 2;
  }
}
```

Fig. 3.13. Definición de la función ***VerificaSiEjecucionActualEsPrimeraVez()***

La variable **nFlagPrimeraEjecucion** si contiene el valor 0x12345678 significa que no es la primera vez que se ejecuta el programa y se utiliza como tarjeta SIM de comienzo la definida en la variable global **nSimSeleccionadaEjecucionProxima.** En caso contrario, el valor de esta variable sería distinto ya que la RAM estuvo sin energía por mucho tiempo. Si este es el caso se selecciona como tarjeta SIM 2 como la de comienzo. Además se inicializa el valor de la variable **nFlagPrimeraEjecucion** con el valor 0x12345678. La variable **nSimSeleccionadaEjecucionProxima** será utilizada por la máquina de estados ***Fsm_BuscaSimConCallReady()***.

Luego de haber llamado a la función ***VerificaSiEjecucionActualEsPrimeraVez()***, se realiza un llamado a ***VerificarCallReady()***, función que se encuentra en la sección de bajo nivel, y se almacena su resultado en el **flagCallReadyOK.** Esta última función es invocada cada vez que se llama a ***Fsm_BuscaSimConCallReady()*** con la finalidad de ir buscando en la lista de mensajes que van llegando desde el módulo SIM900 la cadena "CALL READY" en forma periódica.

```
case S_INICIO:
  intento=0;
  buscaSimConCallReadyStatus=0;
  _NOP();
    if (nSimSeleccionadaEjecucionProxima==1)
    {//6i SIM M1 seleccionada
      if (simInsertadaEnSocketSim1)    // Transición A
      {//7i
          estadoActualFsmBuscaSimConCallReady=\
            S_ESPERANDO_CALL_READY_SIMCARD_M1_ARRIBA;
          EncenderModuloSim900SeleccionandoSimM1Arriba();
      }//7f
      else
        if (simInsertadaEnSocketSim2) // Transición B
          nSimSeleccionadaEjecucionProxima=2;
        else                          // Transición C
        {//8i No hay CallReady ni tarjetas SIM instaladas
          estadoActualFsmBuscaSimConCallReady=S_INICIO;
        }//8f
    }//6f

    if (nSimSeleccionadaEjecucionProxima==2)
    {//9i SIM M2 seleccionada
      if (simInsertadaEnSocketSim2)    // Transición D
      {//10i
           estadoActualFsmBuscaSimConCallReady=\
             S_ESPERANDO_CALL_READY_SIMCARD_M2_ABAJO;
           EncenderModuloSim900SeleccionandoSimM2Abajo();
      }//10f
      else
        if (simInsertadaEnSocketSim1) // Transición E
          nSimSeleccionadaEjecucionProxima=1;
        else                          // Transición F
        {//11i No hay CallReady ni tarjetas SIM instaladas
          estadoActualFsmBuscaSimConCallReady=S_INICIO;
        }//11f
    }//9f
  break;
```

Fig. 3.14. Pseudocódigo del estado S_INICIO

El primer estado S_INICIO que se muestra en la figura 3.14, tiene por objetivo encender el SIM900 con la tarjeta SIM seleccionada según la variable **nSimSeleccionadaEjecucionProxima**. En la máquina de estados ***Fsm_AdminConexionRedCelular()***, se invocaba a ***DevuelveSimsInstaladas()*** que tenía por labor determinar las tarjetas SIMs instaladas en los "sockets". Si las tarjetas SIM 1 y SIM 2 están instaladas, las variables globales **simInsertadaEnSocketSim1** y **simInsertadaEnSocketSim2** valen "1", en caso contrario "0". La tarjeta SIM seleccionada por **nSimSeleccionadaEjecucionProxima** solo se utiliza si su correspondiente variable global **simInsertadaEnSocketSim1** o

simInsertadaEnSocketSim2 vale "1", ya que están presentes. Por ejemplo si **nSimSeleccionadaEjecucionProxima==1** y **simInsertadaEnSocketSim1==1** se procede a buscar cobertura con la tarjeta SIM 1 y se enciende el SIM900. Ahora, en el caso que si **simInsertadaEnSocketSim1==0**, se define la tarjeta SIM 2 para ser utilizada en la próxima ejecución al definir **nSimSeleccionadaEjecucionProxima=2.** Si **nSimSeleccionadaEjecucionProxima==1** y **simInsertadaEnSocketSim1==1** se procede a encender el módulo con el circuito conmutador direccionado hacia la tarjeta SIM 1. Además S_ESPERANDO_CALL_READY_SIMCARD_M1_ARRIBA será el próximo estado para esta FSM. Similar es cuando **nSimSeleccionadaEjecucionProxima==2** y **simInsertadaEnSocketSim2==1**, con la única diferencia que el estado próximo será el definido por S_ESPERANDO_CALL_READY_SIMCARD_M2_ABAJO.

Los siguientes 2 estados son idénticos en cuanto a funcionalidad y forma por lo que se realizará la explicación utilizando solamente el pseudocódigo para el estado S_ESPERANDO_CALL_READY_SIMCARD_M1_ARRIBA. La figura 3.15 muestra el pseudocódigo para los últimos dos estados.

```
case S_ESPERANDO_CALL_READY_SIMCARD_M1_ARRIBA:
  if      ((flagCallReadyOK==0)&&(intento<K_MAX_INTENTOS_OBTENER_CALL_READY))   // Transición G
  {
    estadoActualFsmBuscaSimConCallReady=S_ESPERANDO_CALL_READY_SIMCARD_M1_ARRIBA;
    _NOP();
  }
  else if ((flagCallReadyOK==1)&&(intento<K_MAX_INTENTOS_OBTENER_CALL_READY))   // Transición H
  {
    //Encontramos CallReady en SIM M1 ARRIBA
    estadoActualFsmBuscaSimConCallReady=S_INICIO;
    buscaSimConCallReadyStatus=2;
  }
  else if ((flagCallReadyOK==0)&&(intento>=K_MAX_INTENTOS_OBTENER_CALL_READY))  // Transición I
  {
    estadoActualFsmBuscaSimConCallReady=S_INICIO;
    buscaSimConCallReadyStatus=1;
    nSimSeleccionadaEjecucionProxima=2;
  }
  break;

case S_ESPERANDO_CALL_READY_SIMCARD_M2_ABAJO:
  if      ((flagCallReadyOK==0)&&(intento<K_MAX_INTENTOS_OBTENER_CALL_READY))   // Transición J
  {
    estadoActualFsmBuscaSimConCallReady=S_ESPERANDO_CALL_READY_SIMCARD_M2_ABAJO;
    _NOP();
  }
  else if ((flagCallReadyOK==1)&&(intento<K_MAX_INTENTOS_OBTENER_CALL_READY))   // Transición K
  {
    //Encontramos CallReady en SIM M2 ABAJO
    estadoActualFsmBuscaSimConCallReady=S_INICIO;
    buscaSimConCallReadyStatus=3;
  }
  else if ((flagCallReadyOK==0)&&(intento>=K_MAX_INTENTOS_OBTENER_CALL_READY))  // Transición L
  {
    estadoActualFsmBuscaSimConCallReady=S_INICIO;
    buscaSimConCallReadyStatus=1;
    nSimSeleccionadaEjecucionProxima=1;
  }
  break;
```

Fig. 3.15. Pseudocódigo de los estados S_ESPERANDO_CALL_READY_SIMCARD_M1_ARRIBA y S_ESPERANDO_CALL_READY_SIMCARD_M2_ABAJO

Dentro del estado se encuentran 3 casos. El primero se presenta cuando aún no ha llegado el mensaje "CALL READY" y simultáneamente cuando el número de intentos es menor que K_MAX_INTENTOS_OBTENER_CALL_READY, caso que está señalado en el pseudocódigo con el comentario "Transición G". Esto significa que aún se está esperando el mensaje desde el módulo SIM900 por lo tanto es necesario esperar en el mismo estado.

El segundo caso corresponde cuando llega el mensaje "CALL READY" antes del tiempo máximo definido por K_MAX_INTENTOS_OBTENER_CALL_READY, condición marcada con el comentario "Transición H". Cuando esto sucede se establece S_INICIO como el estado próximo. Como llegó "CALL READY", significa que existe cobertura, por lo que no se vuelve a llamar a esta máquina de estados desde ***Fsm_AdminConexionRedCelular().*** Se retorna **buscaSimCallReadyStatus=2.**

El tercer y último caso señalado por el comentario "Transición I", ocurre cuando no ha llegado "CALL READY" dentro del número de intentos definido por la constante K_MAX_INTENTOS_OBTENER_CALL_READY. Cuando esto sucede se establece S_INICIO como el estado próximo. Se retorna **buscaSimCallReadyStatus=1** y se establece **nSimSeleccionadaEjecucionProxima=2** para cuando se vuelva a llamar a esta FSM desde la ***Fsm_AdminConexionRedCelular()***, se comience a buscar cobertura con la tarjeta SIM 2 siempre y cuando ella se encuentre instalada. El estar instalada implica que **simInsertadaEnSocketSim2=1.**

Los valores de retorno de ***Fsm_BuscaSimConCallReady()*** fueron mostrados en la tabla 3.3 de la sección anterior.

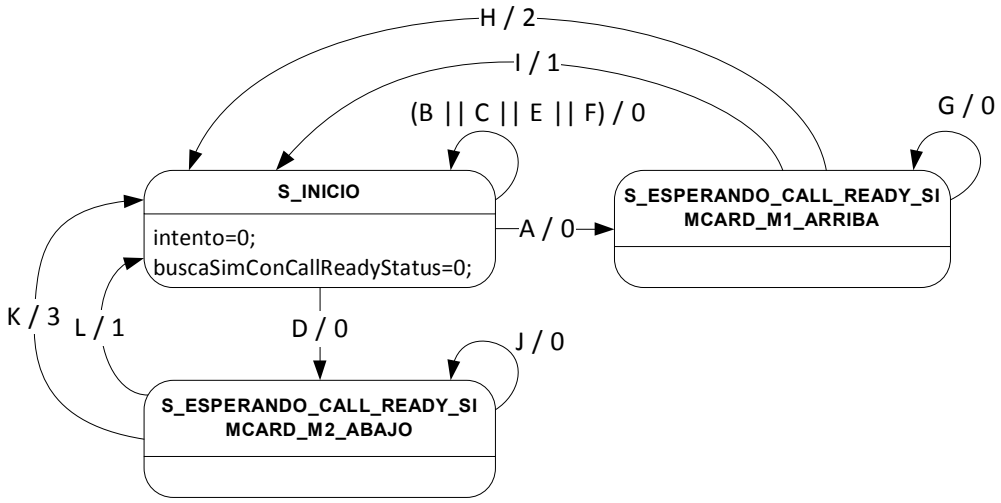


Fig. 3.16. Diagrama de estados simplificado para ***Fsm_BuscaSimConCallReady().***

En la figura 3.16 se puede apreciar un diagrama de estados simplificado para esta FSM. Las transiciones mostradas en el diagrama son obtenidas de los comentarios de cada estado mostrados en las figuras 3.14 y 3.15.

3.6 FSM de Comunicación con el USB Host (*Fsm_ComunicacionPc()*)

Esta función corresponde a la máquina de estados que le entrega la funcionalidad al módem USB especializado de recibir comandos desde el computador cliente. Básicamente existen 2 tipos de comandos, los que se envían al módem cuando éste se encuentra en modo comando y los que se envía cuando se encuentra en modo datos. Todos los comandos en modo comando son enviados siempre desde el computador cliente hacia el módem USB especializado, donde este último responde con un eco para demostrar una recepción correcta. En otras palabras, el servidor no tiene posibilidad de iniciar acción alguna, ya que todas son manejadas por el cliente.

Los comandos en modo datos, son enviados desde el computador cliente hacia el computador servidor, donde este último envía mensajes de respuesta de vuelta hacia el cliente como parte del protocolo. En este modo, el módem USB especializado se encuentra en modo transparente y por ende no interpreta los comandos. Estos comandos serán interpretados por los programas de alto nivel en los computadores cliente y servidor. Tanto la estructura de los comandos para modo datos como los programas de alto nivel serán explicados en el siguiente capítulo. Para la explicación de la presente sección se incluirá el "string" "**<RESTO DEL COMANDO>**" para referirse a los campos de los comandos que serán descritos en el próximo capítulo.

Es importante tener en consideración que si bien estos comandos no son interpretados por el módem USB especializado, su transmisión entre ambos computadores es solamente permitida si ellos son válidos. Esto es realizado con la finalidad de evitar que por un problema de transmisión o recepción el comando pueda llegar erróneo y producir así un funcionamiento errático. Esta labor es realizada por la máquina de estados ***Fsm_ComunicacionPc()*** que será explicada en esta sección.

La única excepción en donde el módem USB especializado interpreta un comando estando en modo datos es el que se utiliza para abortar la comunicación. Este comando nunca llega al computador servidor.

Las definiciones de los comandos para ambos modos se encuentran a continuación.

Comandos en modo comando:

1. COMENZAR: Una vez recibido, se procede con la ejecución "loop" perpetuo principal del programa.

2. MARCAR: Utilizado para discar el número de teléfono donde se encuentra el computador servidor para el acopio de datos. La estructura de este comando es "**MARCAR:<NUM TEL>**". Donde "**<NUM TEL>**" es un "string" y "**:**" un separador.

3. COLGAR: Utilizado para abortar el discado de la llamada realizada por MARCAR.

Comandos en modo datos:

1. COLGAR_MD: Utilizado por el computador cliente para colgar la llamada actual y así abortar la comunicación entre el módem USB especializado y el módem del computador servidor. Este comando es interpretado solamente por el módem USB especializado.

2. C: Utilizado para crear un archivo en el computador remoto. Este comando es interpretado solamente por el computador remoto. Es un "string" de la forma "**C:<RESTO DEL COMANDO>**". Donde "**<RESTO DEL COMANDO >**" es un "string" y "**:**" es un carácter separador.

3. D: Utilizado para enviar los bytes del archivo a transmitir. Este comando es interpretado solamente por el computador remoto. Es un "string" de la forma "**D:<RESTO DEL COMANDO>**". Donde "**<RESTO DEL COMANDO >**" es un "string" y "**:**" es un carácter separador.

4. E: Corresponde a un mensaje de respuesta desde el servidor hacia el computador cliente. Es un "string" de la forma "**E:<RESTO DEL COMANDO>**". Donde "**<RESTO DEL COMANDO >**" es un "string" y "**:**" es un carácter separador.

Para comenzar la explicación de esta máquina de estados, en la figura 3.17 se muestran los estados que puede asumir esta rutina con la definición del tipo **enum EstadoFsmLayerComunicacionPC**.

```
typedef enum
{
  S_ESPERAR_INSTRUCCION_PARTIDA_DESDE_PC,
  S_ESPERAR_COMANDO_DESDE_PC,
  S_DISCANDO,
  S_ESPERANDO_A_MSP_PARA_ENTRAR_EN_MODO_DATA,
  S_ESPERANDO_RESPUESTA_DISCADO_DEL_SIM900,
  S_ESPERANDO_RESPUESTA_COLGAR_MIENTRAS_DISCA,
  S_CONECTADO_A_REMOTO
} EstadoFsmLayerComunicacionPC;
```

Fig. 3.17. Definición del tipo de dato EstadoFsmLayerComunicacionPC

En la figura 3.18 se puede apreciar la estructura general de esta máquina de estados, donde el código que va dentro de cada estado se irá mostrando durante el desarrollo de esta sección. Las primeras líneas de código definen 2 variables.

Primero se define e inicializa la variable **estadoActualFsmLayerComunicacionPC** que es de tipo **EstadoFsmLayerComunicacionPC** que contiene el valor del estado actual en que se encuentra la FSM, el cual al comienzo corresponde al estado S_ESPERAR_INSTRUCCION_PARTIDA_DESDE_PC.

```
char Fsm_ComunicacionPc()
{
  static EstadoFsmLayerComunicacionPC\
          estadoActualFsmLayerComunicacionPC =\
            S_ESPERAR_INSTRUCCION_PARTIDA_DESDE_PC;
  // Flag que representa si se está en modo data
  // o modo comando
  static char flagModoData=0;

  switch(estadoActualFsmLayerComunicacionPC)
  {
    case S_ESPERAR_INSTRUCCION_PARTIDA_DESDE_PC:
    break;

    case S_ESPERAR_COMANDO_DESDE_PC:
    break;

    case S_DISCANDO:
    break;

    case S_ESPERANDO_RESPUESTA_DISCADO_DEL_SIM900:
    break;

    case S_ESPERANDO_RESPUESTA_COLGAR_MIENTRAS_DISCA:
    break;

    case S_ESPERANDO_A_MSP_PARA_ENTRAR_EN_MODO_DATA:
    break;

    case S_CONECTADO_A_REMOTO:
    break;
  }
  return flagModoData;
}
```

Fig. 3.18. Pseudocódigo de la estructura de ***Fsm_ComunicacionPC()***

La segunda variable a declarar es **flagModoData**, la cual tal como dice su nombre, es un "flag" que indica cuando el módulo SIM900 ha entrado en modo datos. Al principio se le asigna "0" ya que no se ha establecido comunicación con el equipo remoto aún. Cuando se entra en modo datos se le asigna valor "1". Este valor se retorna al final de esta función.

Antes de comenzar con la explicación de los estados es importante tener en consideración que existen 2 listas de mensajes, una para el módulo SIM900 y otra para el módulo FT232R. Estas listas contienen los mensajes que llegan desde ambos módulos y su llenado se realiza por rutinas de bajo nivel. Todos los mensajes que provienen desde el computador cliente van quedando en la lista del FT232R. Análogamente, los mensajes que provienen desde el computador remoto que hace de servidor de acopio de datos van quedando almacenados en la lista del SIM900. Estos mensajes son

consultados en las listas mediante ***BuscarMensajeEnListaFt232r()*** y ***BuscarMensajeEnListaSim900()***, siendo ambas funciones parte de las rutinas que conforman la sección de bajo nivel. Tanto la estructura de las listas como las funciones para buscar mensajes en ellas, serán presentadas en el Anexo 2.

El primer estado S_ESPERAR_INSTRUCCION_PARTIDA_DESDE_PC que se presenta en la figura 3.19 tiene por objetivo esperar la instrucción o comando de partida desde el computador cliente para comenzar el bucle perpetuo en el programa principal.

```
case S_ESPERAR_INSTRUCCION_PARTIDA_DESDE_PC:
if (BuscarMensajeEnListaFt232r("COMENZAR"))
  { //Transición A
    //Llegó comando COMENZAR desde PC cliente
    EnviarStringPC("COMENZAR"); //ECO
    EnviarStringPC("Partiendo...");
    estadoActualFsmLayerComunicacionPC = \
      S_ESPERAR_COMANDO_DESDE_PC;
  }
  else
  { //Transición B
    estadoActualFsmLayerComunicacionPC =\
      S_ESPERAR_INSTRUCCION_PARTIDA_DESDE_PC;
  }
break;
```

Fig. 3.19. Definición del estado S_ESPERAR_INSTRUCCION_PARTIDA_DESDE_PC

Mientras no llegue el comando de partida se permanece en el mismo estado. Una vez detectado el comando de partida en la lista de mensajes del FT232R se envía al computador cliente un eco del comando y el "string" "Partiendo…". El próximo estado para este caso será S_ESPERAR_COMANDO_DESDE_PC.

```
case S_ESPERAR_COMANDO_DESDE_PC:
  flagModoData=0;
  if (BuscarMensajeEnListaFt232r("MARCAR:<NUM TEL>"))
  { //Transición C
    //Llegó comando MARCAR desde computador cliente
    EnviarStringPC("MARCAR:<NUM TEL>"); //ECO
    estadoActualFsmLayerComunicacionPC=S_DISCANDO;
  }
  else if(BuscarMensajeEnListaFt232r("COLGAR"))
       {  //Transición D
          //Llegó comando COLGAR desde computador
          //cliente
         EnviarStringPC("COLGAR"); //ECO
         estadoActualFsmLayerComunicacionPC =\
           S_ESPERAR_COMANDO_DESDE_PC;
       }
  else
  { //Transición E
    estadoActualFsmLayerComunicacionPC =\
      S_ESPERAR_COMANDO_DESDE_PC;
  }
break;
```

Fig. 3.20. Definición del estado S_ESPERAR_COMANDO_DESDE_PC

En la figura 3.20 se puede apreciar el código del estado S_ESPERAR_COMANDO_DESDE_PC. Este estado tiene por objetivo esperar un comando para marcar o colgar, el cual cuando es válido se responde con un eco. Cualquier otro "string" que llegue distinto a los comandos esperados es descartado. En el caso que el comando sea "COLGAR" o uno inválido se permanece en el mismo estado. En caso que el comando sea el utilizado para discar, el próximo estado será S_DISCANDO.

```
case S_DISCANDO:
    flagModoData=1;
    EnviarStringPC("Marcando ");
    EnviarStringPC("<NUM TEL>");
    EnviarStringPC("...");
    MarcarNumeroSIM900("<NUM TEL>");
    estadoActualFsmLayerComunicacionPC =\
      S_ESPERANDO_RESPUESTA_DISCADO_DEL_SIM900;
break;
```

Fig. 3.21. Definición del estado S_DISCANDO

El estado S_DISCANDO se muestra en la figura 3.21. En él se asigna **flagModoData=1** para indicar que se está entrando en modo datos. También se envían los "strings" "Marcando", "**<NUM TEL>**" y "**...**" al computador cliente. Luego se disca el número utilizando la función de bajo nivel llamada ***MarcarNumeroSIM900()***. El próximo estado a ejecutar es S_ESPERANDO_RESPUESTA_DISCADO_DEL_SIM900 que se muestra en la figura 3.22.

```
case S_ESPERANDO_RESPUESTA_DISCADO_DEL_SIM900:
  if (BuscarMensajeEnListaFt232r("COLGAR"))
  { //Transición F
    //Llegó comando COLGAR desde computador
    //cliente
     EnviarStringPC("COLGAR"); //ECO
     ColgarSim900();
     EnviarStringPC("Abortado por usuario");
     estadoActualFsmLayerComunicacionPC =\
       S_ESPERANDO_RESPUESTA_COLGAR_MIENTRAS_DISCA;
  }
  else if (VerificarConexionModoDataSim900())
  { //Transición G
    estadoActualFsmLayerComunicacionPC =\
      S_ESPERANDO_A_MSP_PARA_ENTRAR_EN_MODO_DATA;
  }
  else if (VerificarLineaOcupadaSim900())
  { //Transición H
    estadoActualFsmLayerComunicacionPC =\
      S_ESPERAR_COMANDO_DESDE_PC;
  }
break;
```

Fig. 3.22. Definición del estado S_ESPERANDO_RESPUESTA_DISCADO_DEL_SIM900

El estado S_ESPERANDO_RESPUESTA_DISCADO_DEL_SIM900 tiene por objetivo tal como dice su nombre, esperar la respuesta del módulo celular SIM900 ante el comando marcar. Mientras se está en este estado es posible abortar la comunicación que está siendo cursada mediante el comando "COLGAR". En el caso que se decida colgar, se envía el eco hacia el computador cliente además de enviar el "string" "Abortado por el usuario" y S_ESPERANDO_RESPUESTA_COLGAR_MIENTRAS_DISCA será el próximo estado.

En el caso que no se decida colgar, y que se verifique que se pudo entrar en modo datos que en otras palaras significa que se estableció la conexión con el remoto, el próximo estado será S_ESPERANDO_A_MSP_PARA_ENTRAR_EN_MODO_DATA.

En el caso que no se haya decidido colgar y no se haya entrado en modo datos aún, se verifica si la línea está ocupada, que en caso de ser afirmativo, se establece S_ESPERAR_COMANDO_DESDE_PC como el próximo estado para comenzar de nuevo.

El estado S_ESPERANDO_RESPUESTA_COLGAR_MIENTRAS_DISCA se muestra en la figura 3.23. En este estado se permanece mientras no se tenga la respuesta del módulo SIM900 frente al comando colgar efectuado durante el proceso de discado realizado en el estado S_ESPERANDO_RESPUESTA_DISCADO_DEL_SIM900. El mensaje que se busca obtener en este caso para tener seguridad que el colgado se realizó correctamente es el "string" "ERROR", que en el caso de detectarse se envía el "string" "Colgado OK" al computador cliente y se establece el próximo estado en el definido por S_ESPERAR_COMANDO_DESDE_PC. En caso que el mensaje no haya llegado aún, se permanece en el estado actual.

```
case S_ESPERANDO_RESPUESTA_COLGAR_MIENTRAS_DISCA:
  if (BuscarMensajeEnListaSim900("ERROR"))
  { //Transición I
    EnviarStringPC("Colgado OK");
    estadoActualFsmLayerComunicacionPC =\
      S_ESPERAR_COMANDO_DESDE_PC;
  }
  else  //Transición J
    estadoActualFsmLayerComunicacionPC =\
        S_ESPERANDO_RESPUESTA_COLGAR_MIENTRAS_DISCA;
break;
```

Fig. 3.23. Definición del estado S_ESPERANDO_RESPUESTA_COLGAR_MIENTRAS_DISCA

El estado S_ESPERANDO_A_MSP_PARA_ENTRAR_EN_MODO_DATA, que se muestra en la figura 3.24, tiene por objetivo esperar a las rutinas de bajo nivel que realicen todos los procesos de limpieza de búferes y listas de mensajes antes de proceder al funcionamiento en modo datos. Mientras se espera, se permanece en el mismo estado. Cuando estos procesos son terminados se establece el próximo estado S_CONECTADO_A_REMOTO. Los procesos de limpieza son detallados en el Anexo 2.

```
case S_ESPERANDO_A_MSP_PARA_ENTRAR_EN_MODO_DATA:
  estadoActualFsmLayerComunicacionPC =\
    S_ESPERANDO_A_MSP_PARA_ENTRAR_EN_MODO_DATA;
  if (EntrarEnModoData())  //Transición K
    estadoActualFsmLayerComunicacionPC =\
      S_CONECTADO_A_REMOTO;
break;
```

Fig. 3.24. Definición del estado S_ESPERANDO_A_MSP_PARA_ENTRAR_EN_MODO_DATA

El último estado de la rutina ***Fsm_ComunicacionPc()***, el cual corresponde a S_CONECTADO_A_REMOTO se presenta en la figura 3.25.

La primera vez que se ejecuta este estado se envía al computador cliente el "string" "CONECTADO A REMOTO". Una vez enviado el mensaje se asigna a la variable **statusConectadoARemotoPasadaAnterior** el valor "0" para evitar que se envíe el mismo mensaje en forma repetitiva las próximas veces que se ejecute el mismo estado.

Si se desea colgar mientras se está en modo datos, es necesario que se envíe el comando "COLGAR MD". Al recibir este comando se envía el eco del comando al computador cliente como también el "string" "COLGADO OK" para indicar al computador cliente que el proceso de aborto de la conexión con el remoto mediante la función ***ColgarSim900ModoDataVolverModoComando()*** se llevó a cabo con éxito. Para indicar el paso por esta sección de código se asigna el valor "2" a la variable **statusConectadoARemotoPasadaAnterior**. Una vez realizado esto se establece S_ESPERAR_COMANDO_DESDE_PC como estado próximo.

Cada vez que se ejecuta este estado, se verifica si se ha llegado establecer un enlace de datos con el remoto mediante la función de bajo nivel ***VerificarModoDataSim900()***. En caso de existir el enlace la función retorna "1" y en el caso contrario "0".

Cuando se retorna "0" puede deberse a dos causas, la primera es que se cayó la conexión debido a algún problema con el servidor o con la línea de teléfonos a la cual este último se encuentra conectado. La segunda causa puede deberse a que la cobertura ha sido interrumpida producto de baja intensidad RF u otro problema en el lado donde se encuentra el computador cliente. Cualquiera sea el caso que produce que la función ***VerificarModoData-***

```
case S_CONECTADO_A_REMOTO:
  //Conectados podemos TX data ahora!!
  if (statusConectadoARemotoPasadaAnterior!=0)
  {  //Transición L
     EnviarStringPC("CONECTADO A REMOTO...");
     statusConectadoARemotoPasadaAnterior=0;
  }
  if (BuscarMensajeEnListaFt232r("COLGAR_MD"))
  {  //Transición M
     //Llegó comando COLGAR desde computador
     //cliente para colgar en forma voluntaria
     EnviarStringPC("COLGAR_MD"); //ECO
     ColgarSim900ModoDataVolverModoComando();
     EnviarStringPC("Colgado OK");
     statusConectadoARemotoPasadaAnterior=1;
     estadoActualFsmLayerComunicacionPC =\
       S_ESPERAR_COMANDO_DESDE_PC;
  }
  if (!VerificarModoDataSim900())
  {  //Transición N
     //Se cayó conexión y/o cobertura
     //caso NO voluntario
    EntrarEnModoComando();
    EnviarStringPC("Conexión perdida");
    statusConectadoARemotoPasadaAnterior=2;
    estadoActualFsmLayerComunicacionPC =\
      S_ESPERAR_COMANDO_DESDE_PC;
    _NOP();
  }
  else
  {
    //* *CODIGO SECCION COMUNICACIÓN REMOTO * *
  }
break;
```

Fig. 3.25. Definición del estado S_CONECTADO_A_REMOTO

Sim900() retorne “0”, se envía al computador cliente el mensaje “Conexión perdida” y se establece S_ESPERAR_COMANDO_DESDE_PC como el próximo estado a ser ejecutado.

Cuando la función ***VerificarModoDataSim900()*** retorna “1” significa que existe conexión de datos activa y se prosigue con la ejecución de la sección definida con la leyenda `* *CODIGO SECCION COMUNICACIÓN REMOTO * *`, cuyo código se muestra en la figura 3.26.

En esta última sección de código, se realiza el filtrado de “strings” para descartar todos aquellos que no sean comandos de la forma definida al comienzo de esta sección. En la figura 3.26 se pueden apreciar en los comentarios cuales son aquellos comandos que van desde el cliente hacia el servidor y viceversa. Los comentarios `//ECO` son agregados al final de cada línea donde el comando enviado es una respuesta de tipo eco. En la figura 3.27 se puede apreciar un diagrama de estados simplificado para esta FSM. Las transiciones mostradas en el diagrama son obtenidas de los comentarios de cada estado mostrados en las figuras 3.19, 3.20, 3.21, 3.22, 3.23, 3.24, 3.25 y 3.26.

```
if(BuscarMensajeEnListaFt232r("C:<RESTO DEL COMANDO>"))
{ //Transición O
  //Llegó comando CREAR ARCHIVO desde computador cliente
  EnviarStringPC("C:<RESTO DEL COMANDO>"); //ECO
  EnviaDataPcRemoto("C:<RESTO DEL COMANDO>");
}
if(BuscarMensajeEnListaFt232r("D:<RESTO DEL COMANDO>"))
{ //Transición P
  //Llegó comando COMIENZO DATA ARCHIVO desde computador cliente
  EnviarStringPC("C:<RESTO DEL COMANDO>"); //ECO
  EnviaDataPcRemoto("C:<RESTO DEL COMANDO>");
}
if(BuscarMensajeEnListaSim900("C:<RESTO DEL COMANDO>"))
{ //Transición Q
  //Respuesta desde servidor para el comando C:
  EnviarStringPC("C:<RESTO DEL COMANDO>");
  EnviaDataPcRemoto("C:<RESTO DEL COMANDO>"); //ECO SERVIDOR
}
if(BuscarMensajeEnListaSim900("E:<RESTO DEL COMANDO>"))
{ // Transición R
  // Respuesta desde servidor para el comando C: ante archivo
  // existente
  EnviarStringPC("E:<RESTO DEL COMANDO>");
  EnviaDataPcRemoto("E:<RESTO DEL COMANDO>"); //ECO SERVIDOR
}
else if(BuscarMensajeEnListaSim900("D:<RESTO DEL COMANDO>"))
{ // Transición S
  // Respuesta desde servidor para el comando D:
  EnviarStringPC("D:<RESTO DEL COMANDO>");
  EnviaDataPcRemoto("D:<RESTO DEL COMANDO>"); //ECO SERVIDOR
}
```

Fig. 3.26. `* *CODIGO SECCION COMUNICACIÓN REMOTO * *`

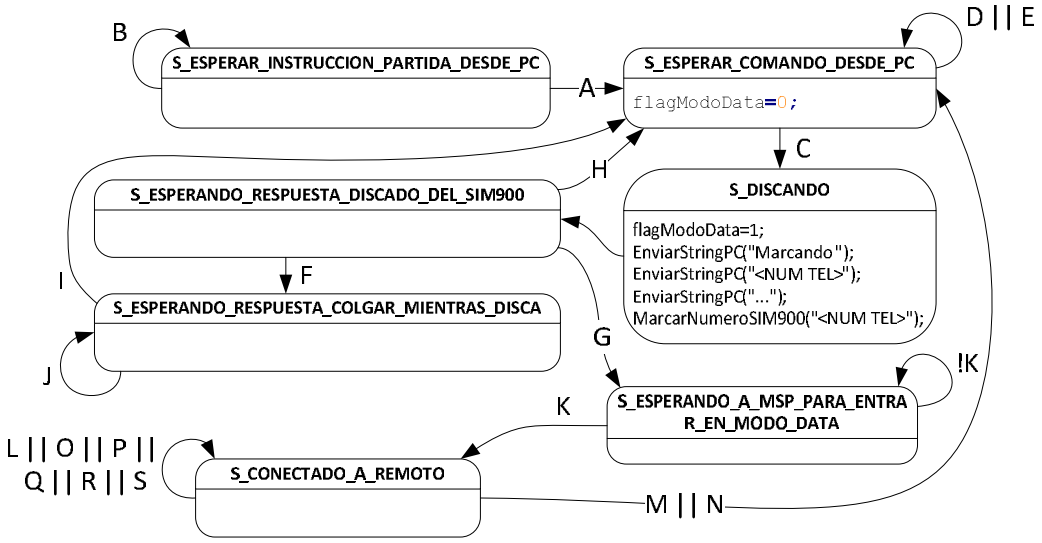


Fig. 3.27. Diagrama de estados simplificado para ***Fsm_ComunicacionPc().***

CAPÍTULO 4

4. Diseño del software lado cliente y lado servidor

En este capítulo se mostrará en forma general la estructura de los programas utilizados tanto en el computador cliente como en el computador servidor. El programa lado cliente tiene por objetivo controlar el módem USB especializado mediante el envío de mensajes que son interpretados por el firmware del MSP430. Además el programa expone una interfaz gráfica o GUI donde es posible visualizar distintos indicadores que reflejan el estado en que se encuentra el módem USB especializado. La interfaz gráfica a su vez otorga la posibilidad de cargar el archivo de texto a ser transmitido al servidor.

El programa lado servidor tiene por objetivo recibir los comandos en modo datos para crear el archivo o reanudar el archivo de texto.

Además ambos programas contienen la lógica que permite manejar el protocolo de comunicaciones para la conexión.

El lenguaje de programación para ambos programas es Microsoft C#.

4.1 Estructura del software lado cliente

En esta sección se pretende dar una explicación general de cómo está estructurado el software lado cliente. Se mostrará la interfaz gráfica diseñada, lógica de comunicación con el módem USB especializado y la lógica de comunicación con el servidor.

4.1.1 Interfaz gráfica GUI (Graphic User Interface)

La interfaz gráfica para esta aplicación se muestra en la figura 4.1. En ella se pueden apreciar varios elementos que se encuentran marcados con letras de la “a” a la “l”, cuyas descripciones se muestran a continuación:

a. Puerto COM donde se detectó la presencia del módem USB especializado.
b. Muestra las tarjetas SIM instaladas. Cuando el triángulo es de color rojo indica SIM instalada.
c. Indica si existe cobertura con la red celular. Es equivalente a decir que se inició sesión.
d. Indica estado del microcontrolador. Solamente utilizado para depuración.

e. Indica estado del módulo SIM900. En caso de estar encendido se muestra "ON" en caso contrario "OFF"
f. Número telefónico donde se encuentra el módem del computador servidor.
g. Botones "Marcar" y "Colgar" la comunicación. Se habilitan solamente cuando se ha detectado cobertura.
h. Indica el estado en que se encuentra la conexión. Los valores que puede asumir son:
    1. "INACTIVO": No hay cobertura.
    2. "Listo para discar": Hay cobertura y es posible marcar un número.
    3. "Discando": Discando número de teléfono.
    4. "Esperando Respuesta Discado": Esperando respuesta del servidor.
    5. "CONECTADO CON REMOTO": Conexión establecida con el servidor correctamente.
    6. "ARCHIVO TRANSMITIDO": Archivo transmitido con éxito

i. Botón para seleccionar archivo de texto a transmitir.
j. Visualización de información recibida desde el módem USB especializado, utilizado para fines de depuración solamente.
k. No implementado. Sin utilización.
l. Muestra el contenido del archivo seleccionado para transmitir.

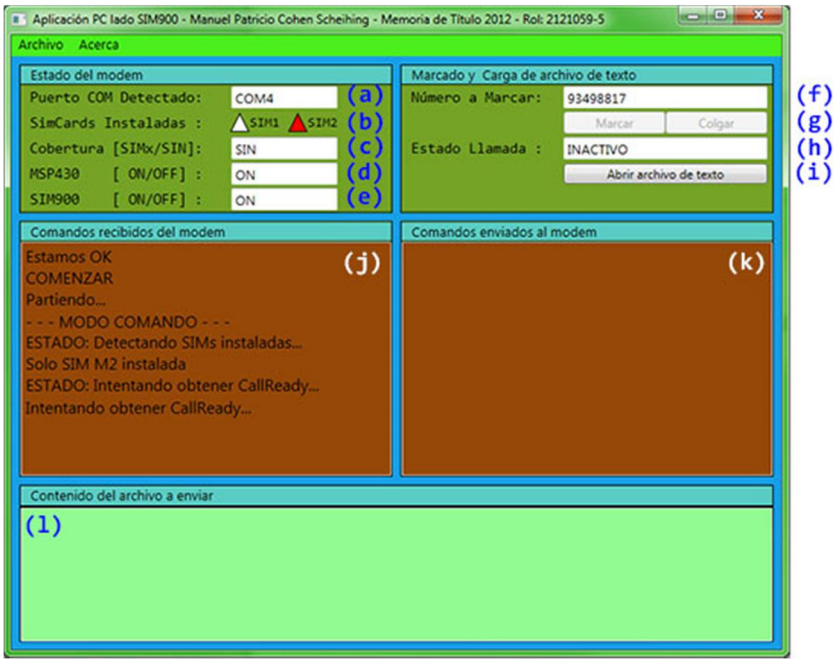


Fig. 4.1. Vista de la interfaz gráfica o GUI de la aplicación lado cliente

4.1.2 Estructura de los mensajes entre los computadores cliente y servidor

Una vez que se realiza la conexión entre ambos computadores, éstos emplean un sistema de mensajes para comunicarse y realizar la tarea de transmisión del archivo de texto. Estos mensajes son parte del protocolo de comunicaciones, que son enviados una vez que ambos computadores se encuentran conectados, y por ende el módem USB especializado se encuentra en modo datos. Los comandos enviados desde el computador cliente al servidor así como las respuestas de este último se encuentran a continuación.

1. Comando tipo "**C:**". Enviado por el computador cliente crear un archivo en el servidor. Este comando es interpretado solamente por este último ya que el módem USB especializado es transparente en modo datos.

   Su estructura es "**C:<NOM_ARCHIVO>:<NUM PAQUETES>**". Donde ambas "**<NOM_ARCHIVO>**" y "**<NUM PAQUETES>**" son "strings" que contienen el nombre del archivo a crear y su número de paquetes respectivamente. Cada paquete corresponde a 130 bytes de datos. El carácter "**:**" es un separador.

   Ante este comando el servidor puede responder de dos distintas maneras, la primera corresponde a cuando el archivo no existe mientras que la segunda corresponde a cuando el archivo existe y debe ser reanudada su transmisión a partir del último paquete recibido con éxito.

   En caso que el archivo no exista en el servidor, este último responde con el "string" "**C:<NOM_ARCHIVO> Creado OK**" para indicar que el archivo fue creado con éxito. En caso de que el archivo exista el servidor responde con el mensaje "**E:<ULTIMO_BYTE>**" donde "**<ULTIMO_BYTE>**" es un "string" que indica el último byte recibido con éxito. Luego de enviar el mensaje de respuesta, el servidor se prepara para recibir datos.

2. Comando tipo "**D:**". Utilizado para enviar un paquete de datos al servidor. Todos los paquetes son de 130 bytes con la excepción del último donde la cantidad de bytes puede ser menor. Este comando es utilizado una vez que el programa lado cliente ha recibido una respuesta desde el servidor como las que se detallan en el punto 1. Su estructura es "**D:<NUM BYTES>:<NUM PAQUETE ACTUAL>:<SUMA TRUNC>:<DATA>**". Donde "**<NUM BYTES>**" es el número de bytes del paquete actual, "**<NUM PAQUETE ACTUAL>**" el número de paquete siendo transmitido, "**<SUMA TRUNC>**" suma truncada para de-

tección de errores y "**<DATA>**" los bytes de datos del paquete actual. El carácter "**:**" es un separador. "**<SUMA TRUNC>**" no se utiliza ya que está pensada para desarrollos futuros.

Las respuestas del servidor ante este comando pueden ser 2. La primera corresponde a cuando el paquete recién transmitido a este último fue recibido en forma satisfactoria para lo cual el mensaje enviado al computador cliente es "**D:OK:<NUM BYTES>:<NUM PAQUETE ACTUAL>**". Con este mensaje se reporta al computador cliente que la transmisión del paquete número "**<NUM PAQUETE ACTUAL>**" cuya cantidad de bytes es "**<NUM BYTES>**" fue recibido correctamente. En caso que no hayan sido recibidos correctamente los bytes del paquete, el mensaje enviado al computador cliente es "**NOK**".

4.1.3 Programas que componen el software

El programa lado servidor se compone de varias clases las cuales se muestran en la figura 4.2. En ella se pueden apreciar la separación que existe entre las clases de alto y bajo nivel. A continuación se dará una descripción básica de cada nivel. En las subsecciones siguientes de este capítulo solo se ahondará en los niveles 3 y 4.

En las clases de bajo nivel primero se encuentra el archivo "FTD2XX_NET.DLL" disponible en el sitio web de FTDI [34]. En dicho archivo se encuentra la clase que contiene los métodos necesarios para tener un control completo del o los dispositivos FTDI conectados. Para el caso de este trabajo solamente se considera un dispositivo FTDI el cual corresponde al controlador USB FT232R que es parte del módem USB especializado.

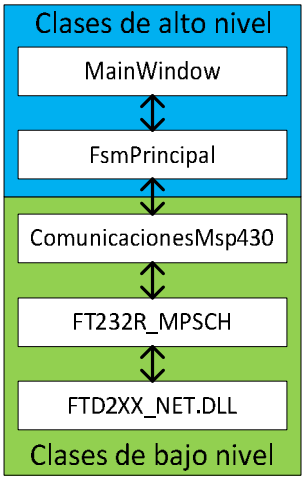


Fig. 4.2. Clases que componen al programa lado cliente

En segundo nivel se encuentra la clase FT232R_MPSCH que fue diseñada durante este trabajo de memoria. Tiene por objetivo encapsular y facilitar el uso de la clase provista por el archivo ".dll" del primer nivel. En esta clase se proveen métodos que permiten hacer utilización del módulo FT232R de una manera más fácil para el programador en cuanto a la configuración de este último. A su vez la clase expone un evento, similar a una interrupción en el caso de un microcontrolador, el cual se ejecuta cada vez que la clase contenida en el archivo ".dll" del primer nivel señala que hay bytes en el búfer de recepción para ser consumidos/leídos.

En tercer nivel se encuentra la clase **ComunicacionesMsp430**. En esta clase se configura la COM VIRTUAL o UART mediante la utilización de la clase FT232R_MPSCH. La configuración se realiza en el método ***ConfiguraUART()*** cuyos detalles pueden ser apreciados en el código fuente adjunto en el disco CD [35] de este trabajo de memoria. Esta misma clase también expone un evento el cual es invocado cada vez que ha llegado un mensaje por la COM VIRTUAL. Donde los mensajes llegados pueden ser respuestas del módem USB especializado como también respuestas desde el computador servidor. Esta clase será explicada en forma general en la subsección 4.1.5.

En el cuarto nivel se encuentra la clase **FsmPrincipal** que implementa la máquina de estados principal del programa lado cliente. Esta clase es la que se pretende explicar en forma general en la subsección 4.1.4.

En el quinto nivel se encuentra la clase **MainWindow** que es aquella que contiene las inicializaciones de los distintos objetos que conforman la interfaz gráfica como también la declaración del objeto **fsmPrincipal** que es una instancia de la clase **FsmPrincipal**.

4.1.4 Clase FSM Principal

Esta clase contiene varios métodos donde dos de ellos son los que explicaran en esta subsección. Ellos corresponden a ***FsmRedCelular()*** y a ***FsmConexionRemoto().*** En el primero se maneja toda la lógica para comunicarse con el módem USB especializado cuando se encuentra en modo comando. El segundo método contiene la lógica de comunicación a ser utilizada con el servidor o computador remoto. El llamado a ambas máquinas de estado se realiza cada vez que ha llegado un mensaje válido desde el módem USB especializado, el cual es señalizado por un evento implementado dentro de la clase **ComunicacionesMsp430**. En las figuras 4.3 y 4.4 se muestran los estados que pueden asumir ambas máquinas de estado.

```
enum EstadoFsmConexionRedCelular
{
    EsperandoReporteMcu,
    Partir,
    DetectandoSimsInstaladas,
    EsperandoCallReady,
    VerificandoCobertura,
    ListoParaDiscar,
    EsperandoEcoComandoDiscarMsp,
    Discando,
    EsperandoRespuestaDiscado,
    ConectadoConRemoto
};
```

Fig. 4.3. Estados de ***FsmRedCelular()***

```
enum EstadoFsmConexionDatosRemoto
{
    EnviarComandoCrearArchivo,
    EsperarRespuestaCrearArchivo,
    TransmisionPaquete,
    EsperarRespuestaTransmisionPaquete,
};
```

Fig. 4.4. Estados de ***FsmConexionRemoto()***

En el constructor de la clase se llama al método de configuración de la COM VIRTUAL implementada en la clase **ComunicacionesMsp430**, donde en ella se busca el puerto COM donde se encuentra el módem USB especializado y se configuran sus parámetros. Los cuales son 115.200 baudios por segundo, 8 bits de datos, 1 bit de parada, sin paridad y con control de flujo por hardware RTS/CTS. Estos son los parámetros de configuración que se utilizaron también para configurar la UART del microcontrolador MSP430 que serán mostradas en el Anexo 2.

En el primer estado de ***FsmRedCelular()*** , el cual es **EsperandoReporteMcu**, se envía el comando "COMENZAR" al módem USB especializado, y se espera el "string" "**Estamos OK**" de respuesta que envía el MSP430 para indicar que se encuentra listo para comenzar. Una vez recibido este "string", se procede al estado **Partir**.

En el estado **Partir** se espera la llegada del mensaje "- - - MODO COMANDO - - -", el cual es enviado por el módem USB especializado cuando se encuentra en modo comando. Una vez recibido el mensaje, se procede al estado **DetectandoSimsInstaladas**.

En el estado **DetectandoSimsInstaladas**, se espera un mensaje que indique cuales tarjetas SIM se encuentran físicamente instaladas en los "sockets". Existen cuatro mensajes distintos que pueden ser enviados en este momento, los cuales son: "Ninguna SIM instalada", "Solo SIM M1 instalada", "Solo SIM M2 instalada" y "SIM M1 y SIM M2 instaladas".

En el caso que el primer mensaje es recibido, se permanece en el mismo estado. Para el caso de los últimos 3 mensajes se procede al estado **EsperandoCallReady.** La GUI es actualizada para mostrar las tarjetas SIM instaladas en el marcador (b) de la figura 4.1.

En el estado **EsperandoCallReady** se espera que llegue un mensaje indicando que se ha encontrado cobertura con alguna de las tarjetas SIM instaladas. El mensaje que se espera puede ser uno de los cuatro posibles, "Encontramos SIM M1 ARRIBA con CallReady", "Encontramos SIM M2 ABAJO con CallReady", "ESTADO: Detectando SIMs instaladas..." y "Se superaron los K_MAX_INTENTOS de obtener Call Ready. Cambiando de SIM...".

Para el caso del primero y segundo mensaje significa que se ha encontrado cobertura por primera vez. El próximo estado será para este caso **VerificandoCobertura**.

En el estado **VerificandoCobertura** permite determinar si la cobertura es estable o no. Para decir que la cobertura es estable, se requiere la llegada de 3 mensajes seguidos indicando que existe cobertura y se establece el próximo estado en **ListoParaDiscar.**

Una vez hecho esto, se habilita el botón "Marcar" marcado con (g) y se cambia el contenido de (h) a "Listo para discar" en la figura 4.1.

En el estado **ListoParaDiscar** se espera que el usuario disque un número de teléfono mediante el botón "Marcar" en cuyo caso se procede al estado **EsperandoEcoComandoDiscarMsp**. Se permanece en este estado mientras no se disque un número de teléfono y exista cobertura. En caso que ésta última se pierde se procede al estado **DetectandoSimsInstaladas** para volver a repetir la ejecución desde ese estado.

En el estado **EsperandoEcoComandoDiscarMsp**, se espera el eco del comando para discar, el cual mientras no haya llegado, se permanece en este estado. En el caso que el eco es recibido, se procede al estado **Discando**.

En el estado **Discando** se actualiza el indicador señalado por (h) de la figura 4.1 con el mensaje "Discando". Se permanece en este estado hasta que se reciba el mensaje "- - - MODO DATA - - -" desde el módem USB especializado que indica que se está intentando entrar en modo datos. Cuando llega este mensaje se procede al estado **EsperandoRespuestaDiscado.**

En el estado **EsperandoRespuestaDiscado**, se habilita el botón "Colgar" y se espera la llegada de uno de tres tipos de mensajes. Mientras no se reciba ninguno de estos 3 mensajes, se permanece en este estado. Los mensajes a recibir son "Colgado OK", "- - - MODO COMANDO - - -" o "CONECTADO A REMOTO...". Si se recibe primer mensaje esto significa que el botón "Colgar" fue presionado en una ejecución anterior de este estado, en cuyo caso el estado próximo será **ListoParaDiscar**. Si se recibe el segundo mensaje, significa que la línea de teléfono en la que se encuentra el servidor se encuentra ocupada o sin respuesta, en cuyo caso el estado próximo también será **ListoParaDiscar**. En caso que el mensaje recibido sea el tercero, significa que se ha establecido la conexión satisfactoriamente al servidor y se ha entrado en modo datos, en cuyo caso, el estado próximo es **ConectadoConRemoto**.

El último estado de esta máquina de estados es **ConectadoConRemoto.** Se permanece en este estado siempre que los mensajes llegados sean distintos de "Colgado OK" y "- - - MODO COMANDO - - -" y se actualiza el indicador señalado por (h) de la figura 4.1 con el mensaje "CONECTADO CON REMOTO". En caso que se decide abortar la comunicación en forma voluntaria desde el computador cliente mediante el botón "Colgar" llegará desde el módem USB especializado el mensaje "Colgado OK". En el caso que haya perdido la cobertura o que simplemente se haya perdido la conexión por un problema en el servidor o línea telefónica, se procede al estado **ListoParaDiscar**. En

caso que la pérdida de conexión no se haya debido a problemas de cobertura, se permanecerá en el estado **ListoParaDiscar**, donde se puede volver a realizar el discado telefónico.

Una vez que se está conectado al servidor, cada vez que llegue un mensaje por la COM VIRTUAL se llamará a la máquina de estados ***FsmConexionRemoto()*** que tiene por objetivo manejar el protocolo de comunicaciones con el servidor y verificar que el archivo de texto a transferir haya sido seleccionado mediante el botón "Abrir archivo de texto" señalado en la figura 4.1 por (i), en cuyo caso se comienza por el primer estado **EnviarComandoCrearArchivo**. En caso que el archivo no haya sido seleccionado aún, la máquina de estados retornará sin resultados en forma permanente. Los estados que puede asumir se muestran en la figura 4.4.

El primer estado esta máquina de estados es **EnviarComandoCrearArchivo** en el cual se envía al servidor "**C:<NOM_ARCHIVO>:<NUM PAQUETES>**" que corresponde al comando crear archivo. Una vez enviado el comando, se procede al estado **EsperarRespuestaCrearArchivo**.

En el estado **EsperarRespuestaCrearArchivo** se permanece hasta recibir la respuesta del servidor. Las respuestas válidas desde el servidor a esperar durante este estado son "**C:<NOM_ARCHIVO> Creado OK**" o "**E:<ULTIMO_BYTE>**", donde se confirma la creación del archivo en servidor o se indica el último byte recibido con éxito respectivamente. En caso que llegara una respuesta errónea por algún motivo se permanece en el mismo estado hasta recibir una respuesta válida.

Para el caso de ambas respuestas "**C:<NOM_ARCHIVO> Creado OK**" y de "**E:<ULTIMO_BYTE>**"**,** se procede con el estado **TransmisionPaquete.** Para el caso de que el mensaje recibido sea "**C:<NOM_ARCHIVO> Creado OK**" se procede a enviar el primer paquete mediante "**D:<NUM BYTES>:<NUM PAQUETE ACTUAL>:<SUMA TRUNC>:<DATA>**", donde "**<NUM PAQUETE ACTUAL>**"="0".

Para el caso que es mensaje recibido es "**E:<ULTIMO_BYTE>**" se calcula a partir del "string" "**<ULTIMO_BYTE>**" el último paquete que es necesario retransmitir y se almacena en "**<NUM PAQUETE ACTUAL>**". Luego se envía el comando "**D:<NUM BYTES>:<NUM PAQUETE ACTUAL>:<SUMA TRUNC>:<DATA>**" al servidor.

El último estado corresponde a **TransmisionPaquete**, el cual tiene por objetivo recibir la respuesta del servidor ante la primera transmisión del primer paquete realizada en

el estado **EsperarRespuestaCrearArchivo**, enviar los paquetes restantes y verificar las respuestas del servidor acusando el recibo correcto de estos.

En este estado se pueden recibir desde el servidor el mensaje "**D:OK:<NUM BYTES>:<NUM PAQUETE ACTUAL>**" o "**NOK**". Cuando llega el primer mensaje significa que el paquete cuyo número es "**<NUM PAQUETE ACTUAL>**" fue recibido en forma correcta, en este caso se incrementa el valor de "**<NUM PAQUETE ACTUAL>**" en "1" y se realiza su envío mediante el comando "**D:<NUM BYTES>:<NUM PAQUETE ACTUAL>:<SUMA TRUNC>:<DATA>**" en caso que queden paquetes restantes. En caso que es el último paquete, se finaliza la comunicación y se actualiza la GUI indicando el mensaje "ARCHIVO TRANSMITIDO" en (h) de la figura 4.1.

4.1.5 Clase de Comunicaciones (ComunicacionesMsp430)

Esta clase de comunicaciones tiene por finalidad proveer una capa de abstracción para la FsmPrincipal en cuanto al manejo de la COM VIRTUAL. La configuración de esta última se realiza a través del método ***ConfiguraUART()*** que se muestra en la figura 4.5. Este método detecta el puerto COM VIRTUAL donde se encuentra el módem USB especializado y es posible definir su velocidad como todas sus características de configuración. Para este caso se utiliza una velocidad de 115.200[bps], 8 bits de datos, 1 bit de parada, no paridad y control de flujo por hardware. La detección automática tanto como la configuración se realiza definiendo los distintos campos del objeto **_uart** y luego llamando al método ***Configurador()*** de la clase FT232R_MPSCH. En caso que la configuración sea exitosa se retorna el puerto COM donde se encuentra el módem USB especializado el cual es utilizado para modificar el campo marcado por (a)

```
private void ConfiguraUART(out string puertoCOM)
{
    int resultadoAbrirUart = -1;
    puertoCOM = "";
    _uart.DataRecibidaEvent += new LlegoDataEventHandle(_uart_DataRecibidaEvent);
    _uart.BaudRateProp = UART_FT232R.BRateEnum.Bps115200;
    _uart.DataBitsProp = UART_FT232R.DataBitsEnum.Bits8;
    _uart.StopBitsProp = UART_FT232R.StopBitsEnum.Bits1;
    _uart.ParidadProp = UART_FT232R.ParidadEnum.Ninguna;
    _uart.ControlFlujoProp = UART_FT232R.ControlFlujoEnum.RTS_CTS;
    _uart.FlushBuffer();
    _uart.Configurador();
    resultadoAbrirUart = _uart.AbreUART().CompareTo("OK");
    if (resultadoAbrirUart == 0)
        puertoCOM = _uart.DevuelvePuertoCOM();
    else
        puertoCOM = "";
}
```

Fig. 4.5. Método ***ConfiguraUART()*** de la clase ComunicacionesMsp430

en la GUI mostrada en la figura 4.1. La detección del módem USB especializado será explicada en el Anexo 3.

Cada vez que llegan hay bytes para ser leídos desde el "pool" del driver de la COM VIRTUAL, señalizado por la clase FT232R_MPSCH, se ejecuta el evento de la clase ComunicacionesMsp430 llamado ***_uart_DataRecibidaEvent***() el cual se muestra en la figura 4.6.

```
void _uart_DataRecibidaEvent(byte[] data)
{
    string datos = System.Text.Encoding.UTF8.GetString(data);
    //Lee la data y cada vez que ingresa la sobre-escribe
    //Invoca al delegado (apunta al evento "si_DataReceived")
    //produciendo en evento, y le envía la "data"
    bufferEntrada += datos;
    ProcesarBuffer(ref bufferEntrada);
}
```

Fig. 4.6. Evento ***_uart_DataRecibidaEvent()*** de la clase ComunicacionesMsp430

Los bytes que vienen llegando van siendo almacenados en la variable "string" **bufferEntrada** cuyo contenido es entregado al método ***ProcesarBuffer()*** que se muestra en la figura 4.7.

```
private void ProcesarBuffer(ref string s)
{
    int start = 0;
    int end = 0;
    bool continuarLoop = true;
    string mensajeString;
    while (continuarLoop == true)
    {
        start = s.IndexOf(char.ConvertFromUtf32(0x02));
        end = s.IndexOf(char.ConvertFromUtf32(0x02), start + 1);
        if ((start != -1) && (end != -1))
        {
            mensajeString = s.Substring(start + 1, end - start - 1);
            s = s.Substring(end + 1, s.Length - end - 1);

            MensajeLlegadoEventArgs mensajeLlegadoEventArgs = new MensajeLlegadoEventArgs();
            mensajeLlegadoEventArgs.Mensaje = mensajeString;
            LlegoMensaje(this, mensajeLlegadoEventArgs);
            ;
        }
        else
        {
            continuarLoop = false;
        }
    }
}
```

Fig. 4.7. Método ***ProcesarBuffer()*** de la clase ComunicacionesMsp430

El método ***ProcesarBuffer()*** tiene por objetivo procesar los bytes que se encuentran en la variable "string" **bufferEntrada** para encontrar mensajes delimitados por los carac-

teres **0x02**. Cada vez que se encuentra un mensaje se produce un evento en clase que ha instanciado a **ComunicacionesMsp430**, que es en otras palabras **FsmPrincipal**.

En esta clase también se exponen los métodos para enviar los distintos tipos de comandos tanto para el módem USB especializado como para el servidor. Los comandos implementados en esta clase se encuentran definidos por los siguientes métodos cuyos nombres fueron elegidos de tal forma que sean auto-explicativos:

- ***EnviaComandoComenzar()***
- ***EnviaComandoColgar()***
- ***EnviaComandoMarcar(****string* ***numTel)***
- ***CrearArchivoRemoto(****string* ***nombre,*** *string* ***numPaquetes)***
- ***EnviaComandoDataArchivo(****string* ***numPaquete,*** *string* ***numBytes,*** *string* ***sumaBytesTruncada,*** *string* ***data)***

  Donde ***numTel*** corresponde al número de teléfono a discar, ***nombre*** al nombre del archivo a transmitir, ***numPaquetes*** el número de paquetes a transmitir, ***numPaquete*** número de paquete actual, ***numBytes*** número de bytes restantes, ***sumaBytesTruncada*** suma truncada para detección de errores y ***data*** datos del paquete actual del archivo de texto a transmitir al servidor.

4.2 Estructura del software lado servidor

En esta sección se pretende dar una explicación general de cómo está estructurado el software lado servidor. Se mostrará la interfaz gráfica diseñada, lógica de comunicación con el módem convencional y la lógica de comunicación con el cliente.

4.2.1 Interfaz gráfica GUI (Graphic User Interface)

La interfaz gráfica, que se muestra en la figura 4.8, para este programa es muy sencilla en comparación al software lado cliente. Posee tres objetos "text box" marcados con las letras de la "a" a la "c". Sus descripciones se muestran a continuación:

a. Indica el estado de la comunicación con el computador cliente. Sus valores posibles son "SIN CONEXION" y "CONECTADO CON CLIENTE".
b. Muestra los comandos recibidos desde el computador cliente en forma secuencial.
c. Muestra el contenido del archivo de texto o a medida que van siendo recibidos los paquetes.

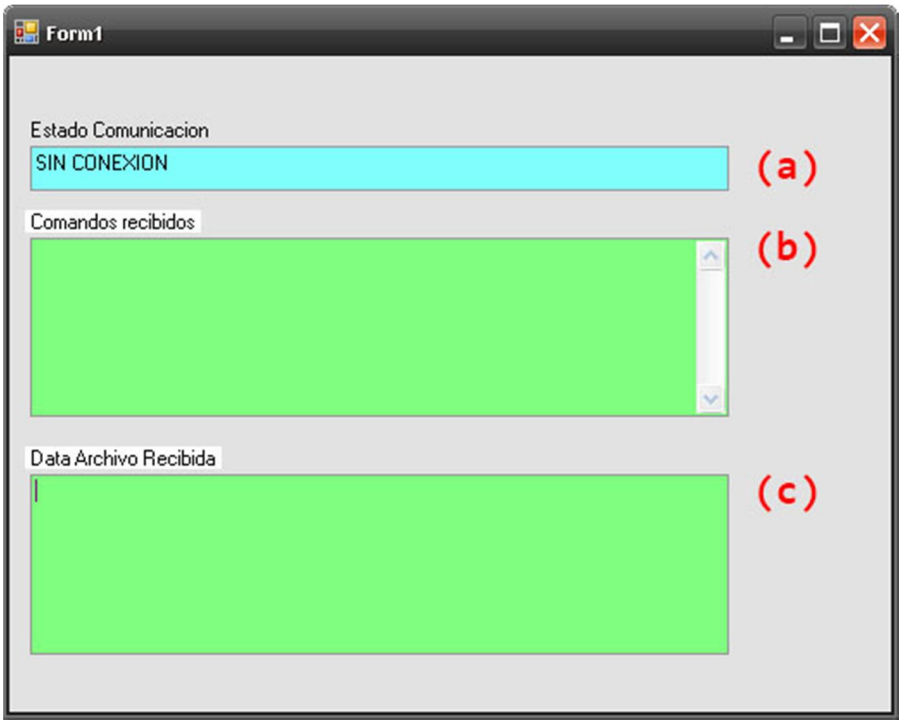


Fig. 4.8. Vista de la interfaz gráfica o GUI de la aplicación lado servidor.

4.2.2 Programas que componen el software

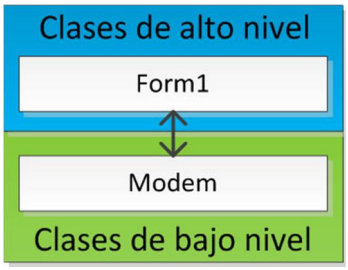


Fig. 4.9. Clases que componen al programa lado servidor

El software implementado en el servidor, que se compone de las clases mostradas en la figura 4.9, tiene la finalidad de poder comunicarse con el software lado cliente cuando existe una comunicación de datos establecida. Cuando esto sucede el módem USB especializado funciona en modo transparente y por ende los comandos y respuestas son solamente interpretados por ambos computadores cliente y servidor con la excepción del comando "COLGAR_MD" el cual permite salir del modo datos al cortar la comunicación. Una vez cortada la comunicación se retorna a modo comando. El comando "COLGAR_MD" se define en la sección 3.6.

La única clase de bajo nivel es "Modem" la cual tiene por objetivo manejar el puerto COM donde se encuentra el módem convencional en el computador servidor. En este programa no se realiza una búsqueda automática del puerto convencional como en el caso del programa cliente donde se detecta la COM VIRTUAL del módem USB especializado.

```
public void ActualizarGui(string mensaje)
{
    if (mensaje == "RING")
    {
        modem.Contestar();
        CambiarMensajeEstado("CONECTADO CON CLIENTE");
    }
    else if (mensaje.Contains("C:"))
    {
        if (AbrirArchivo(<NOM_ARCHIVO>) == "Archivo creado")
        {
            modem.Enviar("C:<NOM_ARCHIVO> Creado OK>");
            <ULTIMO_BYTE> = "0";
        }
        Else (AbrirArchivo(<NOM_ARCHIVO>) == "Existente")
        {
            <ULTIMO_BYTE> = ExtraerUltimoByte(<NOM_ARCHIVO>);
            modem.Enviar("E:<ULTIMO_BYTE>");
        }
    }
    else if (mensaje.Contains("D:"))
    {
      modem.Enviar("D:OK:<NUM BYTES>:<NUM PAQUETE ACTUAL>");
      txtBoxDataArchivoRecibida.Text += <DATA>;
      AgregarDataAlArchivo(<DATA>);
      if (<NUM PAQUETE ACTUAL>== (<NUM PAQUETES>-"1"))
      {
            Cerrar(<NOM_ARCHIVO>);
            modem.Colgar(); ();
      }
    }
}
```

Fig. 4.10. Método ***ActualizarGui()*** de la clase Form1

La única clase de alto nivel es Form1. Esta clase contiene las inicializaciones de los distintos objetos que conforman la interfaz gráfica como también la declaración del objeto **modem** que es una instancia de la clase Modem. Además implementa la lógica de comunicación con el computador cliente. Esta clase es la que se pretende explicar en forma general en la subsección 4.2.3.

4.2.3 Clase principal (Form1)

En el constructor de esta clase se inicializan los objetos que componen el programa junto con declarar el objeto **modem** que es una instancia de la clase Modem que se utilizará para comunicarse con el módem convencional. La rutina en donde se encuentra la lógica de comunicación con el computador cliente se encuentra en el método ***ActualizarGui()*** mostrado en la figura 4.10. Este método es llamado cada vez que hay un mensaje esperando ser consumido, condición que es señalizada por la clase Modem.

Cuando el computador cliente disca el número de teléfono del computador servidor, se recibe en este último el mensaje "RING", con una cadencia equivalente a la que se produciría en la campanilla de un teléfono convencional si estuviese conectado a la línea. Cuando este mensaje se recibe, se llama al método ***Contestar()*** de la clase **Modem** con la cual se da la instrucción al módem convencional de contestar la llamada.

Una vez que se contesta la llamada se actualiza el mensaje del indicador marcado por (a) en la figura 4.7 a "CONECTADO CON CLIENTE".

Cuando se recibe desde el computador cliente un mensaje para crear archivo, pueden ocurrir 2 escenarios. En el primero el archivo no existe y por ende se crea. Se envía la respuesta al computador cliente "**C:<NOM_ARCHIVO>Creado OK**" y a **<NOM_ARCHIVO>** se le asigna el valor "0".

En el segundo escenario, el archivo existe por lo que se abre y se extrae el número del último byte, el cual se deja en "**<ULTIMO_BYTE>**" y posteriormente se le envía la respuesta "**E:<ULTIMO_BYTE>**" al computador cliente.

Una vez recibido el comando para crear archivo, llegarán desde el computador cliente comandos con los bytes de los paquetes. Este comando es "**D:<NUMBYTES>:<NUM PAQUETE ACTUAL>:<SUMA TRUNC>:<DATA>**". La respuesta que se envía al computador cliente es "**D:OK:<NUM BYTES>:<NUM PAQUETE ACTUAL>**". Para el caso del último paquete, el archivo se cierra y se finaliza la conexión.

4.2.4 Clase para el manejo del módem convencional (Modem)

Esta clase tiene por objetivo manejar el módem convencional que se encuentra instalado en el computador servidor. Su constructor que puede apreciar en la figura 4.11, es llamado por la clase Form1.

```
public Modem()
{
    _serialPort = new SerialPort(COM_Port, 9600, Parity.None, 8, StopBits.One);
    _serialPort.Handshake = Handshake.RequestToSend;//.None;//RTS/CTS
    _serialPort.DataReceived += new SerialDataReceivedEventHandler(serialPort_DataReceivedEvent);
    _serialPort.ReadTimeout = 5000;
    _serialPort.WriteTimeout = 5000;
}
```

Fig. 4.11. Constructor de la clase Modem

En el constructor se definen las características de comunicación para el puerto COM donde se encuentra instalado el módem convencional. La configuración se realiza mediante el objeto **SerialPort** [36] que viene incluido en el ".NET Framework". Se configura para una velocidad de 9600[bps], no paridad, 8 bits de datos, 1 bit de parada y control de flujo por hardware RTS/CTS.

Cada vez que llegan hay bytes para ser leídos desde el "pool" del driver del puerto COM , señalizado por la clase **SerialPort**, se ejecuta el evento de la clase Modem llamado ***serialPort_DataReceivedEvent()*** el cual se muestra en la figura 4.12.

```
private void serialPort_DataReceivedEvent(object sender, SerialDataReceivedEventArgs e)
{
    string datos = _serialPort.ReadExisting();
    //Invoca al delegado (apunta al evento "si_DataReceived")
    //produciendo en evento, y le envía la "data"
    bufferEntrada += datos;
    ProcesarBuffer (ref bufferEntrada);
}
```

Fig. 4.12. Evento ***serialPort_DataReceivedEvent()***

Los bytes que vienen llegando van siendo almacenados en la variable "string" **bufferEntrada** cuyo contenido es entregado al método ***ProcesarBuffer()*** cuya estructura es muy similar a la del método con el mismo nombre mostrada en la figura 4.7 para el programa lado cliente.

Con el método ***ProcesarBuffer()***,se procesan los bytes que se encuentran en la variable "string" **bufferEntrada** para encontrar mensajes delimitados por caracteres separadores. Cada vez que se encuentra un mensaje se produce un evento en clase que ha instanciado a Modem, que es en otras palabras Form1.

En esta clase también se encuentran los comandos para realizar el marcado, colgado y limpieza de los búferes de entrada y salida del puerto COM.

Detalles mayores del funcionamiento del software se encuentran en los archivos fuente incluidos en el disco CD [35] de este trabajo de memoria.

CAPÍTULO 5

5. Resultados y evaluación

Es este capítulo se pretende demostrar el funcionamiento con distintos escenarios de pruebas. Cada escenario de prueba irá en una sección diferente.

5.1 Detección del módem USB especializado

Para que el hardware sea reconocido bajo el sistema operativo Microsoft Windows, es necesario utilizar un "driver" el cual fue modificado en forma simple para que el descriptor sea "Modem USB Memoria MPCohen 2012". No se ahondará en este tema en este trabajo, sin embargo para mayor información el lector puede referirse al disco CD adjunto con esta memoria [35].

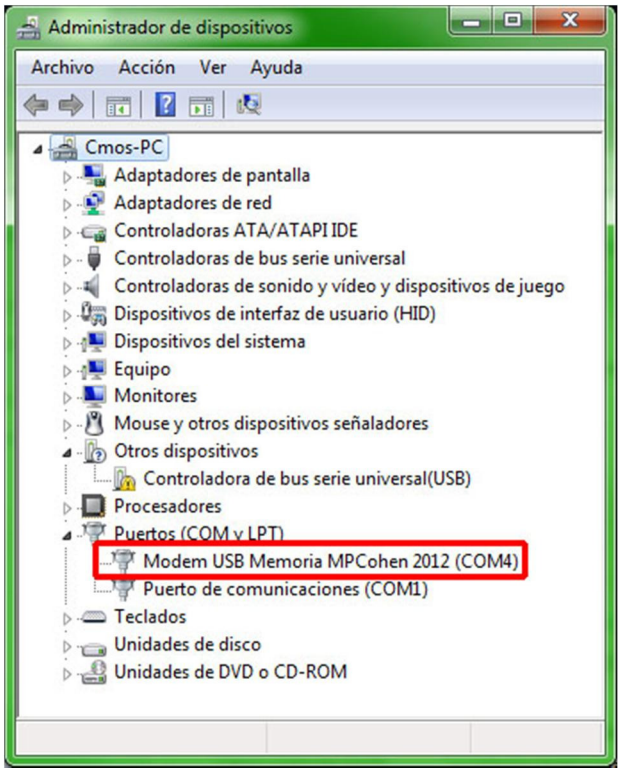


Fig. 5.1. Hardware detectado en Administrador de dispositivos

Al observar el "Administrador de Dispositivos" de Windows, el cual se muestra en la figura 5.1, es posible observar en la lista el hardware correspondiente al módem USB especializado destacado en rojo.

Esto demuestra que el módem USB especializado ha sido reconocido por el sistema operativo y que el puerto COM asignado por este último es el 4.

5.2 Detección de tarjetas SIM instaladas

En la figura 4.1 se mostró una vista de la GUI del programa lado cliente. En la figura 5.2 se puede apreciar el cambio que se produce en los indicadores marcados con (b) de la figura 4.1 para las 4 posibilidades respecto a la presencia de tarjetas SIM en los

"sockets". Para la figura 5.1 los casos son: (a) Ninguna SIM, (b) SIM 1 instalada, (c) SIM 2 instalada y (d) SIMs 1 y 2 instaladas. Además se aprecia el puerto detectado.

Fig. 5.2. Los 4 casos posibles de tarjetas SIM presentes en los "sockets"

5.3 Detección de cobertura

Para esta prueba se insertan las 2 tarjetas SIM en sus "sockets". Al primer intento con la tarjeta SIM 2 no fue posible encontrar cobertura ya que de adrede para esta prueba se desconecta la antena con la finalidad que se conmute a la tarjeta SIM 1 y la cobertura se encuentre con ella. Una vez realizada la conmutación, se instala la antena nuevamente y el módem USB especializado logra obtener cobertura. En la figura 5.3, se muestra una vista de la GUI del programa lado cliente.

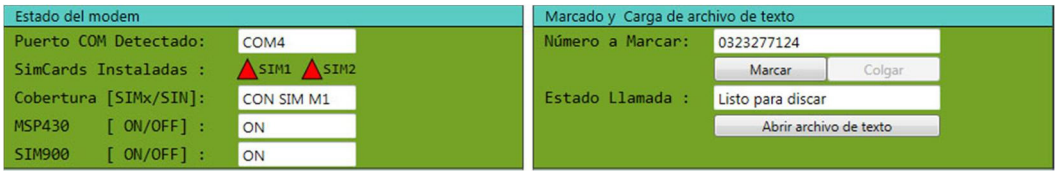


Fig. 5.3. Vista de la interfaz gráfica para el caso de detección de cobertura con la SIM 1

La secuencia de mensajes recibidos por el módem USB especializado se muestra por motivos de espacio en la figura 5.4, ya que debido al largo de la lista, era necesario mover el "scroll" de la "text box" de la aplicación para visualizarla completamente.

Estamos OK
COMENZAR
Partiendo...
- - - MODO COMANDO - - -
ESTADO: Detectando SIMs instaladas...
SIM M1 y SIM M2 instaladas
ESTADO: Intentando obtener CallReady...
Intentando obtener CallReady...
Encontramos SIM M2 ABAJO con CallReady
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Se cayo la cobertura. Reiniciando...
ESTADO: Detectando SIMs instaladas...
SIM M1 y SIM M2 instaladas
ESTADO: Intentando obtener CallReady...
Intentando obtener CallReady...
Encontramos SIM M1 ARRIBA con CallReady
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
Esperando
Verificando Cobertura...
1_valCounter:_3_Con cobertura
Verificando Cobertura...
0_valCounter:_4_Con cobertura
Verificando Cobertura...
1_valCounter:_5_Con cobertura
Verificando Cobertura...
0_valCounter:_6_Con cobertura
Verificando Cobertura...
1_valCounter:_7_Con cobertura

Fig. 5.4. Mensajes recibidos desde el módem USB especializado durante la prueba de obtención de cobertura

5.4 Conexión con servidor remoto

Para esta prueba se utilizará la tarjeta SIM 1 solamente ya que la otra que se tiene no posee saldo para cursar llamadas.

Desde el computador cliente se disca el número del computador cliente. Inicialmente el programa lado servidor se encuentra tal como se muestra en la figura 4.8. El programa cliente señala que se encuentra con cobertura con la tarjeta instalada y listo para discar según la figura 5.5.

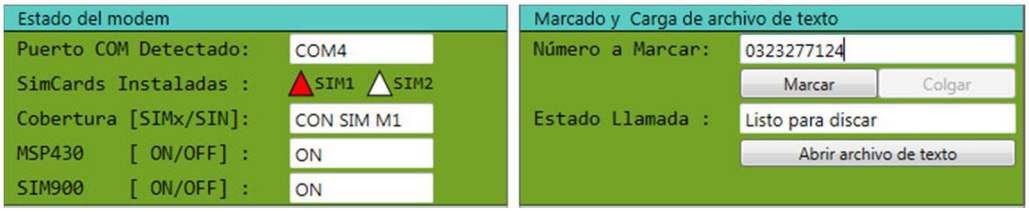


Fig. 5.5. Estado del programa cliente informando cobertura con la tarjeta SIM instalada.

Luego de haber ingresado el número a discar donde se encuentra el servidor, se presiona el botón "Marcar", el cual se encuentra habilitado ya que hay cobertura.

En la figura 5.6 se puede apreciar una vista de la aplicación lado cliente cuando se encuentra establecida la conexión de datos con el servidor. En la figura 5.7 se aprecia la vista de la aplicación del servidor para el mismo caso.

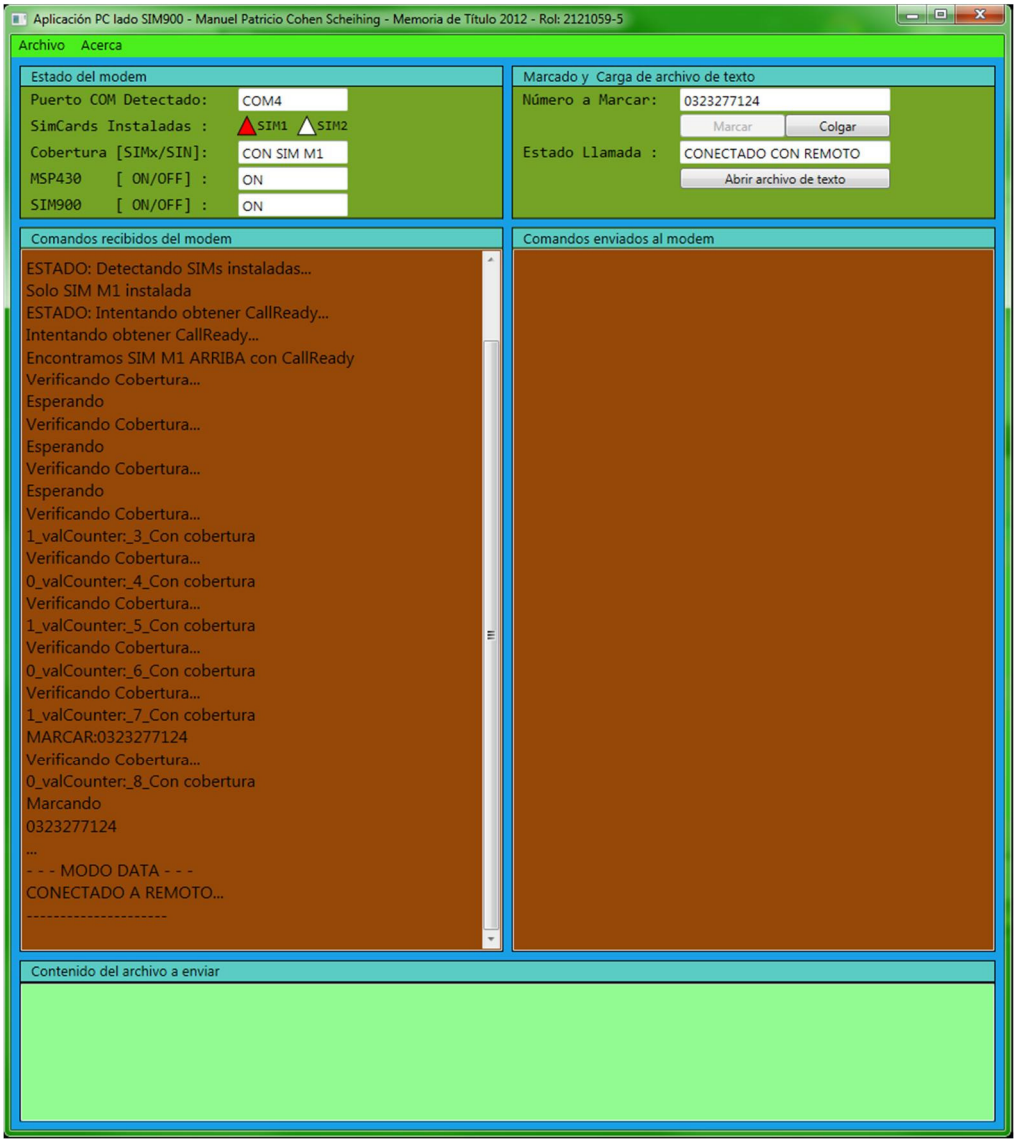


Fig. 5.6. Vista de la aplicación lado cliente para cuando se tiene una conexión de datos activa con el servidor

En el caso que se quiera abortar la comunicación bastará con presionar el botón "Colgar", el cual se encuentra habilitado por el hecho de existir una conexión de datos activa. En el caso que no hay conexión, el este botón está deshabilitado.

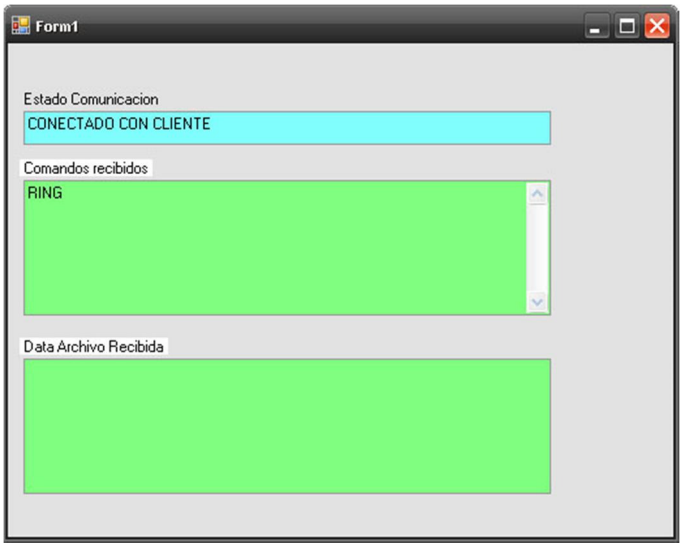


Fig. 5.7. Vista de la aplicación lado servidor para cuando se tiene una conexión de datos activa con el cliente

En la aplicación del lado servidor de la figura 5.7, se puede apreciar el indicador que señala que existe una conexión con el computador cliente.

También es posible apreciar un mensaje "RING" que llegó en el momento que el computador cliente discó el número de teléfono donde se encuentra el módem convencional del computador servidor.

5.5 Transmisión de un archivo de texto y resumen ante falla

En esta sección se pretende realizar una demostración de la transmisión de un archivo de texto cuando hay un corte de la comunicación producto de un problema en la línea telefónica. Para simular este escenario, se desconecta el cable telefónico del módem del computador servidor durante la transmisión.

El contenido del archivo a transmitir es el siguiente:

"*Galaxies have been historically categorized according to their apparent shape, usually referred to as their visual morphology. A common form is the elliptical galaxy,[6] which has an ellipse-shaped light profile. Spiral galaxies are disk-shaped with dusty, curving arms. Those with irregular or unusual shapes are known as irregular galaxies and typically originate from disruption by the gravitational pull of neighboring galaxies. Such interactions between nearby galaxies, which may ultimately result in a merger, sometimes induce significantly increased incidents of star formation leading to starburst galaxies. Smaller galaxies lacking a coherent structure are referred to as irregular galaxies.*"

En la figura 5.8 se puede apreciar una vista de la interfaz gráfica para el escenario donde se cae la conexión. En la parte inferior de la GUI se encuentra el contenido del archivo.

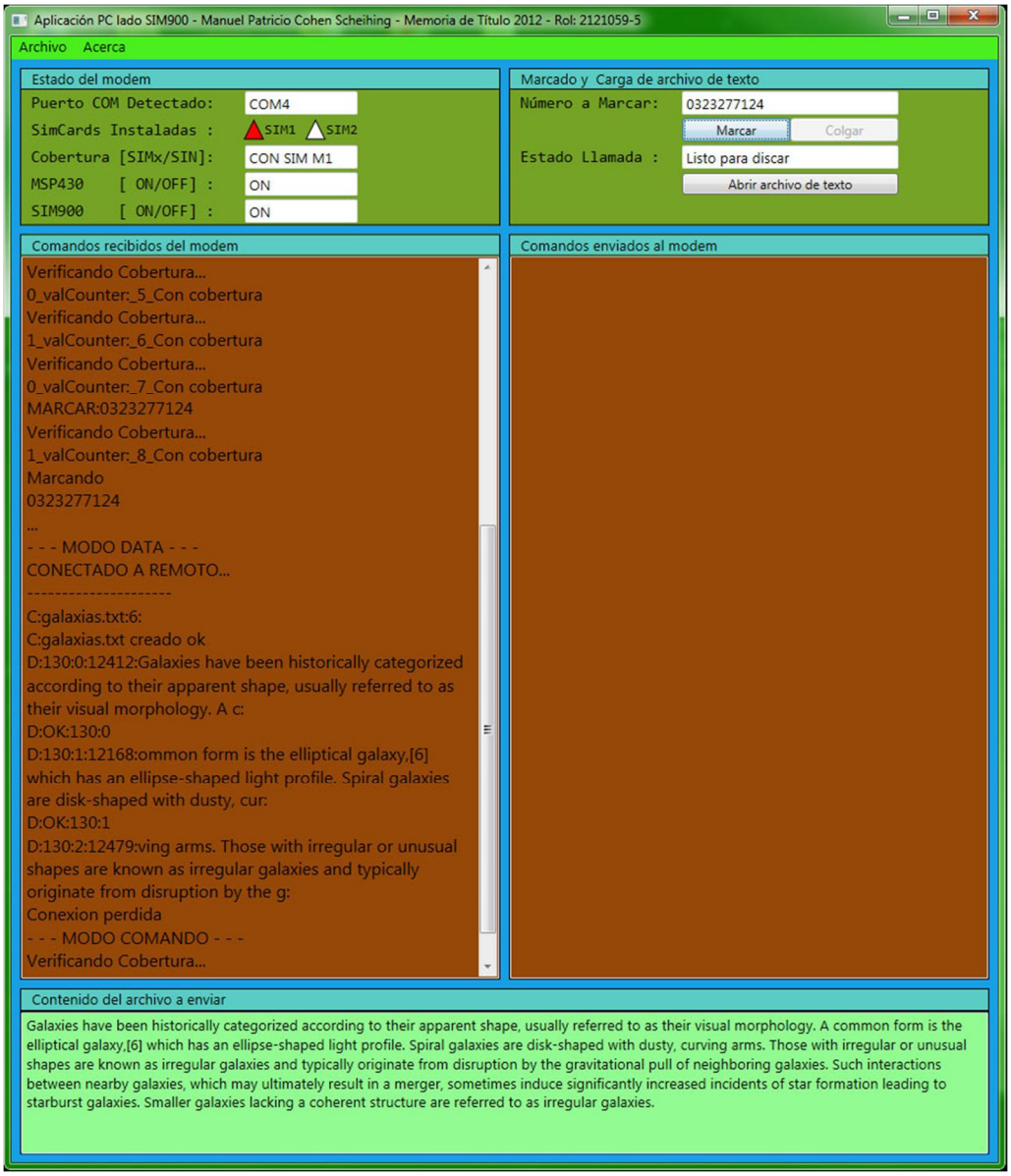


Fig. 5.8. Vista de la aplicación lado cliente para cuando se cae la conexión en medio de la transmisión

Los mensajes que se pueden apreciar en la figura 5.9, muestran la secuencia de envío de comandos. Los mensajes que están en letra **negrita** son aquellos enviados como respuestas desde el servidor al computador cliente. Tal como se puede apreciar, en el tercer paquete se presentó el problema de conexión.

C:galaxias.txt:6:
C:galaxias.txt creado ok

D:130:0:12412:Galaxies have been historically categorized according to their apparent shape, usually referred to as their visual morphology. A c:
D:OK:130:0

D:130:1:12168:ommon form is the elliptical galaxy,[6] which has an ellipse-shaped light profile. Spiral galaxies are disk-shaped with dusty, cur:
D:OK:130:1

D:130:2:12479:ving arms. Those with irregular or unusual shapes are known as irregular galaxies and typically originate from disruption by the g:

Conexion perdida

- - - MODO COMANDO - - -
Verificando Cobertura...

Fig. 5.9. Mensajes recibidos desde el módem USB especializado durante la prueba de transmisión con corte de comunicación

RING

C:galaxias.txt:6:

D:130:0:12412:Galaxies have been historically categorized according to their apparent shape, usually referred to as their visual morphology. A c:

D:130:1:12168:ommon form is the elliptical galaxy,[6] which has an ellipse-shaped light profile. Spiral galaxies are disk-shaped with dusty, cur:

NO CARRIER

Fig. 5.10. Mensajes recibidos desde el módem convencional durante la prueba de transmisión con corte de comunicación

En la figura 5.10 se puede apreciar el listado de mensajes de la vista de la GUI de la aplicación del servidor que se muestra en la figura 5.11 en el momento que se produce el fallo en la conexión. Al comparar los mensajes que muestran ambas aplicaciones se verifica que se ha perdido un paquete.

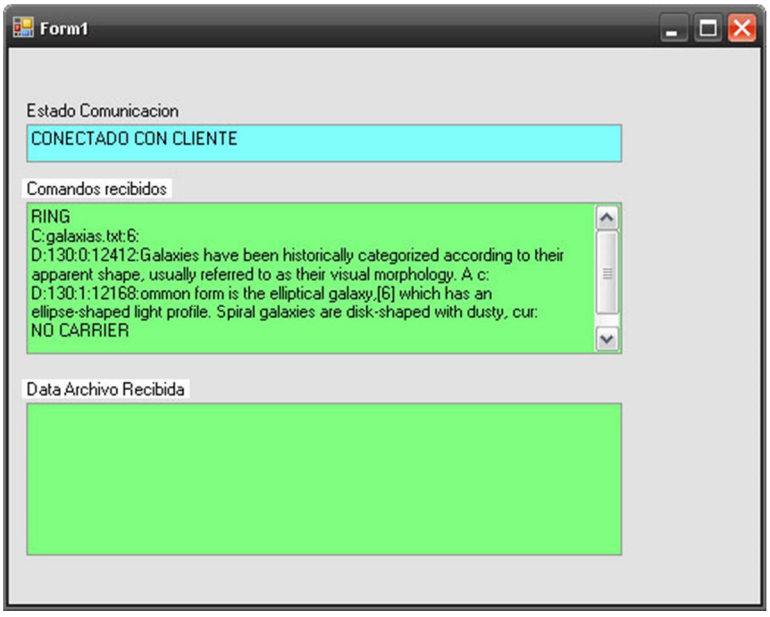


Fig. 5.11. Vista de la interfaz gráfica del programa lado servidor durante la prueba de transmisión con corte de cobertura.

C:galaxias.txt:6:
E:260:

D:130:2:12479:ving arms. Those with irregular or unusual shapes are known as irregular galaxies and typically originate from disruption by the g:
D:OK:130:2

D:130:3:12491:ravitational pull of neighboring galaxies. Such interactions between nearby galaxies, which may ultimately result in a merger, som:
D:OK:130:3

D:130:4:12587:etimes induce significantly increased incidents of star formation leading to starburst galaxies. Smaller galaxies lacking a cohere:
D:OK:130:4

D:51:5:4922:nt structure are referred to as irregular galaxies.:
D:OK:51:5

COLGAR

Fig. 5.12 Mensajes recibidos desde el módem USB especializado durante la reanudación de la transmisión.

El contenido del archivo recibido en el lado servidor es el siguiente:

"*Galaxies have been historically categorized according to their apparent shape, usually referred to as their visual morphology. A common form is the elliptical galaxy,[6] which has an ellipse-shaped light profile. Spiral galaxies are disk-shaped with dusty, cur*"

Luego se vuelve a discar el número de teléfono donde se encuentra el servidor y se reanuda la transmisión. La vista de los mensajes en la GUI de la aplicación lado cliente se muestra en la figura 5.12. La GUI para este caso no se muestra por motivo de espacio.

Se puede apreciar que el servidor ante el comando de creación de archivo responde indicando que el archivo existe y que su último byte recibido fue el byte número 260.

Las líneas siguientes muestran las transmisiones de paquetes desde el que se transmitió mal anteriormente hasta el último. En la figura 5.13 se puede apreciar una vista de la interfaz gráfica de la aplicación lado servidor durante la retransmisión.

Finalmente el contenido del archivo una vez completada la reanudación de la transmisión se muestra a continuación.

"*Galaxies have been historically categorized according to their apparent shape, usually referred to as their visual morphology. A common form is the elliptical galaxy,[6] which has an ellipse-shaped light profile. Spiral galaxies are disk-shaped with dusty, curving arms. Those with irregular or unusual shapes are known as irregular galaxies and typically originate from disruption by the gravitational pull of neighboring galaxies. Such interactions between nearby galaxies, which may ultimately result in a merger, sometimes induce significantly increased incidents of star formation leading to starburst galaxies. Smaller galaxies lacking a coherent structure are referred to as irregular galaxies.*"

Lo que corresponde perfectamente al archivo original.

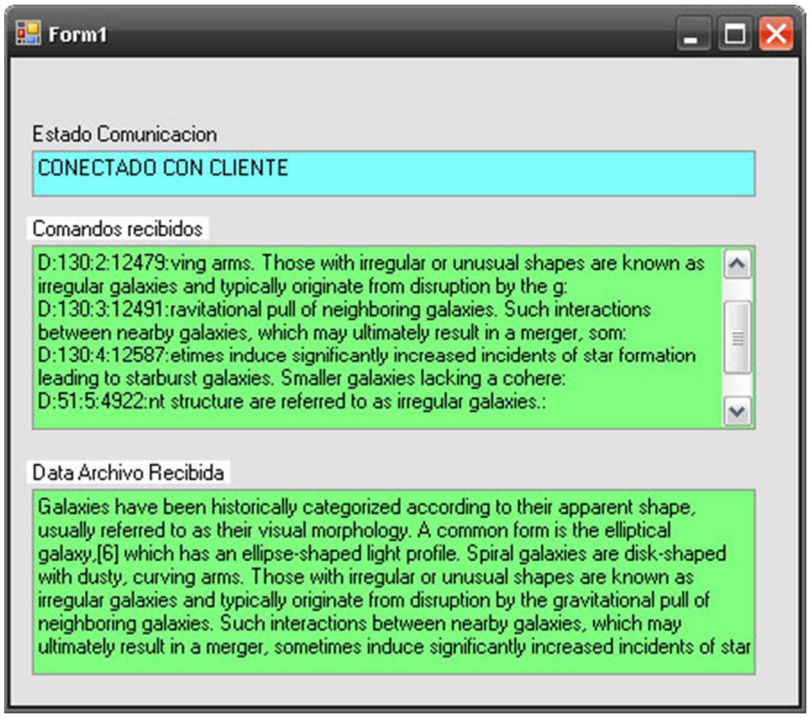


Fig. 5.13. Vista de la interfaz gráfica del programa lado servidor durante la retransmisión

CAPÍTULO 6

6. Conclusiones y trabajo futuro

En módem USB especializado cumple con todos los requerimientos del proyecto elecciones y por lo demostrado en el capítulo 5 funciona correctamente. Los programas lado cliente y servidor implementados son solamente un medio para poder probar la funcionalidad del hardware diseñado ya que éstos no serán utilizados en la aplicación final del proyecto elecciones. Ambas aplicaciones tanto lado cliente como lado servidor para la solución final serán diseñadas por otros ingenieros.

Respecto al circuito impreso o PCB del módem USB especializado, el realizado durante este trabajo de memoria es solamente un prototipo y debe ser depurado para adaptarse a la solución final. La PCB final a desarrollar debe ser de tamaño y forma apropiadas para ser instaladas en un gabinete estándar de computador personal. Muchos de los conectores presentes en la PCB deberán ser eliminados ya eran utilizados solamente para fines de diseño y no serán requeridos en la solución definitiva.

El hecho de que el módem USB especializado funciones con comandos propietarios lo hace más seguro en comparación a las soluciones convencionales de otros módems encontrados en el mercado que utilizan comandos AT [37] universales.

Ciertos tipos de comunicaciones en los cuales van a ser transmitidos datos muy sensibles, como en el caso del proyecto elecciones, es necesario que el enlace sea seguro. Esta es una de las razones principales por las cuales se prefiere utilizar la línea telefónica convencional que es una red de conmutación de circuitos en vez de utilizar el internet que es una red pública de conmutación de paquetes. El Internet por el hecho de ser pública hace que sea más probable recibir ataque de “hackers” lo cual para un proyecto de esta envergadura es un problema muy grave. El “string” correspondiente a la suma truncada fue calculado pero no fue utilizado en lado servidor en este trabajo de memoria. Para trabajos futuros debe considerarse para verificar y/o corregir posibles errores en los datos debido a problemas en la comunicación.

Esta solución de hardware podría ser utilizada en diversas aplicaciones donde se requieran niveles de seguridad importantes como es el caso de transmisiones militares o bancarias. Para estos casos los diseñadores de las aplicaciones cliente y servidor deben incorporar alguna capa software adicional para codificar la información y con ello lograr hacer más difícil a un “hacker” la obtención de información clasificada.

ANEXO 1

A1. Aspectos generales del diseño de la PCB

Para realizar el diseño de una placa de circuito impreso o PCB es necesario utilizar una aplicación CAD/EDA que permita realizar tanto el plano esquemático como el layout. Para este trabajo de memoria se utilizó el software Altium Designer® [37] por ser un paquete de software muy completo y a un costo asequible.

El proceso de diseño comienza con las ideas expuestas en el capítulo 2, donde se plantean los requerimientos y se deciden las componentes hardware que serán parte de la PCB. Luego se procede a la etapa de creación de símbolos esquemáticos y "footprints" para cada componente seleccionada, los cuales son agregados a una librería local. Un "footprint" es una representación física del empaquetado del dispositivo que presenta regiones de cobre donde deberían ir soldados los "pines" de éste. Además puede contener serigrafía para ayudar a la orientación y alineación del dispositivo.

Durante el diseño es importante tener algunas consideraciones en cuanto a las frecuencias de operación de algunas componentes como es el caso de microcontrolador MSP430, el módulo celular SIM900 y el módulo USB FT232R. Estas consideraciones son generalmente para evitar radiación y recepción de ruido electromagnético hacia fuera y dentro del mismo circuito. Dentro de estas consideraciones está la de siempre proveer un camino de retorno para las señales cercano al camino de ida, con la finalidad de disminuir la posibilidad de formación de lazos que funcionen como antenas accidentales. Esta es la razón por la cual se utilizan planos de tierra aparte de proveer blindaje en algunos casos.

La selección de componentes pasivas que se agregan a los circuitos integrados es muy importante, en especial los condensadores de bypass. Los condensadores de bypass se colocan muy cercanos a los circuitos integrados con la finalidad de que las corrientes de alta frecuencia sean suministradas por ellos y no por el regulador de voltaje. Esto se debe a que los reguladores de voltaje no pueden regular correctamente cuando hay solicitudes de corriente de frecuencias mayores de unos pocos cientos de kilo Hertz. Es por ello que los condensadores de bypass se comportan como fuentes ideales para altas frecuencias y por ende para no dañar su desempeño, deben emplazarse muy cerca de los circuitos integrados con la finalidad de no agregar inductancias en serie.

En general los condensadores reales a las frecuencias involucradas no son condensadores puros y en general pueden ser modelados tal como se muestra en la figura A1.1.

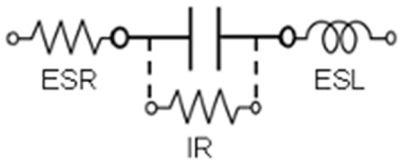


Fig. A1.1. Modelo de un condensador real teniendo en cuenta sus parásitos

La respuesta en frecuencia de un condensador es de la forma mostrada en la figura A1.2. A medida que la frecuencia es mayor, la impedancia tiende a subir después de la resonancia donde se da el mínimo. Este aumento de impedancia hace que el condensador no se comporte como una fuente ideal para altas frecuencia y por ello dañando su desempeño. Utilizar condensadores en paralelo es una técnica utilizada para poder obtener la capacidad total necesaria y que su impedancia se encuentre dentro de los límites establecidos para la banda de frecuencias de interés.

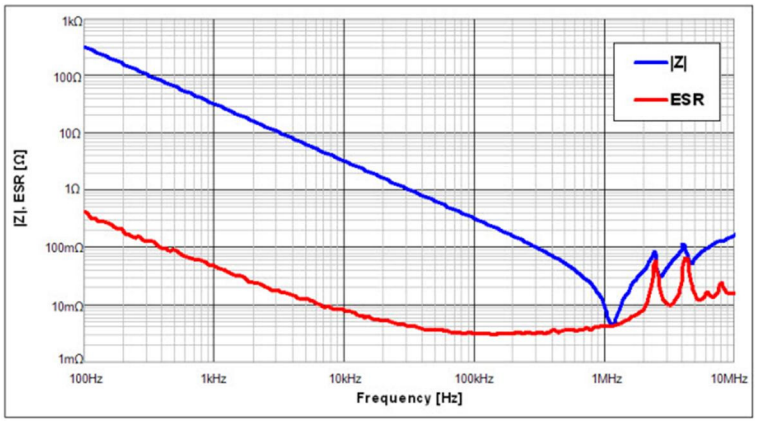


Fig. A1.2. Impedancia en función de la frecuencia para un condensador real.

Para el microcontrolador MSP430, el módulo celular SIM900 y el módulo USB FT232R que poseen relojes de varios mega Hertz, se utilizan condensadores de bypass cerámicos que tienen la virtud de poseer una frecuencia de resonancia localizada en los cientos de mega Hertz y por ende son más que apropiados para esta aplicación.

En la figura A1.3 se muestra una vista del layout terminado de la PCB del módem USB especializado y en la figura A1.4 se muestra el circuito fabricado y ensamblado conectado al computador cliente. En la figura A1.5 hay una vista completa del esquemático del módem USB especializado.

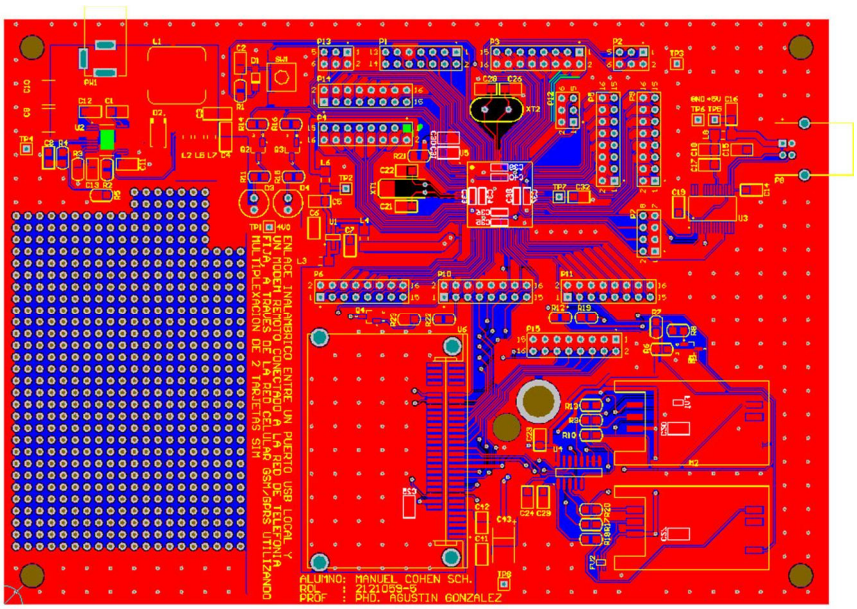


Fig. A1.3. Vista del layout del módem USB especializado

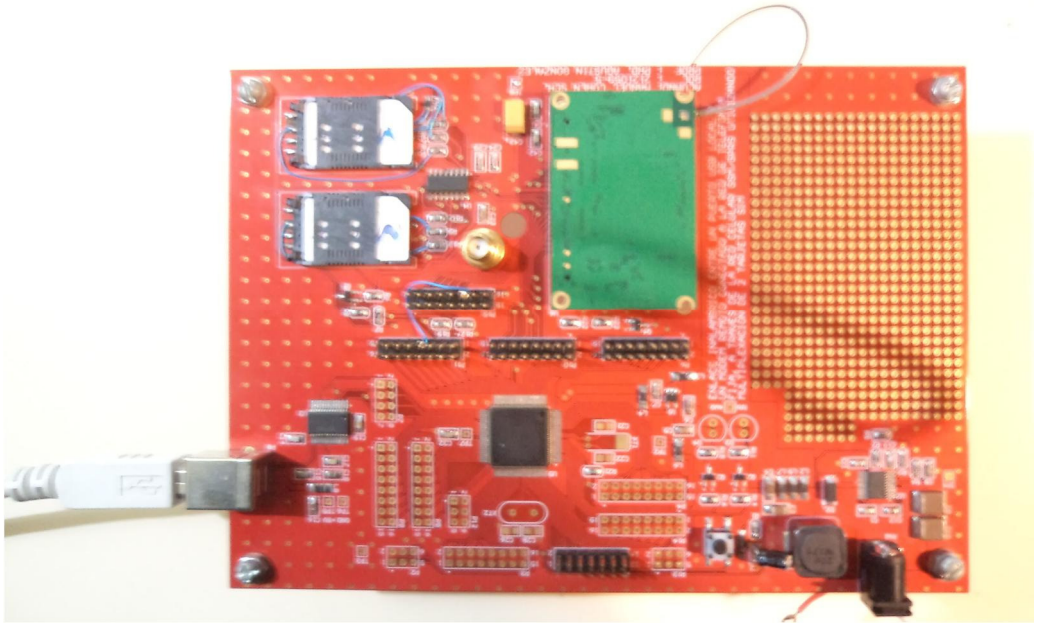


Fig. A1.4. Vista del módem USB especializado fabricado

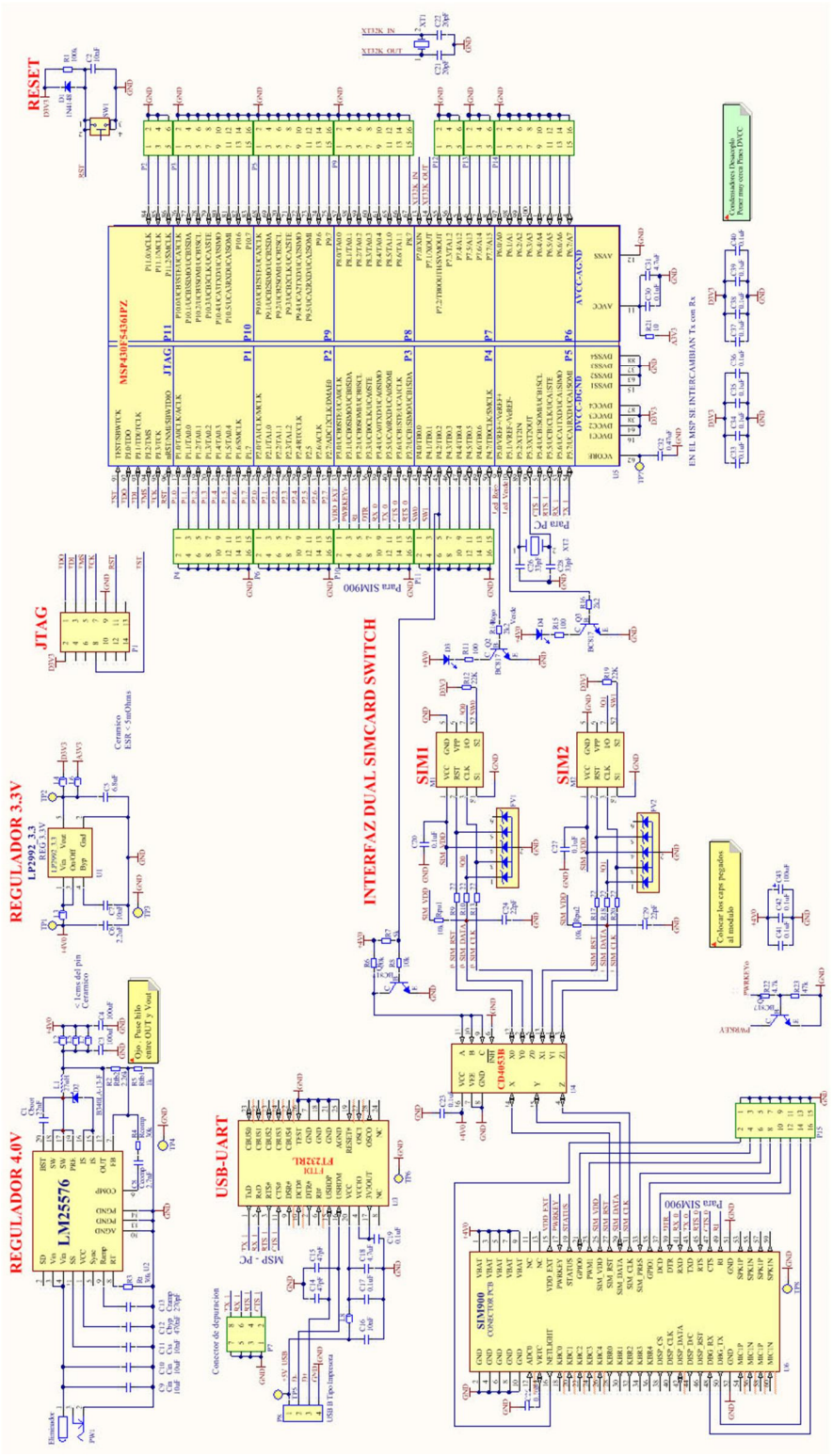


Fig. A1.5. Vista del esquemático del módem USB especializado

ANEXO 2

A2. Rutinas de bajo nivel

En este anexo se pretende mostrar la implementación de rutinas de bajo nivel del módem USB especializado las cuales se encuentran en los archivos h_MOD_UART0.c y h_HAL_SIM900TEC.c. Estas rutinas permiten la comunicación con el módulo SIM900 y el módulo FT232R a través de las interfaces USCI que se encuentran dentro de los periféricos del microcontrolador MSP430F5436 tal como se muestra en el capítulo 2. La discusión en este anexo se circunscribe solamente a las rutinas para el módulo SIM900 ya que las utilizadas para el FT232R son muy similares. Es posible decir que las rutinas del FT232 son un subconjunto de las utilizadas para el SIM900 ya que las de este último incluye más funcionalidades.

Las rutinas del archivo h_HAL_SIM900TEC.c serán explicadas en la sección A2.1 mientras que las del archivo h_MOD_UART0.c en la sección A2.2.

A2.1 Rutinas de abstracción de hardware (HAL)

Las rutinas de bajo nivel que controlan al módulo celular se encuentran en el archivo h_HAL_SIM900TEC.c. Dentro de ellas se encuentran:

1. Rutinas de inicialización
2. Rutinas de envío de comandos al módulo
3. Rutinas buscadores de mensajes
4. Rutinas de control del multiplexor de tarjetas SIM
5. Rutinas varias para el módulo SIM900

En las subsecciones siguientes se pretende exponer los distintos tipos de rutinas haciendo énfasis solamente en aquellas que son de importancia crucial para el funcionamiento del sistema. De todas maneras si el lector quiere comprender en detalle el código deberá consultar el disco CD [35] adjunto con este trabajo de memoria.

A2.1.1 Rutinas de inicialización

Dentro de estas rutinas o funciones se encuentran las que inicializan la USCI 0 en modo UART y la que enciende el módulo SIM900. Ambas se muestran en las figuras A2.1 y A2.2 respectivamente.

```
void InitUart0Sim900(void)
{
  mNoPuedoRecibirDesdeSIM900;   //Inicialmente no permitimos que nos transmitan (mientras nos inicializamos)
  P4DIR |= SimSW;               //Sal: SimSW control de SimCard
  P3DIR |= PWRKEYo;             //Sal: PWRKEYo control de encendido del SIM900
  P3DIR |= DTR;                 //Sal: Se utiliza en el control de flujo entre DTE(MSP) y CTE(Remoto)
  P3DIR |= RTS_0;               //Sal
  P3DIR &=~CTS_0;               //Ent
  P3DIR &=~RI;                  //No la usaremos por lo tanto queda en entrada HiZ

  if (P3IN&VDD_EXT)             //Si el SIM900 esta encendido, lo apagamos

    P3OUT |= PWRKEYo;           //Presionamos el PWRKEY polarizando Q4. Para ello mandamos un 1
    Retardo1Seg();              //Lo mantenemos presionado por 2 segs.
    Retardo1
    P3OUT &=~PWRKEYo;           //Soltamos el PWRKEY quedando Q4 off. Para ello mandamos un 0

  Retardo1Seg();                //Esperamos 1 seg
  InitUartUca0Sim900();         //Llamamos a rutina de inicialización de la Uart0 Uca0
}
```

Fig. A2.1 Rutina de inicialización de la USCI 0 en modo UART

La rutina de inicialización de la UART de la figura A2.1 se ejecuta solamente al comienzo invocando a una macro que utiliza el control de flujo para impedir la transmisión de datos desde el módulo hacia el microcontrolador durante la inicialización del periférico. En las líneas siguientes se definen las direcciones de las señales de los puertos de comunicaciones mediante la escritura en los registros PxDIR de 8 bits. Un bit en 1 indica salida mientras que un 0 indica entrada (condición por defecto). Las constantes SimSW y PWRKEYo corresponden a la localización de las señales indicadas con el mismo nombre en el plano esquemático adjunto en el Anexo 1. La primera es para controlar el multiplexor de tarjetas SIM y la segunda para controlar el encendido del módulo SIM900. La misma explicación aplica para las restantes 4 líneas donde se definen las direcciones de las señales involucradas en el control de flujo tanto a nivel local entre computador cliente y módem USB especializado como entre ambos computadores remotos.

Los valores de las constantes se encuentran en el archivo se encuentran definidos en el archivo h_MapeoPinesMSP430.h y se muestran en la tabla A2.1.

Tabla A2.1 Constantes utilizadas para la rutina ***InitUart0Sim900()***

Constante	Valor
SimSw	0x04
PWRKEYo	0x02
DTR	0x08
RTS_0	0x80
CTS_0	0x40
RI	0x04
VDD_EXT	0x01

La señal RI que es una abreviación para "RING INDICATOR" no se utiliza, ya que la condición de ring también es enviada desde el SIM900 al MSP430 como un "string" por medio de la UART.

La señal VDD_EXT indica que el regulador de voltaje interno del módulo SIM900 se encuentra operativo y es indicación de que el módulo se encuentra energizado.

La detección de si el módulo se encuentra encendido es muy importante. Esto se debe a que cuando la tarjeta SIM actual se conmuta a la otra es necesario volver a encender el módulo para que ésta sea detectada. En el caso que se encuentre encendido se apaga mediante la señal PWRKEYo tal como se indica en la figura 2.4 de la sección 2.2.

Una vez realizado todo lo anterior se invoca a la rutina ***InitUartUca0Sim900()*** donde se define la velocidad de la conexión entre otras características. Esta rutina se encuentra en el archivo h_MOD_UART0.c cuya explicación se realizará en el sección siguiente de este anexo.

```
void EncenderSim900(void)
{
  ResetearBufferEntradaSIM900();
  EliminarTodosLosMensajesDeListaSim900();

  P3OUT |= PWRKEYo;  //Presionamos el PWRKEY polarizando Q4. Para ello mandamos un 1
  Retardo1Seg();     //Lo mantenemos presionado por 2 segs
  Retardo1Seg();
  P3OUT &=~PWRKEYo;  //Soltamos el PWRKEY quedando Q4 off. Para ello mandamos un 0
                     //Ahora el SIM900 debería estar OFF
  Retardo1Seg();
  Retardo1Seg();
  if(BuscarMensajeEnListaSim900("NORMAL POWER DOWN"))
  {
      //Estaba encendido antes y fue apagado
      mSiPuedoRecibirDesdeSIM900;
      P3OUT |= PWRKEYo;  //Presionamos el PWRKEY polarizando Q4. Para ello mandamos un 1
      Retardo1Seg();     //Lo mantenemos presionado por 2 segs
      Retardo1Seg();
      P3OUT &=~PWRKEYo;  //Soltamos el PWRKEY quedando Q4 off. Para ello mandamos un 0
                         //Ahora el SIM900 debería estar prendido
      ResetearBufferEntradaSIM900();
      EliminarTodosLosMensajesDeListaSim900();
  }
  //Estaba apagado antes y fue encendido
    _NOP();
}
```

Fig. A2.2 Rutina de encendido del módulo SIM900

Para la rutina de encendido del módulo celular que se muestra en la figura A.2.2, en las 2 primeras líneas se invocan a las rutinas ***ResetearBufferEntradaSIM900()*** y ***EliminarTodosLosMensajesDeListaSim900()***, ambas funciones pertenecientes al archivo h_MOD_UART0.c, las cuales tienen por misión limpiar el buffer de entrada y eliminar todos los mensajes detectados en el buffer de la lista de mensaje. A continuación se utiliza la secuencia de encendido mostrada en la sección 2.2. En caso que llega el mensaje "NORMAL POWER DOWN" desde el módulo implica que éste fue apagado

por la secuencia de encendido y por ende la secuencia debe ser repetida para tenerlo nuevamente encendido. El volver a encenderlo es importante para cuando se conmuta la tarjeta SIM como se explicó al comienzo de esta subsección. La señal VDD_EXT solo se utiliza al comienzo de la ejecución del programa ya que para encendidos y apagados recurrentes presenta inestabilidades lo que no permitía determinar correctamente el estado del módulo.

A2.1.2 Rutinas de envío de comandos al módulo

El módulo SIM900 requiere que los comandos sean enviados con la estructura "**<COMANDO><CR>**" donde "**<COMANDO>**" corresponde al comando AT utilizado y **<CR>** un "byte" que representa un retorno de carro cuyo valor es "0x0D". La rutina utilizada para esta tarea y que efectúa los envíos con esta estructura es ***EnviarStringSIM900()*** se muestra en la figura A2.3. La variable argumento ***string** es un puntero a "string" cuya dirección de memoria apunta al inicio de la cadena representada por "**<COMANDO>**". La expresión **UCA0IFG&UCTXIFG** cuando vale "0" indica que el transmisor de la UART está preparado para enviar el próximo byte. Cuando vale "1" se espera a que este último esté listo. La macro **mSim900NoPuedeRecibir** vale "1" cuando el módulo celular no puede aceptar más bytes debido a que su buffer está lleno, en caso contrario vale "0".

```
void EnviarStringSIM900(char *string)
{
  char *pt_byte=string;

  if(!flagBufferEntradaLleno)
  {
    while (*pt_byte!=0x00)              //Mientras no sea el fin del string
    {
      while (mSim900NoPuedeRecibir);    //Mientras el SIM900 no pueda atendernos, esperamos
      while (!(UCA0IFG&UCTXIFG));       //Mientras el buffer de salida no este vacío, esperamos
      UCA0TXBUF=*pt_byte;               //Enviamos el byte
      pt_byte++;                        //Apuntamos al próximo byte
    }
    while (!(UCA0IFG&UCTXIFG));         //Mientras el buffer de salida no este vacío, esperamos
    UCA0TXBUF=0x0D;                     //Enviamos el byte de cierre de string
  }
  else
    _NOP();
  _NOP();
}
```

Fig. A2.3 Rutina de envío de comandos hacia el módulo SIM900

La rutina ***EnviarByteHaciaSIM900()*** mostrada en la figura A.2.4 es utilizada para enviar bytes asilados al módulo celular.

```
void EnviarByteHaciaSIM900(char byte)
{
  while (mSim900NoPuedeRecibir);       //Mientras el SIM900 no pueda atendernos, esperamos
  while (!(UCA0IFG&UCTXIFG));          //Mientras el buffer de salida no este vacío, esperamos
  UCA0TXBUF=byte;                      //Enviamos el byte
}
```

Fig. A2.4 Rutina de envío de byte simple hacia el módulo SIM900

A2.1.3 Rutinas buscadoras de mensajes

Los mensajes que van llegando desde el módulo celular van siendo almacenados en una lista por rutinas que serán explicadas en la siguiente sección. El nombre de esta lista es **listaMensajesSim900**, la cual es un arreglo bidimensional cuyo tamaño es definido por las constantes K_N_MENSAJES y K_N_BYTES_POR_MENSAJE. La primera constante define el número de mensajes que puede contener la lista mientras que la segunda define el número de bytes que cada mensaje puede contener. Además existe un arreglo unidimensional llamado **flagMensajesSim900Leidos** de tamaño definido por K_N_MENSAJES, que tiene por objetivo contener marcas para cada mensaje de tal forma que pueda saberse si un mensaje ha sido consumido o leído. Si contiene para un mensaje en particular un "1" indica que ya ha sido consumido, en caso contrario un "0". Los mensajes que han sido consumidos serán eliminados de la lista **listaMensajesSim900** por un recolector de basura para dejar espacio para los mensajes futuros. El recolector de basura se explicará en la siguiente sección.

La estructura de la lista **listaMensajesSim900** y del arreglo unidimensional **flagMensajesSim900Leidos** se muestran la figura A2.5.

	(A)	(B)
Índice Mensaje	Mensaje	Mensaje
0	MENSAJE _1	0 o 1
1	MENSAJE _2	0 o 1
	.	.
	.	.
	.	.
K_N_MENSAJES-1	MENSAJE _(K_N_MENSAJES-1)	0 o 1

Fig. A2.5 Vista de **listaMensajesSim900** (A) **y flagMensajesSim900Leidos** (B)

La rutina encargada de buscar un mensaje determinado en la lista se llama ***BuscarMensajeEnListaSim900()*** y su implementación se encuentra en la figura A2.6.

La variable de entrada ***inputString** es un puntero a "string" que contiene el contenido del mensaje que se pretende buscar en la lista.

Se utiliza el "flag" **flagConsultadoListaDeMensajesSim900=1** para indicarle a otros procesos que puedan utilizar la lista, que ésta está siendo utilizada y no se pueden efectuar cambios en ella. Al final de la rutina se le asigna el valor "0" para indicar que la lista ha sido liberada.

Cada respuesta que llega del SIM consta de 2 “strings” donde el primero contiene la información requerida y el segundo contiene siempre el “string” “**OK**”. Cada vez que se consume un mensaje, se marcan ambos para ser eliminados.

```
char BuscarMensajeEnListaSim900(char *inputString)
{
  int charIndex=0;
  flagConsultadoListaDeMensajesSim900=1; //Levantamos flag para indicar que estamos ocupando la lista
  //Barremos los mensajes recibidos para ver si esta el que buscamos
  for (charIndex=0;charIndex<(K_N_MENSAJES-1);charIndex++)
  {
    _NOP();
    if ((strcmp(inputString,listaMensajesSim900[charIndex])==0)&&(flagMensajesSim900Leidos[charIndex]==0))
    {
      //Encontre el mensaje :)
      flagMensajesSim900Leidos[charIndex]=1; //Ponemos la marca de que este es un mensaje leído o consumido
                                             //y que puede ser eliminado por el recolector de mensajes
      //if utilizado para eliminar el OK que entrega el módulo SIM900 ademas de la respuesta al comando
      if ((strcmp("OK",listaMensajesSim900[charIndex+1])==0)&&(flagMensajesSim900Leidos[charIndex+1]==0))
        flagMensajesSim900Leidos[charIndex+1]=1;
      _NOP();
      flagConsultadoListaDeMensajesSim900=0; //Bajamos flag para indicar que no  estamos ocupando la lista
      return 1;
    }
  }
  flagConsultadoListaDeMensajesSim900=0; //Bajamos flag para indicar queno  estamos ocupando la lista
  return 0;
}
```

Fig. A2.6 Rutina de búsqueda de mensajes en la lista del módulo SIM900

Existe otra rutina llamada ***BuscarParteMensajeEnListaSim900()*** que es muy similar a la ***BuscarMensajeEnListaSim900()*** que entrega la funcionalidad adicional de separar el mensaje recibido en “substrings” que contengan datos de interés. Un ejemplo para el caso del módulo FT232R sería cuando llega el comando “**MARCAR:<NUM TEL>**” desde el computador cliente en el cual se requiere separar del mensaje el “substring” “**<NUM TEL>**” para realizar el discado del número. Por ser muy similar la estructura de esta función a la ya presentada, su explicación no será realizada. Detalles pueden ser consultados en el CD incluido con este trabajo de memoria [35].

A2.1.4 Rutinas de control del multiplexor de tarjetas SIM y presencia

Dentro de estas rutinas se encuentra ***CambiarSim()*** que tiene la funcionalidad de poder conmutar la tarjeta SIM y ***BuscarPresenciaSim()*** cuyo objetivo es buscar cuales de ellas se encuentran físicamente instaladas en los “sockets”.

Las rutinas ***CambiarSim()*** y ***BuscarPresenciaSim()*** se muestran en las figura A2.7.

La función ***CambiarSim()*** conmuta el transistor Q1 de acuerdo al valor de la variable de entrada **Sim**. La conmutación de este transistor opera el multiplexor CD4053 para conmutar de tarjeta SIM.

```
void CambiarSim(char Sim)
{
  if (Sim==M1SIM_Arriba)
    P4OUT |= SimSW;      //0 -> funciona M1, caso contrario M2
  else
  if (Sim==M2SIM_Abajo)
    P4OUT &=~SimSW;      //1 -> funciona M2
}


char BuscarPresenciaSim(void)
{
  //Vale 1 si hay SIM instalada en el socket 1
  char presenciaSim1=(~(P4IN&SW0)&SW0)>>0;
  //Vale 1 si hay SIM instalada en el socket 2
  char presenciaSIm2=(~(P4IN&SW1)&SW1)>>1;
  if (presenciaSIm2)
    if (presenciaSim1) return 3;  //SIM M1 y SIM M2
    else               return 2;  //Solo SIM M2
  else
    if (presenciaSim1) return 1;  //Solo SIM M1
    else               return 0;  //Ninguna instalada
}
```

Fig. A2.7 Rutinas de conmutación y presencia de tarjetas SIM

La función ***BuscarPresenciaSim()*** consulta el estado de los "switches" que se encuentran en cada "socket" al leer los bits señalados por las señales **SW0** y **SW1**. La funcionalidad de estos "switches" se explica en la sección 2.3.

A2.1.5 Rutinas de varias para el módulo SIM900

Las 4 primeras rutinas detalladas en la figura A2.8 son implementadas realizando llamados a la rutinas ***CambiarSim(), EncenderSim900()*** y ***EnviarStringSIM900()***.

```
void EncenderModuloSim900SeleccionandoSimM1Arriba(void)
{
  CambiarSim(M1SIM_Arriba);
  EncenderSim900();
}

void EncenderModuloSim900SeleccionandoSimM2Abajo(void)
{
  CambiarSim(M2SIM_Abajo);
  EncenderSim900();
}

void MarcarNumeroSIM900(char *numero)
{
  char comandoConNumero[20]="ATD";
  strcat(comandoConNumero, numero);
  EnviarStringSIM900(comandoConNumero);
}

void ColgarSim900(void)
{
  EnviarStringSIM900("ATH0");
}
```

Fig. A2.8 Rutinas ***EncenderModuloSim900SeleccionandoSimM1Arriba()***, ***EncenderModuloSim900SeleccionandoSimM1Arriba()***, ***MarcarNumeroSIM900()*** y ***ColgarSim900()***

La finalidad de ellas es ponerle nombres más apropiados a los comandos AT [38] requeridos para efectuar las distintas operaciones requeridas por el módem USB especializado.

Las funciones no requieren ser explicadas debido a que los nombres representan sus objetivos.

Las 4 siguientes funciones detalladas en la figura A2.9 son implementadas realizando llamados a la rutinas ***EnviarByteHaciaSIM900()*** **y** ***BuscarMensajeEnListaSim900().***

```
void ColgarSim900ModoDataVolverModoComando(void)
{
  Retardo1Seg();
  EnviarByteHaciaSIM900('+');
  EnviarByteHaciaSIM900('+');
  EnviarByteHaciaSIM900('+');
  Retardo1Seg();
  Retardo1Seg();

  _NOP();
  while(!BuscarMensajeEnListaSim900("OK"))
  _NOP();
  ColgarSim900();
  RetardoNSeg(2);
  while(!BuscarMensajeEnListaSim900("OK"))
  _NOP();
  _NOP();
}

char EntrarEnModoData(void)
{
  char resultado;
  resultado = PrepararUartParaModoDataSim900();
  _NOP();
  return resultado;
}

char EntrarEnModoComando(void)
{
  char resultado;
  resultado = PrepararUartParaModoComandoSim900();
  _NOP();
  return resultado;
}

char VerificarConexionModoDataSim900(void)
{
  char resultado;
  resultado = BuscarMensajeEnListaSim900("CONNECT 9600");
  _NOP();
  return resultado;
}
```

Fig. A2.9 Rutinas ***ColgarSim900ModoDataVolverModoComando ()***, ***EntrarEnModoData(), EntrarEnModoComando(), VerificarConexionModoDataSim900()***

La primera función ***ColgarSim900ModoDataVolverModoComando()*** es utilizada para colgar la comunicación mientras se está en modo datos. Para realizar esto en todo módem es necesario escribir en modo datos la secuencia de bytes "+++" con lo cual

se vuelve a modo comando. Se espera hasta que el módulo responda "**OK**" para indicar que fue reconocido el comando. Una vez en modo comando se envía el comando para colgar utilizando la función ***ColgarSim900()***.

Las funciones ***EntrarEnModoData()*** y ***EntrarEnModoComando()*** son utilizadas para entrar en modo datos y modo comando respectivamente. Ambas funciones realizan llamados a otras funciones que se encuentran en el archivo h_MOD_UART0.c, cuya misión es borrar los búferes de entrada y la lista de mensajes con el cambio de modo.

Para poder comprobar si hay conexión de datos con el computador servidor, se llama a la rutina ***VerificarConexionModoDataSim900()***. Lo único que realiza esta función es verificar mediante un llamado a ***BuscarMensajeEnListaSim900()*** si llegó el mensaje "CONNECT 9600" el cual es indicador de existencia de conexión.

```
void EnviaComandoDeVerificacionSesionEnRedCelular(void)
{
  EnviarStringSIM900("AT+CPAS");
}

char VerificarSiHaySesionEnRedCelular(void)
{
  char respuesta=2;
  if  (BuscarMensajeEnListaSim900("+CPAS: 0")==1)
  {
    respuesta=1;
  }
  if (BuscarMensajeEnListaSim900("+CPAS: 2")==1)
  {
    respuesta=0;
  }
  return respuesta;
}

char VerificarCallReady(void)
{
  if  (BuscarMensajeEnListaSim900("Call Ready"))
    return 1;
  else
    return 0;
}

void EnviaDataSim900(char *data)
{
  mSiPuedoRecibirDesdeDceRemoto;
  EnviarStringSIM900(data);
}

char VerificarModoDataSim900(void)
{
  if (EncuestaDcdSim900()) //Estamos en modo Data
    return 1;
  else
    return 0;
}
```

Fig. A2.10 Rutinas ***EnviaComandoDeVerificacionSesionEnRedCelular()***, ***VerificarSiHaySesionEnRedCelular()***, ***VerificarCallReady()***, ***EnviaDataSim900()*** y ***VerificarModoDataSim900()***

Las últimas cinco rutinas se muestran en la figura A2.10. La primera función invoca a la función ***EnviarStringSIM900()*** para enviar el comando "AT+CPAS" para encuestar al módulo celular el estado de la cobertura con la red celular. En caso que la respuesta es "1" hay cobertura, en caso contrario no hay.

La siguiente función ***VerificarSiHaySesionEnRedCelular()***, es para verificar la respuesta entregada por ***EnviaComandoDeVerificacionSesionEnRedCelular()***. Es llamada por las rutinas de medio nivel después de unos segundos de haber enviado el primer comando ya que su respuesta no es instantánea. Retorna "1" en caso de haber cobertura y "0" en caso contrario.

La función ***VerificarCallReady()*** es utilizada para verificar si hay cobertura. Para determinar esto se invoca a la función ***BuscarMensajeEnListaSim900()*** con la que se verifica si el mensaje "Call Ready" se encentra en la lista.

La función ***EnviaDataSim900()*** está implementada por la función ***EnviarStringSIM900()*** más una macro que señaliza el control de flujo de tal forma que el módem USB especializado pueda recibir datos desde el servidor o computador remoto.

La última función la cual es ***VerificarModoDataSim900()*** es utilizada como alternativa para comprobar en cualquier momento que exista cobertura si tener que depender solamente del mensaje "**CONNECT 9600**". Para ello se encuesta señal digital DCD que en inglés significa "Data carrier detect" mediante la rutina que se muestra en la figura A2.11. Si su lectura es "0" no hay conexión, en caso contrario hay conexión de datos.

```
char EncuestaDcdSim900(void)
{
  if ((P4IN&DCD_0)==0x00) //Estamos en modo Data
    return 1;
  else
    return 0;
}
```

Fig. A2.11 Rutina de encuesta de la señal DCD

A2.2 Rutinas de manejo de UART

Esta sección corresponde a los programas que se encuentran en el archivo h_MOD_UART0.c que tienen por finalidad el manejo de UART donde se reciben los datos del módulo SIM900. Dentro de las funciones principales que se encuentran en este archivo están las siguientes:

1. Rutina de configuración de parámetros de datos de la UART 0
2. Rutina de interrupción de recepción
3. Rutina de extracción de mensajes
4. Rutinas de limpieza de listas y búferes

A2.2.1 Rutina de configuración de parámetros de datos de la UART 0

La rutina que configura los parámetros de la UART se llama ***InitUartUca0Sim900()*** y es invocada tal como se explicó en la subsección A2.1.1. En ella se configuran los parámetros como la velocidad de 115.200 bps para los datos entre el módulo SIM900 y el microcontrolador MSP430, el reloj asignado al periférico y el largo de los bytes de datos (pueden ser 7 o 8 bits). La rutina se puede apreciar en la figura A2.12. Además se habilita la interrupción de recepción en la última línea.

```
void InitUartUca0Sim900(void)
{
  P3SEL |= RX_0 | TX_0 ;              //P3.4, P3.5 modo UART
  UCA0CTL1 |= UCSWRST;                //Se resetea maquina de estados USCI y se mantiene
reseteada
  UCA0CTL1 |= UCSSEL_2;               //Clock de la USCI = SMCLK
  UCA0BR0   = 0x9c;                   //LSByte BR
  UCA0BR1   = 0x00;                   //MSByte BR
  UCA0MCTL |= UCBRS_4 + UCBRF_4;      //Registro de modulacion
  UCA0CTL1 &=~UCSWRST;                //Se resetea maquina de estados USCI y se mantiene
reseteada
  while ((UCA0STAT&UCBUSY));          //Mientras este en operacion la USCI, esperamos
  UCA0IE   |=UCRXIE;                  //Enable USCI_A1 RX interrupt
}
```

Fig. A2.12 Rutina de definición de parámetros de la UART 0

A2.2.2 Rutina de interrupción de recepción

Los bytes que se reciben del módulo SIM900 se van ingresando en un buffer de entrada llamado **bufferEntradaBytesDesdeSIM900**. Esto se hace automáticamente por la interrupción de recepción que se muestra en la figura A2.13.

Al entrar a esta rutina se deshabilita la interrupción y al salir se vuelve a habilitar para evitar por alguna falla que se aniden interrupciones.

Los bytes en el buffer van siendo almacenados mediante el incremento del cursor **cursorEscrituraBufferEntradaSIM900**. Cuando el cursor alcanza un valor mayor o igual a la constante definida como UMBRAL_SUPERIOR_POSICION_BYTE se pone en "1" el flag que indica que el buffer de entrada está próximo a llenarse. El nombre de este flag es **flagBufferEntradaLleno** y es utilizado para señalizar a la función ***EnviarStringSIM900()*** de la subsección A2.1.2 que no envíe otro comando cuando ésta última sea utilizada.

```
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
  switch(__even_in_range(UCA0IV,4))
  {
  case 0:break;
  case 2:
    UCA0IE &=~UCRXIE;
    bufferEntradaBytesDesdeSIM900[cursorEscrituraBufferEntradaSIM900] = UCA0RXBUF;
    if (cursorEscrituraBufferEntradaSIM900>=UMBRAL_SUPERIOR_POSICION_BYTE)
    {
      flagBufferEntradaLleno=1;
    }
    else
      {mSiPuedoRecibirDesdeSIM900;}
    cursorEscrituraBufferEntradaSIM900++;
    UCA0IE |= UCRXIE;
    break;
  case 4:break;
  default: break;
  }
}
```

Fig. A2.13 Rutina de interrupción de recepción de la UART 0

A2.2.3 Rutina de extracción de mensajes

La rutina llamada ***ExtraeMensajesDesdeArregloDeEntradaSim900()***, que tiene por misión extraer los mensajes del buffer denominado por **bufferEntradaBytesDesdeSIM900**. Los mensajes que llegan desde el módulo celular tienen la estructura "**<0x0D><0x0A><MENSAJE><0x0D><0x0A>**", en otras palabras el mensaje viene dentro de un "sándwich" de retornos de carro y avance de línea. Los mensajes van siendo almacenados en la lista **listaMensajesSim900.**

La rutina se encuentra implementada como la estructurada como máquina de estados por lo que se presentará primero la estructura general y luego se irán mostrando en forma paulatina sus estados. La estructura general se muestra en la figura A2.14. La máquina de estados se ejecuta siempre que **cursorLecturaBufferEntradaSIM900** sea menor a **cursorEscrituraBufferEntradaSIM900** para evitar que se traten de extraer mensajes de bytes que todavía no han sido escritos en el buffer de entrada.

La máquina de estados está conformada por 4 estados, donde los primeros 3 se muestran en la figura A2.15 y el último en la figura A2.16.

El primero llamado S_ESPERANDO_1ER_0D tiene por objetivo esperar el primer byte "**<0x0D>**" de la secuencia. El segundo llamado S_ESPERANDO_1ER_0A tiene por objetivo esperar el primer byte "**<0x0A>**" de la secuencia. En el tercer estado llamado S_COLECCION_BYTES se recolectan los bytes del "string" "**<MENSAJE>**" y se espera la condición de término de recolección, la cual se da cuando el byte analizado es

```
void ExtraeMensajesDesdeArregloDeEntradaSim900(void)
{
  char ByteLeido;
  static int ultimocursorEscrituraBufferEntradaSIM900;
  while((cursorLecturaBufferEntradaSIM900<cursorEscrituraBufferEntradaSIM900))
  {
    ByteLeido = bufferEntradaBytesDesdeSIM900[cursorLecturaBufferEntradaSIM900];
    switch (estadoActualFsmBuscaMensajes)
    {
      case S_ESPERANDO_1ER_0D:
        /* Aca va el codigo a ejecutar
        para este estado */
        break;

      case S_ESPERANDO_1ER_0A:
        /* Aca va el codigo a ejecutar
        para este estado */
        break;

      case S_COLECCION_BYTES:
        /* Aca va el codigo a ejecutar
        para este estado */
        break;

      case S_ESPERANDO_2ND_0A:
        /* Aca va el codigo a ejecutar
        para este estado */
        break;
    }
    cursorLecturaBufferEntradaSIM900++;
  }
}
```

Fig. A2.14 Estructura general de la rutina de extracción de mensajes

"**<0x0D>**". En el cuarto y último estado llamado S_ESPERANDO_2ND_0A, se espera el segundo byte "**<0x0A>**" de término de la secuencia.

```
  case S_ESPERANDO_1ER_0D:
    if      (ByteLeido!=0x0D)     estadoActualFsmBuscaMensajes=S_ESPERANDO_1ER_0D;
    else                          estadoActualFsmBuscaMensajes=S_ESPERANDO_1ER_0A;
    break;

  case S_ESPERANDO_1ER_0A:
    if      (ByteLeido==0x0D)     estadoActualFsmBuscaMensajes=S_ESPERANDO_1ER_0A;
    else if (ByteLeido==0x0A)     estadoActualFsmBuscaMensajes=S_COLECCION_BYTES;
    else                          estadoActualFsmBuscaMensajes=S_ESPERANDO_1ER_0D;
    break;

  case S_COLECCION_BYTES:
    if      (ByteLeido==0x0D)     estadoActualFsmBuscaMensajes=S_ESPERANDO_2ND_0A;
    else if (ByteLeido==0x0A)     estadoActualFsmBuscaMensajes=S_ESPERANDO_1ER_0D;
    else if((ByteLeido!=0x0D)&&(ByteLeido!=0x0A))
         {
           estadoActualFsmBuscaMensajes=S_COLECCION_BYTES;
           listaMensajesSim900[nMensajeSim900][nCaracterSim900] = ByteLeido;
           nCaracterSim900++;
         }
```

Fig. A2.15 Los primeros 3 estados de la rutina de extracción de mensajes

```
case S_ESPERANDO_2ND_0A:
    if      (ByteLeido==0x0A)
    {
       if (nMensajeSim900<(K_N_MENSAJES-1))
       { //Si el indice del mensaje a almacenar es menor que K_N_MENSAJES-1
         //(queda un   espacio libre para un mensaje)
         nMensajeSim900++;   //Incrementamos el cursor de mensaje para el próximo
         nCaracterSim900=0;
       }
       else                 //En caso contrario, si nos quedamos sin espacios libres
        flagListaMensajesSim900Llena=1;     //levantamos flag de lista llena
    }
    estadoActualFsmBuscaMensajes=S_ESPERANDO_1ER_0D;
    break;
```

Fig. A2.16 Ultimo estado de la rutina de extracción de mensajes

A2.2.4 Rutinas de limpieza de listas y búferes

En esta subsección se mostrarán las rutinas utilizadas para realizar mantenimiento a los búferes de recepción de la UART y la lista de mensajes.

Cuando el buffer de recepción está por llenarse es necesario parar el envío de datos y realizar una limpieza. Esta limpieza significa el borrado de todos los datos del arreglo y a su vez reiniciar el cursor de escritura y lectura en “cero”. Esto es realizado al llamar a la rutina ***ResetearBufferEntradaSIM900()*** que se muestra en la figura A2.17.

```
void ResetearBufferEntradaSIM900(void)
{
  for (int i=0; i<TAMANO_BUFFER_EN_BYTES_SIM900; i++)
    bufferEntradaBytesDesdeSIM900[i]=0x00;
  cursorLecturaBufferEntradaSIM900=0;
  cursorEscrituraBufferEntradaSIM900=0;
  flagBufferEntradaLleno=0;  //Señalizamos que se encuentra vació el buffer de entrada
  mSiPuedoRecibirDesdeSIM900;//Permitimos al SIM900 enviarnos data
}
```

Fig. A2.17 Código de la rutina que limpia el buffer de entrada

También es necesario realizar mantenimiento a la lista de mensajes donde se almacenan los que van llegando desde el módulo SIM900. Los que han sido marcados como consumidos o leídos ya no son necesarios y por ello deben eliminarse. Para realizar esta tarea se utiliza la rutina ***EliminarMensajesLeidosDeListaSim900()*** la cual se muestra en la figura A2.18. En ella se buscan todos los mensajes que han sido consumidos al barrer el arreglo **flagMensajesSim900Leidos** y buscar aquellos mensajes que están marcados con “1”. Al momento de encontrar uno, éste se elimina y se realiza un corrimiento de los mensajes para utilizar el espacio del recién eliminado.

```
void EliminarMensajesLeidosDeListaSim900(void)
{
  int j;
  //Se borran todos los datos del arreglo listaMensajesSim900
  for (j=0;j<(nMensajeSim900);j++)
  {
    while(flagMensajesSim900Leidos[j]==1)
    {
      if(j<nMensajeSim900)
      {
        for (int z=j; z<nMensajeSim900; z++)
        {
          flagMensajesSim900Leidos[z]=flagMensajesSim900Leidos[z+1];
          for (int k=0; k<K_N_BYTES_POR_MENSAJE; k++)
            listaMensajesSim900[z][k]=listaMensajesSim900[z+1][k];
        }
        nMensajeSim900--;
        _NOP();
      }
      else
      {
     flagMensajesSim900Leidos[j]=0;
         EscribirRenglonEnCero(j);
         nMensajeSim900--;
      }
    }
  }
}
```

Fig. A2.18 Código de la rutina de eliminación de mensajes consumidos/leídos

A2.3 Rutina de mantenimiento y recolector de basura

Existe una interrupción gatillada por uno de los "timers" del microcontrolador que tiene la misión de realizar mantenimiento en las listas de mensajes. Cuando se habla de mantenimiento se quiere expresar las labores necesarias para mantener actualizada la lista de mensajes. Las operaciones de mantenimiento requieren la búsqueda de nuevos mensajes mediante ***ExtraeMensajesDesdeArregloDeEntradaSim900()*** presentada en la subsección A2.2.3 y eliminación de los mensajes ya marcados como consumidos/leídos.

```
void ExtraeMensajesDesdeArregloDeEntradaYBorrarLeidos-
Sim900(void)
{
  if (flagConsultadoListaDeMensajesSim900==0) //Desocupado
  {
    ExtraeMensajesDesdeArregloDeEntradaSim900();
    EliminarMensajesLeidosDeListaSim900();
  }
  _NOP();
}
```

Fig. A2.19 Código de la función ***ExtraeMensajesDesdeArregloDeEntradaSim900()***

La eliminación de ellos se realiza mediante la función ***EliminarMensajesLeidosDeListaSim900()*** presentada en la subsección A2.2. La rutina de interrupción llama a la rutina ***ExtraeMensajesDesdeArregloDeEntradaYBorrarLeidosSim900()*** la cual invoca a

las funciones ***ExtraeMensajesDesdeArregloDeEntradaSim900()*** y ***EliminarMensajesLeidosDeListaSim900()*** cuyo código se muestra en la figura A2.19. La rutina de interrupción se muestra en la figura A2.20.

```
void ExtraeMensajesDesdeArregloDeEntradaYBorrarLeidos-
Sim900(void)
{
  if (flagConsultadoListaDeMensajesSim900==0) //Desocupado
  {
    ExtraeMensajesDesdeArregloDeEntradaSim900();
    EliminarMensajesLeidosDeListaSim900();
  }
  _NOP();
}
```

Fig. A2.20 Código de interrupción donde se realizan las operaciones de mantenimiento

Referencias

[1] Vote counting system. [En línea]
<http://en.wikipedia.org/wiki/Vote_counting_system> [consulta: 1 Agosto 2013]

[2] Ian Urbina. States Move to Allow Overseas and Military Voters to Cast Ballots by Internet. [En línea] The New York Times Internet, 8 de mayo, 2010
<http://www.nytimes.com/2010/05/09/us/politics/09voting.html?pagewanted=all&_r=0> [consulta: 1 Agosto 2013]

[3] DRE voting machine. [En línea]
<http://en.wikipedia.org/wiki/DRE_voting_machine> [consulta: 1 Agosto 2013]

[4] James Brendan. Everything you wanted to know about voting machines. [En línea] Yahoo News, 5 de noviembre, 2012
<http://news.yahoo.com/blogs/ticket/rough-guide-voting-machines-175213233.html> [consulta: 1 Agosto 2013]

[5] Federal Election Commission United States of America. Voting Systems Standards Volume I: Performance Standards [En línea]
<http://www.eac.gov/assets/1/Page/Voting%20System%20Standards%20Volume%20I.pdf> [consulta: 1 Agosto 2013]

[6] Electronic voting examples. [En línea]
<http://en.wikipedia.org/wiki/Electronic_voting_examples>
[consulta: 1 Agosto 2013]

[7] Subtel Chile, Abonados Móviles. [En línea]
<http://www.subtel.gob.cl/images/stories/apoyo_articulos/informacion_estadistica/series_estadisticas/1_abonados_moviles_dic12_190313_v1.xlsx>,
[hoja:" 3.1.Abonados", Celda: "D19"] [consulta: 3 Julio 2013]

[8] Subtel Chile, Series líneas telefónicas. [En línea]
<http://www.subtel.gob.cl/images/stories/apoyo_articulos/informacion_estadistica/series_estadisticas/1_series_lineas_telefonicas_dic12_190313_v1.xlsx >,
[hoja:" 1.17_Líneas por comuna", Celda: "BW367"] [consulta: 3 Julio 2013]

[9] "USB Class Codes". [En línea]
<http://www.usb.org/developers/defined_class/> [consulta: 3 Julio 2013]

[10] "Nokia 5200". [En línea]
<http://nds1.nokia.com/phones/files/guides/Nokia_5200_UG_es.pdf>
[consulta: 3 Julio 2013]

[11] "Nokia 5300". [En línea]
<http://nds1.nokia.com/phones/files/guides/Nokia_5300_UG_es.pdf>
[consulta: 3 Julio 2013]

[12] "Nokia Dual SIM". [En línea]
<http://www.nokia.com/in-en/dual-sim/> [consulta: 3 Julio 2013]

[13] "Teltonika PCI/G10". [En línea]
<http://www.teltonika.lt/uploads/docs/ModemPCI%20G10%20User%20Manual%20EN.pdf>
[consulta: 3 Julio 2013]

[14] "Teltonika USB/H7.2". [En línea]
<http://www.teltonika.lt/es/pages/view/?id=842> [consulta: 3 Julio 2013]

[15] "3G MODEM USB INTERNAL". [En línea]
<http://www.digicom.it/digisit/prodotti.nsf/ENProdottiIDX/3GModemUsbInternal>
[consulta: 3 Julio 2013]

[16] "Digicom - 3G USB Internal". [En línea]
<http://www.tigal.com/product/2765 >
[consulta: 3 Julio 2013]

[17] "SIM900 Wireless Module". [En línea]
<http://www.module-igbt.com/gsm-gprs-module/sim900-wireless-module-/prod_4207.html>
[consulta: 3 Julio 2013]

[18] "SIM Socket". [En línea]
<https://www.sparkfun.com/products/548>
[consulta: 3 Julio 2013]

[19] "MSP430F5436IPZR IC MCU 16BIT 192KB FLASH 100LQFP Digikey".
[En línea]
<http://www.digikey.com/product-detail/en/MSP430F5436IPZR/296-23765-2-ND/1951896>
[consulta: 3 Julio 2013]

[20] "CD4053BPWR IC MUX/DEMUX TRIPLE 2X1 16TSSOP Digikey".
[En línea]
<http://www.digikey.com/product-detail/en/CD4053BPWR/296-14117-1-

ND/525922>
[consulta: 3 Julio 2013]

[21] "TG.22.0112 ANT 5-BAND GSM DIPOLE Digikey". [En línea]
<http://www.digikey.com/product-search/en?WT.z_header=search_go&lang=en&site=us&keywords=TG.22.0112+&x=-838&y=-51>
[consulta: 3 Julio 2013]

[22] "080-0001 CABLE U.FL TO SMA BULKHEAD Digikey". [En línea]
< http://www.digikey.com/product-detail/en/080-0001/080-0001-ND/2696495 >
[consulta: 3 Julio 2013]

[23] "COM port redirector". [En línea]
<http://en.wikipedia.org/wiki/COM_port_redirector#Virtual_serial_port >
[consulta: 3 Julio 2013]

[24] "CD4051B, CD4052B, CD4053B". [En línea]
<http://www.ti.com/lit/ds/symlink/cd4051b.pdf>
[consulta: 3 Julio 2013]

[25] "Subscriber identity module". [En línea]
<http://en.wikipedia.org/wiki/Subscriber_identity_module>
[consulta: 3 Julio 2013]

[26] "SIM900". [En línea]
<http://wm.sim.com/producten.aspx?id=1019>
[consulta: 4 Julio 2013]

[27] "Entel PCS". [En línea]
<http://www.entelpcs.cl/compania/gsm.iws>
[consulta: 4 Julio 2013]

[28] " SIM900_Hardware Design_V2.02". [En línea]
<http://wm.sim.com/upfile/20126416756f.pdf>
[consulta: 4 Julio 2013]

[29] "Application Note—Implementing Dual SIM Functionality using the SIM900". [En línea]
<http://www.otto.co.za/store/dataSheets/product/be6207b0a3442dc3a0a08bd2529300e7.pdf>
[consulta: 4 Julio 2013]

[30] " Future Technology Devices International Ltd. FT232R USB UART IC ". [En línea]
<http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf >
[consulta: 4 Julio 2013]

[31] " MSP430F5438, MSP430F5437, MSP430F5436, MSP430F5435, MSP430F5419, MSP430F5418 ". [En línea]
<http://www.ti.com/lit/ds/symlink/msp430f5436.pdf >
[consulta: 4 Julio 2013]

[32] "Converter (Integrated Switch) - Step-Down (Buck) Converter - LM25576 - TI.com". [En línea]
<http://www.ti.com/product/lm25576>
[consulta: 4 Julio 2013]

[33] "WEBENCH® Power Designer - TI.com". [En línea]
<http://www.ti.com/lsds/ti/analog/webench/power.page>
[consulta: 4 Julio 2013]

[34] "C# Examples". [En línea]
<http://www.ftdichip.com/Support/SoftwareExamples/CodeExamples/CSharp.htm >
[consulta: 4 Octubre 2013]

[35] "COHEN, Manuel Patricio. Desarrollo de módem celular USB multicompañía aplicado a la transferencia de archivos de texto [CD Rom] Valparaíso UTFSM. Departamento de Electrónica, 2013. 1 disco compacto para computador, archivos de texto de código fuente.

[36] "SerialPort Class (System.IO.Ports) - MSDN - Microsoft". [En línea]
<http://msdn.microsoft.com/en-us/library/system.io.ports.serialport.aspx>
[consulta: 9 Octubre 2013]

[37] "Altium:Next Generation Electronic Design". [En línea]
<http://www.altium.com/>
[consulta: 9 Octubre 2013]

[38] "Conjunto de comandos Hayes - Wikipedia, la enciclopedia libre". [En línea]
<http://es.wikipedia.org/wiki/Conjunto_de_comandos_Hayes>
[consulta: 9 Octubre 2013]