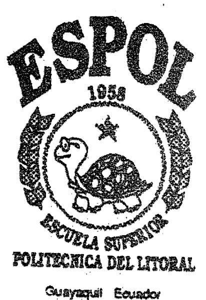
T
005.1
DOM



ESCUELA SUPERIOR POLITECNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL DE ACCESO MANEJADO POR UN SERVIDOR REMOTO A TRAVÉS DE UNA RED ETHERNET; PARA SER UTILIZADO EN LAS DIFERENTES ÁREAS E INSTALACIONES DE LA FIEC"

TESIS DE GRADO

Previa a la Obtención del Título de:

INGENIERO EN ELECTRICIDAD

Especialización: ELECTRONICA

Presentada por:
David Nicolás Domínguez Bonini

Guayaquil – Ecuador

DECLARACION EXPRESA

"La responsabilidad por los hechos, ideas y doctrinas expuestos en esta tesis corresponden exclusivamente a su autor, y, el patrimonio intelectual de la misma, a la ESCUELA SUPERIOR POLITECNICA DEL LITORAL".

David Domínguez Bonini

ING. ARMANDO ALTAMIRANO

SUBDECANO FIEC

ING. CARLOS MONSALVE

DIRECTOR DE TESIS

ING. HUGO VILLAVICENCIO

MIEMBRO DEL TRIBUNAL

ING. NELSON LAYEDRA

MIEMBRO DEL TRIBUNAL

RESUMEN

Este trabajo describe el proceso de diseño e implementación del prototipo de un controlador de acceso integrado para utilizarse en el esquema de seguridad de los laboratorios de la FIEC (facultad de Ingeniería en Electricidad y Computación).

El trabajo se compone de cinco capítulos, los cuales describen de forma sistemática los siguientes pasos del proceso de diseño del equipo mencionado:

1. Definición de requisitos de diseño y fundamentos teóricos del proyecto.

2. Diseño conceptual y análisis de la arquitectura de hardware del proyecto.

3. Construcción del hardware del proyecto.

4. Diseño conceptual y análisis de la arquitectura de software del proyecto.

5. Descripción del funcionamiento y modo de operación del equipo diseñado.

INDICE GENERAL

RESUMEN 3

INDICE GENERAL 4

INDICE DE FIGURAS 8

INDICE DE TABLAS 10

INTRODUCCION 11

CAPITULO I: GENERALIDADES Y ANTECEDENTES 12

1.1 REQUISITOS DE SEGURIDAD EN LOS LABORATORIOS DE LA ESPOL 12

1.1.1 Proyectos previos en el área *13*

1.1.2 Contribución del presente trabajo *14*

1.2 ESQUEMA DE INTERNETWORKING DE LA ESPOL 15

1.2.1 Las redes Ethernet *16*

1.2.2 La arquitectura TCP/IP *19*

CAPITULO II: CONCEPCIÓN Y DISEÑO DEL PROYECTO 24

2.1 OBJETIVOS DEL PROYECTO 24

2.1.1 Bloques conceptuales del proyecto *26*

2.1.2 Software del proyecto *28*

2.1.3 Hardware *30*

2.2 DISEÑO GENERAL DEL PROYECTO 36

2.2.1 Unidad central de proceso *37*

2.2.2 Diseño de la interface de red *52*

2.2.3 Diseño de las interfaces de datos adicionales *70*

CAPITULO III: IMPLEMENTACIÓN Y CONSTRUCCIÓN DEL HARDWARE **77**

3.1 INTEGRACIÓN DE BLOQUES CONCEPTUALES DEL PROYECTO 77

3.1.1 Selección de dispositivos de sincronización *78*

3.1.2 Análisis de requerimientos de potencia del sistema *81*

3.1.3 Minimización de componentes del sistema *82*

3.2 DISEÑO DE LA PLACA DE CIRCUITO IMPRESO 83

3.2.1 Automatización del diseño *84*

3.2.2 Criterios de diseño *86*

Implementación de la platina de circuito impreso *89*

3.3 DISEÑO DE LAS UNIDADES DE LÓGICA PROGRAMABLE 91

3.3.1 Herramientas de diseño *91*

3.3.2 Diseño e implementación *93*

3.4 CONSTRUCCIÓN FINAL DEL HARDWARE DEL PROYECTO 97

3.4.1 Materiales y equipos utilizados *98*

3.4.2 Construcción del prototipo *102*

3.4.3 Descripción y uso del hardware del proyecto *108*

3.4.4 Modificaciones y recomendaciones *117*

CAPITULO IV: DISEÑO DEL SOFTWARE **120**

4.1 ESQUEMA CONCEPTUAL DEL SOFTWARE DEL PROYECTO 120

4.2 DISEÑO DEL CÓDIGO DEL CLIENTE 121

4.2.1 Criterios de diseño *121*

4.2.2 Herramientas utilizadas *126*

4.2.3 Organización del Software *131*

4.2.4 Descripción detallada del software cliente *136*

4.3 DISEÑO DEL CÓDIGO DEL SERVIDOR 194

4.3.1 Implementación y diseño del software *194*

4.3.2 Herramientas utilizadas *197*

4.3.3 Descripción general del código a modificar. *198*

4.3.4 Modificaciones realizadas *199*

CAPITULO V: IMPLEMENTACIÓN E INTEGRACIÓN FINAL DEL SISTEMA 204

5.1 COMPONENTES Y EQUIPOS REQUERIDOS 204

5.1.1 Requisitos para la implementación del cliente *204*

5.1.2 Requisitos para la implementación del servidor *207*

5.2 UTILIZACIÓN DEL SISTEMA DE SEGURIDAD 210

5.2.1 Utilización del sistema cliente *211*

CONCLUSIONES Y RECOMENDACIONES 221

ANEXO A: DIAGRAMAS ELÉCTRICOS DEL CLIENTE DE SEGURIDAD 225

ANEXO B: PLACA DE CIRCUITO IMPRESO DEL CLIENTE DE SEGURIDAD 229

ANEXO C: FUENTES DE SOFTWARE DEL CLIENTE DE SEGURIDAD 233

9.1 ARCHIVO SEC_CLI.H 233

9.2 ARCHIVO SEC_CLI.C 239

9.3 ARCHIVO IO_ROUT.C 245

9.4 ARCHIVO DISPLAY.C 251

9.5 ARCHIVO H_INIT.C 253

9.6 Archivo ETHERNET.C 258

9.7 Archivo ARP.C 263

9.8 Archivo IP.C 269

9.9 Archivo UDP.C 275

9.10 Archivo BOOTP.C 278

9.11 Archivo SEC_APP.C 282

ANEXO D: FUENTES DEL SERVIDOR DE SEGURIDAD 288

10.1 Archivo SERVER.C 288

10.2 Archivo SERVER.H 293

10.3 Archivo VALIACCE.C 293

10.4 Archivo MODIFIES.H 318

ANEXO E: FUENTES DE LA LÓGICA PROGRAMABLE 319

11.1 Archivo PAL1.PLD 319

11.2 Archivo PAL2.PLD 320

11.3 Archivo PAL1.JED 321

11.4 Archivo PAL2.JED 322

ANEXO F: FUENTES ADICIONALES DE INFORMACIÓN Y REFERENCIA 323

13 BIBLIOGRAFÍA 326

INDICE DE FIGURAS

FIGURA 1: TOPOLOGÍA DE RED ETHERNET 17

FIGURA 2: ARQUITECTURA Y PAQUETE ETHERNET 18

FIGURA 3: COMPARACIÓN ENTRE ARQUITECTURAS DE RED OSI Y RED TCP/IP 20

FIGURA 4: BLOQUES CONCEPTUALES DE SOFTWARE 28

FIGURA 5: BLOQUES CONCEPTUALES DE HARDWARE 31

FIGURA 6: ESTRUCTURA INTERNA DEL MICROCONTROLADOR 8051/31. 42

FIGURA 7: DIAGRAMA DE LA ARQUITECTURA UTILIZADA. 44

FIGURA 8: DIAGRAMA DE LA UNIDAD CENTRAL DE PROCESO 46

FIGURA 9: ORGANIZACIÓN DEL ÁREA DE MEMORIA EXTERNA 51

FIGURA 10 : ESQUEMA DE ACCESO A LA RED CON EL DP83902A. 55

FIGURA 11: INTERFACE DE RED CON ACCESO PROGRAMADO. 59

FIGURA 12: ESQUEMA DE CONTROL DEL PUERTO DE ENTRADA Y SALIDA DE DATOS. 62

FIGURA 13: ESQUEMA DEL INTERFACE DE CONFIGURACIÓN DEL ST-NIC 64

FIGURA 14: DIAGRAMA DE LA INTERFACE ENTRE EL ST-NIC Y EL 80C31 67

FIGURA 15: INTERFACE DE TECLADO DEL SISTEMA 71

FIGURA 16: INTERFACE DE CONTROL DEL DISPLAY 74

FIGURA 17: ESQUEMA DEL INTERFACE HACIA EL PORTERO ELÉCTRICO 75

FIGURA 18: COLOCACIÓN DE ELEMENTOS EN LA PLACA DE CIRCUITO IMPRESO 89

FIGURA 19: CONFIGURACIÓN DE LA LÓGICA PROGRAMABLE 96

FIGURA 20: VÍAS CON AGUJERO PLATINADO Y VÍAS CONECTADAS CON ALAMBRE 103

Figura 21: Componentes principales y conexiones del sistema 109

Figura 22: Conector de alimentación 111

Figura 23: Configuración estándar del DIP switch 115

Figura 24: Diagrama organizacional del software cliente................................. 132

Figura 25: Diagrama de flujo simplificado del software cliente..................... 134

Figura 26: Formato de paquete de la aplicación de seguridad 179

Figura 27: Componentes del servidor de seguridad 195

Figura 28: Pantalla de Inicialización.. 212

Figura 29: Iconos del Sistema Cliente 212

Figura 30: Estado normal del sistema 213

Figura 31: Proceso de ingreso de código 214

Figura 32: Posibles conclusiones del proceso de ingreso de código 215

Figura 33: Pantalla de autorización de acceso inmediato................................. 216

Figura 34: Pantalla de requerimiento de clave secreta 217

Figura 35: Ejemplo de pantalla de acceso denegado 217

Figura 36: Pantalla de indicación de falta de respuesta del servidor........... 218

Figura 37: Ingreso de la clave de acceso 219

Figura 38: Clave secreta correctamente ingresada 219

Figura 39: Reintento de ingresar clave secreta y exceso de intentos de ingreso 220

Figura 40: Expiración del proceso de ingreso de clave 220

INDICE DE TABLAS

TABLA 1: ENTRADAS Y SALIDAS DE LA LÓGICA PROGRAMABLE 94

TABLA 2: LISTA DE MATERIALES DEL PROYECTO 101

TABLA 3: OPERACIÓN DEL SWITCH DE CONFIGURACIÓN 115

TABLA 4: LISTA DE PROBLEMAS DE DISEÑO DEL PROTOTIPO DEL SISTEMA 117

TABLA 5: ARCHIVOS PERTENECIENTES AL CÓDIGO DEL CLIENTE 137

TABLA 6: ARCHIVOS DEL PROGRAMA DE PROCESAMIENTO DE PEDIDOS DEL SERVIDOR 198

TABLA 7: IMPLEMENTACIÓN DE PARÁMETROS DEL CLIENTE EN EL PAQUETE BOOTP ...210

INTRODUCCION

El presente proyecto tiene como propósito complementar los esquemas de seguridad que la FIEC (Facultad de Ingeniería en Electricidad y Computación) tiene definidos para el control de acceso a sus laboratorios, oficinas y demás dependencias. Este esquema comprende la implementación de una base de datos que contenga la información necesaria para asignar y discriminar los privilegios de acceso del personal y estudiantes de la universidad; a esta base de datos se referirían los controladores de acceso de las distintas dependencias de la universidad para autentificar al personal y decidir si se le da acceso o no.

El objetivo principal de este proyecto consiste en el diseño e implementación de un prototipo de controlador de acceso, el cual deberá ser compatible con el esquema de seguridad arriba mencionado.

Se recomienda al lector de este trabajo que se familiarice con la documentación de apoyo y referencia que se enumera en el anexo F y en la bibliografía, ya que a lo largo del desarrollo de esta tesis se tratarán temas cuya comprensión será muy difícil sin un estudio previo del material adicional.

CAPITULO I

1 Generalidades y Antecedentes

1.1 Requisitos de seguridad en los laboratorios de la ESPOL

El caso de los laboratorios de la ESPOL, y específicamente el de los laboratorios de computación de la FIEC, merece nuestra especial atención, ya que la implementación de una política de seguridad para estas áreas ha sido tema de varios proyectos y tópicos en los últimos años. Las necesidades de seguridad y control de acceso de estas dependencias se pueden resumir de la siguiente manera:

- Identificación de personal o estudiantes que desean ingresar.
- Resolución de privilegios de acceso de la persona, de acuerdo a políticas previamente establecidas.

El primer punto se refiere a la implementación de un esquema para identificar a las personas que desean ingresar a una dependencia dada, las categorías de ingreso pueden definirse a breves rasgos como: personal docente, personal de servicio, ayudantes de laboratorio, estudiantes y visitantes.

A partir de la identificación de la persona y su inclusión en una de las categorías arriba mencionadas se pasa al segundo punto, que implica la decisión de dejarlo pasar o no, de acuerdo a políticas de acceso definidas por la administración del laboratorio. Como

ejemplo, una política de acceso puede ser que no se permiten visitantes en las horas de la tarde, o que no se permite la entrada a estudiantes o visitantes si no hay supervisores en el laboratorio. Las políticas de acceso deben ser muy flexibles, para que puedan acomodar a la gran variedad de requerimientos de acceso que se tiene en un espacio público. Además, dado el gran volumen de personas que por una u otra razón tienen que tener acceso a las dependencias de la universidad, debe ser posible definir políticas generales, que se apliquen a grupos de personas, y no solo a individuos. Esto es importante para mantener la carga administrativa del sistema de seguridad a un nivel manejable.

1.1.1 Proyectos previos en el área

Como previamente se había mencionado, el tema del control de acceso a los laboratorios de la FIEC ha sido tratado en varios trabajos y tópicos de graduación anteriores. En estos trabajos se han realizado significativos avances hacia el logro de los objetivos designados. El avance más significativo y de más relevancia para el presente trabajo fue hecho por el tópico de graduación "Redes LAN y WAN"*(12)** en el año de 1996, en el cual se desarrolló el sistema de base de datos que se utilizará para identificar y asignar privilegios a los usuarios de los laboratorios. Este sistema se compone de un servidor de control de acceso, una base de datos de almacenamiento de usuarios y políticas de acceso, y una aplicación de administración de la base de datos. Además el

---

* El número entre paréntesis indica una referencia bibliográfica.

tópico desarrolló dos clientes de software para este sistema, los cuales son los encargados de obtener del usuario la información necesaria para identificarlo, y se encargan también de accionar las cerraduras para permitir el acceso a la dependencia.

1.1.2 Contribución del presente trabajo

Uno de los mayores inconvenientes de los proyectos que se han realizado en relación con este tema es la falta de un controlador o cliente de acceso que cumpla los requerimientos percibidos como necesarios para la labor que este dispositivo debe realizar. Estos requerimientos son:

- Bajo costo
- Tamaño reducido
- Facilidad de instalación y mantenimiento
- Facilidad de expansión
- Resistencia al abuso y amabilidad hacia el usuario

Previamente al presente proyecto, los controladores de acceso se han implementado mediante el uso de PCs de bajo costo, normalmente máquinas 386, con capacidad limitada de memoria, con sistemas operativos como MS-DOS o Windows 3.1. Aunque el costo presente de una de estas máquinas puede llegar a ser bastante reducido, estas presentan serios problemas en las áreas de expansión, mantenimiento y resistencia al abuso, debido a que son básicamente equipos de escritorio obsoletos, lo que implica que no están diseñados para operar en exteriores o ambientes adversos, que no resisten altos

volúmenes de uso y abuso, y que su estandarización y por ende su instalación y mantenimiento sería difícil, debido a la inevitable heterogeneidad de los dispositivos que se tendría que instalar.

El presente proyecto pretende reemplazar estos equipos con un cliente especializado, de reducido costo y tamaño, de diseño homogéneo y sin partes móviles. Las ventajas de un dispositivo de este tipo sobre los anteriormente utilizados serían bastante significativas. El uso de componentes de bajo costo, y la especialización del diseño, contribuirían a abaratar y miniaturizar el dispositivo, mientras que el uso de un diseño único y localmente desarrollado, y sin componentes desgastables, haría su instalación y mantenimiento mucho menos exigente y costosa. En cuanto al área de facilidad de uso, la propia especialización del diseño contribuirá a mantener su complejidad en niveles reducidos.

1.2 Esquema de internetworking de la ESPOL

Una de las premisas iniciales del proyecto de seguridad que se ha desarrollado en la FIEC, es que este no requerirá infraestructura de comunicaciones adicional, sino que en lo posible utilizará las facilidades de transmisión de información ya existentes en la ESPOL. Debido a esto procederemos a describir y analizar la infraestructura de networking que existe actualmente en la universidad, ya que esta es la que se ha utilizado y se utilizará para implementar el sistema de seguridad y control de acceso a los laboratorios y dependencias.

Las facilidades de internetworking de la ESPOL en la actualidad tienen como componente base una red central ATM, a la cual se conectan las diferentes unidades académicas mediante enlaces de fibra óptica. Esta red esta a su vez conectada al Internet mediante un radioenlace. Las distintas unidades académicas tienen a su vez redes subsidiarias de computadoras, utilizando Ethernet como medio físico y la arquitectura TCP/IP como estándar de comunicaciones.

Justamente debido a la amplia aceptación de Ethernet, y de la arquitectura TCP/IP, se decidió que la implementación de la interface de red de este proyecto utilizaría estas tecnologías para implementar la conectividad y transferencia de información hacia el servidor. En los subtemas restantes de este capítulo haremos un resumen de las características de estas dos tecnologías, para dar una base teórica a los temas que se tratarán en los capítulos restantes.

1.2.1 Las redes Ethernet

Ethernet es una tecnología de redes de área local, basada en una topología lógica de bus, es decir que todas las estaciones de la red están conectadas a un mismo medio físico, el cual comparten. Debido a estas características del medio, y al hecho de que es una tecnología de banda base (todas las transmisiones se hacen en un solo ancho de banda), solo una estación puede enviar información al medio físico en un momento dado. Si dos o más estaciones tratan de transmitir al mismo tiempo, se produce una colisión, en cuyo

caso las estaciones deben abortar la transmisión y reintentarla después de un periodo de espera aleatorio. La información enviada por una estación es recibida por todas las otras estaciones conectadas al segmento de red, las cuales deben discriminar el destino de la información, basándose en un encabezado incluido por el transmisor en el paquete de información enviado.

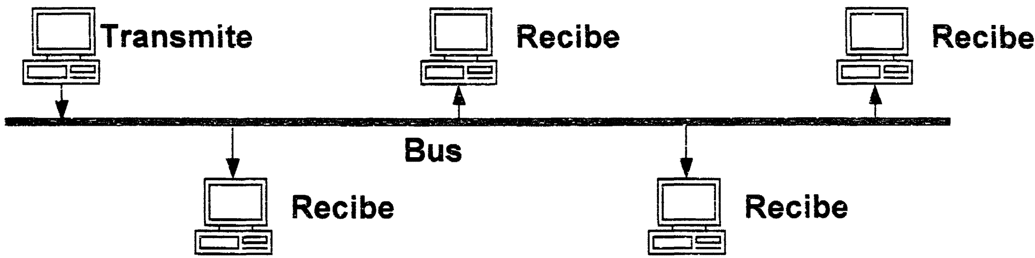


Figura 1: Topología de red Ethernet

Originalmente la topología física de las redes Ethernet se basaba en un bus de cable coaxial grueso, con largo máximo de 500 metros, y con una velocidad de transmisión pico de 10 Mbps, lo que hoy llamamos Ethernet 10Base5; con el paso del tiempo se han hecho varios adelantos en la configuración física de las redes Ethernet, los cuales han aumentado la velocidad de transmisión de la red, así como su extensión máxima, y han abaratado el costo de instalación y mantenimiento de la misma. En la actualidad, la topología física de Ethernet que más se utiliza es Ethernet 10BaseT, la cual consiste en una configuración en estrella de segmentos físicos de cable de cobre trenzado, conectados a un dispositivo central llamado concentrador (hub), el cual se encarga de distribuir la señal entre todos los segmentos para formar la tradicional topología lógica de bus de las redes Ethernet.

Otras evoluciones importantes que ha tenido la tecnología Ethernet han sido la estandarización de las características físicas, eléctricas y lógicas de la red, en la especificación IEEE 802.3, y la adición de la capa de control de enlace definida en la especificación IEEE 802.2. Sin embargo, debido a que los organismos rectores de la estandarización de la arquitectura TCP/IP nunca adoptaron estas modificaciones, las redes Ethernet que implementan esta arquitectura deben utilizar el estándar antiguo, comúnmente conocido como Ethernet II. En los siguientes párrafos estudiaremos un poco más a fondo las características lógicas de este estándar.

Dir. de destino	Dir. de origen	Tipo	Datos	FCS

Figura 2: Arquitectura y paquete Ethernet

La Figura 2 muestra el formato lógico de un paquete Ethernet II; los dos primeros campos contienen las direcciones Ethernet del transmisor y del destino del paquete. Las direcciones Ethernet son números binarios de 48 bits de largo, los cuales deben ser únicos a cada estación para que la red funcione correctamente. El campo siguiente es un número de 16 bits que indica el tipo de información que va encapsulada en el área de datos del paquete Ethernet. Los valores contenidos en este campo marcan la mayor diferencia lógica entre el encabezado Ethernet II y el encabezado IEEE 802.3, ya que este último usa este campo para guardar el largo del paquete Ethernet, y deja la identificación del contenido del mismo al encabezado de enlace de datos IEEE 802.2. Para permitir una cierta interoperabilidad entre los dos estándares, la mayoría de los códigos de identificación de protocolo de Ethernet II empiezan a partir del valor 1500,

que es el largo máximo permitido para el área de datos de un paquete Ethernet. El campo marcado en la figura como FCS (Frame checksum) es una sumatoria de control de error (de 32 bits de longitud), la cual es normalmente introducida automáticamente por el hardware de Ethernet de la estación.

De los distintos campos del paquete Ethernet, los que más nos interesan son los tres primeros, ya que nos dan la información necesaria para procesar correctamente el paquete; la dirección de destino indica la máquina en el segmento de red que debe recibir el paquete. Esta dirección puede también tener un valor de broadcast (ff-ff-ff-ff-ff-ff) el cual indica que todas las máquinas del segmento de red deben procesar el paquete. Una vez establecido el destino del paquete, se utiliza el campo de tipo para decidir a que capa superior de protocolo se entrega el paquete para su procesamiento posterior. Ejemplos interesantes para nosotros son: 0x800 y 0x806 que corresponden respectivamente a la arquitectura TCP/IP y al protocolo Ethernet ARP, los cuales estudiaremos a continuación.

1.2.2 La arquitectura TCP/IP

Contrariamente a lo que se pueda pensar en un principio, TCP/IP no es un protocolo de comunicaciones, sino más una serie capas de protocolos que hacen posible la implementación de redes de área local, área amplia y hasta mundial. Estas capas de protocolo están organizadas aproximadamente de acuerdo al modelo OSI de redes, por lo que nos basaremos en las jerarquías que este define para nuestro estudio de TCP/IP.

Como podemos ver en la Figura 3, en el modelo de internetworking de TCP/IP, las dos primeras capas, la capa física y la capa de enlace de datos, son implementadas de manera totalmente independiente a las capas superiores de la arquitectura. Esto permite una total abstracción de los detalles de diseño y operación de la tecnología y topología de red utilizada. De esta manera, a la arquitectura TCP/IP le es indiferente si esta operando sobre una red Ethernet, Token Ring, o ATM; su operación será invariable.

Arquitectura OSI	Arquitectura TCP/IP
Aplicación	Aplicaciones
Presentación	
Sesión	
Transporte	TCP/UDP
Red	IP/ICMP
Enlace de Datos	Capas inferiores de protocolo
Medio Físico	

Figura 3: Comparación entre arquitecturas de red OSI y red TCP/IP

Si TCP/IP es indiferente a los detalles de operación de la red sobre la que trabaja, podríamos preguntarnos como se puede enviar información desde una estación a otra usando esta arquitectura. Esta función es realizada por la capa base de la arquitectura TCP/IP propiamente dicha, la capa de red o capa 3 del modelo OSI, cuya función conceptual es realizada por el protocolo IP (Internet Protocol). IP es un protocolo de comunicaciones cuya función *es decidir la mejor forma de enviar un paquete de información a un destino determinado.* Para lograr esto, IP implementa un espacio estático de direccionamiento, basado en direcciones de 32 bits de largo. Cada estación

de la red debe tener una dirección IP única. Teniendo como base una dirección de origen y otra de destino, IP es capaz de discernir la ruta más eficaz que un paquete de información tiene para llegar a su destino. Es importante destacar que IP no es responsable de garantizar que la información llegue a su destino. Siendo un protocolo de red, su operación se limita al enrutamiento de la información. Una cuestión importante que hay que aclarar es cual es el mecanismo mediante el cual IP puede enviar un paquete de información a su destino *físico* en una red, si hemos dicho con anterioridad que el esquema de direccionamiento de IP es independiente de los detalles de las arquitecturas de red de las capas inferiores de protocolo. La respuesta es que IP tiene un mecanismo para relacionar la dirección física de una estación con su dirección lógica; este mecanismo es el protocolo ARP (Address Resolution Protocol), el cual se encarga de interactuar con las capas inferiores de la arquitectura para encontrar el equivalente físico de cualquier dirección IP que se desee procesar.

En la capa de protocolo inmediatamente superior a IP, trabajan los denominados protocolos de transporte, cuya función es asegurar que la información enviada sea recibida correctamente, no solo por la estación de destino, sino también por la aplicación o proceso de software al que van destinados. Este es un concepto importante, ya que desde un principio TCP/IP fue diseñado para ser utilizado en sistemas multiusuario, con muchas aplicaciones distintas ejecutándose al mismo tiempo.

Existen dos protocolos principales que operan en la capa de transporte, TCP (Transmission Control Protocol) y UDP (User Datagram Protocol). La diferencia

fundamental entre ellos es el modelo de interacción que utilizan; UDP es un protocolo sin conexión, lo cual significa que la interacción entre emisor y receptor no requiere el previo acuerdo de ambas partes. El emisor envía información sin el previo consentimiento del receptor, el cual puede a su vez recibir o descartar la información a su propia discreción. TCP en cambio es un protocolo orientado a conexión, lo cual implica que cualquier envío de información debe ser previamente acordado por ambas partes, lo que se llama establecer una conexión. Otra diferencia importante es que TCP envía su información en forma de flujos de caracteres, sin fronteras visibles entre los mismos, mientras que UDP envía solo paquetes discretos de información. En cuanto a la manera en que se discrimina el proceso de software al que va destinada una información, ambos protocolos usan el concepto de *puertos*, los cuales son identificadores numéricos que permiten discriminar los procesos de origen y destino de un flujo de datos en la capa de transporte.

Es importante comprender las diferencias entre TCP y UDP, ya que una parte de este trabajo fue decidir cual de los dos protocolos se iba a utilizar para la implementación de la capa de transporte de nuestro sistema.

El modelo OSI implementa varias capas adicionales de interacción por encima de la capa de transporte, pero la arquitectura TCP/IP normalmente delega las funciones realizadas en estas capas a las aplicaciones que vayan a utilizarlo.

Una vez cubiertas estas generalidades teóricas, en los capítulos siguientes haremos un análisis detallado del diseño conceptual y práctico, tanto del hardware como del software de este proyecto.

Capitulo II

2 Concepción y diseño del proyecto

2.1 Objetivos del proyecto

Los objetivos formales de este proyecto se definieron originalmente como:

1. El diseño e implementación de un sistema de control para manejar los siguientes periféricos:
   - Una lectora de banda magnética o código de barras.
   - Un teclado numérico para ingreso de códigos
   - Un display para interacción con el usuario
   - Un portero eléctrico

2. El diseño e implementación de una interface Ethernet de comunicaciones para el ya mencionado controlador de acceso, la mínima funcionalidad de esta interface es proveer un puerto AUI (Attachment Unit Interface).

3. Desarrollo del software de control del servidor remoto, de acuerdo a especificación NDIS (Network Driver Interface Specification).

Sin embargo, a lo largo del desarrollo del proyecto, se hicieron ciertos aumentos y modificaciones a los objetivos originales, los cuales detallamos a continuación.

- Se decidió añadir un puerto 10BaseT a la interface Ethernet, para simplificar y abaratar el acceso a esta muy popular variante de la tecnología Ethernet.

- Se decidió usar la arquitectura TCP/IP como medio de conectividad con el servidor remoto, cuando el objetivo original era el desarrollo de un medio propietario de comunicación cliente-servidor. Se llegó a esta decisión por dos motivos fundamentales: El primero es que el uso de TCP/IP facilita y flexibiliza enormemente el desarrollo del software del servidor, ya que permite la utilización de los recursos de conectividad de TCP/IP ya disponibles en todos los sistemas operativos modernos, y permite la interacción del sistema con múltiples plataformas de hardware y software, y no solo con sistemas Microsoft Windows y OS/2, como hubiera sido el caso si se hubiera utilizado un protocolo arbitrario compatible con la especificación NDIS de Microsoft y 3Com. El segundo motivo es que el uso de TCP/IP permite que el cliente desarrollado interactúe con los sistemas de control diseñados durante los proyectos anteriores que se han efectuado en este campo, todos los cuales han trabajado sobre TCP/IP.

- Se tomó la decisión de evitar el desarrollo de un prototipo de servidor de acceso, sino que simplemente se modificaría el software diseñado por los grupos anteriores para permitirle operar con el cliente diseñado en este proyecto. Esta decisión se

tomó debido a que el trabajo realizado por los grupos anteriores en lo que respecta al desarrollo del servidor de acceso es bastante completo y adecuado a las necesidades del proyecto, por lo que se consideró que cualquier trabajo extra en esa parte del sistema era superfluo, y más bien era muy deseable la interoperabilidad inmediata entre el cliente desarrollado en este proyecto y el servidor desarrollado por los proyectos anteriores.

2.1.1 Bloques conceptuales del proyecto

La división conceptual más obvia que se puede hacer en el desarrollo de este proyecto es entre hardware y software. Ambos bloques deben estar correctamente diseñados e implementados para obtener un sistema eficiente y que cumpla con los criterios de diseño del mismo. Por eso no se puede decir que en este proyecto es más importante el hardware que el software, o viceversa; cada parte cumple una función indispensable. Sin embargo, hay una distinción muy importante que marcó la forma en que se desarrollo este proyecto. *Es mucho más difícil y costoso hacer cambios en el hardware que en el software*. Esto no es así en todos los casos, pero en este en particular, dadas las condiciones en que se iba a desarrollar este proyecto, se consideró deseable comenzar con el diseño de una arquitectura de hardware viable que cumpla todos los requisitos de diseño del proyecto, y a partir de ella pasar al diseño del software a utilizarse en el proyecto. Lo que esto significó, es que no se escribió ni una sola línea de código hasta tener una arquitectura de hardware diseñada e implementada; esto significó además, que el desarrollo del software se hizo a partir de lo que era posible implementar en la

arquitectura de hardware ya diseñada. En el diseño de un proyecto comercial esta forma de trabajar sería costosa e ineficiente, por lo que el lector se preguntará por que se la escogió, en vez de usar un esquema de desarrollo concurrente. Para explicar esto, dedicaremos el siguiente espacio a la descripción de las condiciones de desarrollo de este proyecto.

Dados los requerimientos de diseño delineados en la sección 1.1.2, desde el principio se decidió que el uso de un sistema de microprocesador de uso general (por ejemplo una tarjeta tipo Intel x86) sería demasiado costoso y paradójicamente inflexible. El uso de una arquitectura especializada de tipo integrado, basada en un microcontrolador presentaba grandes ventajas para el desarrollo del hardware del controlador. Pero el uso de este tipo de arquitecturas presentaba a su vez fuertes retos de diseño, ya que todo el sistema de control debía ser diseñado e integrado desde cero, en vez de partir mayormente de partes prefabricadas. Ya que las facilidades disponibles para este tipo de desarrollo eran primitivas o inexistentes, cualquier error de diseño hubiera sido sumamente difícil de diagnosticar y corregir, así que se decidió que todas las etapas de diseño de hardware del proyecto se realizarían de la manera más conservadora y flexible posible, con el propósito de minimizar fuentes de error. El resultado de esto es un diseño bastante fácil de entender y diagnosticar, pero muy ineficiente con relación a lo que se puede conseguir con los componentes utilizados. Cuando decimos ineficiente, no estamos implicando que el hardware y software del proyecto funcionan incorrectamente, sino que nuestro diseño no explota a cabalidad las capacidades de los componentes utilizados. Esto fue necesario para evitar complicaciones causadas por

técnicas complejas de diseño; complicaciones que no era posible diagnosticar debido a la falta del equipo apropiado. Veremos múltiples ejemplos de este diseño conservador a lo largo de las discusiones posteriores. Estas simplificaciones y desperdicios representan retos para futuras optimizaciones del proyecto. Otro punto importante es que el desarrollo de este trabajo fue individual, y sin ninguna experiencia previa que permitiera obviar pasos de diseño, lo cual conspiró contra un desarrollo más rápido o integrado del proyecto.

2.1.2 Software del proyecto

Definida ya la división conceptual fundamental de nuestro proyecto, podemos profundizar en la subdivisión de funciones del mismo, empezando por la subdivisión de bloques conceptuales del software a implementar, ya que es relativamente simple comparado con los bloques conceptuales de hardware.

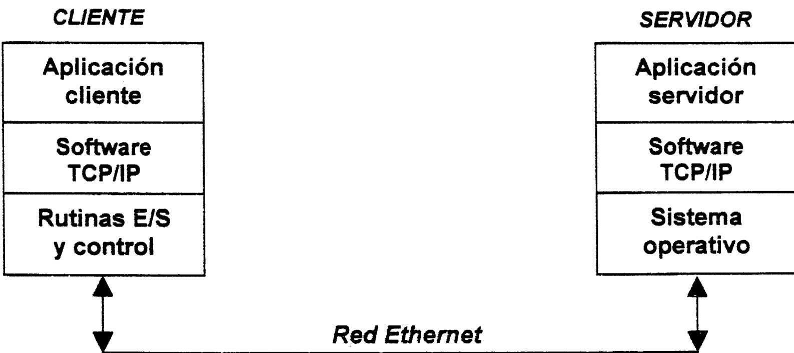


Figura 4: Bloques conceptuales de software

Como vemos en la Figura 4, la división fundamental de nuestro software es entre cliente y servidor, siendo el cliente nuestro sistema integrado a diseñar, y el servidor es una

aplicación ejecutándose en un computador remoto. Para el presente proyecto, el servidor es una aplicación desarrollada por el tópico "Redes LAN y WAN"(12), ligeramente modificada para interoperar con el cliente aquí descrito. Esta aplicación corre sobre el sistema operativo Windows NT de Microsoft, y sirve como punto de interface hacia un sistema de base de datos SQL Server.

El software tanto en el cliente como en el servidor tiene el mismo tipo de divisiones conceptuales: la aplicación propiamente dicha, que es la que ejecuta las funciones necesarias para que tanto cliente como servidor realicen su interacción, el software TCP/IP que es el que hace posible la comunicación entre las aplicaciones en el cliente y el servidor, y las rutinas de Entrada-Salida y control que realizan la interacción con el hardware tanto en el cliente como en el servidor. Obviamente en el caso del servidor estas últimas funciones son realizadas por el sistema operativo del computador; y en el caso específico de este servidor, incluso el conjunto de protocolos de TCP/IP son una parte integral del sistema operativo, pero en interés de la claridad los hemos puesto como un bloque conceptual separado.

Dado el hecho de que el diseño e implementación de la funcionalidad de software se desarrolló posteriormente al diseño del hardware del proyecto, hemos decidido dejar la descripción detallada de la funcionalidad de software del proyecto para el capítulo IV, concentrándonos por ahora en el desarrollo de la funcionalidad del hardware del proyecto; esto permitirá que nuestra comprensión de las decisiones realizadas para el

diseño del software y hardware sea más fácil, ya que se podrá saber la cantidad de información y criterios disponibles al momento en que estas fueron tomadas.

2.1.3 Hardware

Una vez definida la forma básica del modelo de interacción de software, se puede empezar el estudio detallado de la funcionalidad de hardware necesaria para la implementación de nuestro proyecto. Al revisar este diseño hay que recordar que debido a las condiciones de diseño planteadas anteriormente, y a los cambios realizados en los objetivos del proyecto, el hardware del proyecto se diseñó antes de conocer los detalles de la forma final de la interacción de software cliente-servidor. Esta falta de información llevó a ciertos compromisos de diseño, ya que funciones de hardware que se consideraban innecesarias en el modelo inicial de interacción de hardware (como un espacio de memoria no volátil para configuración), terminaron siendo altamente recomendables en una etapa posterior del proyecto en que era demasiado costoso modificar la arquitectura para acomodarlas. Afortunadamente, ninguna de estas consideraciones fue de magnitud suficiente para hacer necesaria la reconstrucción y/o rediseño de la arquitectura, y pudieron ser solucionadas con adiciones a las funciones de software. Como ejemplo podemos indicar que debido a la falta de una memoria no volátil de configuración, nuestro sistema debe implementar un protocolo de red para autoconfiguración (BOOTP). Futuras mejoras a la arquitectura harían posible la adición de este espacio de memoria, y harían innecesaria nuestra implementación de BOOTP. Tomando en cuenta estos detalles, pasemos a analizar los bloques conceptuales que se utilizaron para diseñar la arquitectura de hardware del controlador de acceso.

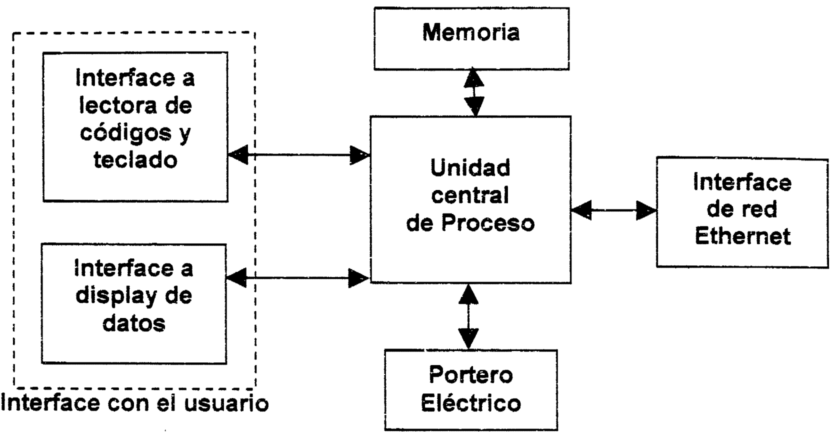


Figura 5: Bloques conceptuales de hardware

La Figura 5 muestra la organización conceptual del hardware del proyecto. Como podemos ver, es una arquitectura muy convencional, organizada alrededor de una unidad central de proceso, con interfaces hacia los distintos periféricos que contiene el sistema. Algo que llama la atención es el hecho de que las interfaces de datos hacia los dispositivos lectores y hacia el teclado han sido unificadas, cuando podía pensarse que deberían ser entidades separadas. Esto se debe a que casi desde los inicios del proyecto se supo que esta interface iba a ser una conexión tipo teclado AT, muy común en sistemas de computadoras personales, ya que se disponía de dispositivos lectores con esa arquitectura. Esto trae sus ventajas y desventajas, las cuales serán analizadas más adelante. Pasemos ahora a hacer un análisis más detallado de cada una de estos bloques conceptuales, y las funciones que deben desempeñar.

2.1.3.1 Unidad central de proceso

La unidad central de proceso del sistema, es la encargada de coordinar y dirigir las actividades de todos los periféricos del mismo. Como podemos ver en la Figura 5, la unidad central de proceso es el bloque central hacia y desde el cual fluye la información de todos los periféricos del sistema. Un detalle importante es que en la figura se ha separado la memoria del sistema como un bloque aparte; esto, como veremos más adelante, se hizo debido a las especiales características de la unidad de procesamiento utilizada; pero se puede asumir sin ninguna ambigüedad que la memoria del sistema es una parte integral de la unidad central de proceso del mismo, ya que la gran mayoría de sistemas de procesamiento integrado incluyen su espacio de memoria dentro del mismo dispositivo de control, y solo en casos especiales se incluye memoria externa adicional.

Definiendo con exactitud las funciones que debe realizar la unidad central de proceso, tendríamos lo siguiente:

- Ejecución del software de control de acceso, protocolos de red, y rutinas de bajo nivel para control de periféricos.

- Provisión de interfaces de hardware hacia los periféricos y memoria del sistema, de una manera que facilite su operación, abarate costos, y minimice potenciales problemas.

2.1.3.2 Interface de red

La interface de red es el componente del sistema que se encargará de recibir y enviar datos desde y hacia la red Ethernet a la que va a estar conectado nuestro equipo. Esto implica que los datos a transmitir deberán ser recibidos de la unidad central de proceso del sistema, y codificados para hacer posible su transmisión. Así mismo, los datos recibidos de la red por la interface deberán ser descodificados, y entregados a la unidad central de proceso. La interface deberá tener al menos un puerto AUI, lo cual es requerimiento de los objetivos del proyecto, además se hace recomendable la adición de un puerto 10BaseT, debido a la popularidad de esta topología

2.1.3.3 Interfaces con el usuario

El objetivo principal de este proyecto es justamente el diseño de un equipo que sirva de interface entre un servidor de seguridad y control de acceso, y los equipos de ingreso de datos y de control que estarán a disponibilidad del usuario. Hay que enfatizar este punto para que quede claro que el proyecto no tendrá ningún uso práctico si las interfaces de entrada y salida de datos, o interfaces con el usuario, como también podrían llamarse, son inadecuadas. Como se delineó en los objetivos, debemos proveer interfaces para una lectora de banda magnética o de código de barras, un teclado numérico, y una pantalla que ofrezca retroalimentación al usuario.

Como ya se indicó, desde muy temprano en el proyecto se tomó la decisión de que la interface con las lectoras de datos y con el teclado sería una interface de teclado AT estándar; esto se debió simplemente a que la ESPOL tenía disponible lectoras y teclados numéricos que se ajustaban a ese tipo de interfaces. Una particularidad importante de los equipos provistos por la ESPOL es que estaban diseñados para ser utilizados en el puerto de teclado de un computador personal. Lo que esto significa es que la lectora (en este caso de código de barras) y el teclado numérico utilizan *la misma* interface física para su conexión. Esto significa que nuestro proyecto no necesita implementar interfaces separadas para estos equipos, lo cual es obviamente bueno desde el punto de visto de la simplicidad y la economía, pero introduce otras dificultades, como se verá más adelante.

Otro punto importante que tuvo que ser discutido en esta etapa del proyecto, es exactamente que tipo de unidad de lectora de datos se iba a utilizar. Esto es importante, por que las lectoras de banda magnética y las lectoras de código de barras son equipos bastante distintos, con funciones distintas, y desafortunadamente para nosotros, con formatos de salida de datos distintos. Una tarjeta de banda magnética estándar tiene tres campos de lectura de datos, llamados tracks 1, 2 y 3. El primero de estos campos es alfanumérico y los otros dos solo contienen números. El problema es que las lectoras de banda magnética tienden a enviar al usuario los tres campos al ser activadas, a diferencia de las lectoras de código de barras, que envían solo un campo de datos numéricos. Aunque este tipo de incompatibilidad no presenta realmente muchas complicaciones, y puede ser resuelto mediante software, para mantener la simplicidad

del dispositivo, y debido a que ya fue utilizado con éxito en proyectos anteriores, se decidió usar una lectora de código de barras como dispositivo de entrada de datos.

En el lado opuesto del flujo de información está la retroalimentación hacia el usuario. En la gran mayoría de dispositivos de control de acceso basados en tarjetas, se evita la implementación de un display de datos, prefiriéndose el uso de sonidos para indicar al usuario el éxito o fracaso de su transacción. Sin embargo, en nuestro caso, esto se vuelve complicado, debido a dos factores:

- Un requerimiento de las especificaciones de seguridad indicadas en el primer capítulo es que debe haber una categoría de visitantes dentro de las políticas de acceso del sistema de seguridad. Estos visitantes obviamente no tendrán una identificación de la ESPOL, por lo que deben usar su número de cédula como código de identificación, el cual tendrían que ingresar manualmente en el teclado numérico. Este tipo de tarea se dificultaría bastante con la ausencia de un display de retroalimentación, ya que el usuario no podrá ver si ha ingresado correctamente los dígitos.

- El segundo factor es el hecho de que la mayoría de los usuarios deberán ingresar manualmente una clave privada de acceso para terminar el proceso de identificación, lo cual hace necesario un display por las mismas razones del punto anterior.

Este display no tiene que ofrecer funcionalidades extravagantes al usuario, como ya indicamos, su función es dar mensajes simples de retroalimentación en caso de errores del usuario o fallas del sistema. Un punto importante es que el display debe ser económico, resistente a la intemperie, y razonablemente fácil de incorporar al sistema.

El dispositivo de entrada y salida de datos que nos queda por mencionar es bastante trivial, pero no por eso menos vital que cualquier otro componente del proyecto; nos referimos al actuador que abrirá la puerta una vez que se haya completado el proceso de identificación del usuario. Este dispositivo debe ser lo más simple y barato posible, y sin embargo debe ser confiable, ya que el funcionamiento del sistema depende de él.

2.2 Diseño general del proyecto

Una vez que se han definido los bloques funcionales necesarios para la construcción de un sistema que cumpla con los objetivos del proyecto, debemos pasar a diseñar e integrar estos bloques, para convertir el concepto desarrollado en una arquitectura viable.

Debido a que la arquitectura utilizada en la unidad central de proceso del sistema tiende a dictar y limitar las características de los periféricos del sistema, la mejor aproximación a la descripción del diseño es desarrollar esta primero y posteriormente definir el diseño de los periféricos. Sin embargo, hay que recalcar que en la práctica esto no es necesariamente la mejor forma de diseñar el sistema. Como ejemplo, podríamos simplemente escoger una arquitectura para nuestro interface Ethernet, y desarrollar la

unidad central de proceso de más bajo costo que nos permita interactuar con la interface escogida. Sin embargo, estas consideraciones se vuelven inmateriales una vez que el diseño está listo, y para no complicar innecesariamente nuestra descripción, describiremos primero el diseño de la unidad central de proceso.

2.2.1 Unidad central de proceso

2.2.1.1 Criterios y requerimientos de diseño

Existe una serie de argumentos fundamentales que influyen en la decisión de que arquitectura utilizar para la unidad central de proceso de un proyecto, siendo los más importantes el costo, la complejidad de las tareas que debe realizar la unidad, y el grado de especialización del diseño de la misma.

En nuestro caso en particular, necesitamos diseñar una unidad central de proceso que realice tareas sumamente complejas (implementación de la arquitectura TCP/IP e interacción con varios periféricos complejos), al menor costo posible. Ya que nuestro diseño no será utilizado para ninguna otra función, este puede ser tan especializado como sea necesario, lo cual es bueno, ya que la especialización de un diseño tiende a bajar sus costos.

Debido a que nuestra unidad deberá ejecutar procedimientos de software complejos, desde el inicio se descartó la posibilidad de usar controladores discretos, y se proyectó la utilización de un microprocesador, ya que este permite la mayor flexibilidad y

Al igual que el bus de datos externo, la mayoría de las funciones del controlador, como los contadores, puerto serial, controlador de interrupciones, etc., utilizan líneas de los puertos de E/S del controlador. Esto se hace para mantener bajo el número de pines del circuito integrado del controlador, pero tiene la desventaja de que en implementaciones con bus externo (como la nuestra) el número de líneas de E/S disponibles puede volverse bastante limitado.

La Figura 6 muestra la manera en que está organizado un sistema de bus externo alrededor de un controlador 8051/31. En este caso particular, el bus externo se conecta a una memoria EPROM, la cual contiene el microcódigo que se ejecutará en el procesador. Nótese el uso de los pines EA, ALE y PSEN[1]. Hay que decir que el diagrama muestra únicamente la implementación del espacio de memoria de código; debemos recordar que el espacio de memoria RAM externo del 8051/31 es totalmente independiente de su espacio de memoria de código. Si se desea implementar memoria RAM externa, o usar periféricos direccionados, a la implementación de la figura se debe añadir el uso de las señales RD y WR del procesador.

Cabe destacar que existen muchas variaciones de la arquitectura básica que mostramos en la Figura 6 , como por ejemplo la unificación de los espacios de memoria de código y de datos, y otras variaciones utilizadas en situaciones específicas, pero todas ellas se basan en la utilización de las mismas señales y funcionalidades del procesador.

---

[1] Para una descripción detallada de la operación del 8051/31, referirse al anexo F.

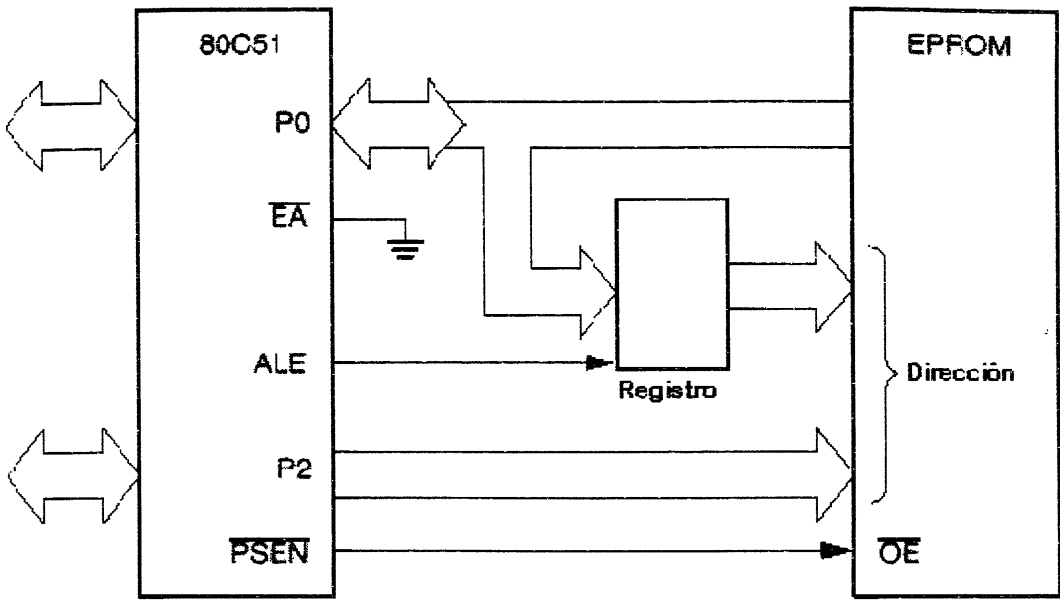


Figura 7: Diagrama de la arquitectura utilizada. Imagen tomada de *80C51 Family Architecture.* (5)

En nuestro caso, debido a que usaremos varios periféricos direccionados, aparte de las memorias de código y datos, tendremos que utilizar lógica de descodificación para discernir cual es el periférico que se desea utilizar en un momento determinado.

2.2.1.3 Descripción de la arquitectura utilizada

Definidos ya los requerimientos del diseño de nuestra unidad central de proceso, y las características principales del dispositivo a utilizarse para implementarla, podemos pasar a referirnos en mayor detalle a la implementación que realizamos para la unidad central de proceso de nuestro sistema.

Quisiéramos recalcar, antes de empezar nuestra descripción, que la comprensión de los temas discutidos aquí requiere haber estudiado detenidamente la documentación técnica referente al procesador 8051; debido a su extensión y complejidad, esta información no será examinada en este trabajo, sino que ha sido referenciada en el anexo F.

En la Figura 8 podemos ver un diagrama esquemático de nuestro diseño para la unidad central de proceso del sistema. Antes de revisarlo debemos recalcar que este es un esquema general de nuestro diseño; los diagramas eléctricos detallados están incluidos en el anexo A.

Observemos la forma en que ha sido implementado el bus externo de datos del sistema. Como habíamos mencionado anteriormente, tenemos dos bloques de memoria distintos, el bloque de código y el bloque de datos (memoria RAM y periféricos). Aunque estos dos bloques comparten el mismo bus de datos y dirección, su operación es totalmente independiente, ya que tienen subsistemas de control distintos; específicamente el bloque de código es controlado por la señal PSEN, mientras que el bloque de datos es controlado por las señales WR y RD. La señal ALE se encarga de ingresar el byte bajo de dirección en el registro de mantenimiento 74LS373.

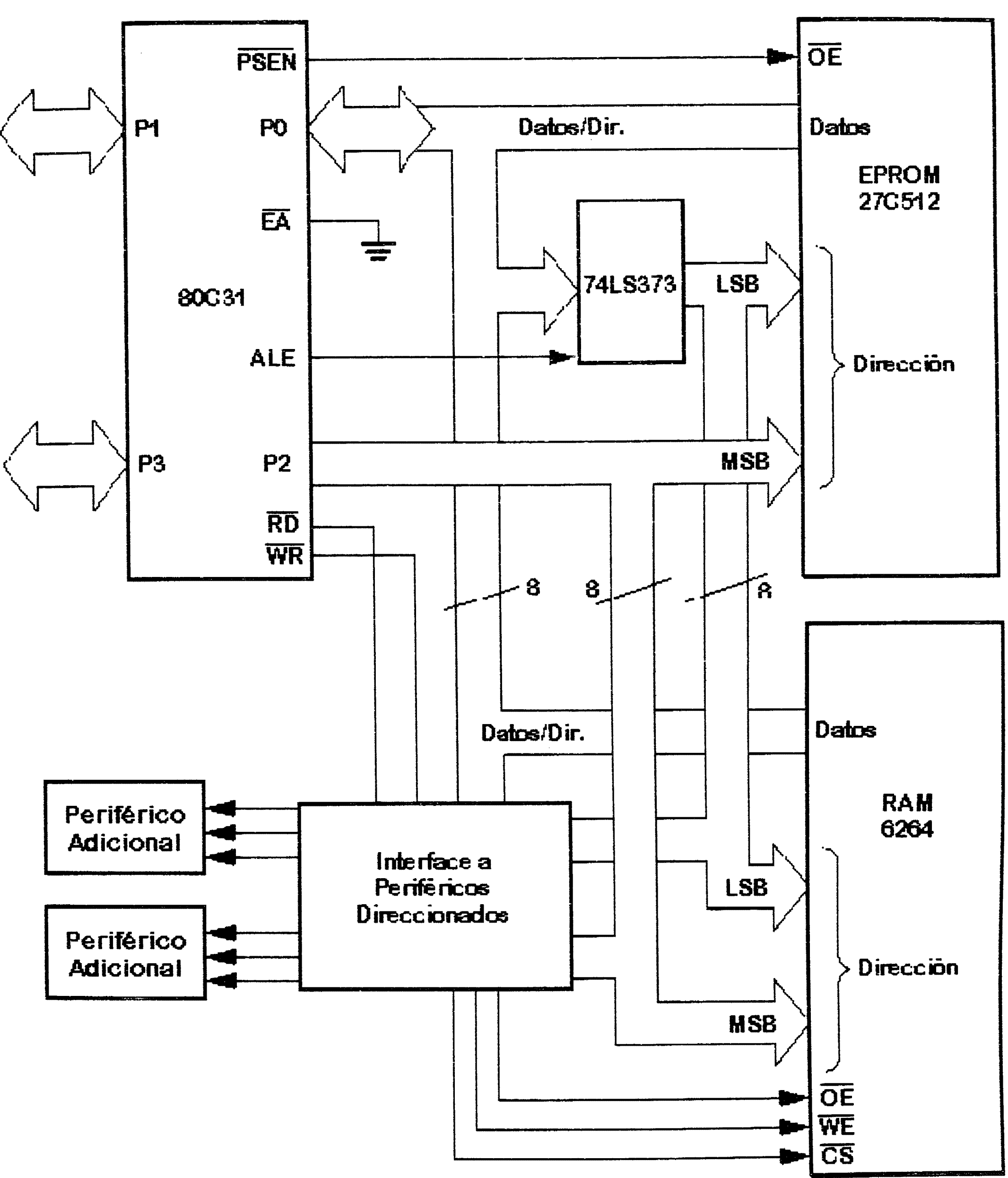


Figura 8: Diagrama de la unidad central de proceso

Podemos notar que se ha asignado una memoria EPROM 27C512 al espacio de memoria de código, lo cual nos da 64 Kbytes de espacio para nuestro software. Aquí hay que destacar que la enorme mayoría de programas y rutinas de control para sistemas integrados ocupan cantidades muy modestas de memoria, y la mayoría podría fácilmente caber en el espacio de memoria integrada de un 8051 (4 Kbytes); sin embargo, debemos recordar que nuestro proyecto es mucho más complejo en cuanto a funciones de software que un proyecto integrado normal, y algo mucho más importante aún: a esta altura del diseño, *no tenemos aún idea del tamaño que va a tener el código de nuestra implementación de software*, ya que este aún no ha sido implementado. En un ambiente de diseño en equipo, con desarrollo simultáneo, esta dificultad podría evitarse, pero en nuestro caso no tenemos más remedio que utilizar la memoria más grande que el dispositivo 8031 es capaz de direccionar, para prevenir el diseño de una arquitectura que cause demasiadas restricciones de complejidad a nuestro software.

En nuestro espacio de memoria de datos hemos utilizado una memoria SRAM 6264, de 8 Kbytes. Nos gustaría decir que nos basamos en criterios totalmente lógicos y estructurados para escoger el tamaño de la memoria de datos, pero por las mismas razones por las que debimos escoger un tamaño máximo para nuestra memoria de código, no teníamos ningún criterio específico para saber el tamaño óptimo de la memoria de datos. En este caso no podíamos darnos el lujo de simplemente escoger también el tamaño máximo de memoria, por dos razones: las memorias SRAM son mucho más caras que una simple EPROM, y además necesitábamos reservar espacio de direccionamiento para los periféricos del sistema. Investigando un poco en el Internet,

encontramos referencias a productos comerciales que implementaban la arquitectura TCP/IP en sistemas integrados, los cuales decían necesitar únicamente 1Kbyte de espacio de variables para funcionar. Tomando esto como referencia, y sabiendo que se iba a necesitar memoria adicional para almacenamiento de paquetes de red, se eliminó alternativas hasta quedar con dos opciones: memorias de 8Kbytes o de 16Kbytes. En interés de la economía del proyecto se tomó la decisión de usar la memoria de 8 Kbytes. Esta decisión implicó un cierto riesgo de dificultades en la etapa de desarrollo de software, pero afortunadamente, como veremos más adelante, este tamaño de memoria RAM resultó ser adecuado para nuestras necesidades.

El uso de 8 Kbytes de nuestro espacio de direccionamiento de datos nos deja 56 Kbytes de espacio de direcciones que podemos utilizar para acceso a periféricos direccionados. Como definición, un periférico direccionado es aquel que es accesado a discreción del microprocesador, mediante un bus de datos compartido. Más adelante examinaremos como se organizó este espacio, y la arquitectura utilizada para descodificar el acceso a periféricos.

Finalmente, podemos ver que en el 8031, que en este caso ya definimos como un dispositivo CMOS tipo 80C31, tenemos dos puertos de entrada/salida de datos (P1 y P3) libres para utilizarlos en acceso a periféricos simples. Hay que aclarar sin embargo, que no todas las líneas de estos puertos están libres ya que, por ejemplo las líneas de control de memoria RD y WR usan pines compartidos con el puerto P3, de manera que ese puerto tendría solo 6 líneas disponibles para otros periféricos.

2.2.1.4 Interface con periféricos

Para implementar las interfaces hacia periféricos de nuestra unidad central de proceso, tenemos dos opciones fundamentales de diseño: La utilización del bus externo de datos del sistema, o la conexión directa de periféricos a los puertos de entrada y salida de datos del microcontrolador 80C31. La opción que se escoja depende estrictamente del tipo de periférico que se este utilizando. Si se tienen periféricos con entradas y salidas orientadas a registros o a espacios de memoria, la opción óptima es integrarlos al bus de datos del sistema, específicamente al área de memoria de datos, que como ya dijimos anteriormente tiene 56 Kbytes de espacio de almacenamiento libres. En cambio, si el periférico basa su operación en señales discretas de entrada o salida, la mejor opción es integrarlo a los puertos de entrada/salida de datos del 80C31. Cabe destacar que hay que tener un cierto equilibrio de criterios al tomar estas decisiones, para conseguir el diseño más económico posible. Esto se debe a que estos dos tipos de conectividad tienen características de diseño totalmente opuestas. Es decir, los periféricos direccionados requieren lógica adicional para integrarse al sistema, lo que los hace relativamente costosos; sin embargo, el amplio espacio de memoria disponible hace posible la conexión (potencialmente) de una gran cantidad de estos periféricos. En cambio, los periféricos conectados directamente al controlador no necesitan (si se los diseña correctamente) ningún tipo de lógica de interface, pero se puede conectar un número muy limitado de ellos, ya que el número de líneas de entrada y salida disponibles es pequeño (y más aún en un sistema con bus de datos externo).

Ya que hemos mencionado que la conectividad de periféricos directos requiere poco o nada en cuanto a lógica de interface, pasemos directamente a analizar el esquema que utilizaremos para integrar los periféricos direccionados.

A estas alturas del diseño, ya debemos definir cuantos periféricos va a tener nuestro sistema, y de que tipo, para poder asignarles recursos de manera efectiva. En varias secciones de este trabajo hemos mencionado que los objetivos del proyecto son interactuar con:

- Una interface de red
- Una lectora de código de barras y un teclado numérico
- Un display de datos
- Un actuador para el portero eléctrico

La interface de red, cuyo diseño veremos inmediatamente después de esta sección, es un híbrido en cuanto a la forma de conectividad, ya que tiene líneas de conexión directa y buses de conexión orientados a registro, así que una parte de su conectividad será integrada al bus de datos del sistema, y otra será integrada directamente al procesador. Como ya habíamos adelantado, la lectora y el teclado numérico comparten el mismo interface, el cual, como veremos más adelante, se puede implementar con conexión directa al procesador. Lo mismo se aplica a la interface con el portero eléctrico, que no

necesita más que una línea lógica para operar. El display de datos, en cambio, va a ser un dispositivo accesado totalmente desde el bus de datos del sistema.

De acuerdo a esto, vamos a tener dos periféricos utilizando el bus de datos externo para operar. Los cuales sumados al espacio utilizado por la memoria RAM externa, nos suma tres espacios distintos de direccionamiento externo que necesitamos para nuestro sistema. Para implementar un sistema de descodificación que permita direccionar estos tres dispositivos, necesitaremos dividir el espacio de memoria externa en cuatro áreas de 16 Kbytes como se muestra a continuación.

$A_{15} = 0 / A_{14} = 0$	**Memoria RAM** **0000 – 3FFF**
$A_{15} = 0 / A_{14} = 1$	**Display de datos** **4000 – 7FFF**
$A_{15} = 1 / A_{14} = 0$	**Interface de Red** **8000 – BFFF**
$A_{15} = 1 / A_{14} = 1$	**Disponible** **C000 – FFFF**

Figura 9: Organización del área de memoria externa

La división de la memoria externa en cuatro áreas de 16 Kbytes de ancho nos presenta dos ventajas significativas: Primeramente nos permite reducir a dos el número de líneas de datos necesarias para discernir entre las diferentes áreas. Además, nos permite, de ser necesario, aumentar sin dificultades el espacio de memoria RAM de 8 a 16 Kbytes.

La implementación de una arquitectura que nos permita descodificar este tipo de organización de memoria es bastante simple, de las 16 líneas del bus de direcciones del sistema, asignamos las dos más significativas, A_{15} y A_{14} a ser el indicador de cual área del bus de direcciones se desea direccionar, de manera que el rango de direcciones correspondientes a cada área de memoria es el que se muestra en la Figura 9.

Una vez explicado el esquema utilizado para la conexión de los periféricos, pasemos a analizar la implementación de estos, y en el caso de los periféricos prefabricados, la implementación de la interface hacia los mismos.

2.2.2 Diseño de la interface de red

En esta sección vamos a describir el diseño de la interface de red de nuestro sistema. Como explicamos en la descripción del bloque conceptual de red, la interface es responsable de enviar y recibir información hacia y desde la red Ethernet a la que estará conectado nuestro sistema.

Para integrar una interface de red a nuestro sistema, resulta importante que su diseño sea lo más simple y compacto posible. Para conseguir esto es muy importante la selección de la arquitectura que se va a usar para la interface, proceso que describiremos a continuación.

2.2.2.1 Selección del controlador de acceso a la red

En la actualidad, las arquitecturas utilizadas para acceder a redes de área local se basan en su gran mayoría en diseños altamente integrados, normalmente contenidos en uno o dos circuitos integrados, los cuales requerirán, dependiendo de la implementación utilizada, pocos o ningún componente subsidiario para su funcionamiento. Obviamente, este tipo de arquitectura se presta magníficamente para los propósitos de nuestro proyecto, ya que permite reducir costos y complejidad del subsistema de acceso a la red de manera notable. Al igual que con la elección del MCU para nuestra unidad central de proceso, el proceso de selección de un controlador de acceso a redes Ethernet fue una labor básicamente investigativa, basándose nuestra selección en los siguientes requerimientos:

- La arquitectura a utilizar debe ser conocida y probada
- Los componentes deben ser obtenidos fácilmente y de costo razonable
- La arquitectura debe ser compacta y fácil de implementar

El primer punto es importante para nuestro proyecto, ya que una arquitectura popular tendrá una gran cantidad de recursos de desarrollo disponibles. Por la misma razón se insiste en el segundo punto, ya que no servirá de nada utilizar una arquitectura exótica u obsoleta, que después sea imposible producir en masa por falta de partes o costo excesivo. Y finalmente, recordando siempre las limitadas facilidades disponibles para experimentación y resolución de problemas de diseño, la arquitectura a utilizar debe ser fácil de implementar.

No fue necesario ir muy lejos en la búsqueda de una arquitectura que satisfaga nuestros requerimientos; el controlador DP83902A de National Semiconductor resultó ser una solución ideal tanto en el área de popularidad, facilidad de obtención y facilidad de diseño. Por estas razones, nuestra arquitectura se basa en una de las más populares y estándares implementaciones de este dispositivo, el cual describiremos en mayor detalle a continuación.

2.2.2.2 El controlador Ethernet DP83902A ST-NIC

El DP83902A ST-NIC (Serial Network Interface Controller for Twisted Pair) es un circuito integrado analógico-digital, el cual integra todas las funcionalidades necesarias para implementar una interface de red 10Base2, 10Base5 o 10BaseT. El dispositivo tiene la capacidad de recibir información de la red a través de una interfaz AUI o 10baset, descodificarla, deserializarla, y transferirla en formatos de palabra de 8, 16 o 32 bits al bus de datos del sistema al que esta conectado. Para realizar transmisiones, el dispositivo debe recibir la información del sistema usando el mecanismo ya descrito, empaquetarla de acuerdo al estándar Ethernet, codificarla y transmitirla, con automatización completa de control de errores y colisiones.

La ventaja para el diseñador en el uso de un dispositivo como este, es que se puede obviar la gran mayoría de los detalles de una implementación Ethernet, ya que el dispositivo los maneja automáticamente, permitiendo al diseñador concentrarse en otros puntos del diseño y minimizando posibles fuentes de error.

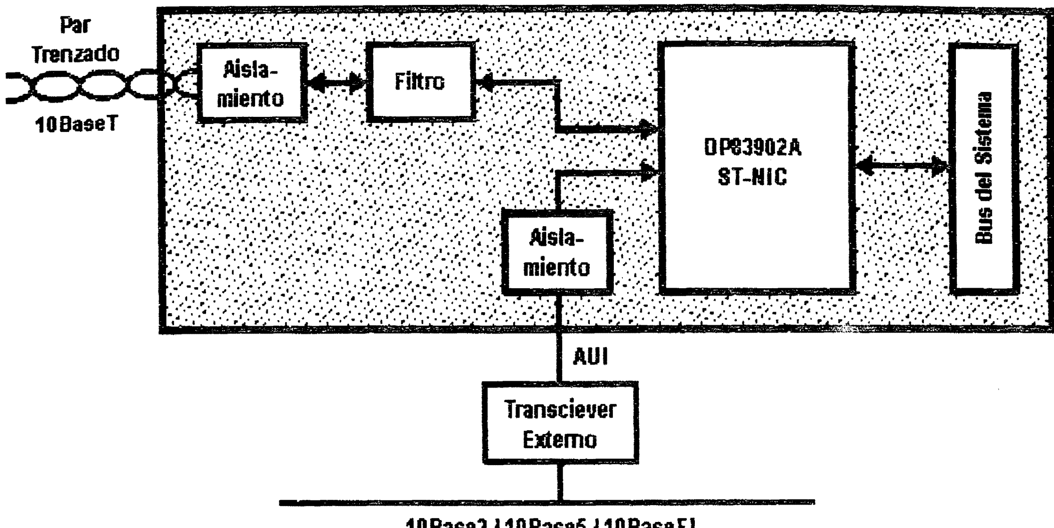


Figura 10 : Esquema de acceso a la red con el DP83902A. Tomado de *DP83902A ST-NIC Serial Network Interface Controller for Twisted Pair.*(11)

La Figura 10 nos muestra un diagrama de bloques del subsistema de acceso a la red, alrededor del controlador de acceso DP83902A. Como podemos ver en la figura, el sistema tiene una sección de acceso al medio físico, implementada con componentes analógicos, y una sección de interface con el bus del sistema al cual está conectado.

Una de las ventajas de usar una arquitectura popular como la que estamos estudiando, es que no hay nada nuevo que descubrir; por ejemplo, todas las funciones analógicas del sistema están claramente definidas y diagramadas en las especificaciones del dispositivo, y en la literatura de apoyo disponible, de manera que no hay necesidad de hacer ningún esfuerzo especial de diseño, sino limitarse a utilizar los ejemplos y modelos certificados para esta arquitectura.

En cuanto al diseño de la interface de acceso al bus de sistema, se aplican las mismas ventajas, ya que el DP83902A es la base de las extremadamente populares arquitecturas de acceso a red NE1000 y NE2000 de Novell, de manera que es posible usar una solución prediseñada, y simplemente hacer pequeñas modificaciones para adaptarla a nuestro sistema.

Habiendo ya dicho que la arquitectura del segmento analógico de la interface de red es básicamente inamovible, podemos, en las siguientes secciones, dedicarnos a estudiar la implementación de la sección digital del sistema. Se recomienda que antes de proceder a las siguientes secciones se estudie a fondo la documentación del DP83902A, la cual cubre en detalle la funcionalidad y características del dispositivo.

2.2.2.3 Arquitectura utilizada

El estudio de la documentación del DP83902A nos muestra que existen varias alternativas para la implementación de la sección digital del subsistema de acceso a la red. Sin embargo, todas estas alternativas basan su funcionamiento en una premisa fundamental, que resume el funcionamiento del DP83902A ST-NIC: *Todas las interacciones de lectura y escritura de datos de red entre el ST-NIC y la unidad central de proceso del sistema se basan en operaciones de acceso a memoria.*

Esto quiere decir que para enviar un paquete de información hacia la red, el sistema operativo tiene que escribirlo en un bloque de memoria RAM, y ordenarle al ST-NIC que lo lea. Así mismo, cuando este último recibe información de la red, la escribe en un

bloque predeterminado de memoria, e informa a la unidad central de proceso que se ha recibido un paquete de la red.

El ST-NIC hace sus operaciones de acceso a memoria usando un esquema DMA (Direct Memory Access), mediante el cual el ST-NIC toma control del bus de datos que lo conecta a la memoria, y realiza la operación de lectura de memoria directamente, sin intervención de la unidad central de proceso del sistema. El ST-NIC posee dos subsistemas independientes de acceso directo a memoria, llamados también *canales DMA*. El canal principal se denomina DMA local, y es usado por el ST-NIC para realizar todas sus operaciones de lectura y escritura de datos de red. El canal secundario, o DMA remoto, puede ser utilizado como un buffer entre el sistema DMA del ST-NIC y el bus de la unidad central de proceso; esto significa que mediante el uso del canal DMA remoto se puede aislar al ST-NIC del bus central del sistema, lo cual es útil en nuestra implementación, como veremos a continuación.

Retomando el tema de nuestra arquitectura de acceso a la red, tenemos que decidir que esquema de interface con la unidad central de proceso debemos utilizar. La facilidad DMA que ofrece el ST-NIC es una excelente solución, la cual ofrece rapidez, efectividad, y la enorme ventaja de poder utilizar directamente la memoria de la unidad central de proceso para las operaciones de red. Las ventajas de esta solución la hacen muy popular en sistemas de alto rendimiento, e inicialmente se entretuvo la idea de utilizarla en nuestra implementación; sin embargo esta idea no prosperó debido a varias dificultades técnicas: Primeramente, la arquitectura del 8051/31 no incluye un

mecanismo de arbitraje del bus externo de datos, o al menos la capacidad de insertar estados de espera en sus operaciones de bus externo; lo cual, aunque no lo imposibilita, si hace muy difícil la implementación de un sistema DMA. Además, las tolerancias del ST-NIC a la latencia del bus (el tiempo que el procesador tarda en entregar el uso del bus) son bastante exigentes, por lo que se consideró que el uso de una arquitectura DMA era un objetivo demasiado ambicioso y riesgoso para este proyecto.

Una vez descartado el uso de acceso DMA a la memoria del sistema, se consideró el uso de una arquitectura de *Acceso Programado*, lo cual significa que la interacción entre la unidad central de proceso y el periférico, en este caso el ST-NIC no es automática, sino que se realiza bajo el control de un algoritmo ejecutado en el procesador del sistema. El ST-NIC permite la implementación de una arquitectura de acceso programado mediante el uso del canal DMA remoto. En este esquema, la unidad central de proceso del sistema escribe o lee la información que se desea enviar al ST-NIC en un registro bidireccional, conocido también como *puerto de entrada y salida de datos (I/O Port).* Una vez que la información es escrita en el puerto, el sistema informa al ST-NIC que hay datos disponibles en el puerto, y el ST-NIC los lee a su vez del puerto, usando el canal DMA remoto, y los almacena en una memoria temporal o buffer (recordemos que el ST-NIC solo puede leer o escribir información en accesos a memoria). Una vez en la memoria temporal, la información puede ser leída a discreción del ST-NIC, mediante el canal DMA local.

La desventaja de esta operación, es que las escrituras y lecturas al puerto de datos, deben hacerse byte por byte, de manera asincrónica, y bajo control del procesador, a diferencia de los accesos DMA, que son automáticos y en ráfaga. Hay que destacar, sin embargo, que en nuestro caso esta desventaja es una ventaja relativa, ya que la naturaleza asincrónica de la operación le da al sistema una tolerancia a fallos y una flexibilidad, que, dadas las limitaciones de nuestro proyecto, bien valen la pena.

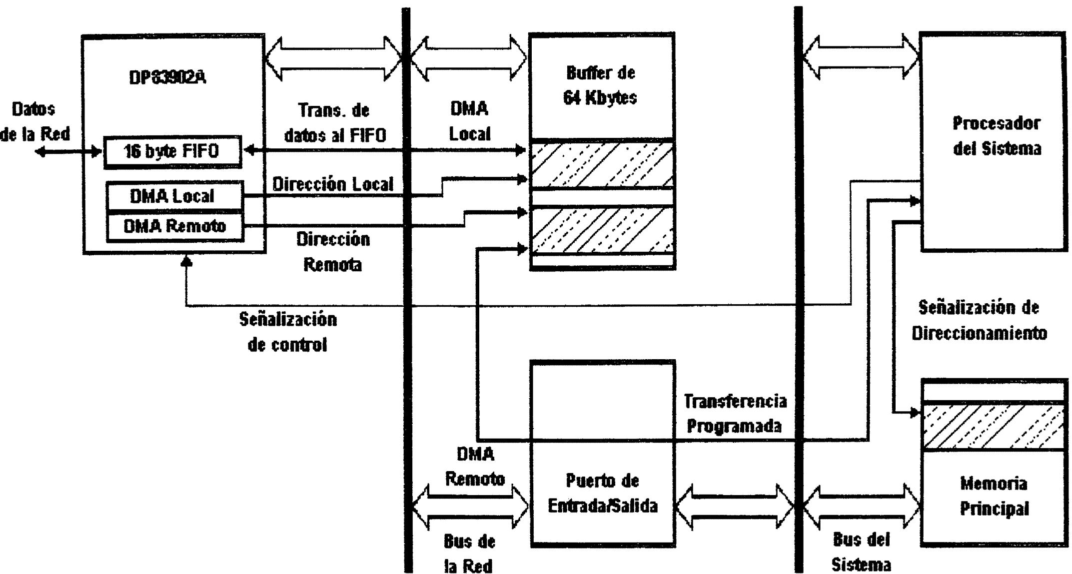


Figura 11: Interface de red con acceso programado. Basado en *DP83902A ST-NIC Serial Network Interface Controller for Twisted Pair.* (11)

El diagrama esquemático de la Figura 11 nos muestra los componentes e interacciones de una interface de red hacia el DP93902A ST-NIC basada en el uso de transferencias programadas. La característica más importante de este sistema es la presencia de dos buses de datos separados, uno es el bus principal del sistema, controlado por el microprocesador, y el otro es el bus de datos de la red, controlado por el ST-NIC. El

subsistema de red tiene su propio espacio de memoria de almacenamiento, y todas las transferencias de información entre los dos buses se realizan a través del puerto de entrada y salida de datos. Lo que se logra con esta arquitectura, es aislar la actividad de transferencia de datos desde y hacia la red, de la actividad normal del sistema; de esta manera nos evitamos todos los problemas relacionados con arbitraje de acceso al bus y latencias, lo cual como explicamos es importante en un sistema tan elemental como el nuestro. Al estudiar el diagrama, podemos notar que las transferencias de datos entre la unidad central de proceso y el puerto de entrada y salida de datos del sistema se realizan directamente desde el microprocesador, bajo estricto control de software.

Como habíamos mencionado, esta arquitectura requiere la utilización de una memoria de almacenamiento temporal en el subsistema de acceso a la red. Aquí nos encontramos nuevamente con el dilema del dimensionamiento de esta memoria. El ST-NIC puede direccionar hasta 64 Kbytes en el bus de datos al cual está conectado, pero obviamente la memoria a utilizarse debe ser más pequeña que eso, dadas las limitaciones de costo de nuestro sistema. Afortunadamente, la arquitectura que mostramos en la figura es básicamente la misma que se utiliza en todas las tarjetas de red compatibles con NE1000 o NE2000, así que podemos fácilmente basarnos en el tamaño de memoria utilizado en esos equipos para dimensionar nuestra memoria de buffer. Encontramos que en la enorme mayoría de implementaciones se utilizan 8 Kbytes de memoria para buses de 8 bits, y 16 Kbytes de memoria para buses de 16 bits. Ya que nuestro sistema y nuestro bus de datos usan palabras de 8 bits de ancho, escogimos 8 Kbytes como el tamaño de nuestro buffer de red.

Una vez concluida esta descripción general de la arquitectura utilizada para nuestra interface de red, podemos pasar a analizar la forma de integrar este subsistema a la unidad central de proceso de nuestro proyecto.

2.2.2.4 Interface con la unidad central de proceso

Para el diseño de una interface entre el ST-NIC y nuestro sistema, tenemos que tener claros los dos diferentes modelos de comunicación que utiliza el ST-NIC. Si revisamos la Figura 11 notaremos que dentro de los subsistemas de lectura de memoria y acceso al puerto de datos, el ST-NIC tiene una serie de líneas de señalización para controlar el progreso de las transferencias DMA. De estas líneas, las pertenecientes al DMA local no deben preocuparnos, ya que se usan para controlar el acceso a la memoria de buffer de red, la cual está bajo el control completo y automático del ST-NIC. El DMA remoto, en cambio, interactúa de manera asincrónica con el puerto de entrada y salida de datos, y ya que este último está (como vemos en la figura) bajo el control directo del microprocesador, es inevitable que algunas de las líneas del DMA remoto tengan que llegar al microprocesador.

Además, un aspecto de la operación del ST-NIC que no está cubierto en el diagrama de la Figura 11 es el subsistema utilizado para programar y controlar este complejo dispositivo. El esquema de programación y control del ST-NIC se basa en accesos a registros direccionados, y trabaja poniendo al ST-NIC en modo de *esclavo,* mediante una línea CS (Chip Select). El concepto de operación de este sistema es que la unidad

central de proceso informa, mediante la línea CS del ST-NIC, que desea escribir o leer datos de los registros de configuración. El dispositivo procede a ponerse en modo esclavo, lee la dirección del registro al cual que sistema necesita acceso en las líneas selección de registro del ST-NIC (R_0 – R_4), y según el valor de las señales SRD y SWR, escribe o lee el byte presente en el bus de datos del ST-NIC hacia los registros. Obviamente, mientras el dispositivo está en modo esclavo, no se puede hacer transferencias DMA de ningún tipo, lo cual nos trae dificultades que analizaremos más adelante.

Una vez descrito el método que se usa para configurar el DP83902A, tenemos los criterios necesarios para estudiar el diseño de nuestra interface entre el ST-NIC y la unidad central de proceso.

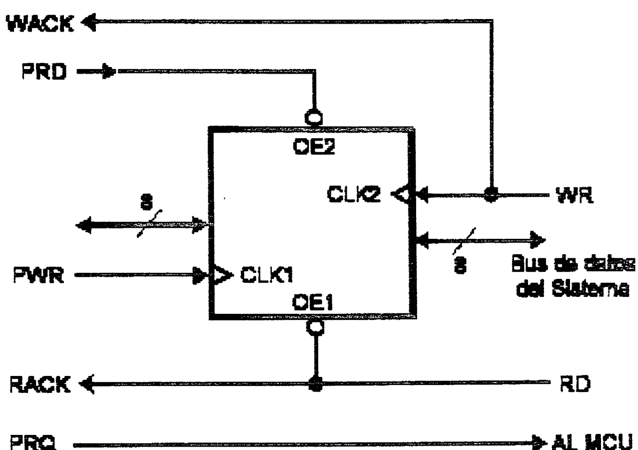


Figura 12: Esquema de control del puerto de entrada y salida de datos. Tomado de *DP83902A ST-NIC Serial Network Interface Controller for Twisted Pair.* (11)

La Figura 12 nos muestra el método genérico utilizado para la transferencia de información entre la unidad central de proceso y el ST-NIC. El mecanismo empieza a funcionar cuando el ST-NIC informa, mediante la señal PRQ (Port Request), que está listo para realizar una transferencia de datos a través del puerto. El sentido de esta transferencia ya es conocido por ambos lados del puerto, ya que las operaciones de transferencia son hechas bajo control del software de la unidad central de proceso, el cual se encarga previamente de programar al ST-NIC, para que inicie un proceso de lectura o escritura en el puerto. Si es el sistema el que está escribiendo al puerto, al detectar la señal PRQ este iniciaría un ciclo de escritura hacia el puerto de E/S, el cual estaría implementado como un periférico direccionado en el bus de datos del sistema. Aquí hay que recordar que la señal marcada como WR, y que es la que ingresa el dato al puerto, no es directamente la señal del MCU, sino una señal esclava generada por la lógica de descodificación de dirección para los periféricos. Esta misma señal es utilizada para informar al ST-NIC (conectándola a la entrada WACK) que el dato ha sido escrito al puerto. En ese momento, el ST-NIC usará la señal PRD para leer los datos del puerto y guardarlos en la memoria buffer de la red, para posteriormente ingresarlos a su sistema de transmisión mediante el DMA local. SI la operación realizada es la última del ciclo de lectura o escritura del puerto, el ST-NIC apagará la señal PRQ.

En el otro sentido, la operación funciona de la misma manera, pero usando las señales PWR, RD y RACK. Hay que volver a recalcar que en nuestra implementación, todas estas actividades de transferencia se realizan bajo el control del procesador, el cual debe

programar al ST-NIC para hacer una operación de lectura o escritura de datos al puerto, esperar a que este responda mediante la señal PRQ, y orquestar la operación de transferencia. También es importante notar que estas operaciones son totalmente asincrónicas; ambos lados esperarán el tiempo que sea necesario a que el otro lado confirme que ha finalizado correctamente la transferencia de datos. Esto es importante para nosotros, ya que aunque es un método lento e ineficiente, es a la vez robusto y fácil de diagnosticar.

Ahora podemos revisar el esquema que debemos utilizar para realizar las tareas de lectura y escritura de los registros de configuración del ST-NIC, que como dijimos se basan en un sistema independiente al que usaremos para realizar las transferencias de datos de red.

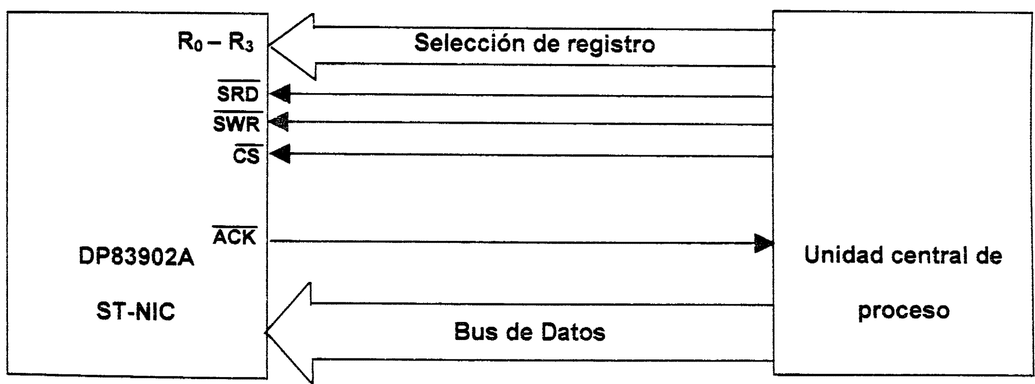


Figura 13: Esquema del interface de configuración del ST-NIC

El diagrama de la Figura 13 contiene el esquema genérico de la interacción de configuración entre el ST-NIC y la unidad central de proceso del sistema. Como habíamos adelantado, para programar al ST-NIC, el procesador del sistema debe

ordenarle que pase a modo de esclavo, mediante la señal CS, e informarle, mediante las señales $R_0 - R_3$, la dirección del registro que se desea leer o modificar. Podría pensarse que el procesador puede inmediatamente escribir o leer la información necesaria, es decir operar de manera sincrónica; sin embargo, esto no puede ser debido a que se usa el bus de datos de la red para transferir el contenido de los registros. Aquí debemos recordar que el bus de datos de la red está bajo el total control del ST-NIC, el cual en cualquier momento dado puede estarlo utilizando para escribir datos llegados de la red mediante el canal DMA local. Debido a esto, el ST-NIC implementa la señal ACK, la cual usa para detener el proceso de acceso a registros, hasta que el bus de la red este libre y el ST-NIC se encuentre listo para ponerse en modo de esclavo. El tiempo de espera hasta la activación de la señal ACK no es predecible, así que tuvimos que abandonar la idea de usar transferencias sincrónicas para el acceso a registros.

Teniendo que implementar varias líneas de señalización entre el ST-NIC y el sistema, incluyendo las cuatro líneas de selección de registro, nos encontramos con el dilema de implementarlas como un periférico direccionado o como un periférico de acceso directo. Dado que estas líneas deben operar en secuencias asincrónicas, no se prestan para ser implementadas en forma direccionada, (con la probable excepción de las líneas de selección de registro) así que se decidió implementarlas como líneas de acceso directo al MCU 80C31.

Habiendo revisado los esquemas de operación de las interfaces entre el ST-NIC y nuestra unidad central de proceso, pasaremos ahora a revisar la implementación que finalmente se hizo a partir de estos esquemas.

Antes de pasar al estudio de la implementación de la interface debemos señalar que debido a la necesidad de minimizar lo más posible el diseño, se tomó la decisión de integrar al puerto de entrada y salida de datos las funciones de acceso a registros. Esta forma de implementación nos ahorra el uso de un circuito integrado de aislamiento de bus, pero paradójicamente parece complicar nuestro diseño con la adición lógica de control para integrar los dos subsistemas de transferencia de datos. Sin embargo, como notaremos en los capítulos subsiguientes, esta aparente deficiencia de diseño no causa ningún impacto, ya que estaremos en capacidad de integrar la lógica no solo de esta interface, sino de todas las interfaces de acceso a periféricos del proyecto.

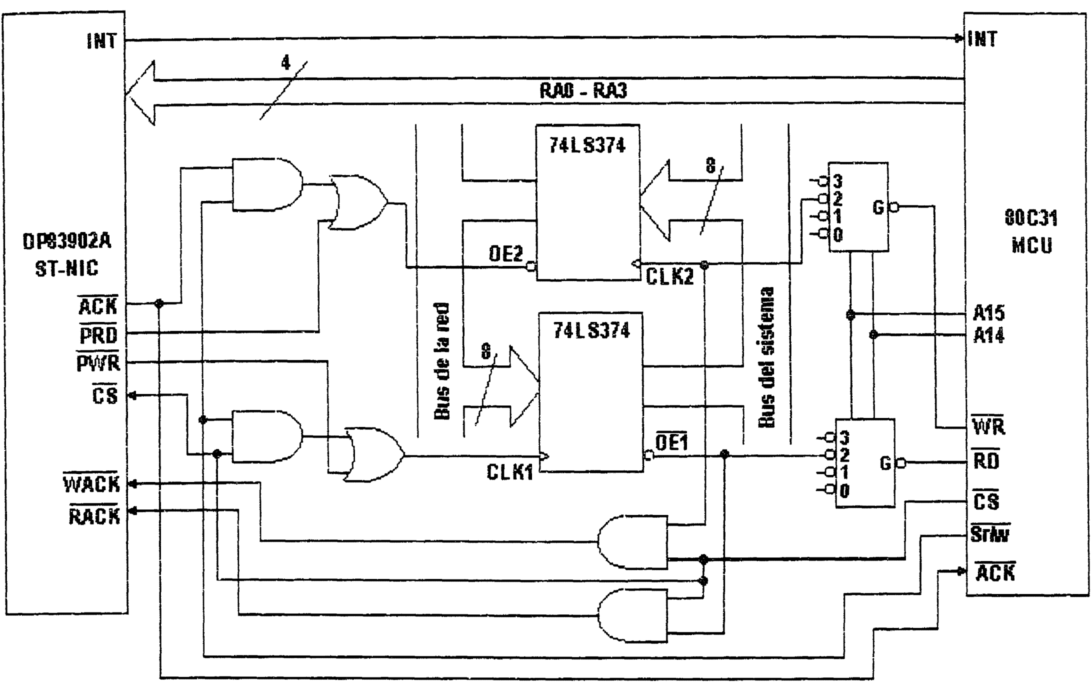


Figura 14: Diagrama de la interface entre el ST-NIC y el 80C31

En la Figura 14 podemos apreciar la forma en que se ha implementado la interface entre el ST-NIC y el procesador del sistema. El puerto de entrada y salida de datos hacia el bus de la interface de red ha sido implementado con dos registros de 8 bits 74LS374. Podemos además notar la estructura de la lógica de direccionamiento de periféricos de la unidad central de proceso, la cual, como habíamos mencionado, utiliza las dos líneas más significativas del bus de direcciones (A_{15} y A_{14}) del 80C31 para descodificar entre los cuatro espacios definidos para periféricos. La señal de salida del decodificador es utilizada para controlar el lado de sistema del puerto de entrada y salida de datos, y manejar las entradas RACK y WACK del ST-NIC. Podemos notar que la activación de estas señales está subordinada al estado de la señal CS, la cual es generada por software en una de las líneas de salida directa del 80C31; esto es necesario porque el puerto es

utilizado también para pasar datos en el modo esclavo, el cual no usa RACK y WACK, por lo que al activarse CS estas líneas son aisladas del estado del puerto.

En el lado del ST-NIC podemos notar la lógica que se ha tenido que implementar para integrar la operación del ST-NIC en modo normal y en modo esclavo. Podemos ver que la operación de lectura de datos del puerto es gobernada por la señal ACK, y por la señal Sr/w. Esta última es simplemente una bandera generada por el software del procesador para diferenciar entre operaciones de lectura y escritura. El proceso de escritura de datos al puerto desde el ST-NIC está en cambio gobernado por la señal CS. La razón para esto está en la forma en que el ST-NIC opera en modo esclavo, y puede discernirse observando los diagramas de tiempo incluidos en la documentación del fabricante.

Siguiendo con la descripción, vemos que las líneas de selección de registro RA_0 a RA_3 han sido implementadas directamente entre el procesador del sistema y el ST-NIC. Inicialmente se consideró implementar estas líneas como salidas direccionadas, lo cual es altamente deseable, ya que son líneas simultáneas y su remoción de los puertos de salida del procesador dejaría cuatro líneas libres para otros usos. Sin embargo, este tipo de implementación, aunque posible, se consideró un potencial punto problemático en el diseño, así que se descartó en favor de la más simple conexión directa.

Otra línea que aparece en el diagrama, y hasta el momento no había sido mencionada, es la línea INT del ST-NIC, que es la que este usa para informar al procesador sobre

eventos que requieren de su atención, como por ejemplo, la llegada de un paquete desde la red. Si observamos los diagramas detallados del sistema, veremos que esta línea llega a una de las entradas de interrupción externa del 80C31, lo cual nos permite implementar una interrupción para procesar eventos de red. Sin embargo, esto no es absolutamente necesario (y eventualmente no fue utilizado) ya que la señal INT mantiene su nivel hasta ser desactivada directamente por el procesador del sistema (Mediante una operación de escritura a los registros del ST-NIC).

Con esta descripción de la interface entre el subsistema de red, gobernado por el ST-NIC, y la unidad central de proceso, terminamos nuestro estudio de la interface de red de nuestro proyecto. Sobra decir que no se han cubierto todos los detalles de la implementación, ya que estos se pueden notar en los diagramas detallados del sistema (presentados en el anexo A), y su propósito ser entendido mediante el estudio de la documentación del ST-NIC y del 80C31, cuya lectura previa y consulta simultánea (especialmente de los diagramas de estado y tiempo del ST-NIC) es absolutamente necesaria para la comprensión de la implementación de este proyecto.

Hay que enfatizar también que muchas partes de la implementación de esta interface y de nuestro sistema en general han sido omitidas de este trabajo debido a que son explícitamente diagramadas y ampliamente discutidas en la documentación de los fabricantes de los dispositivos utilizados. En estos casos, nos limitamos a utilizar el diseño sugerido/requerido por el fabricante, y delegamos la explicación de los detalles del mismo a la documentación ya mencionada. El ejemplo más claro de esto, como ya

hemos dicho, es la sección analógica de la interface de red, la cual se basa en un diseño genérico utilizado por todos los fabricantes de equipos de red basados en el DP83902A y sus equivalentes.

Nuestro siguiente subtema tratará sobre las interfaces de conectividad hacia el resto de periféricos del sistema que estamos diseñando. Ya que estas interfaces son mucho menos complicadas en diseño e implementación que la interface de red, su estudio ha sido integrado en un solo subtema, al que pasamos a continuación.

2.2.3 Diseño de las interfaces de datos adicionales

Una vez estudiada la implementación de nuestra interface de red, debemos analizar como se implementaron las interfaces hacia los otros dispositivos con los que necesita interactuar nuestro sistema, los cuales son: la lectora de código de barras, el teclado numérico, el display de datos, y el portero eléctrico.

Como ya habíamos explicado, el teclado numérico y la lectora de código de barras usarán la misma interface. Debido a que estos dos dispositivos son los que el sistema utiliza para recoger la información de los usuarios, la llamaremos interface de ingreso de datos, y la analizaremos a continuación.

2.2.3.1 Interface de ingreso de datos del usuario

Tanto la lectora de código de barras como el teclado numérico que se utilizaron para este proyecto son dispositivos con interface compatible con el puerto de teclado de las computadoras personales tipo IBM AT. Debido a esto, para poder interactuar con estos dispositivos, nuestro sistema debe implementar una interface compatible con este estándar.

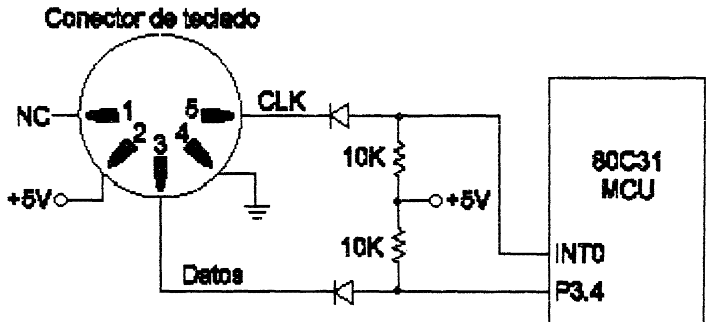


Figura 15: Interface de teclado del sistema

La interface de teclado IBM AT, es un puerto de comunicaciones seriales bidireccionales, basado en una señal de datos, y una señal de sincronía. Aparte de estas dos señales, la interface provee alimentación eléctrica para el dispositivo conectado, y tiene también una línea de inicialización. La mayoría de computadores personales utilizan circuitos integrados especializados para controlar sus teclados; sin embargo, se consideró que la adición de otro dispositivo para implementar una interface tan simple no era conveniente en nuestro caso, así que se decidió integrar la interface de ingreso de datos a los puertos de salida del microcontrolador 80C31.

En la Figura 15 tenemos el detalle de la interface utilizado para conectar el controlador 80C31 a los dispositivos de ingreso de datos. Como podemos ver, la interface es bastante simple, compuesto únicamente de dos líneas de entrada directa al 80C31 que se conectan al puerto DIN, el cual es el conector estándar para dispositivos tipo teclado AT. Los componentes discretos que podemos observar en el diagrama son necesarios ya que el estándar de las interfaces de teclado AT exige que las líneas de comunicación de la interface sean de colector abierto, para poder hacer bidireccional el flujo de datos. En nuestro caso no necesitamos flujos bidireccionales, pero dado que los dispositivos conectados a esta interface van a tener líneas de colector abierto (ya que son implementados de acuerdo al estándar), necesitamos los resistores y diodos para fijar el estado lógico de las líneas de la interface.

Hay que destacar en este momento, que todas las tareas de recolección de datos en esta interface son realizadas mediante software, el cual será analizado en los capítulos siguientes. Para hacer posible la operación de las rutinas de control de esta interface, este está conectado a una de las líneas de interrupción externa del 80C31. Entonces, entre esta interface, y la interface de red ya analizado anteriormente, hemos utilizado ya las dos líneas de interrupciones externas del 80C31.

Analicemos ahora la interface que utilizamos para controlar el display de datos conectado a nuestro sistema.

2.2.3.2 Interface de control del display

Antes de describir el diseño de una interface de display, se debe tomar la decisión de que display utilizar, ya que las características del mismo no están definidas por los objetivos y especificaciones del proyecto.

Lo que necesitamos que nuestro display haga, es ayudar al usuario en la operación de nuestro sistema mediante el envío de mensajes de status y reportes de errores. Tomando en cuenta estas necesidades, decidimos utilizar un display alfanumérico de 2 líneas de 20 caracteres cada una. El display que obtuvimos, es de tipo fluorescente, y es un dispositivo inteligente con una interface para conexión a buses de microprocesadores.

Estas características lo hacen muy adecuado para nuestras necesidades, ya que la tarea de conectarlo se vuelve muy sencilla. Tan solo debemos integrarlo como un periférico más de nuestro bus de datos.

Otra ventaja que tiene el tipo de display que vamos a utilizar, es que su diseño es muy estandarizado, existiendo muchos fabricantes que hacen displays con interfaces y programación idénticas, lo cual facilitará la obtención de equipos de similares características.

Nuestro display puede operar con buses de ocho o cuatro bits, y ser utilizado cor interfaces de bus tipo Intel (con señales de control WR y RD). El display tiene ademá

una señal de selección de registro (RS) para efectuar su programación. Analicemos ahora la interface que implementamos para conectar nuestro sistema al display.

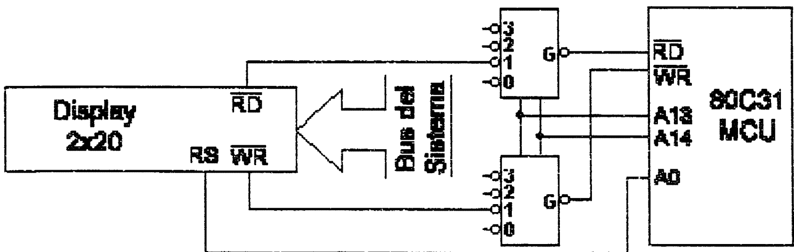


Figura 16: Interface de control del display

En la Figura 16 podemos apreciar la interface utilizada para interactuar con el display. Podemos notar que está completamente basado en la interface estándar para periféricos direccionados de nuestra unidad central de proceso. Para poder controlar la línea RS del display, utilizamos la línea menos significativa del bus de direcciones del sistema. De esta manera, podemos diferenciar entre comandos enviados al display, y datos enviados al display simplemente haciendo par o impar la dirección a la que enviemos los datos.

Para finalizar, debemos analizar la interface usado para aquel periférico que es el motivo de la creación de este sistema: El portero eléctrico.

2.2.3.3 Interface al portero eléctrico

Un portero eléctrico no es más que un dispositivo electromecánico diseñado para abrir una puerta cuando se le envía un impulso eléctrico. Este impulso eléctrico puede ser de diferentes voltajes, aunque en el equipo utilizado para esta tesis, el voltaje será de 110

voltios nominales de corriente alterna. Dado que nuestro equipo debe poder interactuar con una amplia gama de tipos de portero eléctrico, se consideró que la interface más eficiente sería el uso de un actuador o relay, el cual sería controlado por una línea de salida de nuestro microcontrolador.

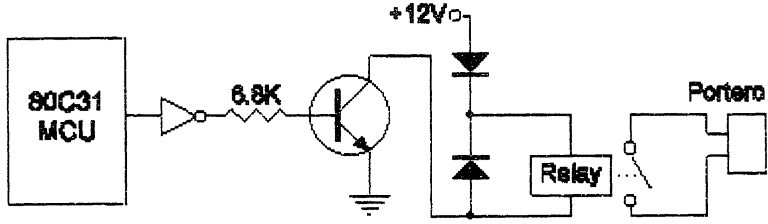


Figura 17: Esquema del interface hacia el portero eléctrico

Podemos ver en la figura la forma en que se ha implementado la interface. La línea de salida del 80C31 llega a un inversor, el cual es utilizado por dos razones: Primero, porque la línea del 80C31 no puede generar suficiente corriente para manejar la base del transistor; y segundo, porque las líneas del 80C31, tienen como estado nominal al inicio el valor de 1, lo que haría que nuestro relay se active al encenderse el aparato. El transistor se usa para aislar la parte digital del circuito de la parte que maneja el relay, ya que esta última trabaja a 12V. Notamos también el uso de diodos, los cuales absorben los picos de corriente generados por las transiciones de voltaje en el relay.

Hay que destacar, que esta parte del circuito ha sido implementada con un relay de baja potencia, el cual posiblemente no resulte adecuado para uso prolongado, pero es suficiente para la función de prototipo que cumple nuestro sistema. La implementación de un relay de mejores características no requiere cambios importantes en la interface,

siempre y cuando su voltaje de activación sea parecido al del relay utilizado por este trabajo (5V). El único cambio que probablemente se tenga que hacer, es modificar el valor de la resistencia de base del transistor, para compensar el cambio de resistencia del relay (si es que lo hubiera). Obviamente, se tendría que modificar el formato de la placa de circuito impreso para acomodar un relay más voluminoso, pero como veremos en el siguiente capítulo, esta tarea se puede realizar muy fácilmente, si las dimensiones del relay no son excesivamente grandes.

Con el análisis del circuito de interface hacia el portero eléctrico, concluimos nuestro estudio del diseño del hardware del proyecto. En el siguiente capítulo estudiaremos la implementación y construcción del mismo.

CAPITULO III

3 Implementación y construcción del Hardware

3.1 Integración de bloques conceptuales del proyecto

Una vez concluido el estudio de nuestro diseño, debemos analizar lo que se hizo para convertir este diseño en una implementación válida. Como hemos mencionado ya varias veces, hay muchas partes de nuestro proyecto cuya operación se basa en un diseño predeterminado, el cual no se presta a modificaciones, o estas simplemente no han sido necesarias para nuestro proyecto. Ejemplos de esto son: La parte analógica de la interface de red, la estructura de la unidad central de proceso, la interface de ingreso de datos. Todos estos componentes han sido simplemente implementados basándose en guías encontradas en la documentación de los dispositivos en los que están basados, y a guías de implementación publicadas por terceros. Debido a esto, recomendamos consultar la bibliografía de nuestro trabajo para mayores detalles sobre estas implementaciones.

Las descripciones de nuestro diseño que hemos presentado hasta este momento solo nos muestran los detalles relativos a las funciones que el equipo debe realizar, por lo que aquí debemos tratar de cubrir otros temas necesarios para la comprensión del funcionamiento del proyecto: Estos temas son:

- Selección de dispositivos de sincronización (Osciladores)
- Análisis de requerimientos de potencia del sistema

- Minimización de componentes del sistema

3.1.1 Selección de dispositivos de sincronización.

Se entiende como un dispositivo de sincronización a aquel que provee al sistema de señales de reloj, las cuales el sistema utiliza para sincronizar sus diferentes subsistemas lógicos. Para realizar esta tarea de selección debemos primero conocer los requerimientos de sincronización de los diferentes dispositivos utilizados en nuestro sistema.

La parte más importante de nuestro proyecto, la unidad central de proceso, está basada en el dispositivo 80C31 de Intel. El controlador específico que usamos en nuestro proyecto utiliza un diseño estático, el cual puede trabajar en frecuencias desde 0 hasta 16 Mhz (Cuando decimos que trabaja a 0 Mhz, significa que el controlador puede mantener su estado en la ausencia de pulsos de sincronización). Dados los altos requerimientos del tipo de software que vamos a utilizar en nuestro proyecto, sería conveniente utilizar la velocidad más alta soportada por nuestro procesador. Desde otro punto de vista en cambio, la naturaleza experimental de nuestro proyecto, y la forma en que las sensibilidades al ruido y la emisión de interferencia aumentan con la frecuencia, harían deseable usar la frecuencia más baja posible a la que pueda trabajar nuestro proyecto.

Cuando se estaba en las etapas iniciales del proyecto, y se estaba explorando el funcionamiento de periféricos como el display fluorescente y la interface de teclado, se

escogió una velocidad de reloj de 1.834 Mhz, una velocidad muy popular en sistemas integrados, ya que permite al puerto serial del 80C31 generar un amplio rango de velocidades estándares. Sin embargo, esta frecuencia pronto se demostró inutilizable, ya que como veremos cuando lleguemos al estudio del software del proyecto, las rutinas de interrupción utilizadas para el control de la interface de ingreso de datos son muy sensibles a la velocidad del sistema, ya que se basan en interrupciones sucesivas, las cuales en sistemas muy lentos tienden a superponerse, lo cual es obviamente muy indeseable. Se descubrió que la mínima velocidad de reloj que se podía usar para que las rutinas de la interface de teclado trabajen correctamente es de alrededor de 8 Mhz (a velocidades menores, el controlador no podía manejar el flujo de datos entrante). Se podría pensar entonces, teniendo en cuenta la sensibilidad de la interface de teclado a la velocidad del sistema, en subir su velocidad a la máxima soportada por nuestro controlador, 16Mhz; sin embargo, otra consideración se interpuso en nuestro camino; esta consideración es el display fluorescente. De acuerdo a la hoja de datos provista por el fabricante, nuestro display puede ser accesado por buses que trabajen a una velocidad máxima de 10Mhz. Esto puso límites superiores e inferiores muy estrechos a la elección de velocidad de reloj de nuestro sistema. Sin embargo, pudimos romper esta limitación mediante la experimentación. Nuestras pruebas (y el posterior desarrollo y experimentación del sistema) demostraron que el modelo de display que utilizamos trabaja correctamente a velocidades mucho mayores que las indicadas por el fabricante. Se hicieron pruebas a velocidades de hasta 15Mhz, sin muestras de fallos en la operación del display. La razón para la discrepancia parece estar, en nuestra opinión, en que el display que utilizamos está diseñado para compatibilidad con una muy popular

serie de displays LCD, los cuales no pueden operar con buses de frecuencia mayor a 10Mhz. Para enfatizar la compatibilidad con estos equipos, el fabricante aparentemente decidió indicar prestaciones del dispositivo que son mas bajas de lo que realmente puede manejar en la realidad.

De todas formas, para precautelar contra posibles consecuencias imprevistas o no detectadas por nuestras pruebas, decidimos fijar la velocidad de reloj de nuestra unidad central de proceso a 12 Mhz. Sin embargo, es posible experimentar con frecuencias mayores, ya que el oscilador que utilizamos en nuestro proyecto tiene una salida adicional a 15Mhz.

Habiendo definido la frecuencia a la que va a trabajar nuestra unidad central de proceso, podríamos pensar que todo el sistema va a trabajar a la misma frecuencia. Esto no es así, ya que la parte analógica del ST-NIC debe obligadamente trabajar a una frecuencia de 20Mhz, ya que tiene que generar flujos de datos a 10 Mbps en sus salidas Ethernet. Debido a esto, tuvimos que utilizar un oscilador separado para nuestra interface de red. Hubiera sido posible hacer correr la parte digital del ST-NIC a la velocidad de la unidad central de proceso, ya que este dispositivo tiene una entrada de reloj independiente para controlar este subsistema. Sin embargo, ya que todas las comunicaciones entre la unidad central de proceso y la interface de red son asincrónicas, permitimos que esta última operara a 20 Mhz. No se han detectado consecuencias adversas de esta decisión.

3.1.2 Análisis de requerimientos de potencia del sistema.

Cuando nos referimos a requerimientos de potencia de nuestro sistema, no solo hablamos de los voltajes y corrientes que necesitan sus componentes para funcionar, sino de todos los fenómenos que influyen sobre la alimentación de poder a nuestro proyecto, y por lo tanto sobre su correcto funcionamiento. Lo primero que hay que considerar, en efecto, es el valor de la alimentación eléctrica que debemos utilizar. Muchos de los componentes de nuestro sistema, incluyendo al ST-NIC y a los componentes TTL, requieren alimentación de 5V, por lo que toda la parte digital de nuestro circuito funciona a 5V. Sin embargo, la salida AUI de nuestra interface de red requiere, por especificaciones del estándar, una salida de poder de 12V. Además, el circuito de alimentación del relay en nuestra interface con el portero eléctrico funciona también a 12V. Esto último puede quizás ser cambiado, buscando un relay que pueda accionarse a voltajes mas bajos, o usando optoacopladores, pero por el momento, y debido a que la salida AUI ya requería de potencia de 12V, decidimos utilizarla también en nuestro relay.

Otro punto muy importante concerniente a la alimentación de poder a nuestro sistema es el uso de capacitores de acoplamiento, el cual se vuelve muy crítico en un circuito que utiliza componentes de alta velocidad de conmutación. Para evitar problemas de introducción de señales falsas, y toda la serie de fallas que se producen debido a oscilaciones de la alimentación, hemos provisto a nuestro circuito de capacitores de acoplamiento para cada circuito integrado, y en el caso del ST-NIC, ya que así lo

recomienda el fabricante, se usó cuatro capacitores de acoplamiento. La presencia de capacitores de acoplamiento en si no es suficiente, sino que hay que proveer de una fuente para que estos capacitores puedan reponer la carga que pierden al realizar su labor; para esto se han localizado una serie de capacitores de mayor capacidad en sitios estratégicos de nuestro circuito, como se puede ver en los diagramas del mismo.

Como veremos cuando analicemos la construcción de nuestra placa de circuito, todavía hay mucho por hacer para mejorar las condiciones de alimentación eléctrica de nuestro circuito, pero dadas las escasas facilidades disponibles y nuestra poca experiencia en esa área, se ha tenido que utilizar un esquema poco óptimo, pero que es probablemente adecuado para nuestro prototipo. Cuando decimos que el esquema no es optimizado, nos referimos al hecho de que realmente no se ha hecho ningún tipo de estudio detallado sobre la implementación de la red de alimentación del equipo, tomando en cuenta análisis de ruido, emisiones electromagnéticas, corrientes parásitas, etc. Este tipo de estudio es muy importante para el desarrollo de un producto comercial, pero puede ser obviado para un prototipo.

3.1.3 Minimización de componentes del sistema

Cuando analizamos el diseño de las interfaces de nuestros periféricos y nuestra unidad central de proceso, notamos que ha sido necesario el uso de varios componentes de lógica digital, como inversores, puertas AND, OR y decodificadores. Dada la necesidad de simplificar el diseño de nuestro sistema, se consideró desde el principio del proyecto

la integración de esta lógica en circuitos de *lógica programada* o PALs (Programable Array Logic). La ventaja del uso de PALs radica en que se puede integrar en un solo circuito integrado una serie de funciones lógicas que usando lógica discreta necesitarían el uso de varios dispositivos, lo cual complica bastante el diseño de la placa de circuito impreso. Por ejemplo, nuestro circuito integra en dos PALs una funcionalidad lógica que requeriría cuatro dispositivos discretos de la familia TTL. Hay que destacar que el uso de PALs no es necesariamente mas barato en cuestión de materiales que el uso de componentes discretos, pero las facilidades que ofrecen en las áreas de diseño de circuito, tamaño del producto final, y facilidad de manufactura, las hacen una alternativa atractiva.

Una vez estudiados estos temas, podemos pasar a revisar la etapa de diseño de nuestra placa de circuito impreso.

3.2 Diseño de la placa de circuito impreso

Uno de los principales retos de este proyecto, es conseguir que el tamaño del producto final del mismo sea lo más reducido posible. Para conseguir este propósito hemos hecho numerosas optimizaciones, algunas de las cuales ya hemos revisado. Pero como veremos a continuación, la etapa del proyecto en donde se pueden hacer las mejores optimizaciones de tamaño es en el diseño de la placa de circuito impreso. Y veremos también que el tamaño no es la única variable del proyecto que depende de un buen diseño de la placa. Toda la operación del circuito depende de este, y algo más

importante aún, la facilidad de manufactura del aparato depende totalmente de la manera en que se ha hecho este diseño; entonces podríamos tener un perfecto ejemplo de diseño digital, que no sirve de nada porque no se lo puede fabricar a un costo reducido.

Habiendo notado la importancia de esta parte del diseño de nuestro proyecto, pasemos a revisarla en detalle.

3.2.1 Automatización del diseño

Para permitir un diseño acelerado de una placa de circuito impreso, existen hoy en día una serie de herramientas computacionales integradas, las cuales ofrecen un alto grado de automatización de los procesos de diseño. Para el desarrollo de este proyecto utilizamos el sistema de desarrollo de circuitos impresos de la compañía Protel. Este sistema se compone de varias herramientas de software. La primera etapa de un diseño de este tipo es la digitalización del diagrama esquemático del proyecto. Para realizar esta tarea se utilizó el programa Protel Advanced Schematics. En esta aplicación se dibuja el diagrama del circuito, con todas las conexiones que este va a tener; además se especifica el tipo de componentes que se va a utilizar. A partir de esta información, este programa genera lo que se conoce como una *tabla de conexiones (netlist)* la cual es utilizada por la siguiente etapa del proceso de automatización de diseño. Esta tabla contiene toda la información necesaria de conexiones, número y tipo de elementos utilizados, etc.

La siguiente etapa en el proceso de automatización de diseño es el desarrollo de la platina de circuito impreso en sí. La tabla de conexiones es cargada en el programa Protel Advanced PCB, el cual la utiliza para dibujar los componentes que se van a incluir en el diseño, y para establecer las conexiones que existen entre ellos. El trabajo, sin embargo, apenas empieza. Los componentes deben ser organizados de la manera más óptima posible, para facilitar su interconexión. Una vez realizada esta tarea, se debe empezar el diseño real de la circuitería de cobre. El éxito o fracaso de esta tarea depende de la eficacia con la que se haya realizado la labor de colocación de los elementos. Hay que destacar que el software utilizado ofrece métodos automáticos para las tareas de colocación, e incluso para el ruteo de conexiones. Sin embargo, la calidad del acabado dado por estas herramientas no se compara con lo que se puede lograr haciendo el diseño de manera manual. Esto no quiere decir que los algoritmos automáticos no tienen uso, son muy útiles si se los usa como pasos previos, para dar ideas al diseñador y para detectar fallas en la forma en que el diseñador ha realizado su trabajo. Por ejemplo, una manera efectiva de comprobar que el trabajo de colocación de partes ha sido realizado correctamente, es hacer una iteración de autoruteo de conexiones, la cual será exitosa o no de acuerdo a la forma en que se haya hecho la colocación de partes. De esta manera el diseñador puede rápidamente hacer varios borradores de su diseño, hasta llegar a una forma óptima.

3.2.2 Criterios de diseño

Como ya hemos mencionado en varias ocasiones, uno de los objetivos de nuestro proyecto, es hacer una implementación compacta del mismo. Este es básicamente el principal criterio de diseño que hemos aplicado al trabajo de creación de la placa de circuito impreso, más no es el único. Podemos resumir rápidamente los criterios utilizados:

- Minimizar el tamaño y volumen del producto final del proyecto.
- Eliminar o disminuir potenciales problemas de fabricación posterior.
- Minimizar el costo de producción del dispositivo.

Para minimizar el tamaño de nuestro proyecto hemos usado distintos métodos, varios de los cuales ya han sido explicados. En lo que concierne al diseño de la placa de circuito impreso, nuestro esfuerzo de minimización solo puede ir en dos direcciones: disminución del tamaño de los componentes, y aumento de la densidad de componentes en la platina. Inicialmente consideramos la utilización de componentes montados en superficie para nuestro diseño, y aunque su correcto uso puede proporcionar grandes ahorros de espacio, estos traen problemas de fabricación y logística. De los elementos de nuestro circuito solo dos, el ST-NIC y el 80C31, son montados en superficie (SMDs). El resto de los elementos son convencionales. En cuanto a la densidad de elementos en el proyecto, se temía que la decisión de utilizar componentes normales nos obligaría a aumentar significativamente el área del circuito, pero esto finalmente no fue así, ya que pudimos optimizar el diseño de manera que el área del mismo no se

extienda. Hay que destacar que el campo de minimización de tamaño es el área en que nuestro proyecto se presta a mayores mejoras.

Nuestras proyecciones indican que el uso de tecnologías modernas de fabricación de platinas y componentes montados en superficie, además de un diseño profesional, podrían reducir el tamaño de este proyecto a la mitad.

En el área de factibilidad de producción del diseño, fue donde se nos presentaron algunos de los problemas más graves de nuestro proyecto. La tecnología de fabricación de circuitos impresos en nuestro país es primitiva, con un atraso que calculamos en 20 años respecto a la media mundial. Las herramientas de diseño que utilizamos son, en cambio, bastante modernas, por lo que nuestra obvia tendencia fue realizar diseños modernos que no se podían fabricar en nuestro país. Mediante simplificaciones del diseño, pudimos llevarlo a un nivel en que fue posible fabricarlo en el Ecuador y mantener un tamaño de la placa aceptable. Sin embargo, otros factores han hecho que el diseño de placa resultante sea simplemente inadecuado para producción en masa (Si se usa placas de fabricación nacional). La razón de esto es que para poder obtener una densidad de componentes aceptable, se requiere la utilización de al menos dos capas interconectadas de cobre, usando agujeros platinados. Desgraciadamente, ninguno de los fabricantes de placas de circuitos del Ecuador puede hacer agujeros platinados, y un diseño que no los utilice sería excesivamente grande, por lo que se decidió usar un diseño que requiere agujeros platinados, en una platina que no los implementa. El resultado es que todas las interconexiones entre las dos capas de cobre tienen que

hacerse manualmente, lo cual hace el diseño impráctico para manufactura, aunque para la construcción de un prototipo este sí es adecuado. Obviamente este problema sería resuelto por el uso de técnicas de fabricación que implementen agujeros platinados, pero eso nos lleva a nuestro siguiente criterio de diseño, el cual es la minimización de costos. Dado que un diseño como el nuestro no se puede implementar en su totalidad con la tecnología existente aquí, se plantea la cuestión de enviarlo a fabricar al exterior. Averiguaciones realizadas indican que si es posible la construcción de nuestro diseño en Colombia o Venezuela, pero que esto solo sería factible en cantidades grandes, dados los costos de construcción y de envío. La decisión de que sería mas conveniente dependería entonces de la cantidad de unidades que se desee implementar, ya que si esta es pequeña, probablemente sería más conveniente (tomando en cuenta costos de fabricación de la platina y mano de obra de colocación de los componentes) usar platinas de fabricación nacional.

Como resumen podemos decir que el diseño de placa de circuito que hemos realizado puede fabricarse en masa sin mayores problemas, si las placas de circuitos son fabricadas con agujeros platinados. Cabe destacar que la utilización o no-utilización de agujeros platinados no modifica en nada el diseño de la circuitería de cobre de la placa, lo único que cambia es el proceso de fabricación utilizado para la placa de circuito impreso.

Habiendo analizado estos criterios, revisemos el diseño que utilizamos para la platina de circuito de nuestro prototipo.

3.2.3 Implementación de la platina de circuito impreso

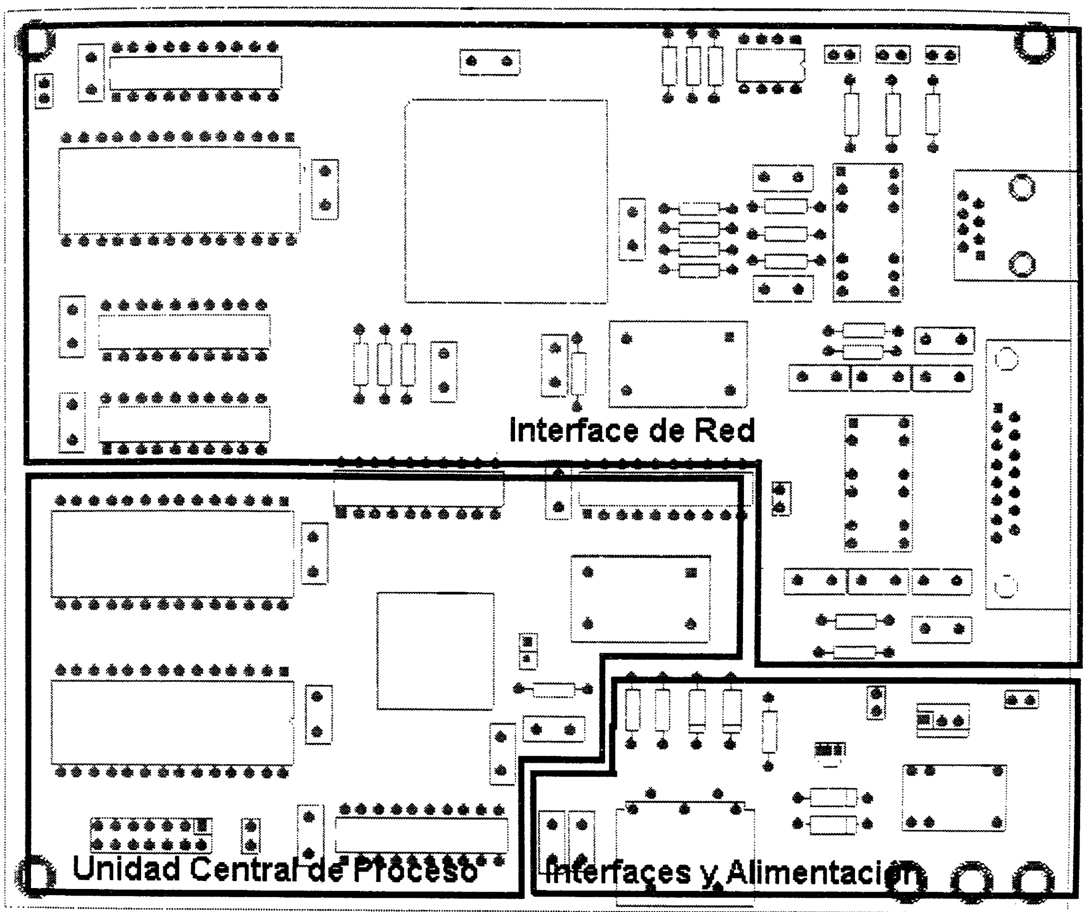


Figura 18: Colocación de elementos en la placa de circuito impreso

En la Figura 18 se muestra la forma en que están colocados los elementos de nuestro circuito en la placa de circuito impreso[2]. Los dispositivos más importantes del diseño han sido identificados para referencia. Quisiéramos decir que el esquema de colocación

---

[2] Los diagramas completos de colocación y ruteo de componentes del proyecto se encuentran detallados en el anexo B.

mostrado es producto de una metodología lógica de diseño, pero esto no es así. Aunque el programa utilizado tiene una utilidad de autocolocación de componentes, esta fue de poco uso (lo cual es común en este tipo de ayudas automatizadas), y la colocación se hizo básicamente mediante prueba y error, observando las conexiones resultantes y asumiendo la colocación que mejor se prestará para las tareas restantes del diseño.

El ruteo de las conexiones del circuito se realizó manualmente, aunque como ya indicamos, se hicieron varias iteraciones previas de autoruteo, para tratar de dilucidar la forma más eficiente de realizar la labor de enlazar los diferentes componentes del circuito.

Podemos observar en el diagrama, que alrededor de un 60% del espacio de la platina es utilizado por la interface de red, y el restante espacio se divide entre la unidad central de proceso y el área dedicada a las interfaces de ingreso de datos y portero eléctrico. En la figura es posible ver con más claridad lo que se puede hacer en el área de minimización del diseño, ya que los componentes que más espacio utilizan, como el ST-NIC, las memorias RAM y los registros, pueden ser implementados en versiones SMD de tamaño bastante inferior al utilizado en este prototipo. Además, el uso de una sola interface de acceso al medio físico en la red sería también un ahorro significativo a la hora de diseñar un modelo de producción de nuestro proyecto. Hay que destacar aquí, que dado que nuestro proyecto es un prototipo, se escogió no implementar circuitería propia de alimentación y regulación de corriente; pero si se piensa en la creación de un

diseño para producción en masa, sería bastante deseable implementar las funciones de regulación y quizás hasta de rectificación de voltaje en la propia platina.

Pasemos ahora a revisar la forma en que se implementó la lógica programable que utilizamos para efectuar las funciones lógicas requeridas por nuestro diseño.

3.3 Diseño de las unidades de lógica programable

3.3.1 Herramientas de diseño

En la actualidad existen una gran cantidad de formas de lógica programable, desde simples reemplazos para unas cuantas puertas discretas, hasta enormes dispositivos reprogramables capaces de ejecutar una infinita diversidad de funciones. Desde un principio, el diseño de circuitos de lógica programable ha estado estrechamente ligado al diseño asistido por computadora, tanto es así que la programación de los más modernos sistemas de ASICs y FPGAs requieren el uso de lenguajes dedicados para su diseño y programación. Hay que destacar que esta es un área del diseño electrónico donde nuestro país se encuentra en un total atraso.

Para nuestro simple propósito de implementar un reemplazo para los distintos componentes lógicos discretos de nuestro sistema, no es estrictamente necesario el uso de herramientas complejas de diseño, ya que dispositivos simples como las PALs que estamos utilizando se programan a base de una matriz de interconexiones, que el usuario puede utilizar para definir la forma en que su diseño responda a estímulos en sus

entradas. Sin embargo, debido a la naturaleza de la tecnología que se usa para programar estos dispositivos, el uso de herramientas computacionales es necesario, ya que los equipos utilizados para programar PALs requieren que se utilice un formato estándar bastante complejo (conocido como formato JEDEC) para ingresar la información de programación al equipo. Dadas estas circunstancias, normalmente la programación de PALs se realiza usando una herramienta de software llamada *compilador JEDEC*, que es un programa que se encarga de convertir las ecuaciones lógicas ingresadas por el usuario en las directivas que el equipo de programación requiere para configurar el dispositivo PAL. Afortunadamente, a diferencia de las otras herramientas de software utilizadas en este proyecto, un compilador JEDEC no es difícil de conseguir. La mayoría de fabricantes de dispositivos programables ofrecen gratuitamente compiladores compatibles con sus dispositivos. En nuestro caso utilizamos un compilador llamado CUPL, el cual es una herramienta de programación de PALs, GALs, PROMs, y muchos otros dispositivos adicionales. CUPL tiene la ventaja adicional de ser una aplicación de Windows, lo que facilita y flexibiliza su utilización en los computadores usados en la actualidad.

Habiendo ya conocido la herramienta que utilizamos para la implementación de la lógica programable a usarse en nuestro proyecto, pasaremos a revisar la implementación en sí de los circuitos integrados a partir de las unidades de lógica discreta usadas en nuestro proyecto.

3.3.2 Diseño e implementación

El diseño de unidades de lógica programable depende de varios factores importantes, los cuales definen el tipo de tecnología que se va a usar (PROM, PAL, GAL, FPGA, etc.), así como el número de dispositivos de lógica programable a implementar. El primer factor es la complejidad del sistema digital a implementarse, específicamente el número de entradas y salidas del mismo. Si estos valores son pequeños, lo mejor es implementar el diseño usando tecnologías simples de lógica programable, como por ejemplo PROMs, PALs o GALs; en cambio, si el número en cuestión es grande (100 líneas, por ejemplo) lo más efectivo sería la utilización de tecnologías de alta densidad como FPGA.

Otro factor importante es el tipo de lógica a implementar, si es una simple tabla de verdad, lo más adecuado es usar PROMs, si es lógica combinatorial o secuencial, y relativamente simple, probablemente se pueda usar PALs o GALs; si la interacción entre las unidades lógicas es compleja, probablemente se requerirá el uso de lógica de alta densidad.

En nuestro caso, tenemos una serie de unidades lógicas combinatoriales simples con un número pequeño de entradas y salidas, así que lo más adecuado es el uso de PALs. Una vez definida esta cuestión, pasamos a la siguiente etapa del proceso de diseño, que es definir el número y tipo de dispositivos a utilizarse. Aquí entra un poco en juego el hecho de que las PALs son dispositivos poco utilizados en nuestro medio, así

básicamente se tuvo que utilizar lo que hay disponible en el mercado, que en este caso es el dispositivo PAL16L8, el cual es una unidad de lógica AND-OR con ocho salidas combinatoriales, y 12 posibles entradas. [3]

Ya que la PAL16L8 es el dispositivo disponible, tenemos simplemente que decidir cuantas de ellas debemos utilizar. Para esto necesitamos definir el número de entradas y salidas necesarias para nuestra lógica programable. Estos datos los detallamos a continuación:

ENTRADAS:	SALIDAS: (15)
ADD15.H	$OE1.L = ADD15.H * \overline{ADD14.H} * \overline{RD.L}$
ADD14.H	$OE2.L = \overline{PRD.L} + (\overline{ACK.L} * \overline{Sr/w.H})$
PRD.L	$CLK1.H = \overline{PWR.L} + (\overline{CS.L} * Sr/w.H)$
PWR.L	$CLK2.H = ADD15.H * \overline{ADD15.H} * \overline{WR.L}$
ACK.L	$SRD.L = \overline{CS.L} * Sr/w.H$
CS.L	$SWR.L = \overline{CS.L} * \overline{Sr/w.H}$
RD.L	$WRlcd.L = \overline{ADD15.H} * ADD14.H * \overline{WR.L}$
WR.L	$RDlcd.L = \overline{ADD15.H} * ADD14.H * \overline{RD.L}$
Sr/w.H	$WRram.L = \overline{ADD15.H} * \overline{ADD14.H} * \overline{WR.L}$
INT0.H	$RDram.L = \overline{ADD15.H} * \overline{ADD14.H} * \overline{RD.L}$
RELAY.H	$WACK.L = ADD15.H * \overline{ADD14.H} * \overline{WR.L} * CS.L$
RESET.H	$RACK.L = ADD15.H * \overline{ADD14.H} * \overline{RD.L} * CS.L$
	$INT0.L = \overline{INT0.H}$
	$RELAY.L = \overline{RELAY.H}$
	$RESET.L = \overline{RESET.H}$

Tabla 1: Entradas y salidas de la lógica programable

[3] Para mayores detalles sobre la organización, características y uso del PAL16L8, consultar el anexo F.

La Tabla 1 detalla las entradas y salidas de todos los componentes lógicos necesarios para el funcionamiento de nuestro sistema. Hay que destacar que en esta tabla están incluidos todos los elementos lógicos no secuenciales que utilizamos en nuestro sistema, tanto en la unidad central de proceso como en los distintos periféricos de la misma; por ejemplo, toda la lógica de direccionamiento de periféricos del bus de datos de la unidad central de proceso está implementada aquí, aunque en nuestros diagramas la mostramos como una serie de decodificadores.

El dato más importante para definir el número de dispositivos que vamos a utilizar es el número de salidas de nuestro sistema; ya que dijimos que la PAL16L8 es un dispositivo de 8 salidas, es claro que vamos a necesitar dos de estos para implementar las quince salidas que tiene nuestro sistema lógico. El número de entradas no representa un problema, ya que es bastante menor al número de entradas disponibles en un sistema de dos PAL16L8 con 15 salidas.

Una vez definido el número de dispositivos a utilizarse, se debe organizar la distribución de entradas y salidas a cada dispositivo. En sistemas con un alto número de entradas, relacionadas a su vez con todas o la mayoría de las salidas, podría haber problemas en la asignación de entradas, ya que por ejemplo podríamos tener un dispositivo con ocho salidas asignadas, las cuales requieren 14 entradas para trabajar correctamente. Dado que la PAL16L8 tiene solo 10 entradas independientes de las salidas, esta combinación de entradas y salidas no podría implementarse. Afortunadamente, nuestro sistema tiene una serie de entradas y salidas poco

relacionadas entre sí, así que no tuvimos que enfrentar este tipo de problemas, y pudimos hacer nuestra asignación de entradas y salidas a nuestra mejor conveniencia. Hay que destacar aquí, que todas las labores que hemos detallado hasta el momento pueden ser realizadas automáticamente por el compilador CUPL, en el cual simplemente hubiéramos tenido que ingresar las entradas y salidas detalladas arriba, y las ecuaciones correspondientes a las salidas, y el compilador se hubiera encargado de asignar entradas y salidas, y usar el número de dispositivos que sean necesarios.

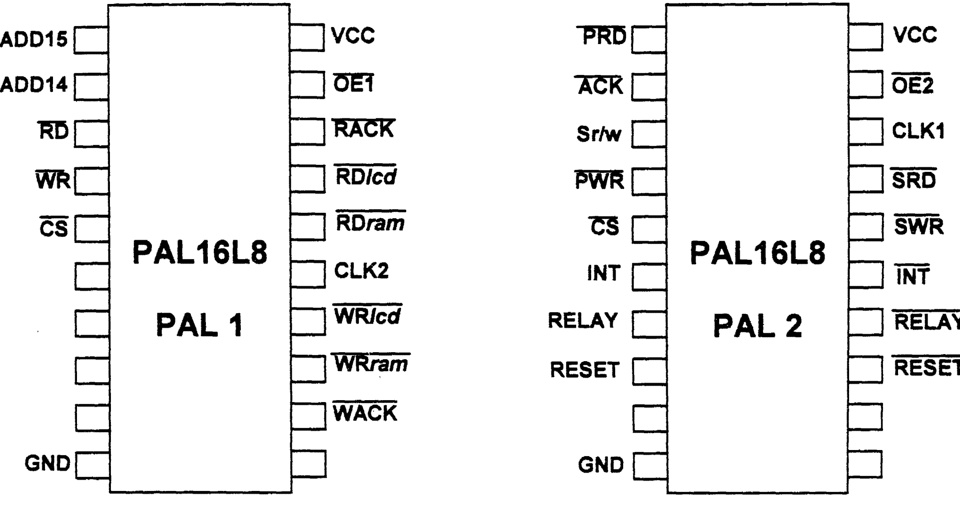


Figura 19: Configuración de la lógica programable

La Figura 19 muestra la organización que se utilizó para nuestra lógica programable. Podemos notar que en ambos dispositivos hay entradas sin utilizar, ya que como dijimos anteriormente, la variable determinante en este caso es el número de salidas de nuestro sistema lógico. Hay que destacar que en el dispositivo PAL 2 se ha implementado tres inversores; de estos, solo dos son absolutamente necesarios para nuestro diseño, el inversor de la entrada INT, que es para invertir la señal de interrupción del ST-NIC (ya

que la lógica de interrupciones del 80C31 utiliza niveles lógicos bajos), y el inversor de la señal RELAY, que es la salida del 80C31 hacia el circuito de activación del portero eléctrico. Esta inversión es necesaria por las causas que ya mencionamos cuando se describió el interface hacia el portero eléctrico. La tercera señal invertida, RESET, fue añadida más tarde en el proceso de desarrollo del sistema, para lograr que el 80C31 controle la inicialización del ST-NIC y del display fluorescente.

Una vez definida la configuración que se va a utilizar para el diseño de los dispositivos de lógica programable, esta información se ingresa al programa CUPL, el cual genera a partir de estos datos el archivo JEDEC que se ingresa al equipo de programación de PALs. El formato utilizado para ingresar los datos en CUPL puede verse en los anexos, así como la salida generada por este programa, con la cual se puede directamente reproducir la configuración arriba descrita. Las facilidades utilizadas para programación de PALs fueron provistas por el laboratorio de sistemas digitales de la ESPOL.

Una vez descrito el proceso de implementación de la lógica programable de nuestro sistema, estamos listos para pasar a estudiar el proceso de manufactura de nuestro prototipo, que es el tema que revisamos a continuación.

3.4 Construcción final del hardware del proyecto

En esta sección analizaremos el proceso de construcción de nuestro prototipo. Nuestro estudio empezará con una descripción de los equipos y materiales necesarios para la

construcción del prototipo, así como una estimación de los costos de su construcción, tanto en lo que corresponde a materiales, como a mano de obra. Esta sección del trabajo es de suma importancia, ya que de sus resultados depende la factibilidad de la manufactura en masa de nuestro dispositivo.

3.4.1 Materiales y equipos utilizados

Como hemos apreciado en las páginas previas de nuestro trabajo, nuestro proyecto tiene requerimientos tecnológicos, que aunque moderados, son ciertamente problemáticos en nuestro medio. Por esto se podría pensar que los requerimientos de equipo necesarios para la construcción e implementación de nuestro proyecto son de elevado costo. Afortunadamente, esto no tiene que ser necesariamente cierto. Aunque para un proceso de construcción adecuado y científico de un trabajo como el nuestro si convendría el uso de herramientas modernas (y de elevado costo), la serie de compromisos que hemos hecho en nuestro diseño en todo lo que corresponde a facilidad de manufactura, inspección y diagnóstico, han tenido como resultado que, aunque el proceso de manufactura de nuestro prototipo se vuelve complejo, el costo de los equipos necesarios para realizarlo se pudo disminuir a niveles razonables.

En lo que respecta a la construcción de nuestro proyecto, el único equipo sofisticado que se considera necesario, es una estación de soldadura con temperatura controlada, debido a que tanto el ST-NIC, como el 80C31 utilizados son elementos sensibles a la temperatura y carga electrostática. El equipo restante que se considera necesario es lo

que se encontraría comúnmente en el maletín de herramientas de cualquier ingeniero o tecnólogo electrónico: un multímetro digital con medición de continuidad, herramientas de pelado y corte de cables, implementos simples y materiales de soldaduras, y herramientas varias como destornilladores, pinzas, etc. La parte más delicada del proceso de construcción del equipo es la colocación y soldadura de los elementos montados en superficie, y aunque no es estrictamente necesario, y no fue utilizado por nosotros debido a su costo relativamente alto, se recomienda el uso de implementos especiales para soldadura de SMDs si se piensa hacer implementaciones en masa de este diseño. Esto se debe a la facilidad con que se puede inutilizar una platina o componente SMD si se realiza incorrectamente el proceso de soldadura. No vamos a presentar un estudio de costo de equipos, porque pensamos que los equipos que se utilizaron en la construcción de este proyecto son de uso indispensable en cualquier taller electrónico, y no deben ser adquiridos únicamente para la construcción de este proyecto, con la posible excepción de herramientas para manejo de SMDs.

En el área de materiales, como ya habíamos mencionado, se hicieron esfuerzos iniciales por implementar la mayor parte de los componentes del circuito usando SMDs, pero por razones ya discutidas, esta iniciativa fue descartada. Debido a esto, con la excepción del ST-NIC y el 80C31, todos los elementos de nuestro proyecto son de fabricación y uso convencional. Hay que destacar que en el caso del ST-NIC, no hay más alternativa que implementarlo usando SMDs, ya que el número de conexiones de este integrado (84) haría impráctica su implementación en formato DIP, por lo que el fabricante no lo provee en ese tipo de paquete. El 80C31 si existe en implementaciones DIP de 40 pines,

pero se consideró que si ya se estaba implementando el ST-NIC como SMD, bien podríamos ahorrar espacio extra e implementar también el 80C31 de esa manera.

La Tabla 2 muestra la lista completa de materiales utilizada para la construcción de nuestro proyecto. Hay que enfatizar que no se encuentran listados los materiales subsidiarios como soldadura, cables de cobre varios, y otros materiales que se puede esperar que existan en un taller de fabricación electrónico. Algo muy importante que debemos tener en cuenta respecto a los materiales de este proyecto, es que debido a las funciones muy específicas que sus componentes realizarán, algunos de ellos serán difíciles o imposibles de obtener en el mercado local, en esta categoría están el ST-NIC, los osciladores y los filtros de red. El lector se preguntará como se consiguieron estos dispositivos en cantidades de muestra para la construcción de nuestro prototipo. La respuesta a esto, es que se obtuvo los componentes desoldándolos de tarjetas electrónicas viejas; esto no se aplica únicamente a los componentes difíciles de obtener, sino a prácticamente todos los componentes de nuestro circuito. Excepciones a esto fueron el ST-NIC, el 80C31, los conectores de red y el display fluorescente, los cuales fueron obtenidos en el extranjero. Obviamente, para fabricación en cantidades mayores, no se puede confiar en ese tipo de técnicas. Afortunadamente, si se los ordena a casas de venta de componentes electrónicas en el extranjero, ninguno de estos componentes, con la posible excepción del ST-NIC y los filtros de red, es difícil de conseguir. En cuanto a estos últimos, se los puede obtener directamente de los fabricantes, ya sea por orden telefónica, o a través del Internet.

Elemento	#	Costo Unitario (sucres)	Costo Total (sucres)	Paquete	Comentario
Intel 80C31-16	1	15000	15000	PLCC 44	Controlador Principal
National DP83902AV	1	90000	90000	PLCC 84	Controlador Interface de Red
27C512	1	22500	22500	DIP 28	EPROM 64Kbytes
6264-10	2	22500	45000	DIP 28	Memoria SRAM 8Kbytes 100ns
74LS373	2	3000	6000	DIP 20	Registro 8 bits
74LS374	2	3000	6000	DIP 20	Registro sincrónico 8 bits
TIBPAL16L8	2	25000	50000	DIP 20	Lógica programable
TCO -711A 20	1	15000	15000	DIP	Oscilador 20 Mhz
F5C-S7 12 -15	1	15000	15000	DIP	Oscilador 12/15 Mhz
YCL 20F001N	1	8000	8000	DIP	Aislamiento y filtro 10BaseT
YCL 16PT-005B	1	8000	8000	DIP	Aislamiento AUI
Relay 5V STSP	1	9000	9000	DIP	Relay portero eléctrico
DIP Switch 4	1	7500	7500	DIP 8	Switch cuádruple
Conector DB 15	1	5000	15000	DB15	Conector AUI
Conector RJ45	1	5000	7000	RJ45	Conector 10BaseT
Conector DIN 5	1	8500	8500	DIN	Conector teclado
Noritake CU20025ECPB-U1J	1	225000	225000	Externo	Display fluorescente 2 líneas x 20 columnas
Cap. 100nF 16V	24	250	6000	Cerámica	Capacitores acoplamiento
Cap. 1uF 16V	5	2600	13000	Tantalio	Capacitores fuente de carga
Res 4.7K ¼ W 20%	6	150	900	¼ W	Resistores pull-up varios
Res 10K ¼ W 20%	2	150	300	¼ W	Resistores pull-up teclado
Res. 330 ¼ W 20%	3	150	450	¼ W	Protección LEDs
Res. 39 ¼ W 5%	4	150	600	¼ W	Filtros AUI
Res. 6.7K 20%	1	150	150	¼ W	Transistor portero eléctrico
Res. 271 ¼ W 1%	2	500	1000	¼ W	Arreglo de señal 10BaseT
Res. 67 ¼ W 1%	2	500	1000	¼ W	Arreglo de señal 10BaseT
Res. 51 ¼ W 1%	2	500	1000	¼ W	Arreglo de señal 10BaseT
Res. 800 ¼ W 1%	1	500	500	¼ W	Arreglo de señal 10BaseT
Diodo 1N914	4	250	600		Diodos de conmutación pequeños
Transistor 2N2222	1	1300	1300	TO-92	Switch del Relay
LED Rojo	2	400	800		LEDs señalización de red
LED Verde	1	400	400		LED señalización de red
Cable plano 14 líneas	1	15000	15000		Conexión display
Conector tres líneas	1	5000	5000		Conector de alimentación
COSTO TOTAL:			**S/. 604550**		

Tabla 2: Lista de Materiales del Proyecto

Como podemos ver en la tabla de materiales, el costo calculado para materiales de nuestro proyecto es de unos seiscientos mil sucres. Para dar una apreciación más exacta del costo de componentes electrónicos, siempre es mejor dar el resultado en dólares,

para no tener problemas debido a las fluctuaciones de tipo de cambio. El costo de los componentes pasado a dólares al tipo de cambio que rige al momento de escribir este trabajo es de aproximadamente 133 dólares. Como podemos ver en la tabla, la mitad de este costo proviene del ST-NIC y del display fluorescente, los cuales por ser los componentes más especializados del proyecto, son también los más costosos.

3.4.2 Construcción del prototipo

Una vez establecida la lista de materiales que necesitamos para la construcción de nuestro proyecto, el siguiente paso es la obtención de los mismos, y la construcción del prototipo en sí. El proceso de construcción empieza con la fabricación de la placa de circuito impreso. Como ya habíamos mencionado, esta pieza normalmente se manda a manufacturar externamente. En nuestro caso, la construcción de la platina de circuito impreso fue encargada a la compañía CIRELEC, que es uno de los mejor equipados fabricantes de circuitos impresos de nuestro medio. Normalmente este tipo de trabajos se facturan de acuerdo al tipo de material utilizado para la platina (baquelita o fibra de vidrio), el tipo de acabado de la circuitería (simple o recubierto), y obviamente de acuerdo al tamaño de la platina en cuestión. Un prerrequisito para la fabricación de la platina es que el diseño de la misma se presente en un medio transparente, ya que el proceso de fabricación es en base a fotolitografía. Ya que quisimos eliminar cualquier tipo de problema referente a la platina, la hicimos construir de la mejor calidad posible, en fibra de vidrio y con líneas recubiertas. Dados estos requisitos, el costo de su construcción fue de ciento diez mil sucres (o unos 25 dólares). Como habíamos

nencionado antes, esta placa fue construida sin agujeros platinados, aunque nuestro diseño requiere su uso.

Una vez que tuvimos la platina de circuito impreso en nuestras manos, el primer paso que debimos realizar para la construcción de nuestro prototipo es la implementación de las interconexiones entre las dos capas de la placa, las cuales en una placa fabricada con tecnología moderna (usando agujeros platinados) vendrían ya implementadas del taller de fabricación de circuitos impresos. Estas interconexiones, también llamadas *vías,* deben necesariamente ser implementadas previamente a cualquier otro trabajo sobre la platina, ya que muchas de ellas se volverán inaccesibles una vez que se empiece a colocar los componentes electrónicos sobre la platina.

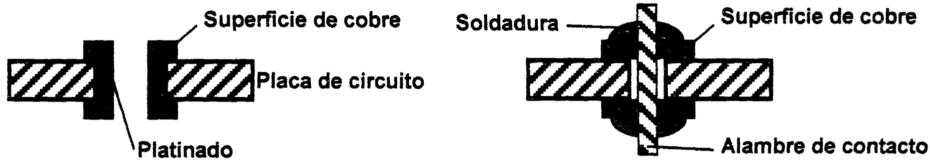


Figura 20: Vías con agujero platinado y vías conectadas con alambre

La Figura 20 muestra la forma de conexión implementada con un agujero platinado, y la manera en que se implementan las conexiones en placas de circuito no platinadas. Podemos ver que el proceso de conexión requiere la inserción de un pequeño alambre de cobre en el agujero, el cual es fijado en el sitio con soldadura de estaño. Algunos lectores podrían pensar que sería más eficiente simplemente llenar el hueco de

soldadura, pero esto desgraciadamente no es posible debido a la tensión superficial de la soldadura derretida, la cual se resiste a ingresar al agujero por si sola.

Aunque el proceso de interconexión de las vías no es difícil; debido a la gran cantidad de estas que existe en una platina compleja como la nuestra, este trabajo es bastante pesado y prolongado. Una vez que se termine de implementar la interconexión de vías, es absolutamente vital comprobar que todas las vías implementadas estén correctamente soldadas, confirmando que hay conexión eléctrica entre los dos lados de la vía, ya que como mencionamos antes, una vez que se colocan los componentes, muchas de las vías ya no podrán ser modificadas. Se recomienda además, para evitar problemas posteriores, la nivelación de las protuberancias existentes sobre las vías como resultado de exceso de cable y de soldadura, ya que estas podrían causar problemas en la colocación de componentes que van sobrepuestos a las vías. La forma más eficaz de remover el material excedente sobre las vías es el uso de una herramienta abrasiva, como un esmeril portátil.

Tras implementar y comprobar las interconexiones de las vías, el siguiente paso de nuestro trabajo de construcción fue la colocación de los circuitos integrados SMD, el DP83902A y el 80C31. Estos integrados deben ser colocados primero, ya que su soldadura es el proceso más delicado de toda la construcción de nuestro proyecto. Debido al diminuto tamaño de sus pines, y a la cercanía entre los mismos, el proceso de soldadura de estos debe realizarse con absoluta concentración, ya que si se comete un error, es extremadamente difícil desoldar uno de estos componentes sin herramientas

especializadas para este propósito. Hay que destacar que existen formas mucho más cómodas y confiables de soldar este tipo de componentes, como el uso de soldadura en pasta. En nuestro caso, sin embargo, dado que solo se iba a soldar dos piezas, no consideramos conveniente la adquisición de esta soldadura, la cual tiene un costo elevado.

Una vez colocados los componentes SMD, se procede a colocar el resto de circuitos integrados del proyecto, los cuales afortunadamente están todos implementados en formato DIP convencional. Al igual que las vías, los agujeros de conexión de nuestros circuitos integrados están implementados con conexión en ambos lados de la placa de circuito impreso, así que debemos asegurarnos que los pines de los circuitos integrados hagan contacto eléctrico con ambos lados de la placa. Esto lo logramos soldando el pin en ambos lados de la placa. Obviamente este proceso es más laborioso que la práctica normal de soldar solo el lado inferior de la placa, pero en nuestro caso no tenemos más alternativa.

Tras la colocación de los circuitos integrados, los procedimientos restantes son relativamente simples, se colocan el resto de componentes, tomando siempre en cuenta que deben estar conectados de ambos lados de la placa (aunque en algunos componentes este paso no es necesario y puede ser ignorado).

Una de las mayores dificultades de la construcción del proyecto fue asegurar la adecuada conectividad de ciertos componentes hacia ambos lados de la placa de

circuito, ya que estos por su diseño no se prestaban para ser conectados de esa manera. El más problemático fue el switch cuádruple, ya que sus pines no están a los lados como en los otros integrados, sino que al colocarse quedan ocultos debajo del componente, por lo que se requiere que este quede ligeramente levantado para poder soldarlo. Hay que notar que otros componentes del circuito, como los osciladores y los filtros de red, también presentan este problema, pero en su caso, este fue previsto desde el diseño de la platina, y se hizo un esfuerzo por usar solo conexiones en la parte inferior de la platina para estos componentes. En el caso del switch, ya que es un componente relativamente sin importancia y puede ser eventualmente reemplazado por jumpers o simples cables, se decidió obviar esta precaución.

La colocación de los componentes y subsecuente revisión de conexiones es el último paso en la construcción del hardware de nuestro proyecto. Un punto importante que hay que resaltar para finalizar este tema, es el costo en mano de obra estimado para la construcción de este proyecto. Este valor depende normalmente del tiempo que se tiene que emplear para la construcción del aparato, y la duración de este tiempo depende en este caso de dos factores principales: el tipo de platina utilizada, y el tipo de proceso utilizado para soldar los SMDs.

El tipo de platina que se use es el factor más importante para estimar el tiempo de construcción de este proyecto, debido a que los procedimientos más laboriosos de la manufactura del prototipo fueron la conexión de vías, y la conexión a ambos lados de los circuitos integrados y demás componentes. Estos procedimientos se verían

completamente obviados por el uso de una placa con agujeros platinados, así obviamente el proceso de construcción se vería bastante reducido.

El tipo de proceso para soldar los SMDs es de menor importancia, en cuanto a tiempo utilizado, pero si debe ser considerado debido a que el soldado convencional de estos componentes tiende a fatigar al ensamblador debido a la concentración y lentitud de trabajo que requiere, y tiene la desventaja de que cualquier error tendrá consecuencias que pueden ir desde una considerable pérdida de tiempo hasta la inutilización de la platina. Debido a esto, se considera que el uso de soldadura en base a pasta traerá una considerable reducción del tiempo utilizado para construir el dispositivo.

Ya hemos mencionado que para la construcción del prototipo utilizamos una placa convencional, así como soldadura normal para los SMDs así que se puede esperar que el tiempo utilizado para la construcción del aparato sea grande. Efectivamente, la construcción del sistema nos tomó aproximadamente unas diez horas, espaciadas a lo largo de dos días de trabajo. Tomando en cuenta que el proyecto es un prototipo difícil de reproducir, por lo que el trabajo se realizó con la mayor lentitud y deliberación del caso, se puede estimar que el proceso de construcción puede reducirse en unas dos horas si se trabaja en serie. Calculamos además que el uso de placas con agujeros platinados y soldadura adecuada para los SMDs podría reducir el tiempo de construcción por lo menos a la mitad del utilizado para la construcción de nuestro prototipo.

No vamos a hacer un análisis de costos de mano de obra, ya que los valores de este tipo de servicio pueden fluctuar considerablemente. Simplemente nos limitaremos a definir que el tiempo requerido para la construcción de este proyecto puede variar entre 10 y 5 horas-hombre (de acuerdo a la tecnología utilizada). Mediante este estimado, se puede rápidamente calcular costos sabiendo el valor de la hora de trabajo para la mano de obra disponible. El nivel de proficiencia tecnológica de la mano de obra a utilizarse no tiene que ser alto, pero se requiere experiencia en soldadura de componentes electrónicos, así como en identificación y colocación de los mismos.

3.4.3 Descripción y uso del hardware del proyecto

Una vez terminado el proceso de construcción de nuestro dispositivo, debemos pasar a hacer un estudio de los componentes principales del mismo, y la forma en que este debe ser conectado a sus diferentes periféricos.

Podemos ver en la Figura 21 un diagrama de nuestro dispositivo, en el cual se pueden identificar los componentes más importantes del sistema, así como los puntos de conexión con dispositivos externos y alimentación. A continuación describiremos detalladamente los componentes de nuestro dispositivo de cuya correcta utilización depende la operación del mismo.

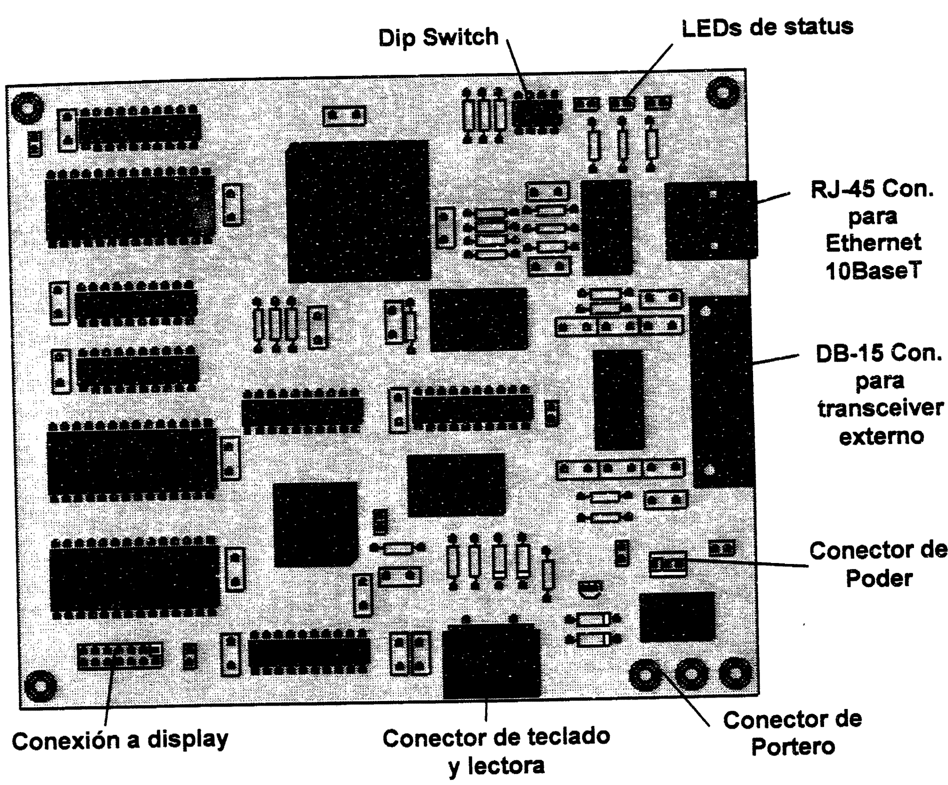


Figura 21: Componentes principales y conexiones del sistema

3.4.3.1 Memoria de código

El espacio de memoria de código de nuestro sistema está, como ya hemos mencionado, implementado en una memoria EPROM de 64 Kbytes. Debido a que el software de nuestro sistema está sujeto a muchas revisiones, esta memoria ha sido implementada en un socket, de manera que puede ser fácilmente removida y reprogramada. Hay que destacar aquí que debido al tipo de platina de circuito que utilizamos, un socket común y corriente no nos hubiera sido útil, ya que hubiera sido imposible realizar las conexiones dobles requeridas por nuestro diseño. Para evitar este problema utilizamos,

tanto para la EPROM como para la lógica programable, sockets de pin hueco, y simplemente nos limitamos a remover el plástico del socket y utilizar solo los pines, los cuales de esta manera pueden ser soldados fácilmente.

Hay que recalcar que nuestro sistema solo puede utilizar memorias del tipo 27512, ya que su organización física es fija, y orientada a esa memoria. Por eso hay que tener cuidado de no tratar de usar memorias como la 27256 o 27128, que tienen igual número de pines, pero están organizadas de manera distinta. Obviamente, si se desea posteriormente reducir el tamaño del espacio de memoria, está modificación se puede realizar sin mayor problema a partir del diagrama esquemático del sistema.

3.4.3.2 Conexión del display

Nuestro display fluorescente se conecta al sistema a través de un cable de 14 líneas, el cual está soldado a nuestra placa de circuito. Inicialmente intentamos implementar está conexión con un socket plano, pero por errores en el diseño, tuvimos que abandonarlo e implementarla de manera permanente. Esta sección de nuestro diseño se presta para bastante mejoría, ya que existe una gran variedad de tipos de conector que se prestan para esta función. Desgraciadamente, nos fue imposible conseguir un conector adecuado para esta parte del sistema (ya que son difíciles de obtener localmente); sin embargo, futuras mejoras podrían implementarlo.

El display recibe su alimentación de energía a través del cable de conexión, de manera que no necesita ninguna otra conexión externa. Si estudiamos la documentación del display, notaremos que hay ciertas preparaciones que hay que realizar en el mismo para poder acoplarlo a nuestro sistema. Esto se debe a que nuestro display puede interoperar con buses de sistemas Motorola, y con buses de sistemas Intel. La selección de bus se hace mediante el cierre de un jumper en el display. Obviamente este es un detalle específico del display que hemos utilizado, y podría variar si se usa otro modelo en el dispositivo final.

3.4.3.3 Conexión de alimentación

Como vemos en la Figura 22, la conexión de alimentación de nuestro sistema está implementada al lado del relay de la salida hacia el portero eléctrico. El conector utilizado es triple, y con un seguro de acoplamiento. Obviamente, como ya hemos discutido anteriormente, futuras versiones del sistema podrían hacer grandes cambios en la forma de alimentación de energía, de manera que la implementación de este conector podría verse sujeta a variación.

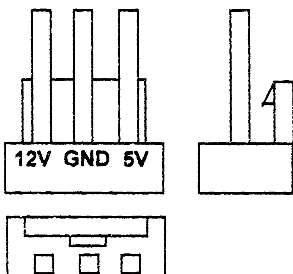


Figura 22: Conector de alimentación

Como podemos notar en la Figura 22, el conector tiene entradas separadas para 5 y 12 voltios de alimentación de corriente continua, y una tierra común para ambos voltajes. Los requerimientos de corriente de nuestro sistema no son muy elevados, aunque se puede hacer bastantes mejoras en el área de ahorro de potencia, pero dado que este es un prototipo, nos pareció mejor hacer un sistema que trabaje correctamente y de manera confiable, que idear esquemas complejos de ahorro de potencia que podrían complicar nuestro diseño. Las mediciones realizadas indican que nuestro sistema consume alrededor de 600 mA (a 5V), con todos sus periféricos conectados. Aquí hay que destacar que estas mediciones se hicieron con los periféricos provistos por la ESPOL; el uso de periféricos distintos conllevaría consumos de energía distintos, por lo que se recomienda un cierto margen de potencia para la alimentación de nuestro sistema. El uso de una fuente de alimentación de un amperio de potencia se considera adecuado para la alimentación del sistema combinado de 5 y 12 V, ya que la alimentación de 12V tiene una menor utilización de potencia, básicamente lo necesario para activar el relay y alimentar un transceiver opcional en el interface de red AUI.

3.4.3.4 Conector de teclado y lectora

El conector de teclado de nuestro sistema nos enlaza con la combinación teclado numérico-lectora de código de barras que hemos utilizado para el desarrollo de nuestro dispositivo. Al igual que con el display, el conector de teclado se encarga de alimentar de energía a los periféricos que se conectan a él. Básicamente cualquier dispositivo que tenga una interface de tipo teclado AT podrá conectarse a este conector e interoperar

con nuestro sistema, con la limitación que el intercambio de datos es unidireccional, ya que nuestro sistema solo es capaz de recibir datos en la interface de teclado. El estándar de interconexión de teclados AT también implementa una línea de inicialización, la cual por el momento no está utilizada en nuestro diseño.

Hasta donde hemos podido notar, no hay ningún problema con la remoción e inserción de dispositivos en este conector mientras el sistema está encendido; ya que el interface es unidireccional y los dispositivos no reciben configuración. Sin embargo, ya que la adición o remoción de dispositivos podría causar picos de voltaje, se recomendaría no hacerlo mientras el sistema está energizado.

3.4.3.5 Conector del portero eléctrico

El conector del portero eléctrico de nuestro sistema, no es más que un par de puntos de contacto conectados a nuestro relay, los cuales se cierran cuando el relay es activado. Podemos decir entonces, que el interface del portero eléctrico de nuestro sistema es de tipo "normalmente abierto".

En nuestro prototipo nos hemos abstenido de instalar un conector dedicado para este interface en la placa de circuito, prefiriendo implementar esta salida con conexiones de cable. Sin embargo, en futuras mejoras del sistema, esta salida podría implementarse con un par de puntos de conexión de alta potencia, quizás con tornillos incorporados.

3.4.3.6 Conector AUI

La salida AUI de nuestro sistema nos permite la conexión del mismo a transceivers externos, los cuales, a su vez, permiten que nuestro sistema interopere con todas las topologías de red Ethernet de 10 Mbps que existen en la actualidad. Ya que la especificación para este tipo de salidas así lo requiere, este conector se encarga de alimentar de corriente al dispositivo conectado al mismo. Esta alimentación es de 12V, y es una de las razones por las que ese nivel de voltaje es necesario en nuestro diseño.

El conector es un DB-15 hembra, el cual, como todos los conectores de tipo AUI, tiene incorporado un seguro deslizante para evitar que el dispositivo al que va conectado se desprenda accidentalmente.

3.4.3.7 Conector 10baseT

Esta salida permite a nuestro sistema conectarse directamente a una red Ethernet de topología 10BaseT, con un conector estándar RJ-45. Esta salida es compatible eléctrica y mecánicamente con las especificación IEEE 802.3, así como con la antigua especificación Ethernet II.

El usuario tiene la opción de utilizar esta salida o la salida AUI ya revisada, más no las dos al mismo tiempo. La selección entre ambas se realiza con el DIP switch, cuyo uso describiremos a continuación.

3.4.3.8 DIP Switch de configuración

Este juego de interruptores se utiliza para configurar la interface de red de nuestro sistema. El estado de estos no tiene efecto sobre ningún otro componente del sistema.

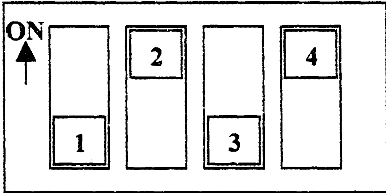


Figura 23: Configuración estándar del DIP switch

La Figura 23 muestra la configuración estándar que utilizamos para el desarrollo y experimentación con nuestro sistema. Las funciones realizadas por cada interruptor del dispositivo están detalladas en la siguiente tabla:

Pos.	Abierto (OFF)	Cerrado (ON)	Estado Estándar
1	En modo 10BaseT, la detección de portadora en la línea de red está activada. En modo AUI, el interruptor es ignorado.	En modo 10BaseT, la detección de portadora de red está desactivada, el LED de enlace permanece encendido.	Abierto
2	Se selecciona la salida AUI de la interface de red.	Se selecciona la salida 10BaseT de la interface de red.	Cerrado
3	Operación normal.	Se aísla el decodificador de salida del ST-NIC para diagnóstico de fábrica.	Abierto
4	La salida 10BaseT es eléctricamente compatible con IEEE 802.3.	La salida 10BaseT es eléctricamente compatible con Ethernet II	Cerrado

Tabla 3: Operación del switch de configuración

De acuerdo a los datos mostrados en la tabla, la configuración que hemos utilizado durante el desarrollo de nuestro proyecto para la interface de red es: Interface10BaseT, detección de portadora activada, y compatibilidad eléctrica con Ethernet II.

Hay que destacar que en la práctica, de estos interruptores, solo es importante la posición de los dos primeros, ya que el tercer interruptor no es necesario para la operación normal del sistema(fue añadido por precaución en caso de problemas con la interface de red) y podrá ser posteriormente eliminado del diseño, y la posición del cuarto interruptor no tiene importancia en la enorme mayoría de situaciones, ya que prácticamente todos los dispositivos de red tipo 10BaseT son compatibles eléctricamente tanto con Ethernet II como con IEEE 802.3.

3.4.3.9 LEDs de status

Para finalizar, revisaremos las funciones desempeñadas por los tres LEDs de status que contiene nuestro sistema. Estos indicadores se utilizan principalmente para diagnosticar la operación de la interface de red, y se distribuyen de la siguiente manera: Dos LEDs color rojo, TX y RX, los cuales se utilizan para indicar la transmisión (TX) o recepción (RX) de información desde la red; además se tiene un LED de color verde, LINK, el cual nos indica la presencia de la portadora de red en modo 10BaseT, y permanece encendido en modo AUI. Este último LED puede utilizarse como una referencia rápida de que el sistema está listo para operar, ya que si estamos usando modo 10BaseT y este LED está apagado, significa que no tenemos portadora de red, y por lo tanto el sistema no podrá establecer comunicaciones en la red. En modo AUI, en cambio, si este LED no está encendido, significa que el sistema está apagado, o que la interface de red está averiada.

Con esto concluimos la descripción de la operación de nuestro prototipo. A continuación detallaremos los cambios y modificaciones que por cualquier razón hemos tenido que hacer a nuestro diseño original, y las modificaciones que se consideran deseables en futuras implementaciones de este dispositivo.

3.4.4 Modificaciones y recomendaciones

El diseño original de nuestro prototipo tuvo una serie de errores y deficiencias, algunas de las cuales fue necesario corregir para el correcto funcionamiento de nuestro dispositivo, y otras en cambio son simples molestias o ineficiencias que podrán ser eliminadas en posteriores iteraciones de diseño. A continuación mostramos una lista de estas fallas, y su status actual:

Problema	Modificación requerida/recomendada	Status actual
Conexión de display invertida. No se puede usar conectores DIL para implementarla.	Modificación de diseño de placa de circuito impreso.	Problema obviado. Se descartó el uso de un conector y se soldó directamente las líneas.
Salida de 80C31 hacia portero eléctrico sin inversor.	Modificación de diseño de placa de circuito impreso. Adición de inversor a lógica programable.	Parcialmente resuelto. El inversor fue añadido a la lógica programable, pero la conexión se implementó mediante cables.
Dificultad en colocación de DIP switch.	Modificación de diseño de placa de circuito impreso y/o reemplazo de DIP switch por jumpers.	No resuelto. DIP switch implementado sin modificaciones.
Red de inicialización deficiente.	Modificación de diseño de placa de circuito impreso.	Resuelto mediante cables externos. Modificación pendiente.
Red de alimentación deficiente.	Modificación de diseño de placa de circuito impreso.	No resuelto.
EPROM de tamaño excesivo.	Modificación de diseño lógico y de placa de circuito impreso.	No resuelto.

Tabla 4: Lista de problemas de diseño del prototipo del sistema

Consideramos que de estas fallas, la más importante es la deficiencia en el diseño de nuestra red de alimentación, la cual es marginalmente adecuada para los requerimientos de nuestro sistema, más no sería conveniente para un dispositivo de uso comercial, como ya explicamos anteriormente.

La mayoría de los problemas que mencionamos aquí ya han sido discutidos en otras secciones de este trabajo, con la excepción de la red de inicialización del sistema. Esta red se compone de la circuitería encargada de inicializar los diferentes componentes del sistema para su correcto funcionamiento. En nuestro sistema es posible realizar la inicialización de cuatro componentes distintos: El 80C31, el ST-NIC, el display fluorescente, y los dispositivos conectados a la entrada de teclado. De estos, solo el ST-NIC y el 80C31 tienen implementada circuitería de inicialización. En el caso de display, esta no es absolutamente necesaria, ya que el aparato se inicializa correctamente al encenderse, y además es posible inicializarlo mediante comandos de software. En el caso del conector de teclado, los diseños en los que nos basamos para implementar su arquitectura no implementaban la conexión de inicialización, por lo que decidimos no implementarla tampoco para mantener la complejidad de nuestro diseño lo más baja posible.

El ST-NIC y el 80C31 si poseen circuitería de inicialización, la cual consiste en redes RC comunes. Sin embargo, este diseño causó problemas en la etapa de desarrollo y diagnóstico del sistema, ya que los dos dispositivos tienen redes de inicialización distintas, de manera que la inicialización del uno no garantizaba la inicialización del

otro, a menos que esta se realizara apagando el equipo. Como veremos en el siguiente capitulo, no nos era posible realizar la inicialización de nuestro sistema simplemente apagándolo, por lo que tuvimos que integrar las dos redes de inicialización; esto requirió la implementación de un inversor en nuestra lógica programable, ya que los niveles de inicialización de ambos dispositivos son opuestos. Sin embargo, la integración de la red de inicialización trajo sus propios problemas, ya que debido a su diseño, el ST-NIC tiende a inicializarse incorrectamente si se le envía la señal de inicialización antes de que sus señales de reloj y alimentación se hayan estabilizado. Debido a esto, se tuvo que hacer una modificación final a la red de inicialización; básicamente lo que se hizo fue dedicar una línea de salida del 80C31 a la generación de la señal de inicialización del ST-NIC. Con esto conseguimos que el control de la inicialización del ST-NIC resida en el software del controlador, y no en la red RC usada anteriormente. Esta modificación eliminó los problemas de inicialización del sistema de manera tan eficiente, que se decidió controlar también la inicialización del display fluorescente mediante la misma.

De esta manera finalizamos el estudio del hardware de nuestro proyecto, el siguiente capítulo tratará sobre el concepto, diseño e implementación del software necesario para la operación de nuestro sistema.

Capítulo IV

4 Diseño del software

4.1 Esquema conceptual del software del proyecto

En el capítulo II discutimos la forma en que se desea implementar las interacciones de software en este proyecto. Como dijimos, el software de nuestro proyecto se divide en dos componentes principales: el software cliente, y el software servidor. El software cliente debe implementarse en el sistema de hardware que hemos estado analizando en los dos capítulos anteriores, mientras que el software servidor puede estar en cualquier tipo de computador que sea accesible mediante el esquema de conectividad de red que vamos a utilizar.

Como ya mencionamos en el capítulo II, el software del servidor de nuestro sistema está básicamente ya desarrollado e implementado sobre una plataforma Windows NT, siendo nuestra única responsabilidad el modificarlo para que interactúe con el software cliente implementado en este proyecto.

Tenemos entonces que el énfasis principal del diseño de software de este proyecto va a estar en la implementación del código a ejecutarse en el cliente de nuestro sistema. Este código, como ya revisamos, estará dividido en tres componentes conceptuales básicos: la aplicación de seguridad, la serie de rutinas de comunicaciones TCP/IP que esta utiliza

para comunicarse (comúnmente llamadas *pila* o *stack* TCP/IP), y las rutinas de bajo nivel usadas para controlar los dispositivos de hardware del cliente.

Habiendo refrescado esta información, podemos pasar a analizar el diseño del código de nuestro sistema, empezando por aquel que implementamos para el dispositivo cliente del mismo.

4.2 Diseño del código del cliente

Una de las partes más importantes del proceso de diseño, es la definición de las posibilidades y limitaciones dentro de las cuales se desarrolla nuestro proyecto. Esto nos permite un correcto dimensionamiento de la funcionalidad de todos los componentes de nuestro sistema, incluyendo al software. La tarea de encontrar y definir estos límites es entonces el primer paso que debemos realizar en el diseño del código de nuestro cliente, y es lo que revisaremos a continuación.

4.2.1 Criterios de diseño

El software que debemos diseñar para el cliente de nuestro sistema debe ser capaz de recibir datos de los dispositivos de entrada conectados al mismo (lectora y teclado), debe procesar esos datos, y debe enviarlos al servidor a través de la red TCP/IP a la que va a estar conectado. Así mismo, debe ser capaz de recibir las respuestas del servidor a sus pedidos, procesarlas, y tomar las acciones que estas ordenen. Esta es la funcionalidad básica que debe ser capaz de realizar nuestro código. Nuestra labor es

encontrar la manera de implementar esta funcionalidad dentro de la arquitectura de hardware que hemos diseñado; aquí vale la pena recordar el hecho de que el diseño de nuestra arquitectura de hardware, e incluso la implementación de nuestro prototipo, ya estaban realizados para cuando empezamos la etapa de diseño de software, de manera que no nos es posible (o no resulta barato ni práctico) el hacer cambios a nuestro hardware para acomodar las funcionalidades que requiere nuestro software, básicamente debemos trabajar con lo que tenemos hasta este momento.

Debemos entonces revisar las características del hardware que vamos a utilizar, siendo la más importante la limitación de recursos, la cual fue necesaria para mantener costos y complejidad a un nivel aceptable. Nuestro sistema posee únicamente 8 Kbytes de memoria RAM. Si comparamos esto con los 32 o 64 Mbytes de memoria RAM que es común encontrar en cualquier computador moderno, veremos que no tenemos mucho espacio de almacenamiento para trabajar. El espacio de código del sistema es también limitado, solo 64 Kbytes, de manera que es necesario mantener el software lo más compacto posible, para evitar tener problemas de espacio de código. Otro problema son las limitadas capacidades de procesamiento del 80C31, comparado con otros procesadores existentes en el mercado, lo cual nos indica que debemos tratar de que nuestro código sea lo menos complejo posible, para que su tiempo de ejecución sea aceptable.

De todas estas limitaciones, la más crítica es el espacio de almacenamiento de datos, es decir, la memoria RAM. Basta únicamente decir que un paquete IP puede tener hasta 64

Kbytes de extensión, para saber que la implementación de TCP/IP que vamos realizar va a ser de funcionalidades limitadas, dados los recursos de los que disponemos. El hecho de tener capacidad de memoria limitada nos ha obligado a lo largo de nuestro trabajo a tratar de compactar el uso de variables y espacio de memoria en general, de manera que si una variable puede realizar dos o tres funciones distintas en diferentes procesos de nuestro código, estas se le asignan. El resultado inevitable de esto es un código poco legible y difícil de mantener, pero que es a la vez bastante eficiente en el uso de recursos del sistema.

Una decisión fundamental que se debió realizar con respecto al diseño de nuestro código de cliente, era el tipo de herramienta de programación que se usaría para implementarlo. El 80C31, así como la mayoría de sistemas integrados, no está diseñado para usarse con lenguajes avanzados de programación, sino que normalmente se presta para el desarrollo de procedimientos simples codificados en lenguaje ensamblador, es decir usando directamente el set de instrucciones del procesador. Esto ofrece el potencial de una inigualable eficiencia del código desarrollado. Sin embargo, este tipo de programación es engorrosa y difícil de depurar, y aún más difícil de leer. Otra alternativa disponible es el uso de lenguajes de programación de alto nivel como BASIC; FORTRAN, PASCAL, o C; estos lenguajes ofrecen facilidad de desarrollo y legibilidad, pero tienden a ser ineficientes en cuanto a uso de memoria y extensión del código, lo cual es de vital importancia en nuestro sistema, como ya hemos indicado.

La consideración que finalmente fue determinante para escoger el tipo de herramienta que se iba a utilizar para el desarrollo de nuestro sistema fue la complejidad que podíamos esperar de nuestro código, dadas las múltiples funciones que este debe realizar. Debido a esto, se decidió que lo mejor sería implementar el código usando un lenguaje de alto nivel, ya que su diseño en lenguaje ensamblador sería excesivamente engorroso. Esto nos llevó al dilema de conseguir una herramienta de desarrollo para el 80C31 basada en un lenguaje de alto nivel, preferiblemente en lenguaje C, debido a su flexibilidad para acomodarse a distintas arquitecturas. La tarea de conseguir una herramienta de este tipo no es tan fácil; existen muchas herramientas de diseño para la arquitectura 8051, incluso existen algunas de excelente calidad que son gratuitas, pero todas ellas son basadas en lenguaje ensamblador. Las herramientas basadas en C u otros lenguajes de alto nivel tienden a ser bastante costosas, ya que están dirigidas a mercados restringidos y de poder adquisitivo relativamente alto. Afortunadamente fue posible conseguir uno de estos sistemas de diseño (el cual describiremos más adelante) de manera gratuita, lo cual nos permitió seguir adelante con la idea de usar un lenguaje de alto nivel para nuestra implementación.

El uso de lenguaje C para el diseño de nuestro proyecto nos impuso ciertas dificultades adicionales, ya que este lenguaje utiliza variables propias y pilas de sistema, las cuales están mayormente fuera del control del programador. El 80C31 tiene la capacidad de implementar una pila de variables e instrucciones, pero esta pila solo puede ser implementada en el espacio de memoria RAM interno del dispositivo, el cual mide solo

128 bytes. Esto impone ciertas limitaciones a la complejidad de las interacciones en nuestro código, para evitar sobrecargar la pila.

Hay que decir a esta altura, que los compiladores de C para el 8051/31 pueden, mediante rutinas de software, implementar pilas en el espacio de memoria externa del sistema. El problema de esto, aparte de hacer al código bastante lento, es que estas rutinas (en el caso del compilador que utilizamos) requieren la desactivación de interrupciones mientras se ejecutan. Como veremos más adelante, la rutina de control de teclado de nuestro sistema hace imposible la desactivación fortuita de interrupciones, y por lo tanto el uso de una pila en memoria externa.

Como resultado de esto, tuvimos que usar un modelo de memoria en nuestro compilador C que implemente la pila del sistema en la memoria interna del 80C31. El efecto que esto tiene sobre el desarrollo de nuestro código, es que se ha tratado por todos los medios posibles de minimizar el uso de la pila. El usuario podrá notar a todo lo largo de su estudio de nuestro código, que la mayoría de llamadas de función en el mismo se hacen sin argumentos, en vez de esto se utilizan variables externas y estructuras para pasar datos entre funciones. Esto, combinado con la reutilización de variables, hace a nuestro código notablemente difícil de interpretar, situación que intentamos remediar con una adecuada documentación.

Antes de pasar al estudio del código en sí, haremos una pequeña revisión de las herramientas utilizadas para su diseño y diagnóstico.

4.2.2 Herramientas utilizadas

La implementación y diagnóstico de una arquitectura de software requiere como herramientas base de un editor que permita el desarrollo y modificación del código, y de un compilador que convierta el código a instrucciones de procesador. En nuestro caso en especial, ya que estamos trabajando con una arquitectura integrada, necesitamos además una manera de verificar la correcta operación del código. Existen varios tipos de herramientas para sistemas integrados que permiten este tipo de diagnóstico, las cuales describimos a continuación:

- **Simuladores**. Son aplicaciones de software que simulan la operación completa de un procesador integrado, en nuestro caso el 80C31. Estos programas reciben como entrada de datos el código desarrollado, y lo ejecutan dentro de su "procesador virtual" el cual permite al usuario revisar todos los aspectos de la ejecución del código. Estas herramientas son bastante útiles, pero en nuestro caso, debido a la complejidad de las interacciones de entrada y salida de nuestro sistema, el uso de estas herramientas no es una solución práctica.

- **Monitores**. Son aplicaciones de software que se ejecutan en el procesador del sistema en diagnóstico, y se comunican serialmente con un computador, del cual reciben el código a evaluar, y lo guardan en la memoria RAM del sistema integrado, desde la cual lo ejecutan y evalúan. Estas herramientas son bastante útiles para la

etapa de aprendizaje y experimentación del uso de un sistema integrado, pero debido a que requieren memoria RAM para guardar el código, y requieren arquitecturas especiales (espacios de código y datos superpuestos) para poder ejecutar código desde RAM, no son adecuados para usarse en la experimentación con el hardware ya implementado. Existen varias de estas herramientas disponibles gratuitamente al público en el Internet, lo cual las hace fácilmente las más asequibles de todas las ayudas de diseño para el 80C31.

- **Emuladores**. Estos son combinaciones de software y hardware, que básicamente consisten de un simulador igual a los descritos en el punto anterior, pero que en vez de comunicarse con un "procesador virtual" simulado en software, se comunican a través de un enlace de datos, normalmente serial, con un dispositivo instalado en el sistema que se está diagnosticando, y hace las veces de procesador del mismo. De esta manera se puede estudiar directamente las interacciones entre el sistema y el código desarrollado. Estas herramientas tienen la gran desventaja (para nosotros) de ser bastante costosas, y además no son muy prácticas para sistemas con espacio de código externo, ya que el código tiene que ser reescrito a la memoria de código cada vez que se lo modifica.

- **Emuladores de EPROM**. Esta es una herramienta que como su nombre lo indica simula la funcionalidad de una memoria EPROM. Estos dispositivos normalmente se conectan serialmente a un computador, desde el cual se transfiere serialmente el código a diagnosticar a la memoria del emulador, el cual a su vez está conectado al

socket de la memoria EPROM del sistema integrado, y simula sus funciones. Este tipo de herramienta es muy útil en sistemas que utilizan memoria de código externa, ya que evita la reprogramación de EPROMs, un proceso que en el mejor de los casos toma mas de 20 minutos. Aunque son más baratos que un emulador de procesador, los emuladores de EPROM siguen siendo dispositivos relativamente costosos.

Como ya dijimos anteriormente, se decidió utilizar el lenguaje C como base para el desarrollo del código de nuestro proyecto, aunque hay que destacar que partes de él debido a requerimientos especiales, fueron desarrolladas en lenguaje ensamblador. La herramienta que utilizamos para generar el código C de nuestro proyecto es el sistema de diseño AVCASE 51 de la compañía Avocet Systems. Este sistema es una integración de herramientas de software y hardware para diseño y diagnóstico de programas para el 8051/31. El sistema incluye un editor de texto, un ensamblador, un compilador C, una utilidad de compilación automática, un simulador, y un emulador de procesador. Desgraciadamente, debido a la forma en que se obtuvo esta herramienta, no tuvimos acceso más que a los componentes de software del mismo, y lo que es peor, no pudimos obtener su documentación, por lo que el uso que hemos hecho de sus funciones es muy inferior a su potencial utilidad. Aunque el sistema AVCASE 51 tiene un ensamblador, mucho del desarrollo y experimentación inicial de nuestro proyecto fue realizada usando el ensamblador ASM51 de la compañía Metalink; esta herramienta es de dominio público y viene con documentación completa, por lo que su uso es bastante

sencillo y vale la pena que el desarrollador de software para el 8051/31 conozca de su existencia.

Aparte de estos componentes de desarrollo de código, necesitamos una herramienta de hardware que nos permita evaluar la operación del código en nuestra implementación de hardware; en las etapas iniciales del desarrollo del hardware del proyecto, usamos un sistema experimental de desarrollo, montado sobre un protoboard, y corriendo la aplicación PAULMON 1, que es un monitor para el 8051/31, libremente disponible en el Internet. Este prototipo de desarrollo fue usado para experimentar la operación de las interfaces con la lectora de banda magnética y con el display fluorescente. Desgraciadamente, cuando llegó la hora de experimentar con la interface de red de nuestro sistema, debimos abandonar el uso del monitor, y pasar directamente a la construcción del prototipo del sistema propiamente dicho, ya que la interface de red es demasiado complicada de implementar en un protoboard.

Habiendo construido ya nuestro sistema, necesitábamos una forma eficiente de evaluar el código que íbamos a generar. Hasta entonces el código había sido ejecutado en el sistema de desarrollo que implementamos, mediante el uso del monitor. Pero una vez implementado nuestro prototipo de hardware, el uso del monitor ya no era posible, así que teníamos dos opciones para la experimentación con nuestro código: La reprogramación de EPROMs cada vez que se hiciera una revisión del código, o el uso de un emulador de EPROMs.

La reprogramación de EPROMs hubiera implicado viajes prácticamente diarios a la ESPOL para reprogramar los dispositivos, a un costo en tiempo y dinero absolutamente prohibitivo, así que decidimos usar un emulador de EPROMs como herramienta de desarrollo de software. Debido a que no teníamos presupuesto para la compra de uno de estos dispositivos, dedicamos un periodo de dos semanas de trabajo al diseño e implementación de un emulador limitado, construido en protoboard. Nuestro emulador puede simular las funciones de una EPROM 27512 como la que usamos en nuestro proyecto, siendo capaz de almacenar hasta 64 Kbytes de datos en su espacio de memoria. La interface con el usuario se implementó tomando como base el software monitor PAULMON 1.

El emulador de EPROMs resultó ser una herramienta absolutamente invaluable para el desarrollo de nuestro software, ya que debido a la complejidad de este último, tuvimos que hacer decenas de revisiones del mismo, actividades que se hubieran visto tremendamente dificultadas por la falta de una herramienta eficiente para la evaluación de código. Aunque nosotros construimos nuestro propio emulador, existen versiones comerciales de esta herramienta que soportan todo tipo de EPROMs, con mejores tiempos de respuesta, y a precios en el rango de 100 – 200 dólares. Dado el ahorro en tiempo y dinero que el uso de esta herramienta representa, bien vale la pena adquirirla, si no es posible su construcción.

Para finalizar, notaremos que dado que nuestro sistema tiene como finalidad la interacción con una red de área local, otra herramienta cuyo uso se vuelve indispensable

es un analizador de protocolos. Estas herramientas pueden venir implementadas como software o como dispositivos separados de hardware, y pueden monitorear el tráfico en una red de área local. Desgraciadamente, estas herramientas pueden ser increíblemente caras, con costos que oscilan entre 400 y 10000 dólares. Obviamente estos costos están fuera de nuestras posibilidades; sin embargo, logramos conseguir una versión de evaluación de un producto llamado EtherBoy, de NDG Software, el cual, aunque limitado, sirvió para evaluar las capacidades de red de nuestro sistema, y estudiar su interacción con el software de servidor.

Con esto concluimos nuestra revisión de las herramientas que utilizamos para el desarrollo de nuestro software. En las siguientes secciones pasaremos a revisar de manera detallada la implementación del software de nuestro cliente de seguridad.

4.2.3 Organización del Software

Cuando revisamos el esquema conceptual de la organización de nuestro software, recalcamos que este se dividía, de acuerdo a su funcionalidad, en tres capas distintas: La aplicación, los protocolos de comunicación TCP/IP, y las rutinas de control de hardware.

Para el desarrollo de software de nuestro proyecto se pudo haber utilizado una serie de metodologías distintas, pero debido al hecho de que se trata de un trabajo individual, y a que no tenemos conocimientos estructurados de programación, decidimos realizar

nuestro trabajo de diseño de software desde abajo hacia arriba, es decir, diseñando primero las funciones mas elementales del código, y a partir de estas ir añadiendo capas superiores de complejidad. Esta metodología es adecuada también porque se ajusta muy bien a la filosofía de diseño de la arquitectura TCP/IP, que trata siempre de separar funciones entre sus diferentes capas de operación.

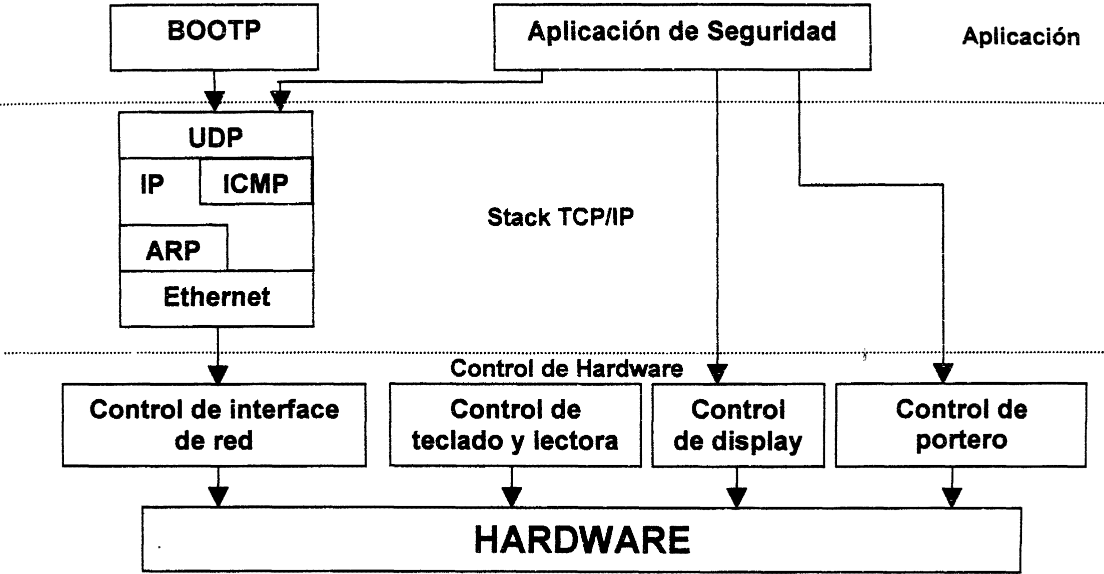


Figura 24: Diagrama organizacional del software cliente

La Figura 24 muestra las interrelaciones existentes entre los distintos bloques de código que vamos a analizar. Algo interesante que podemos ver, es que por primera vez mostramos los diferentes componentes de la arquitectura TCP/IP que vamos a utilizar. Las líneas punteadas nos muestran las fronteras entre los tres bloques conceptuales de software que habíamos definido hasta el momento.

Como ya mencionamos, el desarrollo de nuestro proyecto fue avanzando desde las capas más bajas de funcionalidad, y progresando hacia arriba a medida que se confirmaba el correcto funcionamiento de cada capa de operación.

Como vemos, nuestro sistema tiene distintos procesos que tienen que ejecutarse y coordinarse entre si para que el dispositivo cliente funcione correctamente. Muchos de estos procesos requieren ejecución periódica, e intercambio de datos con otros procesos para poder realizar su trabajo. El hecho de que nuestro software trabaje de esta manera nos planteó ciertas interrogantes importantes con respecto al diseño del mismo. Estas interrogantes se pueden resumir en la pregunta de si nuestro software necesita o no un sistema operativo.

En un computador normal, el sistema operativo realiza una serie de funciones de mantenimiento y coordinación que son indispensables (asignación de memoria, control de procesos, acceso a hardware, etc.), pero que el programador normalmente tiende a menospreciar, ya que el sistema operativo las hace por él. Este no es nuestro caso, ya que estamos trabajando en un sistema integrado, el cual no posee ningún tipo de sistema operativo; la funcionalidad de este sistema debe ser proporcionada por nosotros.

Aunque la mayoría de sistemas integrados no necesitan normalmente un sistema operativo, existen varios productos en el mercado que ofrecen lo que se conoce como RTOS (Real Time Operating System). Un RTOS es una implementación integrada de un sistema operativo multitarea, con funcionalidad de asignación de memoria, control

de procesos, creación de colas, y casi todas las funciones requeridas para ejecución simultánea de distintos procesos. Ya que la documentación sobre la que hemos basado nuestra implementación de la arquitectura TCP/IP *(1)* asume el uso de un sistema operativo multitarea, el uso de uno de estos nos hubiera sido de suma utilidad. Desgraciadamente no pudimos conseguir ninguna versión de un RTOS para el 8051/31, por lo que tuvimos que organizar nuestro software para ejecución lineal, y no para ejecución simultánea.

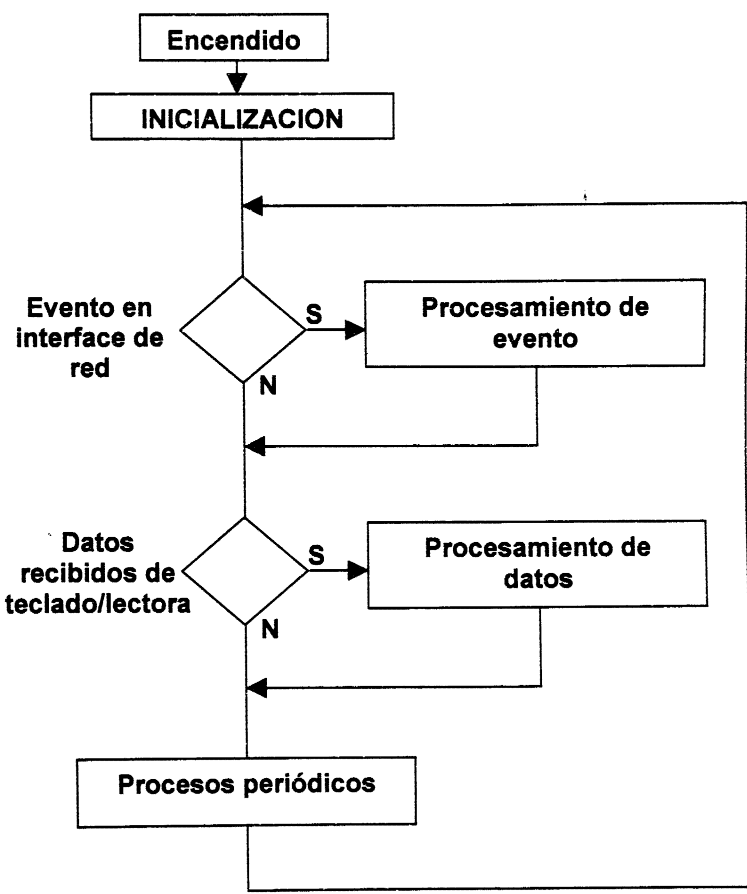


Figura 25: Diagrama de flujo simplificado del software cliente

En la Figura 25 tenemos una representación simplificada del flujo de ejecución de nuestro software. Podemos ver que la ejecución comienza con una etapa de inicialización, en la que se prepara el sistema para su correcto funcionamiento. En esta etapa se configuran los periféricos del sistema, se inicializan los puertos de salida del 80C31, y se preparan las variables globales del sistema.

Una vez realizada la etapa de inicialización, el sistema entra a un lazo infinito en el que va a realizar todas sus funciones. Podemos notar que hay dos formas principales de respuesta del sistema a eventos externos: respuesta a recepción de paquetes en la red, y respuesta a ingreso de datos en la lectora o teclado. Tenemos que advertir que este diagrama es una simplificación de lo que realmente ocurre, pero es una fiel representación de la forma de operar de nuestro sistema. Aparte de las respuestas a eventos, el sistema realiza una serie de operaciones periódicas de mantenimiento, las cuales, como su nombre lo indica, se realizan cada cierto tiempo independientemente de que el sistema reciba o no estímulos externos. Debido a la forma en que está organizado nuestro flujo de ejecución, el sistema se basa fuertemente en el uso de banderas para su operación.

Una desventaja importante que tiene este tipo de organización de software, es que sus tiempos de ejecución no son determinísticos, es decir, no se pueden predecir. Esto se debe a que la presencia de eventos atrasa la ejecución de las tareas de proceso periódicas, de manera que estas no se ejecutan a intervalos regulares, sino cuando el flujo de eventos externos así lo permite. Esto podría hacernos pensar que una serie

regular de eventos externos sucediendo lo bastante rápido podría detener por completo al sistema. Afortunadamente esto no puede suceder, ya que cada procesamiento de evento debe espera un ciclo completo del sistema para tener oportunidad de ejecutarse nuevamente. Sin embargo, la falta de precisión en los tiempos de ejecución del sistema hacen imposible su utilización para ciertas tareas, como por ejemplo la implementación de un reloj de software. Si se requiere tiempos de ejecución determinísticos, no queda más remedio que utilizar una implementación basada en un RTOS, que es justamente el tipo de aplicación para el que estos sistemas operativos fueron pensados. Afortunadamente para nosotros, ninguno de nuestros procesos requiere de manera imperativa tiempos de ejecución determinísticos, así que pudimos trabajar sin problemas con el esquema de ejecución arriba descrito.

Una vez definida a breves rasgos la organización lógica de nuestro software, podemos pasar a la descripción de sus diferentes procesos de manera más detallada.

4.2.4 Descripción detallada del software cliente

Nuestro software cliente está mayormente implementado en el lenguaje ANSI C, y decimos mayormente porque ciertas partes del código han tenido que ser implementadas en ensamblador, dado que deben ejecutarse lo más rápido y eficientemente posible.

El hecho de tener partes del software implementadas en ensamblador no implica trabajo extra de compilación, ya que el sistema de diseño AVCASE 51 hace posible la

incrustación de código ensamblador dentro de rutinas codificadas en C. De esta manera, usamos ensamblador donde sea necesario en interés del rendimiento, y el resto del programa se codifica normalmente en C.

Archivo	Descripción	PERTENECIENTE A:
8051.h	Encabezado estándar de registros del 8051/31	AVCASE 51 (librería propietaria)
Intrpt.h	Encabezado de manejo de interrupciones del 8051/31	AVCASE 51 (librería propietaria)
Stdlib.h	Encabezado de la librería estándar de ANSI C	AVCASE 51 – ANSI C
String.h	Encabezado de la librería de manejo de cadenas de caracteres de ANSI C	AVCASE 51 – ANSI C
Sec_cli.h	Encabezado de variables y estructuras del sistema	Software del cliente
Sec_cli.c	Archivo principal del sistema. Contiene el lazo de ejecución, los manejadores de eventos, y los manejadores de tareas periódicas.	Software del cliente
Io_rout.c	Rutinas de bajo nivel. Contiene rutinas de acceso al hardware de la interface de red y al display fluorescente.	Software del cliente
Display.c	Rutinas y macros de manejo del display	Software del cliente
H_init.c	Rutinas de inicialización de hardware y software del sistema.	Software del cliente
Ethernet.c	Rutinas de entrada y salida a la red ethernet.	Software del cliente
arp.c	Rutinas de control y manejo del cache de ARP, y entrada y salida de paquetes ARP.	Software del cliente
ip.c	Rutinas de entrada y salida de paquetes IP e ICMP.	Software del cliente
Udp.c	Rutinas de entrada y salida de paquetes UDP	Software del cliente
Bootp.c	Rutinas de entrada y salida de paquetes BOOTP, rutinas de resolución de configuraciones del sistema.	Software del cliente
Sec_app.c	Rutinas de entrada y salida de paquetes del protocolo de seguridad del sistema.	Software del cliente

Tabla 5: Archivos pertenecientes al código del cliente

Para facilidad de programación y diagnóstico, nuestro software fue dividido en varios archivos distintos, según la funcionalidad de las rutinas implementadas. Además de estos archivos, el software requiere varios encabezados propietarios del sistema AVCASE 51, así como encabezados estándares de las librerías de ANSI C. En la Tabla 5 mostramos una lista de los archivos necesarios para la correcta compilación de nuestro

software, junto con una descripción de la función que ejercen las rutinas en cada archivo.

Obviamente, este código puede ser compilado únicamente en el compilador C del sistema AVCASE 51. Si se desea usar otro tipo de compiladores C para el 8051, se deben hacer pequeñas modificaciones al código, especialmente en el manejo de código ensamblador y manejo de interrupciones. Las modificaciones exactas que se vayan a hacer dependen del compilador que se piense usar, así que no las describiremos aquí. El resto del código está implementado de acuerdo al estándar ANSI C, así que debería ser portable a otras implementaciones de lenguaje C para el 8051.

A diferencia de los sistemas de computadores personales, en la mayoría de los casos, la salida de un compilador C o un ensamblador para el 8051 (o algún otro procesador integrado) tendrá que ser implementada en una memoria discreta (EPROM, ROM, Flash ROM, EEPROM, etc.); de manera que los compiladores para sistemas integrados normalmente tienen como salida ya sea directamente el código binario, o alguna representación del mismo requerida por los programadores de dispositivos. El formato más utilizado para procesadores Intel es el formato Intel Hex, que es el que usamos en este proyecto. Este formato tiene soporte en la mayoría de programadores de dispositivos modernos, y es aceptado por todos los servicios de implementación de ROMs, así que la codificación de la salida de nuestro compilador no debería ser ningún problema.

Habiendo ya definido la estructura de archivos de nuestro código, estamos listos para empezar a describir la funcionalidad del mismo en forma más detallada, tarea que realizamos a continuación.

4.2.4.1 Variables globales y estructuras de datos: *sec_cli.h*

Dadas las distintas tareas y procedimientos que nuestro sistema debe realizar durante sus operaciones normales, muchas de las cuales se realizan como respuesta a eventos externos, debemos tener una manera de mantener control del estado de los procesos que se están realizando en un momento dado. La forma en que nuestro sistema realiza este control es mediante el uso de banderas, las cuales señalan cuando se necesita la ejecución de una rutina, o cuando el sistema se encuentra en un estado específico. Debido a esta forma de operar, nuestro sistema tiene una serie de variables globales, las cuales están definidas en el archivo sec_cli.h.

Además de las variables globales, este archivo contiene todas las etiquetas del sistema, así como las definiciones de las estructuras de datos usadas a lo largo de todo el código. En este espacio no haremos una descripción detallada de cada elemento del archivo, ya que este último contiene en sus comentarios la información necesaria para comprender la función de cada etiqueta, variable y estructura. Lo que si haremos es una descripción de las variables y estructuras más importantes y su función dentro del sistema.

El archivo sec_cli.h tiene tres secciones principales, cada una de las cuales describiremos a continuación:

4.2.4.1.1 Lista de direcciones de periféricos:

Esta sección contiene las direcciones de memoria de los periféricos direccionados de nuestro sistema. Las variables que contiene esta sección representan direcciones de 16 bits. Como podemos ver al revisar el código las variables aquí listadas tienen asignado un valor hexadecimal precedido por el signo "@". Esta es una convención del sistema de diseño AVCASE 51 que se utiliza para indicar al compilador que ese valor representa una dirección de memoria externa. Obviamente, estos valores no pueden ser cambiados arbitrariamente, ya que representan valores definidos por la arquitectura del hardware de nuestro sistema.

4.2.4.1.2 Variables y constantes globales del sistema:

Esta sección define todas aquellas variables cuyo contenido es visible para todas las rutinas de nuestro sistema. Las variables y etiquetas definidas en esta sección son vitales para el funcionamiento del sistema, y pueden subdividirse de la siguiente manera:

- Variables de configuración del sistema.
- Variables de control de estado y procesos del sistema.
- Variables de temporización del sistema.
- Variables especiales de la rutina de control de la interface de teclado.

Las variables y constantes de configuración del sistema son aquellas que contienen datos relevantes a la interacción del equipo con la red TCP/IP y el servidor de seguridad. La mayoría de estas variables son configuradas mediante el protocolo BOOTP al inicializarse el sistema; la gran excepción a esta regla es la dirección Ethernet del sistema, la cual es una constante que debe estar definida en el código del sistema, ya que de lo contrario la interface de red no podrá operar.

Las variables de control de estado del sistema se encargan de coordinar los distintos procesos que deben realizarse para el funcionamiento del equipo. Una característica interesante de estas variables es que están organizadas como campos de bits. Cada bit de estas variables es una bandera que cumple una función específica en el sistema. La variable *program_status* contiene banderas de control de procesos del sistema, mientras que *res_status* contiene mayormente banderas necesarias para el proceso de configuración y resolución de recursos del sistema. Decimos mayormente, porque solo cinco bits de la variable se usan para procesos de configuración, dejando los tres restantes disponibles para otras tareas de almacenamiento. Aquí debemos recordar la filosofía de diseño que describimos al inicio de la descripción de nuestro código de cliente; estas variables son el mejor ejemplo de optimización de uso de recursos, ya que hemos usado tan solo dos bytes de almacenamiento para guardar 16 banderas. Este ahorro es importante dados los escasos recursos de memoria de nuestro dispositivo.

Las variables de temporización del sistema se encargan de controlar aquellos procesos que tienen que ejecutarse periódicamente, y de la expiración de condiciones momentáneas en la operación del mismo; por ejemplo, el tiempo que el relay permanece encendido al ser activada la interface del portero eléctrico es controlado por la variable de temporización *relay_out*. De las variables de temporización, la más importante es *interval,* ya que esta controla la ejecución de los procesos periódicos del sistema. La mayoría de las otras variables de temporización utilizan a interval como referencia de tiempo, excepto aquellas cuyos tiempos de conteo sean menores al de *interval (relay_out*, por ejemplo). Hay que destacar que en el sistema existen muchas más variables de temporización que aquellas que están definidas junto con *interval*, estas otras variables están contenidas en las estructuras de datos del sistema, y como ya dijimos, son básicamente múltiplos del periodo definido por *interval*, ya que los procesos que las decrementan son controlados por esta variable.

La sección de variables globales también contiene a las variables que se utilizan para controlar la rutina de interrupción que maneja la interface de teclado. Estas variables merecen nuestra atención por dos razones: primero, su declaración está optimizada para que se implementen en la memoria interna del 80C31, lo cual hace su tiempo de acceso mucho más rápido; y segundo, que estas variables son modificadas por la rutina de interrupción, la cual está escrita parcialmente en lenguaje ensamblador.

4.2.4.1.3 Estructuras de datos del sistema:

Nuestro software cliente basa su operación de manera muy pronunciada en el uso de punteros a estructuras, ya que como veremos más adelante, estos son una forma muy eficiente de manipular paquetes de red. La sección que estamos describiendo contiene las definiciones de las estructuras que corresponden al formato de paquete usado por los distintos protocolos de red implementados en nuestro sistema. Además de los formatos de paquetes, usamos estructuras para organizar el cache de resolución del protocolo ARP (Address Resolution Protocol), así como para guardar la información referente a los datos de identificación enviados hacia y recibidos desde el servidor. Advertimos al lector que es muy difícil entender la implementación de estas estructuras sin un conocimiento exacto de los formatos de paquete de la arquitectura TCP/IP. Afortunadamente, el estilo y organización de estructuras de nuestro sistema siguen el patrón usado por Comer y Stevens en *Internetworking with TCP/IP vol. I y II (1),* por lo que su lectura previa y consulta serán de gran ayuda para el estudio de nuestro código.

Con esto concluimos nuestra revisión de las variables globales y estructuras de nuestro sistema. Como ya recalcamos, el lector debe referirse a las fuentes del código para información detallada sobre las funciones de cada variable. A continuación pasaremos a revisar las diferentes funciones que componen nuestro código, empezando por las rutinas de bajo nivel utilizadas para controlar los periféricos de nuestro sistema.

4.2.4.2 Rutinas de control de hardware: *io_rout.c*

El archivo io_rout.c contiene los procedimientos necesarios para que nuestro software interactúe con los dispositivos de hardware implementados en nuestro sistema. Los periféricos que son controlados mediante estas rutinas son: la interface de red, la interface de teclado y el display fluorescente.

4.2.4.2.1 Rutina de interrupción de teclado: *isr_keyb()*

El caso de la interface de teclado es especial, ya que esta es el único componente de nuestro sistema que es controlado mediante una rutina de interrupción. Para entender las razones de esto, debemos conocer la manera en que trabajan los dispositivos compatibles con el interface de teclado AT. Estos dispositivos, como ya hemos dicho, se comunican con el sistema a través de una línea serial sincrónica bidireccional, la cual en nuestro caso ha sido solo parcialmente implementada, ya que nuestro sistema solo puede recibir, y no enviar, datos de la interface de teclado. Debido a que esta interface no implementa ningún método de control de flujo, no hay ninguna manera de detener el flujo de datos que viene desde la misma si el sistema no es capaz de procesarlos lo bastante rápido. El resultado de esto, es que si nuestro sistema es muy lento, los datos recibidos de la interface de teclado tienden a superponerse uno al otro, y resultan inservibles. Ya adelantamos que una de las soluciones para esto es subir al máximo posible la velocidad de reloj del sistema; pero esto no es suficiente, dado que nuestro software no implementa un mecanismo determinístico de control de procesos. Entonces,

la única solución viable para implementar esta rutina de manera confiable, es configurarla como una interrupción de hardware.

La forma de implementar rutinas de interrupción en el 8051/31 varía de acuerdo al entorno de diseño que se utilice; en AVCASE 51, se tiene que implementar un macro especial llamado *set_vector(dir_int, nombre_rutina),* el cual se encarga de crear el puntero necesario en la tabla de vectores de interrupción del 80C31(este macro está implementado en el archivo principal del sistema) para direccionar la interrupción a nuestra rutina. Si revisamos el código, la rutina de control de teclado *isr_keyb()* tiene el calificador especial *interrupt*; este le indica al compilador C que esta es una rutina de interrupción, y su código se debe compilar para que se ejecute lo más rápido posible. Sin embargo, esto no es suficiente, ya que las funciones que tiene que ejecutar esta rutina, que implican desplazamiento de registros, y uso de carry, son difíciles de implementar eficientemente en C, así que decidimos escribir la rutina parcialmente en assembler.

El funcionamiento de la rutina es bastante simple, cuando el sistema detecta la interrupción generada por los datos entrantes, la rutina es activada, y procede a desplazar un bit de datos hacia el sistema (los bits van siendo almacenados en la variable *key_temp*). Cada bit sucesivo que se presenta en la interface de teclado causa una ejecución de la rutina. Cuando el número de bits llega a once (start bit + 8bits de datos + bit de paridad + stop bit), se ha recibido un byte completo de datos, así que la rutina determina entonces si este byte corresponde a un código de carácter o no. Para

comprender este sistema de identificación de código, debemos entender que un teclado AT no reacciona a la activación de una tecla simplemente enviando el código de la tecla, sino que al presionarse esta, envía el código de la tecla mientras esta este presionada, y cuando esta se suelta, envía un código de finalización, el cual es el número hexadecimal 0xF0, seguido nuevamente del código de la tecla. Nuestra rutina ignora (pero almacena en *last_key*) los códigos que recibe, a menos que sean precedidos por un código de finalización. En ese momento la rutina indexa el código, y lo convierte a formato ASCII usando la tabla de indexación *key_map*. Si revisamos esta tabla, veremos que se compone principalmente de ceros, esto se debe a que nuestra tabla solo recoge caracteres numéricos, el resto de caracteres que puedan venir de un dispositivo de teclado serán ignorados. Si el carácter recibido es válido, este se almacena en el buffer de teclado *key_buff*, desde donde puede ser recogido por el resto de rutinas del sistema. Cuando este buffer se llena, la reacción de la rutina es desactivar la interrupción de teclado, para que si este se avería, o esta siendo manipulado ociosamente por los usuarios, la repetida ejecución de la rutina no afecte el rendimiento del sistema. La interrupción es reactivada posteriormente por los manejadores de eventos del sistema.

4.2.4.2.2 Rutina de conteo de seguridad: *isr_watchdog()*

Esta rutina fue añadida al final del proceso de desarrollo del software de nuestro proyecto, específicamente cuando se hizo la modificación final a la red de inicialización

del sistema (ver sección 3.4.4), la cual hizo posible la implementación de este tipo de rutina.

La función realizada por esta rutina se denomina normalmente *watchdog timer* (contador de vigilancia), y consiste en vigilar que el sistema no se congele o estanque por fallas de hardware, picos de corriente, o cualquier condición que lleve a que el flujo de instrucciones se detenga. Para implementarla se utiliza uno de los contadores de hardware del 80C31, el cual es iniciado y reinicializado por el flujo de actividad del sistema; si en algún momento se detiene la actividad del sistema, el contador no es reinicializado, y al llegar a su valor máximo (0xFFFF) genera una interrupción que activa la rutina que estamos revisando. La rutina tiene como única función la total inicialización del sistema, de manera que este pueda recobrarse de la condición que generó su congelamiento. Para esto la rutina realiza todas las operaciones necesarias para retornar al 80C31 a su estado normal de inicialización, borra el contenido de la memoria RAM (externa e interna), y finalmente regresas al sistema a la dirección inicial de ejecución (0x0000). Al hacer esto, el sistema ejecuta nuevamente la rutina de inicialización del hardware y software, con lo que regresará a su operación normal en la mayoría de situaciones.

4.2.4.2.3 Rutina de lectura/escritura a buffers de red: *netb_rw()*

Esta es una rutina muy simple, la cual ejecuta el proceso de lectura y escritura hacia el puerto de datos que conecta a la unidad central de proceso de nuestro sistema con la

interface de red. Esta rutina hace la escritura/lectura de datos tras asegurarse que el ST-NIC ha activado la señal PRQ (Port Request), que recordamos que el ST-NIC usa para indicar al sistema que está listo para iniciar un ciclo de DMA remoto. Tras confirmar que PRQ está activado, la rutina escribe o lee según el estado del argumento *sel*. Si la orden es escribir, el valor del argumento *c* es escrito al puerto; si es leer, se lee el dato existente en el puerto, y se lo retorna en *c*. Podemos notar que el proceso de lectura/escritura al puerto es un simple acceso al espacio de memoria externa, direccionado por STNIC_IO hacia la interface de red. Hay que destacar que esta rutina es para interactuar con el buffer de paquetes de la interface de red, a través del DMA remoto del ST-NIC. Esta rutina no sirve para accesar los registros de configuración del ST-NIC.

4.2.4.2.4 Rutina de lectura/escritura a registros de ST-NIC: *netr_rw()*

Esta es la rutina utilizada para accesar los registros de configuración del controlador de red ST-NIC. Al igual que la rutina anterior, esta se orienta a la lectura/escritura de un byte de datos. Sin embargo, como vemos al examinar el código, esta rutina es más complicada.

La primera actividad realizada por la rutina es escribir la dirección del registro al que se desea acceder en las líneas RA0-RA3 (de selección de registro) del ST-NIC. Aquí podemos notar otro ejemplo de ahorro de recursos, ya que para evitar el uso de otro argumento en esta rutina, la dirección de registros y la variable de selección de lectura y

escritura han sido combinadas en un solo byte. Esto no es tan difícil de interpretar como parece, ya que los números se presentan en hexadecimal, de manera que se dividen fácilmente en bloques de cuatro bits.

Una vez configurados RA0-RA3, se activa o desactiva la señal Sr/w según estemos en una operación de lectura o escritura, y se procede a bajar la señal CS para poner al ST-NIC en modo esclavo.

Como ya mencionamos, aquí nuestro sistema debe esperar a que el ST-NIC responda al pedido de ingreso a modo esclavo bajando la señal ACK. Una vez que ACK está abajo, se puede iniciar el ciclo de lectura/escritura de los registros.

Aquí, sin embargo, se presenta una falla de diseño de nuestro sistema (no detectaba antes de la implementación del mismo), ya que nuestro ciclo de lectura de datos presenta problemas, que aunque resueltos mediante trucos de software, lo hacen más lento que el ciclo de escritura de datos. El problema se debe a que nuestra lógica programable no escribe el valor del registro del ST-NIC al puerto hasta que CS suba. Ahora, si hacemos el ciclo de lectura inmediatamente después de subir CS, la lógica asume que se está haciendo una lectura de modo DMA, ya que CS está arriba, por lo que la señal RACK se activa erróneamente. Para evitar este problema, nuestro ciclo de lectura sube CS, lo cual escribe el dato al puerto, y lo vuelve a bajar para ejecutar la lectura del puerto, efectivamente leyendo el registro dos veces (la segunda lectura es ignorada).

El ciclo de escritura carece de esta falencia, ya que el sistema escribe el dato de configuración al puerto antes de subir CS, lo que indica al ST-NIC que el ciclo de escritura ha terminado.

Hay que destacar que aunque esta rutina adolece de estos problemas, no se ha comprobado ningún efecto adverso en el sistema que pueda ligarse a esta falla de diseño. La corrección de este error implicaría el rediseño de las conexiones eléctricas de la lógica programable del sistema, por lo que se considera impráctica de realizar, dadas las pocas consecuencias de la existencia del error.

4.2.4.2.5 Rutina de lectura/escritura al display de datos: *dis_rw()*

La primera función que debe realizar esta rutina es revisar que el display esté listo para ser accesado. Para esto se hace un ciclo de lectura del display, el cual nos retornará en el bit más significativo el valor de la bandera de acceso del display. Si la bandera está alta, se repite este ciclo de lectura hasta que la bandera baje, y se procede con le resto de la rutina.

El proceso de lectura/escritura es igual al usado en la rutina de acceso a buffers de red, con la diferencia de que según el valor de RS que se desee (rs indica si se está escribiendo datos a la pantalla, o si se está enviando comandos al display), la dirección

de memoria a la que se accesa es distinta. Esta función también implementa ahorros en recursos mediante la unificación de sus argumentos.

4.2.4.3 Rutinas de manejo de display: *display.c*

Las rutinas y macros implementadas en este archivo son utilizadas como complementos de automatización de la rutina *dis_rw()* ya descrita. Esto es necesario, ya que esta rutina al igual que todas las implementadas en el archivo *io_rout.c*, están orientadas a transferencias byte por byte, lo cual las hace engorrosas de utilizar directamente, a menos que se desee escribir/leer únicamente un byte de datos. Además el archivo contiene macros de automatización de diversas tareas de control del display. Estos macros son solo para conveniencia del usuario, ya que son simples reemplazos de la función *dis_rw()*, con los argumentos que sean necesarios.

4.2.4.3.1 Rutina de escritura de cadenas de caracteres: *display()*

Esta rutina sirve para enviar una cadena entera de caracteres al display, mediante un solo llamado de función. Esta rutina ofrece la ventaja adicional de que las cadenas de caracteres pueden ser almacenadas directamente en el argumento de llamado a la función, evitándonos tener que declararlas como constantes. Como veremos a todo lo largo del código de nuestro sistema, se prefiere el uso de esta rutina en todos los casos en que se deba enviar datos al display.

4.2.4.3.2 Macros de automatización de manejo del display

Los macros que están definidos en este archivo son, como ya dijimos, sustituciones del comando que se debe dar al display para obtener el resultado deseado. En estos macros están definidas todas las acciones que el usuario tendrá necesidad de ordenar al display: borrado de pantalla, inicialización de cursor, encendido y apagado de la pantalla y colocación del cursor en sitios específicos.

Sobre esto último, debemos decir que aunque el display tiene solo 20 columnas de ancho, el tamaño de su memoria de caracteres es el doble de eso, así cada línea del display puede tener hasta 40 caracteres, aunque solo 20 sean visibles en un momento dado.

El último macro listado, *set_cgram()*, se utiliza para programar las localidades de memoria usadas para almacenar caracteres generados por el usuario. Este comando es usado por nuestro sistema para generar los iconos de estado del dispositivo.

4.2.4.3.3 Rutina de control de iconos del sistema: *icon_display()*

Esta rutina es la que se encarga de presentar los iconos de estado del sistema en la pantalla. Estos iconos son gráficos generados por el sistema que indican el estado de las dos más importantes variables de configuración de nuestro equipo: la dirección IP del

equipo, y la dirección IP del servidor de seguridad (El formato de los iconos utilizados es mostrado en el Capítulo V).

La rutina utiliza las banderas de resolución de estas variables, para decidir si mostrar o no los iconos de estado correspondientes.

4.2.4.4 Rutinas de inicialización del sistema: *h_init.c*

4.2.4.4.1 Rutina de inicialización: *h_init()*

Esta rutina se encarga de asegurar que el estado inicial de nuestro sistema sea el adecuado para que su operación sea correcta. La primera actividad realizada por la rutina es la inicialización de los registros de configuración del controlador 80C31 y de las líneas de entrada y salida que llegan hacia este controlador desde la interface de teclado y la interface del portero eléctrico. Después, la rutina configura la interrupción externa que se activará cuando haya actividad en la interface de teclado. Cabe destacar que esta interrupción es configurada mas no activada, su activación no se hará hasta que el sistema haya sido totalmente configurado por el protocolo BOOTP. Además, se configura la interrupción que controlará el contador de seguridad del sistema; debido a la función que realiza, esta interrupción estará continuamente activa, a diferencia de la interrupción de la interface de teclado.

Una vez realizadas las configuraciones del hardware del controlador, la rutina pasa a inicializar las variables globales del sistema, incluyendo las variables de configuración de TCP/IP.

El siguiente paso que se realiza es la configuración correcta del display fluorescente. Las configuraciones más importantes del display son la configuración del mismo para operar en un bus de 8 bits, y la programación de los caracteres especiales que se utilizarán para mostrar los iconos de estado de configuración. Una vez terminada la configuración del display, se procede a mostrar el mensaje de inicialización del sistema en la pantalla.

Ahora se procede a la inicialización de la interface de red, siguiendo al pie de la letra el procedimiento de inicialización prescrito en la documentación del ST-NIC. De este procedimiento, los pasos más destacables son:

- Configuración del anillo de recepción de datos y buffer de transmisión. Este es el subsistema encargado de manejar los buffers de red, el anillo de recepción se compone de varios buffers de 256 bytes de largo. Lo que debemos configurar es el espacio de la memoria de buffer que se usará para ubicar el anillo de recepción. Nuestro código lo ubica en el rango de 0x0600 – 0x1FFF, lo cual nos da aproximadamente 6.5K de almacenamiento de datos entrantes. El resto del espacio de memoria de buffer (1536 bytes) se utiliza para el almacenamiento de paquetes salientes.

- Configuración de la dirección física de la interface. Esta es la dirección almacenada en la variable *ethadd* del encabezado principal del sistema, y la interface de red no puede funcionar si no está definida. Obviamente, es importante que este número sea distinto al valor usado en las otras interfaces de red presentes en el segmento. (En realidad tiene que ser único; en nuestro caso, usamos el valor inicial del rango de experimentación de National Semiconductor).

- Configuración del registro de máscara de interrupción. Este registro controla los eventos de la interface de red que generan interrupciones a la unidad central de proceso. Hay varios eventos que pueden generar interrupciones, pero para nuestros propósitos solo es necesario que se generen interrupciones cuando se reciben paquetes, y cuando el anillo de recepción se llena, por lo que solo esos dos eventos están activados en nuestra configuración.

La tarea final de inicialización que realiza nuestra rutina, es la inicialización de los miembros del cache de ARP. Todos los miembros son etiquetados como libres (AE_FREE), y los punteros de las colas de paquetes son inicializados a cero.

4.2.4.4.2 Rutinas de control de interrupciones

El archivo *h_init.c* posee también unas pequeñas pero importantes rutinas utilizadas para activación y desactivación de interrupciones en nuestro sistema. Aunque la

activación de interrupciones no es más complicada que el seteo de un bit en los registros del controlador 80C31, existen consideraciones especiales que hacen necesario el uso de estas rutinas. El problema se da cuando se desactiva la interrupción de teclado, y mientras está desactivada se produce un evento en el teclado. Debido a la forma de operar de la arquitectura 8051/31, si no se inicializan manualmente las banderas de interrupción en el registro TCON del 80C31 al activar nuevamente la interrupción, esta queda desabilitada, y la interface de teclado no responde. Para evitar este problema, nuestra rutina de activación de interrupciones inicializa TCON, y también las líneas de entrada de datos de la interface de teclado antes de activar la interrupción.

4.2.4.5 Rutinas de entrada/salida Ethernet: *ethernet.c*

Este archivo contiene las rutinas que interactuan con la interface de red del sistema, para envío y recepción de paquetes Ethernet, así como para el procesamiento de eventos de red.

4.2.4.5.1 Rutina de salida Ethernet: *ethernet_out()*

La rutina de salida Ethernet se encarga de transferir los paquetes salientes a la memoria de buffer de la interface de red (mediante el uso del DMA remoto), y de dar al ST-NIC el comando necesario para que el paquete transferido sea inmediatamente transmitido a la red Ethernet. El resto del proceso de transmisión (codificación, adición de CRC, detección de colisiones y reenvió) es automáticamente manejado por el ST-NIC.

Para empezar, podemos ver que la rutina tiene como argumento de entrada un puntero a una estructura de paquete Ethernet, esta es una fórmula que será muy común en nuestro sistema, ya que nuestro manejo de paquetes se basa totalmente en punteros a estructuras, que como veremos es una forma muy eficiente de manipular las distintas capas de protocolo de un paquete de red.

Lo primero que nuestra rutina hace es confirmar que la interface de red este activada, chequeando la bandera de sistema correspondiente; si la interface está desactivada (normalmente por un desborde del buffer de entrada) la transmisión se descarta. Hay que destacar que no se da aviso a las capas superiores de protocolo de que se descartó o no la transmisión. Como veremos más adelante, todas las retransmisión de nuestro sistema son realizadas usando los contadores de expiración.

Inmediatamente después, la rutina configura los registros del ST-NIC para empezar el ciclo de escritura del DMA remoto, con el que se transferirá el paquete al buffer de red. Un detalle importante, es que debido a fallas de diseño del ST-NIC (delineadas en su documentación) se debe hacer un proceso de lectura previo (llamado "dummy read" por la documentación del ST-NIC), el cual tienen como propósito único el asegurar que la señal PRQ esté activada correctamente al iniciar el ciclo de lectura. Una vez realizada la prelectura, se configura el DMA remoto para transmitir el paquete hacia el buffer. Esta configuración requiere que conozcamos el largo del paquete a transmitir; para esto hemos precedido nuestra estructura de paquete Ethernet por un campo adicional *etherp.len* (que no es transmitido), el cual contiene el largo del paquete. Si el largo del

paquete es menor al mínimo requerido por la especificación Ethernet (60 bytes), se añade relleno al mismo. Una vez configurados los registros del DMA remoto y los registros de transmisión de paquetes, ejecutamos el proceso de transferencia al buffer de red, y damos el comando necesario para que el paquete sea transmitido a la red.

La función, como ya hemos explicado, no retorna ningún indicador de éxito o fracaso de la transmisión, esto es posible ya que nuestro sistema implementa facilidades de retransmisión en las capas superiores de protocolo.

4.2.4.5.2 Rutina de recepción Ethernet: *ethernet_in()*

La rutina de recepción de paquetes Ethernet realiza las funciones necesarias para transferir el paquete recibido desde el buffer de la red hacia la memoria de sistema. Además la rutina se encarga de llamar al procedimiento adecuado, según el tipo de paquete, para que continúe con el procesamiento del mismo.

La transferencia del paquete se realiza mediante un comando automatizado del DMA remoto (*send packet*), el cual nos permite recibir un paquete entero sin tener que configurar los registros del DMA remoto, este comando se ejecuta, y todo lo que hacemos es empezar la recepción del paquete. Sin embargo, antes de recibirlo, necesitamos sitio donde almacenarlo. Para esto nuestro sistema utiliza las funciones de asignación dinámica de memoria del lenguaje C, las cuales buscan un bloque de memoria desocupado en el cual poder almacenar el paquete, y retornan un puntero hacia

el bloque. Durante el diseño del sistema hubo cierto debate sobre la conveniencia de utilizar este método de almacenamiento versus el uso de cadenas de buffers de tamaño fijo. Se decidió utilizar asignación dinámica, ya que nos pareció mucho más eficiente en el manejo de recursos de memoria.

Una vez asignado un espacio de almacenamiento, el paquete se empieza a transferir desde el buffer de red. En cuanto se finaliza la transmisión, la rutina revisa el campo de tipo de paquete especificado por el formato Ethernet II, para saber que tipo de contenido tiene nuestro paquete. Si el contenido corresponde a uno de los dos protocolos superiores usados por nuestro sistema, que son IP y ARP, el paquete es enviado a la rutina correspondiente, si su contenido es desconocido, el paquete es descartado.

Una vez procesado el paquete, la rutina pasa a revisar si existen más paquetes esperando en el buffer de recepción del ST-NIC, si el buffer está vacío, la rutina procede a desactivar la bandera de recepción de paquetes, si no lo está, la rutina no modifica la bandera y retorna; los paquetes restantes no son procesados inmediatamente para dar tiempo de ejecución al resto de procesos de nuestro sistema.

4.2.4.5.3 Rutina de identificación de eventos de red: *poll_net()*

Esta rutina es llamada por el manejador de eventos de red del sistema cuando detecta una interrupción generada por el ST-NIC. La función de la rutina es preguntar al ST-NIC que tipo de evento de red es el que ha sucedido.

Nuestra interface de red está configurada para generar interrupciones en solo dos tipos de eventos, cuando se recibe un paquete, y cuando se desborda el anillo de recepción del buffer de red. Nuestra rutina entonces implementa manejadores para procesar estos dos tipos de eventos, los cuales cabe decirse, pueden suceder simultáneamente.

Si el evento es la recepción de un paquete, la rutina se limita a activar la bandera de paquete recibido, la cual será usada posteriormente por el sistema para llamar a la rutina de recepción Ethernet. En cambio si el evento es un desborde del buffer de red, nuestra rutina debe ejecutar el procedimiento impuesto por National Semiconductor para la recuperación de desbordes de buffer. Este procedimiento consiste en desactivar la interface de red (parando la recepción y envió de paquetes a la red), y proceder a vaciar el buffer de red mediante repetidos llamados a la rutina de recepción Ethernet. Es posible que al desbordarse el buffer se haya cancelado alguna transmisión pendiente, por lo que la rutina se encarga de retransmitir cualquier paquete cuya transmisión haya sido truncada por el desborde del buffer. Hay que destacar que la documentación del ST-NIC hace mucho énfasis en la implementación al pie de la letra de este procedimiento, por lo que lo hemos reproducido fielmente. Una vez finalizada la evacuación del buffer, la interface de red es activada nuevamente, y la rutina retorna a la operación normal del sistema.

4.2.4.6 Rutinas del Protocolo ARP: *arp.c*

Las rutinas de este archivo se encargan de procesar paquetes ARP entrantes, generar pedidos ARP cuando se requiere una resolución de dirección IP, y de administrar el cache de resolución de ARP. Como el lector podrá ver al revisar el código, las rutinas utilizadas son básicamente las mismas que describen Comer y Stevens en *Internetworking with TCP/IP Vol. II (1)*, las únicas diferencias funcionales entre nuestro código ARP y el desarrollo en el texto mencionado, es que nuestro código está modificado para funcionar en nuestro sistema, el cual no es multitarea, a diferencia del sistema operativo descrito en el texto. Además, nuestro código maneja de manera estática las colas de paquetes ARP, reservando un espacio definido para punteros a paquetes encolados, a diferencia de Comer y Stevens, que usan colas dinámicas provistas por su sistema operativo. De todas maneras, el estudio del código provisto por Comer y Stevens será de gran ayuda para la comprensión de nuestras rutinas ARP.

4.2.4.6.1 Rutina de entrada de ARP: *arp_in()*

Esta rutina es llamada por la rutina de entrada Ethernet cuando se recibe un paquete destinado al protocolo ARP. La primera acción que la rutina realiza es confirmar que el paquete ARP es para la arquitectura de red Ethernet, y para el protocolo IP, usando los campos de identificación de hardware y de protocolo del paquete. Si el paquete es efectivamente válido, la rutina pasa a determinar si el paquete tiene como destino nuestro sistema, comparando su dirección IP de destino con la nuestra. Si el paquete no va dirigido a nuestro equipo, este es descartado. Aquí hay que advertir que este

procedimiento es contrario a lo dictado por el protocolo ARP, el cual dice que aunque el paquete no vaya dirigido a nuestro equipo, el sistema debe procesarlo igualmente, para ahorrar pedidos posteriores. Sin embargo, nosotros consideramos que ya que nuestro sistema mantendrá interacción solo con un número muy limitado de máquinas (probablemente el servidor de seguridad sea la única máquina con la que interactúe), es mejor descartar pedidos que no nos corresponden, para poder mantener el tamaño de nuestro cache ARP al mínimo posible.

Si el paquete va dirigido a nuestro sistema, la rutina revisa el cache ARP en busca de miembros relevantes (mediante la rutina *arp_find()*), y los revalida si existen, los crea si no existen(mediante *arp_add()*) y si hay miembros relevantes que están pendientes (esperando una resolución) ejecuta la rutina *arpq_send()*, que se encarga de transmitir los paquetes encolados esperando la resolución de ese miembro del cache.

Posteriormente, la rutina revisa si el paquete es un pedido de ARP, en cuyo caso se debe generar una respuesta. Para generar la respuesta, la rutina reutiliza el espacio de memoria usado por el paquete entrante, y lo modifica para convertirlo en un paquete de respuesta de ARP. Una vez hecha la conversión, y llenados los campos relevantes, el paquete es pasado a la rutina de salida Ethernet para ser transmitido a la red.

4.2.4.6.2 Rutina de búsqueda en el cache ARP: *arp_find()*

Esta rutina tiene como única función la búsqueda entre los miembros del cache de ARP de uno cuya dirección IP sea igual a la que se ingresó como argumento de la función. Si encuentra un miembro, la función retorna un puntero hacia este miembro, si no, retorna cero.

4.2.4.6.3 Rutina de adición de miembros al cache ARP: *arp_add()*

La función realizada por esta rutina es la de añadir nuevos miembros al cache de ARP cuando se la llama. Hay que destacar que el espacio del cache de ARP es limitado (nuestro cache tiene un máximo de 3 miembros actualmente), de manera que a veces habrá que borrar algún miembro anterior del cache para acomodar el nuevo miembro; para realizar esta función, la rutina *arp_alloc()* es ejecutada. Una vez que nuestra rutina ha conseguido espacio en el cache para el nuevo miembro, este es inicializado (estado puesto como resuelto, tiempo de vida llevado al máximo, y cola de paquetes llevada a cero), y sus campos de direcciones son llenados con la información obtenida del paquete ARP que generó el nuevo miembro.

4.2.4.6.4 Rutina de envío de paquetes encolados: *arpq_send()*

Esta rutina se encarga de revisar si un miembro que ha sido marcado como resuelto tiene paquetes encolados esperando a ser transmitidos; si el miembro tiene paquetes en

su cola, estos son enviados a la rutina de salida Ethernet. Después de transmitir los paquetes, la cola es reinicializada para que pueda aceptar más paquetes.

4.2.4.6.5 Rutina de eliminación de colas: *arp_dq()*

Cuando un miembro pendiente del cache ARP expira (su contador de tiempo de vida llega a cero), y ya se ha hecho el máximo número de pedidos ARP permitidos para ese miembro, el miembro debe ser descartado y su cola de paquetes eliminada. Esta es la función de la rutina *arp_dq()*. La rutina chequea si el miembro expirado tiene paquetes pendientes, y procede a liberar el espacio de memoria usado por estos paquetes y a marcar las colas del miembro como libres, una vez finalizada esta tarea la rutina retorna, y el miembro puede ser marcado como libre para su posterior uso.

4.2.4.6.6 Rutina de asignación de espacio en el cache: *arp_alloc()*

Esta rutina se encarga de buscar entre los miembros del cache de ARP para encontrar uno que se encuentre disponible para usarse. Si la rutina no encuentra miembros disponibles, se debe eliminar uno de los miembros utilizados. Para decidir cual miembro borrar, la rutina utiliza la variable estática *ae_next*, la cual apunta en secuencia de anillo a los distintos miembros del cache. Hay que destacar que esta política de borrado de miembros no es arbitrariamente impuesta por nosotros, sino que es un requerimiento del protocolo ARP.

4.2.4.6.7 Rutina de envío de pedidos ARP: *arp_send()*

La rutina *arp_send()* se encarga de generar y transmitir los pedidos ARP que deben ser inundados a la red para resolver una dirección IP.

La rutina primeramente busca espacio de memoria en el que pueda construir el paquete de pedido. Si no se encuentra espacio, la rutina finaliza; esto no es un problema, ya que los mecanismos de retransmisión de ARP intentarán nuevamente hacer el pedido después de un periodo de tiempo determinado. Una vez que se ha conseguido el espacio para el paquete, la rutina empieza a llenar los distintos campos del paquete ARP, además de los campos del paquete Ethernet en el cual este último va encapsulado. Esto es importante, ya que el lector probablemente se habrá fijado que la rutina de salida Ethernet no hace ningún tipo de construcción de paquetes; los paquetes que llegan a esta rutina deben estar listos para transmitirse hacia la interface de red, así que es responsabilidad de la capa superior de protocolo la construcción del paquete Ethernet. Obviamente esta división de funciones podrá parecer una violación de la delegación de operación de las distintas capas de protocolo, pero se debe a consideraciones puramente prácticas de la implementación del software, y es común a todo lo largo de nuestro proyecto.

Una vez generado el paquete Ethernet con su pedido ARP encapsulado, este es enviado a la rutina de salida Ethernet para su transmisión.

4.2.4.6.8 Rutina de administración del cache ARP: *arp_timer()*

La rutina de administración de ARP, es el primer proceso de mantenimiento periódico que revisamos en nuestro sistema. La rutina es ejecutada cada cierto tiempo por el lazo principal del sistema, y su función es controlar la expiración de los miembros del cache de ARP.

La rutina revisa cada miembro del cache. Si el miembro está libre, es ignorado y se pasa al siguiente. Si el miembro está resuelto y su tiempo de vida ha expirado, el miembro es liberado. Los miembros que están pendientes de resolución y han expirado son revisados para ver si el número de pedidos ARP que han enviado ha llegado al máximo permitido, si es así el miembro es liberado y sus colas de paquetes son descartadas. Si el miembro aún no ha hecho su máximo número de pedidos, se envía un nuevo pedido de inmediato. Todos los miembros restantes del cache decrementan su tiempo de vida.

4.2.4.7 Rutinas de los protocolo IP e ICMP: *ip.c*

Las rutinas contenidas en este archivo se encargan de procesar paquetes entrantes del protocolo IP, encapsular paquetes salientes con el encabezado de IP, y realizar los procedimientos básicos de ruteo soportados por nuestro sistema. Además, este archivo contiene las rutinas de control de entrada y salida del protocolo ICMP. Estas rutinas han sido incluidos en este archivo considerando que ICMP es una parte integral del protocolo IP.

4.2.4.7.1 Rutina de entrada de IP: *ip_in()*

Esta rutina es la encargada de recibir los paquetes entrantes que van dirigidos al protocolo IP. La rutina empieza su labor revisando que la versión del encabezado IP sea correcta (versión 4), y que el largo del encabezado sea de 20 bytes. El encabezado IP puede ser más largo que eso si tiene incluidos los campos de opciones de IP, pero ya que nuestro sistema no soporta el procesamiento de opciones de IP, los paquetes que las incluyen son descartados. Esto nos ayuda a mantener simple la implementación de nuestro código.

El siguiente paso es determinar si el paquete debe ser procesado o no por nuestro sistema, observando si su dirección IP de destino es igual a la nuestra, o si es un paquete de broadcast. Si el paquete no corresponde a ninguno de estos dos casos, este debe ser descartado. Inmediatamente después pasamos a revisar si el paquete IP ha sido fragmentado, en cuyo caso deberá ser descartado, ya que nuestro sistema no soporta fragmentación y reconstrucción de paquetes IP. Esto no es así simplemente por la necesidad de mantener el código compacto, sino también porque nuestro sistema tiene únicamente 8 Kbytes de RAM, mientras que el largo máximo de un paquete IP es de 64Kbytes. Debido a esto, no tiene sentido que implementemos fragmentación de paquetes, si nuestro sistema no va a poder manejar todos los tamaños de paquetes entrantes. El límite del tamaño de los paquetes IP que puede recibir nuestro sistema está limitado entonces por el tamaño máximo de paquete que puede ser enviado en el medio físico Ethernet (1500 bytes).

Una vez que determinamos que el paquete no está fragmentado y por lo tanto puede ser procesado, pasamos a discriminar que tipo de protocolo superior de la arquitectura TCP/IP está encapsulado en el paquete. Nuestro sistema soporta únicamente la recepción del protocolo de transporte UDP, así como del protocolo de diagnóstico ICMP; los paquetes que contengan algún otro protocolo de red serán descartados.

4.2.4.7.2 Rutina de salida de IP: *ip_out()*

Esta rutina se encarga parcialmente de la encapsulación de paquetes con el encabezado IP, y realiza las funciones de ruteo de paquetes implementadas en nuestro sistema.

Cuando decimos que la rutina realiza solo parcialmente la encapsulación de paquetes, nos referimos a que la rutina llena solo una parte de los campos necesarios para transmitir el paquete. Los campos restantes deben ser llenados por los protocolos superiores de la arquitectura TCP/IP. En otras implementaciones de IP, esta necesidad de propagar información de arriba hacia abajo en la estructura de protocolos se maneja mediante argumentos de funciones; pero en nuestro caso, debido a la necesidad de ahorrar espacio en la pila del sistema, decidimos dejar la implementación de estos campos como requisitos previos a la ejecución de la rutina de salida de IP.

La rutina debe además calcular la sumatoria del encabezado IP, para lo cual se debe encerar el campo de sumatoria, calcular esta (llamando a la función *ck_sum()*), y colocar el valor obtenido en el campo de sumatoria del encabezado.

Aparte de llenar los campos relevantes en el paquete IP y el paquete Ethernet, y calcular la sumatoria del encabezado IP, la rutina debe realizar la función de ruteo del paquete. Nuestro sistema soporta ruteo mediante el uso de máscaras de subred, con ruteo de clases como alternativa secundaria. Además, el sistema soporta la definición de un gateway, o ruteador por defecto.

Para rutear, el sistema primero decide si el destino del paquete corresponde a la red local IP o no, esta comprobación se realiza mediante el uso de la máscara de subred definida en el sistema, la cual puede haber sido configurada por el servidor BOOTP, por un pedido ICMP, o calculada a partir de la clase de dirección IP del sistema.

Si el paquete es para la red local, la rutina consulta el cache de ARP para encontrar la dirección física correspondiente al destino del paquete. Si no existe un enlace con una dirección física, el paquete es encolado y se envía un pedido ARP.

Si el paquete no es para la red local, la rutina comprueba si es que este es un broadcast de red. Nuestro sistema soporta únicamente broadcasts locales, así que la dirección Ethernet es inmediatamente llenada de unos, y el paquete es transmitido.

Si el paquete no es un broadcast, se intenta enviarlo al gateway definido para el sistema, si el gateway no está definido, el paquete es descartado.

4.2.4.7.3 Rutina de cálculo de sumatoria: *ck_sum()*

El estándar de IP indica que la sumatoria del encabezado corresponde al complemento unitario de la suma de todos los enteros de 16 bits del encabezado. Esta rutina de sumatoria es casi exactamente igual a la propuesta por Comer y Stevens (1), con la excepción de que, dado que nuestro sistema está fuertemente orientado a operaciones de ocho bits, el argumento de largo de la sumatoria ha sido pasado de enteros de 16 bits a caracteres de ocho bits.

La función realiza primeramente la sumatoria de 16 bits del bloque de datos que se le indique procesar, y la almacena en un entero de 32 bits. Esto le permite almacenar cualquier tipo de avance que pueda tener la sumatoria. Una vez terminada la sumatoria, esta es convertida nuevamente a un entero de 16 bits, mediante operaciones de desplazamiento y suma. Finalmente, se retorna el complemento a uno del resultado obtenido.

4.2.4.7.4 Rutina de entrada ICMP: *icmp_in()*

Esta rutina se encarga de procesar los mensajes ICMP que llegan a nuestro sistema. Nuestra implementación solo reacciona a dos de los mensajes ICMP implementados por el estándar: El mensaje de pedido de eco, y el mensaje de respuesta a un pedido de máscara de subred.

Si el paquete es un pedido de eco, nuestra rutina envía la respuesta usando el mismo espacio de memoria en que está almacenado el pedido. Hay que destacar que si el tamaño del pedido es impar, la rutina debe añadir un byte adicional al paquete para que la función de cálculo de sumatoria trabaje correctamente. Dado que se reutiliza el mismo espacio del pedido, no tenemos que hacer ningún procesamiento del área de datos del pedido de eco.

Si el paquete es una respuesta a un pedido de máscara de subred, nuestra rutina compara su campo de identificación con el que inserta nuestra rutina de envío de pedidos de máscara (los cuatro bytes iniciales de la dirección física del sistema). Si son iguales, se considera válida la respuesta, y se reemplaza la máscara de subred del sistema por la que llegó en el paquete. Ya que normalmente solo los equipos configurados como ruteadores responden a pedidos de máscara, si no existe un gateway definido en el sistema, la rutina asigna la dirección de origen del paquete como gateway del sistema.

4.2.4.7.5 Rutina de salida ICMP: *icmp_out()*

Esta rutina tiene como único uso la generación de pedidos de máscara de subred. Los pedidos de máscara son generados por nuestro sistema si durante el proceso de inicialización de configuración mediante el protocolo BOOTP, no se recibió la información de máscara.

La rutina debe conseguir espacio de memoria para construir el paquete de pedido de máscara, y llenar los campos correspondientes del formato. El único punto destacable de este proceso, es que para llenar el campo de identificación del pedido ICMP, la rutina utiliza, como ya dijimos, los cuatro primeros bytes de la dirección Ethernet. Hacemos esto, ya que no requerimos ningún procedimiento complicado para confirmar estos pedidos, los cuales serán hechos solo durante la inicialización del sistema, y eso solo si el servidor BOOTP no puede proveer una máscara de subred.

4.2.4.8 Rutinas del protocolo UDP: *udp.c*

Las rutinas aquí descritas implementan el único protocolo de transporte que implementamos en nuestro sistema: UDP. Las rutinas se encargan de recibir y procesar paquetes UDP entrantes, y de procesar paquetes UDP salientes antes de pasarlos a IP para su ruteo.

4.2.4.8.1 Rutina de entrada de UDP: *udp_in()*

Esta rutina recibe de IP los paquete calificados como pertenecientes al protocolo UDP. La primera actividad que realiza la rutina es la confirmación de integridad del paquete mediante el cálculo de la sumatoria del mismo (para lo cual llama a la función *udp_cksum()*). Ya que UDP implementa el uso de sumatorias de manera opcional, la rutina revisa si en realidad se ha incluido una sumatoria en el paquete entrante.

Una vez confirmada la validez de la sumatoria del paquete, nuestro software procede inmediatamente a la discriminación del destino del paquete según el valor contenido en el puerto de destinatario del mismo. Nuestro sistema implementa tres posibles destinos de paquetes UDP: el primero es nuestra aplicación de seguridad, el segundo es el protocolo BOOTP de configuración, y el tercero es el servicio de eco de UDP, que es incluido ya que este es utilizado por la popular herramienta de diagnóstico de rutas *traceroute*.

Según sea el caso, la rutina envía el paquete a las rutinas de entrada de los protocolos respectivos, y en el caso del servicio de eco, genera inmediatamente una respuesta usando el mismo paquete entrante como espacio de almacenamiento.

4.2.4.8.2 Rutina de salida de UDP: *udp_out()*

La rutina de salida UDP recibe paquetes de las capas superiores de protocolo, y los procesa antes de transferirlos a IP para su transmisión. Por las mismas razones que en el caso de IP, esta rutina requiere que los campos de IP de destino, ambos puertos de UDP, y el largo del paquete, estén definidos previamente.

La rutina llena los campos del paquete requeridos por IP, calcula la sumatoria del mismo, la invierte si resulta ser cero (ya que cero es el indicador de que la sumatoria está desactivada), y lo pasa a IP.

4.2.4.8.3 Rutina de cálculo de sumatoria de UDP: *udp_cksum()*

El procedimiento de cálculo de sumatoria utilizado por UDP es exactamente el mismo que el usado por IP, pero el cálculo se complica ya que UDP exige que el bloque de datos al que se le aplicará la sumatoria incluya un pseudo-encabezado compuesto de campos específicos del paquete IP y UDP a transmitirse.

La rutina que utilizamos es la misma que describen Comer y Stevens *(1)* en su implementación, y no hubo que aplicarle ningún cambio significativo para utilizarla en nuestro código.

La rutina realiza el cálculo sin ensamblar el pseudo-encabezado, sino que lee sus componentes directamente del paquete y los va adicionando a la sumatoria. Una vez computado el pseudo-encabezado, la rutina suma el paquete UDP, habiendo previamente chequeado si este tiene un número par de caracteres, y añadido un carácter de relleno si es necesario. Tras finalizar el cálculo de la sumatoria, la rutina la transforma a formato de 16-bits y calcula su complemento, el cual es retornado como resultado.

4.2.4.9 Rutinas del protocolo BOOTP: *bootp.c*

En este archivo se encuentran las rutinas utilizadas por el protocolo BOOTP para enviar pedidos de configuración, y para recibir las respectivas respuestas, y configurar el sistema de acuerdo a estas.

4.2.4.9.1 Rutina de salida BOOTP: *bootp_out()*

La rutina de salida de BOOTP tiene la labor de generar un pedido de BOOTP, el cual será inundado a la red Ethernet a la cual está conectado el dispositivo.

La primera acción de la rutina es separar el espacio de memoria para el pedido. Una vez conseguido el espacio, la rutina procede a llenar los distintos campos necesarios para transmitir el pedido. El campo de identificación de UDP se llena, como ya hemos hecho antes, con los bytes iniciales de nuestra dirección Ethernet; esto nos ayudará a discriminar que las respuestas entrantes son realmente generadas a partir de nuestro pedido. Debemos indicar que el paquete BOOTP, tiene muchos más campos que los que nosotros utilizamos en nuestra implementación. Los campos no utilizados son dejados en blanco, lo que indica al servidor que debe ignorarlos.

Después de configurar el paquete BOOTP, la rutina configura los puertos del paquete UDP que contiene a nuestro pedido, para que pueda dirigirse correctamente al servidor BOOTP. Hay que destacar que BOOTP utiliza puertos distintos para clientes y para servidores.

La rutina también configura la dirección IP de destino para realizar un broadcast, ya que al inicializar el equipo no sabremos la dirección del servidor BOOTP. La especificación de BOOTP recomienda que el paquete se marque como no fragmentable, así que la

rutina también realiza esta función. Finalmente el paquete es enviado a UDP para ser procesado y transmitido.

4.2.4.9.2 Rutina de entrada de BOOTP: *bootp_in()*

Esta rutina es llamada por UDP cuando se recibe un paquete BOOTP, y su función es extraer todas las configuraciones que sean relevantes a nuestro sistema del paquete BOOTP. Nuestro sistema utiliza BOOTP para resolver los siguientes parámetros de configuración:

- Dirección IP del sistema.
- Máscara de subred
- Dirección IP de gateway de la red local
- Dirección IP del servidor de seguridad
- Código de identificación de cliente del sistema
- Puerto UDP utilizado por el protocolo de seguridad

De estos valores, la dirección IP solo puede ser obtenida mediante BOOTP, todos los otros valores pueden ser obtenidos de otras formas, calculados, o asumidos. Estas funciones son todas realizadas por esta rutina.

La primera función que realiza la rutina es confirmar que el paquete es una respuesta a un pedido nuestro, comparando su campo de identificación con nuestra dirección física, que como ya dijimos, usamos para llenar ese campo. Una vez validado el paquete,

buscamos la dirección IP que nos ha sido asignada, en el campo correspondiente. Si la dirección existe, la copiamos a la variable *ipadd,* y activamos la bandera IP_RESOLVED. Esta es la función más importante de configuración del sistema, ya que sin ella el protocolo IP no puede trabajar.

Después de resolver nuestra dirección IP, la rutina trata de interpretar el contenido del campo de fabricantes de BOOTP (vendor specific area), el cual es de contenido variable y configurable. Para nuestros propósitos, hemos utilizado el campo recomendado por la especificación de BOOTP. Este campo es identificado por el "número mágico" (Magic Cookie) 99.130.83.99; por lo que nuestra rutina chequea la presencia de este número antes de intentar descodificar el área de fabricantes de BOOTP. Si el número mágico concuerda, la rutina busca en le área de fabricantes los parámetros que necesitamos para configurar nuestro sistema. Hay que destacar aquí que solo la definición de gateway y máscara de subred están incluidas formalmente en la configuración estándar del área de fabricantes. La dirección IP del servidor, puerto UDP, y el código de identificación de cliente son elementos propietarios de nuestro sistema de seguridad, por lo que deben ser configurados manualmente en el servidor BOOTP, o reemplazados en vez de valores que no utilizamos, por ejemplo en vez de la dirección IP del servidor de tiempo de la red, y del servidor de impresión de la red. En nuestro caso, usamos campos libres del área de fabricantes de BOOTP, los cuales están detallados en el Capítulo V.

Si el área de fabricantes del paquete no nos da toda la información que necesitamos, la rutina intentará ahora calcular o asumir la información restante. Lo primero que hace es

confirmar si se obtuvo una máscara de subred; si no es así, la rutina envía un pedido ICMP de máscara (usando la rutina *icmp_out()*), y mientras espera una respuesta, configura la máscara de subred del sistema de acuerdo a la clase IP a la que pertenece la dirección IP asignada por el servidor BOOTP. Si llega posteriormente una respuesta al pedido de máscara de subred, la máscara calculada será reemplazada por la máscara recibida.

Inmediatamente después, la rutina revisa si el código de cliente del sistema ha sido resuelto; si no es así, asigna a nuestro sistema un código de cliente equivalente a los cuatro últimos bytes de la dirección física del cliente, que viene a ser el código "por defecto" que usaremos en nuestro sistema de seguridad. La labor siguiente que realiza la rutina es revisar si se ha resuelto un gateway, y si este no es el caso, intentar resolverlo a partir del campo de gateway de BOOTP del paquete. Después, se verifica si el servidor BOOTP ha especificado un puerto UDP para el protocolo de seguridad, si este no es el caso, se utiliza el puerto 515.

Finalmente, la rutina activa las interrupciones del sistema (mediante *int_on()*), ya que este está en capacidad de operar en este momento. Al activarse las interrupciones, el uso del teclado y de la lectora de código de barras se hace posible por primera vez.

4.2.4.10 Rutinas de la aplicación de seguridad: *sec_app.c*

Este archivo contiene las rutinas de entrada y salida de paquetes de nuestra aplicación propietaria de seguridad. Esta aplicación basa sus comunicaciones cliente servidor en un modelo de comunicaciones sin conexión, razón por la que usamos UDP como protocolo de transporte.

<table>
<tr><td colspan="2">0</td><td colspan="2">16</td><td colspan="4">32 — 64</td></tr>
<tr><td>Tipo</td><td>Status</td><td colspan="2">Transacción</td><td colspan="4">Código de Usuario Bytes 0 – 7</td></tr>
<tr><td colspan="2">C. Usuario 8 - 9</td><td colspan="4">Clave de usuario</td><td colspan="2">C. Cliente 0 - 1</td></tr>
<tr><td colspan="2">C. Cliente 2 - 3</td><td colspan="6">Desocupado</td></tr>
</table>

Figura 26: Formato de paquete de la aplicación de seguridad

La Figura 26 nos muestra el formato que hemos utilizado para las comunicaciones entre cliente y servidor de nuestro sistema de seguridad. El diagrama está organizado en líneas de 64 bits cada una, para permitir una visión más clara de los distintos campos del paquete. El primer campo, de un byte de longitud, indica la función que el paquete desempeña dentro de nuestro protocolo de seguridad. Nuestro sistema implementa los siguientes tipos de paquetes:

- **Pedido de autorización:** Identificado con el código 0, este paquete es enviado del cliente hacia el servidor cuando se requiere la autenticación y autorización de ingreso de un usuario.

- **Respuesta de Autorización:** Este paquete tiene el código 1, y es la respuesta del servidor al pedido de autorización del cliente.

- **Advertencia de Exceso de Errores en Clave:** Identificado con el código 2, este paquete es enviado del cliente hacia el servidor, cuando un usuario falla tres veces en el ingreso de su clave personal, en el caso de que el servidor la haya requerido.

El campo siguiente, también de 1 byte, es utilizado por el servidor en su paquete de respuesta para indicar al cliente la acción que debe tomar respecto al pedido de ingreso del usuario. Nuestro sistema tiene programados diez tipos de respuestas posibles (los cuales siguen los parámetros establecidos por el proyecto de tópico "Redes LAN y WAN"*(12)*, cuyo servidor de seguridad utilizamos), los cuales detallamos a continuación:

- **Autorización de ingreso incondicional (Código de status 0):** Esto indica al cliente que debe proceder a dar acceso al usuario sin pedirle que ingrese su clave secreta, si es que se le ha asignado una.
- **Autorización de ingreso condicional (Código de status 1):** El usuario es requerido de ingresar su clave secreta, la cual ha sido enviada por el servidor en el paquete de respuesta. Si el usuario ingresa correctamente la clave, se le permite ingresar al espacio restringido. Si falla en el ingreso de la clave, se le permiten dos intentos más de ingresarla; si falla en los dos intentos, el proceso de autorización es cancelado, y

se envía un paquete de advertencia al servidor, para que anote el intento fallido de ingreso.

- **No autorizado - Usuario desconocido (Código de status 2):** El servidor envía esta respuesta cuando el código de usuario que ha recibido no se encuentra en su base de datos.

- **No autorizado – No hay supervisor (Código de status 3):** Esta respuesta se recibe cuando el servidor determina que el espacio restringido se encuentra sin supervisión de usuarios privilegiados, y el usuario que trata de ingresar no está autorizado a hacerlo en esas condiciones.

- **No autorizado – Fuera de horario (Código de status 4):** Es cuando el usuario está tratando de ingresar fuera del horario en el que ha sido autorizado para ingresar.

- **No autorizado - Usuario revocado (Código de status 6):** El servidor envía esta respuesta cuando los privilegios de ingreso del usuario han sido revocados por la administración del sistema.

- **No autorizado – No tiene privilegios de ingreso (Código de status 7):** Mediante este mensaje el servidor indica que el usuario no está autorizado a ingresar al espacio restringido controlado por el sistema cliente.

- **No autorizado – Cliente desconocido (Código de status 8):** Este mensaje indica que el servidor no tiene en su base de datos al cliente que envió el pedido de autenticación.

- **No autorizado – Error en base de datos (Código de status 9):** Con este mensaje el servidor indica que ha ocurrido un error interno en su sistema y no puede procesar el pedido de autorización.

Estos mensajes son utilizados por el cliente para permitir o negar el ingreso del usuario, y para indicar a este las razones por las que se procede de esta manera. Hay que destacar que el cliente nunca modifica este campo, esa es una prerrogativa exclusiva del servidor.

El campo siguiente de nuestro paquete contiene un código secuencial de 2 bytes de longitud generado por el cliente al momento de construir el pedido de autorización. Este código se usa para identificar la transacción de autorización, y deberá ser retornado por el servidor en el paquete de respuesta.

A continuación del identificador de transacción viene el campo que contendrá el código de identificación de usuario. Este campo tiene 10 bytes de longitud, ya que debe ser capaz de transportar tanto el formato de matrícula de la ESPOL (8 bytes) como el formato de la cédula de ciudadanía del Ecuador (10 bytes). Este campo es llenado por el

cliente en el pedido de autorización, y debe ser devuelto por el servidor en la respuesta del mismo. El cliente también debe llenar este campo cuando transmite un paquete de advertencia de exceso de intentos de validar clave, ya que el servidor lo usa para añadir el evento a su bitácora.

Después del campo de código de usuario viene el campo asignado para la clave secreta. Cuando el servidor responde a un pedido de autorización con la respuesta de ingreso condicionado, este campo contendrá la clave de cuatro caracteres que el cliente deberá usar para validar el acceso del usuario. Este campo, al igual que el campo de status de respuesta, es sólo llenado por el servidor, el cliente nunca lo modifica. Cuando el cliente manda un paquete de advertencia de exceso de intentos de validar una clave, no transmite la clave, sino que deja el campo en blanco.

El último campo que contiene nuestro paquete es el usado para identificar al sistema cliente cuando este hace cualquier tipo de interacción con el servidor. Este campo tiene cuatro bytes de longitud, y es llenado siempre con el número de identificación de cliente que el sistema debe recibir del servidor BOOTP. Como ya mencionamos, si el sistema no puede configurar este valor, le asignará un valor igual a los cuatro últimos bytes de la dirección Ethernet del sistema.

Una vez definido el mecanismo de interacción de nuestra aplicación cliente-servidor, podemos pasar a revisar el código de las rutinas de entrada y salida de nuestra aplicación de seguridad.

4.2.4.10.1 Rutina de salida de la aplicación de seguridad: *sec_out()*

Esta rutina tiene la doble función de enviar los paquetes de pedido de autorización, y de enviar los paquetes de advertencia de exceso de intentos de validar clave. La primera acción que realiza la rutina, es discriminar cual es el tipo de paquete que se desea enviar. Esto lo hace revisando la bandera PASSWORD_PENDING del sistema; si esta bandera está activa, eso significa que la rutina ha sido llamada por el manejador de validación de claves del lazo principal del sistema, y por lo tanto su función es la de enviar un paquete de advertencia. En el caso contrario (la bandera está desactivada) la rutina asume que se le está pidiendo que genere un pedido de autorización. Si este último es el caso, la rutina debe chequear si este es el primer intento de enviar un pedido de autorización (mediante la bandera RESPONSE_EXPECTED), en cuyo caso procede a preparar la pantalla para mostrar el código de usuario ingresado, y activa la bandera RESPONSE_EXPECTED, que indica que se ha enviado un pedido y se está esperando una respuesta. En cambio, si este no es el primer pedido, la rutina revisa si el número de pedidos enviado ha excedido el máximo de tres, en cuyo caso aborta el proceso e informa al usuario que el servidor no está respondiendo a los pedidos.

La construcción del paquete propiamente dicho es prácticamente idéntica para los dos tipos de paquete que produce la rutina. Pero la creación del paquete de pedido es un poco más complicada, ya que el código de usuario debe ser extraído del buffer de teclado, y organizado correctamente para su inclusión en el campo correspondiente del

paquete. Además, el procedimiento debe enviar el código recibido a la pantalla, para que el usuario pueda ver que su código fue leído correctamente por el sistema.

El procedimiento de envío del paquete de advertencia no tiene estas complicaciones, ya que el código es extraído del almacenamiento de memoria donde se lo guarda hasta ser autorizado.

Una vez llenado el campo de código de usuario, la rutina debe incluir el código de identificación de transacción en el paquete. Para generar este código, la rutina simplemente utiliza el valor presente de la variable *interval*, la cual ya hemos indicado que está cambiando constantemente.

Finalmente el paquete es encapsulado en sus correspondientes encabezados UDP e IP, para ser transmitido. Un punto interesante en este procedimiento, es que si el sistema no ha recibido una dirección de servidor en su etapa de configuración, el campo de dirección IP de destino del paquete es llenado con la dirección de broadcast, de manera que el servidor pueda recibir el paquete y responder. Al recibir la respuesta, el cliente usará la dirección IP de origen de la misma como IP de servidor en pedidos posteriores.

4.2.4.10.2 Rutina de entrada de la aplicación de seguridad: *sec_in.c*

Esta rutina tiene como función el procesamiento de paquetes entrantes destinados a la aplicación de seguridad. Dado que los únicos paquetes que entran al cliente en nuestro

modelo de seguridad son respuestas de autorización provenientes del servidor, el primer procedimiento realizado por la rutina es el filtrado de los paquetes entrantes para asegurar su autenticidad y procedencia.

Primeramente, la rutina descarta los paquetes que provienen de otros clientes, a los cuales puede identificar según su tipo (Pedidos de autorización y advertencias de fallo en la clave). Esto es necesario, ya que como ya vimos, los clientes tienen la facultad de hacer broadcasts de sus envíos cuando no tienen resuelta la dirección IP del servidor de seguridad. Un segundo paso de confirmación, es la comparación del código de identificación del cliente, lo cual indicará que el paquete viene realmente destinado a este sistema. Una vez revisado este código, el cliente debe procesar la dirección IP de origen del paquete. Si el cliente no tiene resuelto su servidor de seguridad, utilizará la dirección de origen para resolver su configuración de servidor; en el caso de que el cliente ya tenga resuelto el servidor, la dirección de origen del paquete debe ser igual a la dirección de servidor definida en el cliente, o el paquete será descartado. Finalmente, la rutina procede a comprobar el identificador de transacción, comparándolo con el utilizado para la última transacción originada en el cliente.

Una vez finalizados estos pasos de seguridad, el cliente verifica que el tipo de paquete sea efectivamente el esperado, es decir una respuesta de autorización. Después de hacer esto la rutina realiza un muy importante paso de seguridad; se chequea que el sistema esté en realidad esperando una respuesta de autorización, mediante la bandera

RESPONSE_EXPECTED. Finalmente, chequeamos el código de usuario del paquete para comprobar que es el mismo que el cliente envió en su última transacción.

El siguiente procedimiento realizado es la descodificación del campo de status de la respuesta del servidor, el cual indicará el procedimiento a seguir con la transacción en progreso. En el caso de que el servidor ordene el ingreso inmediato del usuario, la rutina procede a la activación del portero eléctrico. Cabe destacar que el procedimiento de activación del portero eléctrico no es más complicado que la inicialización de una variable a uno, por lo que no se consideró necesario asignar una rutina especial para realizarlo.

En el caso de que se requiera el ingreso de una clave secreta, la rutina copia la clave que viene en el campo correspondiente del paquete de respuesta, e inicia el procedimiento de ingreso de clave (la etapa de ingreso de clave es realizada en el lazo principal del sistema, como veremos más adelante).

Los casos restantes son negaciones de acceso, así que la única labor realizada por la rutina en esos casos es informar al usuario mediante el display que se le ha negado el acceso, y la razón por la que se hizo esto.

4.2.4.11 Rutina central del sistema: sec_cli.c

Este archivo es el componente principal de nuestro sistema, ya que toda la ejecución del mismo está centrada en los procedimientos realizados por la rutina aquí contenida. Como ya habíamos mencionado, nuestro sistema basa su ejecución en la detección y procesamiento de eventos externos, y en la ejecución de procedimientos periódicos de mantenimiento. Los manejadores principales de eventos y de procesos periódicos se encuentran delineados dentro de la rutina contenida en este archivo. Como es común en las implementaciones de C, la rutina principal del sistema se llama *main(),* nombre que indica al compilador que esta rutina es la entrada a la ejecución del código.

Ya que *sec_cli.c* es el archivo principal de nuestro código, este contiene las declaraciones de inclusión de todos los otros componentes del sistema, tanto aquellos que son encabezados estándares del compilador, como aquellos que forman parte de nuestro código. El orden en que estas declaraciones de inclusión están definidas es importante y no debe ser cambiado sin premeditación, ya que la declaración y definición de variables y funciones de todo el sistema depende del orden de inclusión que se ha establecido.

La rutina *main()* empieza con un llamado al macro *set_vector(),* que es el encargado de implementar el vector de interrupción de la rutina de interrupción de la interface de teclado. Si este macro no se incrementa, la interrupción de teclado no operaría, ya que el microcontrolador no sabría donde está localizado el código de la rutina.

El siguiente paso realizado es la inicialización del sistema mediante la rutina *h_init()*. Esta rutina es ejecutada únicamente una vez, ya que inmediatamente después el sistema ingresa al lazo principal de ejecución, el cual es infinito, y tiene como función la permanente ejecución de los manejadores de eventos y los procesos periódicos del sistema.

Antes de la ejecución de cualquier actividad dentro del lazo principal del sistema, se procede a la inicialización y activación del contador de seguridad del sistema, para que este pueda detectar cualquier anomalía en el flujo de operaciones del lazo.

Los manejadores de eventos están implementados primero en el lazo, seguidos por los procesos de mantenimiento periódico. Hay que destacar que nuestro sistema no tiene un orden de ejecución basado en prioridades, sino que este es estrictamente lineal. Debido a esto, no hay ninguna ventaja real para ningún proceso en el hecho de estar implementado antes que otro en el lazo de ejecución. El primer manejador de eventos es el encargado de detectar interrupciones generadas por la interface de red. Como ya dijimos, este manejador no está implementado como una interrupción propiamente dicha, sino que se limita a monitorear el estado de la salida de interrupción del ST-NIC. Si se detecta una interrupción de red, el manejador llama a la rutina *poll_net()*, que se encarga de definir la causa de la interrupción y tomar la acción apropiada.

Inmediatamente después del manejador de interrupción de red, se implementa el manejador de entrada de paquetes, el cual es activado por la presencia de la bandera PACKET_RECEIVED, la cual es generada por *poll_net()*. Al detectar la presencia de paquetes en el buffer, el manejador llama a la rutina *ethernet_in()* para que los procese. Hay que destacar aquí que debido a la naturaleza lineal de nuestro sistema, esta rutina no retornará hasta que se haya terminado totalmente el procesamiento del paquete entrante, con la posibilidad (por ejemplo en el caso de que el paquete sea un pedido de echo de ICMP) de la generación y transmisión de una respuesta; esto hace que el tiempo de retorno de *ethernet_in()* sea bastante variable.

Después de los manejadores de red, nuestro sistema implementa el manejador de eventos de teclado. Este manejador se activa cuando la variable *key_count* es mayor que cero, lo que indica que existen caracteres en el buffer de teclado. La actividad posterior realizada por este manejador depende del estado en que se encuentre el sistema en ese momento:

- Si el sistema se encuentra en estado normal, el manejador procesa los caracteres ingresados como códigos de usuario. El procesamiento de códigos de usuario consiste en recibir caracteres y mostrarlos en pantalla, hasta que se reciba el carácter ENTER, o hasta que se llene el buffer de teclado, en cuyo caso no se reciben más caracteres excepto ENTER. Una vez terminado el ingreso del código de usuario, el manejador llama a la rutina *sec_out()* para que envíe un pedido de autorización para el código ingresado. El manejador implementa un periodo de espera para ingreso de

caracteres de código, para evitar que el sistema se quede infinitamente en estado de recepción de código si el usuario abandona el proceso. Hay que destacar que el código puede provenir ya sea del teclado numérico como de la lectora de banda magnética.

- Si el sistema se encuentra esperando el ingreso de una clave secreta, los caracteres ingresados serán recibidos por el sistema, pero no serán mostrados en pantalla (se muestra asteriscos en vez de los caracteres de la clave); además, ya que la clave secreta es de cuatro dígitos, el sistema no recibirá más dígitos una vez que el cuarto haya sido ingresado, y procederá inmediatamente a corroborar la validez de la clave ingresada. Si la clave es correcta, el manejador se encarga de activar la interface de portero eléctrico; en cambio si la clave es incorrecta, el usuario recibe dos intentos más de ingresar la clave; si ambos son errados, el manejador llama a la rutina *sec_out()* para que envíe una advertencia de exceso de fallas en ingreso de clave al servidor. Al igual que en el caso de ingreso de código, el ingreso de la clave secreta está limitado en el tiempo por un contador decrementado por los procesos de mantenimiento del sistema.

Hay que destacar que el manejador de la interface de teclado desactiva la interrupción del teclado cuando llama a la rutina de seguridad. Esto es necesario para evitar que el contenido del buffer de teclado cambie debido a intervención del usuario antes de que la rutina termine de procesarlo (hay que decir que esto es muy improbable, ya que la ejecución de la rutina es mucho mas rápida que cualquier usuario).

Después de los manejadores de eventos, nuestra rutina implementa el lazo principal de mantenimiento del sistema. Este lazo es de ejecución condicionada por la variable *interval*. Nuestra rutina ejecuta los procedimientos contenidos en el lazo únicamente cuando la ya nombrada variable (la cual está constantemente siendo incrementada) se inicializa a cero. Ya que *interval* tiene 16 bits de longitud, los procedimientos del lazo de mantenimiento se ejecutarán cada 65535 iteraciones del lazo principal de ejecución del sistema.

Los procedimientos realizados dentro del lazo empiezan con el manejador de configuración del sistema. Este manejador revisa si se tiene resuelto el valor de la dirección IP del sistema; si esto no es así, el manejador llama a la rutina *bootp_out()* para que genere un pedido de configuración BOOTP. Dado que este manejador se encuentra dentro de una lazo infinito, el sistema no parará de enviar pedidos de configuración hasta que su dirección IP haya sido resuelta.

El siguiente procedimiento que se ejecuta dentro del lazo de mantenimiento es *arp_timer()*, que se encarga de controlar el cache de ARP y sus miembros. Después de este procedimiento, se ejecuta una serie de manejadores dedicados a monitorear las banderas de estado del sistema con el objetivo de controlar los contadores de expiración de eventos del sistema. Así se monitorean las bandera RESPONSE_EXPECTED, PASSWORD_EXPECTED, CODE_EXPECTED y DISPLAY_FLAG, las cuales

afirman que el sistema está esperando una respuesta del servidor, un ingreso de clave secreta, un ingreso de código de usuario, y la lectura de datos presentados en pantalla respectivamente. Cada una de estas banderas tiene asociado un contador decremental, el cual al llegar a cero indica que el tiempo de vida del evento ha terminado, y debe ser cancelado. El hecho de que estos manejadores estén dentro del lazo de mantenimiento significa que los periodos de expiración de los eventos que controlan están definidos como múltiplos del ciclo de tiempo definido por la variable *interval*.

Una excepción a esto es representada por el manejador que controla el tiempo de activación de la interface del portero eléctrico. Debido a que el portero debe estar encendido por un tiempo reducido (~ 1 segundo), no es práctica su inclusión dentro del lazo de mantenimiento, ya que el ciclo de ejecución de este último puede ser considerablemente más largo.

Al finalizar la iteración del lazo principal del sistema, se procede a detener y reinicializar el contador de seguridad, para que este listo para vigilar la siguiente iteración.

Con la descripción de la rutina principal del sistema terminamos el estudio detallado de nuestro código de cliente. A continuación revisaremos el código servidor utilizado, y las modificaciones que tuvieron que realizarse para su adaptación a nuestro modelo de interacción.

4.3 *Diseño del código del servidor*

El código del servidor de seguridad de nuestro sistema es una adaptación del código utilizado por el tópico de graduación "Redes LAN y WAN" *(12)* para su sistema de seguridad. La razón por la cual utilizamos este código, en vez de desarrollar uno propio, es que el proyecto anteriormente desarrollado fue una implementación completa y eficiente del servidor de seguridad, por lo cual se consideró innecesario el hacer esfuerzos extras en esa dirección, si existía la factibilidad de modificar el software existente para su uso con nuestro cliente de seguridad.

4.3.1 Implementación y diseño del software

El servidor de seguridad de nuestro sistema es una aplicación de múltiples componentes, los cuales están centrados en una base de datos SQL, controlada por el software comercial Microsoft SQL Server. La base de datos contiene todos los detalles necesarios para el procesamiento de pedidos de autorización de usuario del sistema. Los datos de los usuarios y clientes del sistema son añadidos a la base de datos mediante una aplicación de administración desarrollada como parte del sistema de seguridad. El procesamiento de pedidos propiamente dicho es realizado por un proceso dedicado, el cual recibe los pedidos de los clientes, y realiza consultas a la base de datos para decidir si se debe admitir o no al usuario que realiza el pedido.

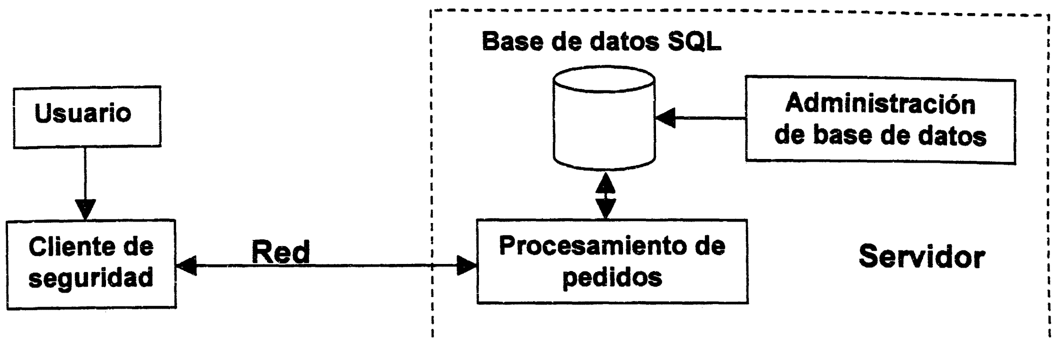


Figura 27: Componentes del servidor de seguridad

Como podemos notar en la Figura 27, de todos los componentes del servidor de seguridad, el único que tiene contacto con los dispositivos clientes es el proceso de recepción de pedidos. Gracias a esta modularidad del sistema, se nos facilitó la tarea de modificarlo para nuestros propósitos, ya que el único componente que fue necesario estudiar y modificar fue el proceso de recepción de paquetes.

El sistema servidor en sí está basado en el sistema operativo Windows NT 4.0 de Microsoft, y sus componentes están escritos en Visual Basic (aplicación de administración) y C (procesamiento de pedidos). La base de datos SQL es un producto comercial, así que su implementación es irrelevante para nosotros.

Habiendo aclarado que el énfasis de nuestras modificaciones va a estar en la aplicación de procesamiento de paquetes, debemos pasar a describir las premisas que utilizamos como guías para la realización de estas modificaciones:

- **Realizar el menor número posible de modificaciones:** Esto es importante; dado que estamos trabajando con código escrito por terceros, el cual es además relativamente complejo y especializado en áreas ajenas a nuestro conocimiento.

- **Realizar en lo posible adiciones de funcionalidad, no modificaciones de funcionalidad:** Para mantener la simplicidad de nuestras modificaciones, hemos tratado de no reescribir secciones completas del código, sino simplemente de añadir en partes claves el código necesario para que el sistema se acople a nuestra forma de trabajar. Solo en casos en que es inevitable hemos recurrido a sobreescribir secciones de código.

- **Modificar preferiblemente en el cliente y no en el servidor:** Dado que el código de cliente ha sido íntegramente diseñado por nosotros, hemos tratado de acoplar nuestro modelo de interacción al usado por el servidor de seguridad; de manera que las modificaciones se hagan en nuestro código, cuya organización dominamos, y no en el código del servidor. El código del servidor solo ha sido modificado en aquellas funciones en las que el trabajo de modificar al cliente hubiera sido más difícil que modificar el servidor (Por ejemplo, en el protocolo de transporte utilizado: UDP vs. TCP).

4.3.2 Herramientas utilizadas

Dado que el código que vamos a modificar está escrito en lenguaje C, se podría pensar que es posible la utilización de cualquier compilador C para la arquitectura Intel x86. Sin embargo, la tarea se ve obstaculizada por la funcionalidad especial que debe implementar este código. Primeramente, el código debe poder interactuar con el sistema operativo de 32 bits de Windows NT. Cabe destacar que la implementación de la aplicación de recepción de pedidos no incluye las ventanas gráficas comúnmente asociadas con una aplicación diseñada para el ambiente Microsoft Windows, sino que es simplemente un proceso automático y aislado, sin interface de usuario del usuario; esto sin embargo no implica que la aplicación no tenga interacción con el sistema operativo, como veremos enseguida. Aparte de la necesidad de integrarse al sistema operativo, el código debe ser capaz de interactuar con los protocolos de la arquitectura TCP/IP implementada por Windows NT; esto implica la utilización de la especificación Winsock, cuya función es estandarizar el diseño de aplicaciones TCP/IP en Windows. Una última funcionalidad que debe implementar nuestra aplicación, es la interacción con el sistema de base de datos SQL sobre el cual trabaja nuestra aplicación.

Estas limitaciones excluyen a los compiladores basados en DOS, y nos dejan con los más modernos sistemas de desarrollo de aplicaciones basados en Windows. La tarea de escoger una herramienta de desarrollo para realizar nuestras modificaciones se vio limitada por la poca variedad de títulos existentes en el mercado. El software que terminamos utilizando es Microsoft Visual C++ versión 4.2. Aunque esta es una

herramienta de diseño con énfasis en C++, debemos destacar que tanto el código original de la aplicación de recepción de pedidos como nuestras modificaciones, son enteramente escritas en lenguaje C estándar.

4.3.3 Descripción general del código a modificar.

El código de la aplicación de procesamiento de pedidos está dividido en varios archivos, como está indicado en la tabla:

Archivo	Descripción	PERTENECIENTE A:
Server.c	Inicialización y lazo principal de proceso del programa	Servidor de seguridad original
Server.h	Encabezado del archivo server.c	Servidor de seguridad original
Valiacce.c	Rutinas de validación de usuario	Servidor de seguridad original
Util.c	Rutinas de reporte de errores para Winsock	Servidor de seguridad original
Util.h	Encabezado del archivo util.c	Servidor de seguridad original
Modifies.h	Encabezado con adiciones y modificaciones de variables y estructuras.	Actualización del software

Tabla 6: Archivos del programa de procesamiento de pedidos del servidor

Como se indica en la Tabla 6, el archivo principal de la aplicación es *server.c*, el cual se encarga de las labores de inicialización de variables, inicialización de la interface Winsock con el stack TCP/IP del sistema operativo, e inicialización de la interface SQL con la base de datos del servidor de seguridad. Además, este archivo contiene el lazo principal de procesos del sistema. Este lazo básicamente se dedica a esperar el ingreso de paquetes hacia la aplicación, y llama a las rutinas de archivo valiacce.c para el procesamiento de los mismos.

El archivo *valiacce.c* contiene las rutinas que analizan la información recibida de los clientes de seguridad, hacen los pedidos de información a la base de datos del sistema, y evalúan si se debe dar acceso o no al usuario que generó el pedido.

El resto de archivos originales del sistema son encabezados simples, o rutinas auxiliares que no afectan la forma de operar del sistema.

Nuestra única adición a la lista de archivos de la aplicación es el encabezado modifies.h, el cual contiene las definiciones de todas las variables y estructuras añadidas por nuestras modificaciones.

4.3.4 Modificaciones realizadas

Las modificaciones realizadas a la aplicación de seguridad pueden entenderse mejor si se observan desde un punto de vista funcional. Por esta razón empezaremos definiendo las variaciones de funcionalidad que consideramos necesarias para que esta aplicación interactúe correctamente con nuestro sistema cliente:

4.3.4.1 Cambio de protocolo de transporte

Nuestro sistema cliente encapsula sus pedidos utilizando el protocolo UDP, mientras que la aplicación de procesamiento de paquetes basaba su interacción con clientes en conexiones TCP. Debido a las complejidades inherentes en una implementación TCP, desde el principio del proyecto se descartó su uso en nuestro cliente integrado; en

cambio, el cambiar el modo de transporte del servidor de TCP a UDP resulta posible gracias a las facilidades dadas por la especificación Winsock. La factibilidad de hacer el cambio de protocolo de transporte fue el factor determinante en la decisión de utilizar el servidor del tópico en vez de desarrollar uno propio de nuestro proyecto.

Para implementar el cambio de protocolo de transporte, lo único que se tuvo que hacer fue modificar los parámetros de las funciones Winsock utilizadas para inicializar la interacción con el stack TCP/IP del sistema, pasándolas de TCP a UDP. Esto se logra mayormente cambiando el tipo de socket utilizado de SOCK_STREAM a SOCK_DGRAM, en la inicialización del mismo. Además, dado que el modelo de interacción UDP no es orientado a conexión, debemos modificar el uso de las funciones de recepción y envío de paquetes, usando *recvfrom()* en vez de *recv()*, y *sendto()* en lugar de *send()*.

4.3.4.2 Acoplamiento de formato de paquetes

Dado que estamos integrando dos sistemas desarrollados por separado, es natural que existan diferencias en el formato de la interacción cliente servidor a nivel de sesión, presentación y aplicación. Dado que el servidor original usaba TCP como protocolo de transporte, su formato de transferencia de datos estaba orientado a simples cadenas de caracteres. En cambio, nuestro cliente de seguridad, ya que usa UDP, está orientado a transmisión desconectada de paquetes de largo invariable, organizados mediante estructuras de datos. Dado que uno de nuestros objetivos es el cambio de protocolo de

transporte, fue natural que también se cambiara el formato de los paquetes, para ajustarse a nuestras necesidades. Gracias al uso de estructuras, el efecto sobre el código ya implementado es muy pequeño.

El código original del servidor basa mucho su interacción entre funciones en el uso de argumentos (a diferencia de nuestro código cliente), de manera que los únicos cambios que debimos hacer para adecuar este código a nuestro formato de paquetes basados en campos de estructuras fue cambiar las secciones del código en las que se definen las variables que van a ser pasadas como argumentos al resto de sistema.

Los únicos argumentos requeridos por el servidor para su operación son las variables *xmatric_ing*, *xclave_ing*, y *xclave_lab_ing*, las cuales definen el código de usuario, clave de acceso y código de cliente respectivamente, para la presente transacción. Entonces, para interoperar con nuestro formato de construcción y procesamiento de paquetes, todo lo que tuvimos que hacer fue implementar la copia del contenido de los campos de usuario, clave y código de cliente de nuestra estructura de paquete hacia estas variables.

Además, tuvimos que añadir piezas de código en las secciones del software de servidor encargadas de la decisión de autorización de acceso. El código añadido no modifica la función del programa, sino que actualiza la estructura de nuestro paquete de acuerdo a la decisión tomada por el código del servidor.

4.3.4.3 Adición de funcionalidades extras

Aquí tratamos sobre la integración de las funciones de cliente y servidor, de manera que aquellas funcionalidades consideradas deseables, e implementadas solo en uno de los sistemas son implementadas también en el otro. El servidor de seguridad original tenía 10 posibles respuestas de autorización, mientras que nuestro cliente original implementaba apenas 5. Las respuestas adicionales fueron incluidas en el código de nuestro cliente. Así mismo, nuestro cliente de seguridad implementa la selectividad en el pedido de la clave secreta, a discreción del servidor; esta función no estaba incluida en el servidor de seguridad, el cual siempre requería el ingreso de la clave. Esta funcionalidad fue agregada al servidor.

La adición de los códigos de respuesta adicionales fue absolutamente trivial, y realizada enteramente en el código del cliente, el cual, como ya observamos, implementa actualmente todas las posibles respuestas que puede generar el servidor de seguridad.

La adición de la selectividad en el pedido de ingreso de la clave secreta fue un poco más complicada, ya que requirió el único cambio que hicimos en el flujo de operaciones del software del servidor. Inicialmente, el software recibía la clave al mismo tiempo que el código de cliente, y procedía a pedir a la base de datos que enviara la clave de acceso correspondiente al código de cliente recibido. El programa comparaba las dos claves, y procedía a denegar el acceso inmediatamente si eran distintas. En el caso de que las claves fueran iguales, el programa procedía a revisar los privilegios de seguridad

definidos para ese usuario, para confirmar la autorización de acceso, el cual aún podía ser denegado en base a estos parámetros adicionales.

Nuestro software eliminó el paso inicial de corroboración de clave (aunque el pedido de la clave a la base de datos sí se realiza, esta es simplemente almacenada por el momento), y pasa directamente a la decisión de autorización de acceso en base a los privilegios del usuario. Una vez decidido si el usuario tiene autorización de acceso, el sistema revisa el valor contenido en la variable que contiene la clave del usuario recibida de la base de datos. Si la variable está en blanco, el acceso es inmediato, y si la variable contiene una clave de acceso, esta clave se añade al paquete de respuesta y se condiciona el ingreso del usuario a la validación de la misma (Código de status 1).

Un problema que resulta de esto, es que dado que el software del servidor no realiza la validación de la clave, sino que esta se realiza en el cliente, el servidor no tenía la posibilidad de anotar las ocurrencias de accesos denegados debido a claves mal ingresadas. Para remediar este problema, se añadió al modelo de interacción el paquete de advertencia de exceso de errores en la clave. Cuando este paquete es recibido por el servidor, este último se limita a recoger los valores de usuario y cliente y llamar a la función encargada de llevar la bitácora de eventos del sistema de seguridad.

Aquí concluimos el estudio del software de nuestro sistema de seguridad. En el siguiente capítulo, estudiaremos los detalles y procedimientos que hay que conocer para implementar y operar correctamente los distintos componentes de nuestro sistema.

Capítulo V

5 Implementación e integración final del sistema

5.1 Componentes y equipos requeridos

Dada la naturaleza modular de nuestro sistema de control de acceso, es necesario que especifiquemos los distintos componentes que son necesarios para su correcta implementación. La mejor forma de organizar los componentes del sistema es utilizar las divisiones conceptuales que hemos definido para su diseño. Como ya conocemos, la principal división conceptual de nuestro sistema es aquella entre cliente y servidor, por lo que podemos empezar definiendo los requerimientos que necesitamos satisfacer para la implementación de estos dos componentes principales.

5.1.1 Requisitos para la implementación del cliente

5.1.1.1 Configuración Inicial

Nuestro sistema cliente es un dispositivo integrado, razón por la cual sus requerimientos de equipo subsidiario son mínimos; el requisito más importante para la correcta implementación del cliente es que el microcódigo que este contiene esté correctamente programado. Como ya estudiamos en los capítulos anteriores, el cliente utiliza el protocolo BOOTP para la configuración de sus subsistemas de software; sin embargo, existe un parámetro de la configuración del mismo que debe estar definido al momento

de la compilación del código del cliente; este parámetro es la dirección física de la interface de red del sistema. Debido a la necesidad de tener direcciones físicas distintas para todos los equipos conectados a un segmento Ethernet, y a la total dependencia del sistema en la existencia de este parámetro, nos vemos obligados a definirlo en la implementación del microcódigo; esta configuración trae las siguientes consecuencias:

- El código de cada cliente que se desee implementar es único, y deberá ser identificado de acuerdo a la dirección física definida en el mismo.

- Se necesitará un inventario de direcciones asignadas para evitar repeticiones de las mismas.

Entonces, tenemos dos opciones principales para la implementación del microcódigo del cliente. La primera es la compilación separada de cada sistema cliente, modificando las fuentes para obtener direcciones físicas distintas (Este procedimiento podría ser totalmente automático, mediante el uso de compilación condicionada, de manera que no representa un obstáculo insalvable para la producción en grandes cantidades de nuestro cliente). La segunda opción es el uso de un solo bloque de código, el cual puede ser modificado posteriormente a su compilación para implementar las direcciones físicas de los distintos clientes; este procedimiento es posible ya que la variable utilizada para almacenar la dirección física del sistema es estática, por lo que su localización de memoria permanecerá constante en todas las compilaciones del código (si las fuentes no son alteradas).

Una vez resuelta la implementación de la dirección física del cliente, los parámetros restantes necesarios para la operación del sistema son obtenidos mediante resolución BOOTP; esto implica la obvia necesidad de implementar un servidor BOOTP, cuya implementación discutiremos mas adelante.

5.1.1.2 Conexión de periféricos

En los capítulos anteriores estudiamos los distintos periféricos que utiliza nuestro cliente para su operación normal. Hay que destacar que en la implementación actual de nuestro dispositivo, no es posible suprimir la implementación de ninguno de los periféricos del sistema, ya que el código del mismo no implementa mecanismos para detectar la presencia o ausencia de los mismos.

El periférico/conexión mas importante a implementarse es la conexión a la red Ethernet con la que va a interactuar nuestro sistema. De la correcta implementación de esta conexión depende toda la operación del sistema, ya que como ya observamos, el equipo la utiliza para su configuración inicial.

Si se va a utilizar el conector 10BaseT de la interface de red, se puede realizar la conexión sin variar el estado inicial de los DIP switches de la interface de red (Ver Figura 23). En cambio, si se necesita utilizar el conector AUI, se debe cambiar la posición del switch 2 para habilitar esta conexión.

Una vez resuelta la implementación de la dirección física del cliente, los parámetros restantes necesarios para la operación del sistema son obtenidos mediante resolución BOOTP; esto implica la obvia necesidad de implementar un servidor BOOTP, cuya implementación discutiremos mas adelante.

5.1.1.2 Conexión de periféricos

En los capítulos anteriores estudiamos los distintos periféricos que utiliza nuestro cliente para su operación normal. Hay que destacar que en la implementación actual de nuestro dispositivo, no es posible suprimir la implementación de ninguno de los periféricos del sistema, ya que el código del mismo no implementa mecanismos para detectar la presencia o ausencia de los mismos.

El periférico/conexión mas importante a implementarse es la conexión a la red Ethernet con la que va a interactuar nuestro sistema. De la correcta implementación de esta conexión depende toda la operación del sistema, ya que como ya observamos, el equipo la utiliza para su configuración inicial.

Si se va a utilizar el conector 10BaseT de la interface de red, se puede realizar la conexión sin variar el estado inicial de los DIP switches de la interface de red (Ver Figura 23). En cambio, si se necesita utilizar el conector AUI, se debe cambiar la posición del switch 2 para habilitar esta conexión.

Los periféricos restantes del cliente no requieren ningún tipo de configuración especial, aparte de su correcta conexión al sistema.

A continuación revisaremos la implementación del servidor del sistema, y los componentes que este requiere para la correcta operación del sistema.

5.1.2 Requisitos para la implementación del servidor

En la sección 4.3.1 revisamos los distintos componentes necesarios para la operación del servidor de seguridad. Estos componentes han sido diseñados para operar sobre el sistema operativo Windows NT 4.0, por lo que esta será la plataforma de software que usaremos para la implementación del servidor.

Ya que nuestro sistema utiliza la arquitectura TCP/IP para transportar los pedidos y respuestas de autenticación del sistema, es necesario que estos protocolos estén correctamente configurados en el sistema a utilizarse como servidor, previamente a la instalación de los componentes de nuestro servidor de seguridad.

5.1.2.1 Instalación y configuración de las aplicaciones del servidor de seguridad

Como ya hemos revisado, nuestro servidor de seguridad tiene tres componentes principales: El servidor de base de datos SQL Server, la aplicación de administración de la base de datos, y la aplicación de procesamiento de pedidos. Estos tres componentes

fueron desarrollados por el tópico "Redes LAN y WAN" *(12)*, y se debe estudiar la documentación desarrollada por ese tópico para conocer los pormenores de la instalación y operación de estos componentes. Hay que destacar que aunque nosotros realizamos modificaciones a la aplicación de procesamiento de pedidos del servidor, estas modificaciones no cambian en absoluto la metodología y requisitos para la instalación de la misma.

Un requisito que hay que cumplir para la correcta operación del sistema de seguridad es que el puerto utilizado para recepción y envío de datos por la aplicación de procesamiento de pedidos no debe estar ocupado por ningún otro programa residente en la plataforma utilizada como servidor. Tanto la aplicación de procesamiento de pedidos del servidor como nuestro cliente integrado han sido programados para utilizar el puerto 515 del protocolo UDP. Hay que destacar que este parámetro puede ser modificado, tanto a nivel de configuración como a nivel de fuentes para permitir la resolución de conflictos de recursos con otros equipos que estén siendo utilizados en la red.

5.1.2.2 Implementación del servidor BOOTP

Como ya hemos mencionado en varias oportunidades, nuestro sistema utiliza el protocolo BOOTP para las labores de configuración del equipo cliente. Debido a esto, nos vemos en la necesidad de implementar un servidor BOOTP en cada una de las redes en las que se va a instalar sistemas clientes. El servidor BOOTP es una aplicación totalmente independiente de las aplicaciones utilizadas por nuestro sistema de

seguridad, de manera que no se requiere que esté ubicado en la misma máquina o sistema operativo que estas. Dado que BOOTP es un protocolo estándar y ampliamente utilizado, se puede usar una gran variedad de implementaciones como servidor para nuestro sistema, los requisitos básicos que debe cumplir la aplicación BOOTP para interactuar correctamente con nuestros sistemas clientes son:

- Resolución de dirección IP
- Resolución de máscara de subred
- Resolución de gateway de la red
- Area de fabricantes configurable por el usuario

Los tres últimos parámetros son opcionales porque nuestro sistema cliente es capaz de calcular o asumir los valores de estos parámetros, o porque su presencia no es esencial (como en el caso del gateway). El caso del área de fabricantes del paquete BOOTP es especial, ya que nuestro sistema la utiliza para resolver la dirección IP del servidor de seguridad, y para resolver el código de identificación del cliente. Como ya hemos mencionado, si el sistema no puede resolver el código de identificación del cliente, utiliza su dirección física como código; en muchos casos este esquema de identificación de clientes será adecuado, por lo que se puede omitir la resolución del código de cliente. En el caso de la resolución del servidor de seguridad, las opciones son más limitadas, ya que si el sistema cliente no puede resolver la dirección IP del servidor de seguridad, este tratará de obtenerla haciendo broadcasts de los primeros pedidos de autenticación que realice. Este modo de operación puede no ser adecuado en ambientes de alto riesgo de

seguridad, y además tiene la desventaja de que el servidor debe residir en la misma red física que el cliente.

Si la aplicación BOOTP que se va a utilizar puede implementar parámetros definidos por el usuario en el área de fabricantes, entonces se hace posible la resolución de los parámetros de código de cliente y dirección IP del servidor de seguridad. Hay que destacar que nuestro sistema utiliza el formato estándar del área de fabricantes, ya que este permite la resolución de la máscara de subred y el gateway de la red local. Para la resolución de los parámetros adicionales de nuestro sistema nos limitamos a utilizar secciones no reglamentadas del formato estándar (las secciones no reglamentadas ocupan los códigos 129 al 255). La implementación de nuestros parámetros adicionales es como sigue:

Parámetro	Código de identificación	Longitud (bytes)
Dirección IP del servidor de seguridad	150	4
Código de cliente	151	4
Puerto UDP del sistema	152	2

Tabla 7: Implementación de parámetros del cliente en el paquete BOOTP

Estos parámetros pueden ser modificados a conveniencia del usuario en las fuentes del código del cliente de nuestro sistema de seguridad.

5.2 Utilización del sistema de seguridad

Una vez que los distintos componentes del sistema han sido correctamente instalados, se puede proceder al estudio de la operación normal del mismo. Nuestro énfasis principal

estará en la operación del sistema cliente, ya que este ha sido desarrollado en su totalidad durante este proyecto. La operación de los componentes del servidor de seguridad está descrita en detalle en la documentación del tópico "Redes LAN y WAN" *(12)*, por lo que no cubriremos a fondo esos temas en este trabajo.

5.2.1 Utilización del sistema cliente

La interacción del sistema cliente con el usuario se realiza enteramente a través de tres dispositivos: la lectora de código de barras, el teclado numérico, y el display fluorescente. Los dos primeros dispositivos son utilizados por el usuario para ingresar los datos que el sistema requiere para autorizar su ingreso; mientras que el display es utilizado por el sistema para entregar información de estado y diagnóstico, tanto para el usuario como para el administrador o encargado de mantenimiento del sistema. A continuación pasaremos a describir paso a paso la operación del sistema cliente, y las acciones que se requieren del usuario para su correcta interacción con el mismo.

5.2.1.1 Arranque del sistema: Pantallas de inicialización

Al encenderse el sistema, este entra en su etapa de inicialización, durante la cual hace pedidos BOOTP para poder configurar sus parámetros de funcionamiento. La primera información que el usuario verá en la pantalla del cliente durante esta etapa será la que se muestra en la figura:

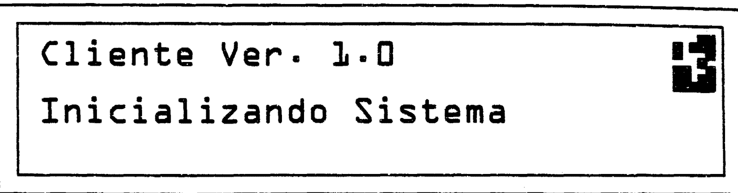


Figura 28: Pantalla de Inicialización

Esta pantalla inicial nos muestra en su primera línea la versión del código cliente guardado en la memoria del sistema, y un mensaje de inicialización en la segunda línea. Además podemos notar la presencia de uno de los iconos de sistema del cliente en la última posición de la primera línea. Este icono, como veremos inmediatamente, nos muestra el estado de la resolución BOOTP de la dirección IP del sistema, la cual en este caso aún no ha sido resuelta. Antes de proseguir, es importante que se conozca el formato y significado de los iconos de sistema utilizados por nuestro cliente.

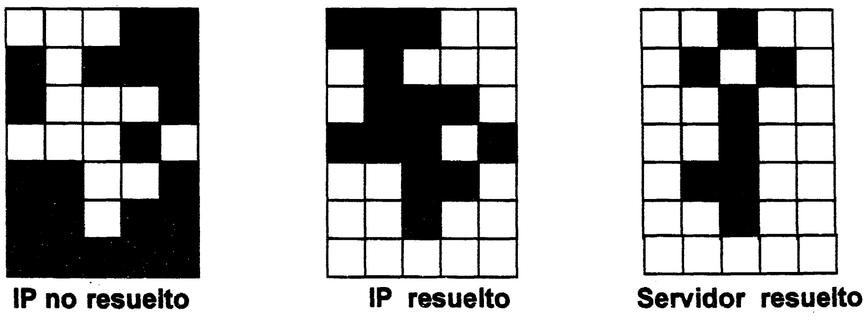


Figura 29: Iconos del Sistema Cliente

La Figura 29 muestra los tres posibles iconos que puede mostrar nuestro sistema cliente en su display de datos. El primer icono nos indica, como ya habíamos dicho, que el sistema aún no tiene una dirección IP asignada. El segundo icono reemplaza al primero

cuando el sistema recibe una respuesta BOOTP con una dirección IP válida. Si además de su dirección IP el sistema también recibió la dirección IP del servidor de seguridad, se mostrará adicionalmente el tercer icono, que indica que el servidor de seguridad ha sido resuelto. Este icono se activa ya sea que la dirección del servidor se resuelva mediante BOOTP, o se extraiga de la respuesta a un pedido del cliente.

La pantalla de inicialización mostrada en la Figura 28 se mantiene activa durante unos cuantos segundos, y es reemplazada por una pantalla en blanco (excepto por el icono de resolución de IP), sin importar que la dirección IP haya sido resuelta o no.

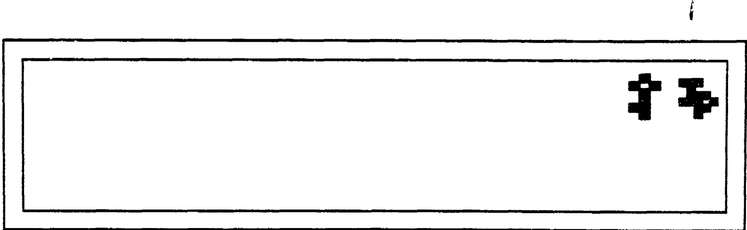


Figura 30: Estado normal del sistema

La Figura 30 nos muestra la pantalla del sistema después de la etapa de inicialización. Como podemos ver, el sistema ha resuelto ya su dirección IP, y la dirección IP del servidor de seguridad, ya que ha emplazado los iconos que así lo indican. En este estado, el sistema está listo para empezar a recibir pedidos de autorización. Hay que destacar, que si el sistema no logra resolver su dirección IP por cualquier motivo, este permanecerá indefinidamente en el proceso de inicialización, con el icono de IP no resuelto desplegado, e ignorará cualquier pedido de autorización de ingreso. En este estado el sistema está periódicamente enviando pedidos BOOTP de configuración.

5.2.1.2 Procesamiento de pedidos de autorización

Un usuario puede iniciar un pedido de autorización de dos maneras distintas: Si posee un carnet de identificación de la ESPOL, puede pasarlo por la lectora para que esta registre su código de identificación; si no posee un carnet, puede escribir directamente en el teclado numérico el código de identificación que le corresponde.

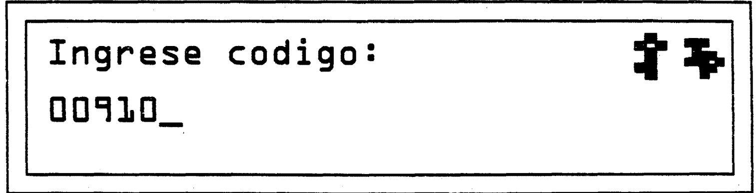


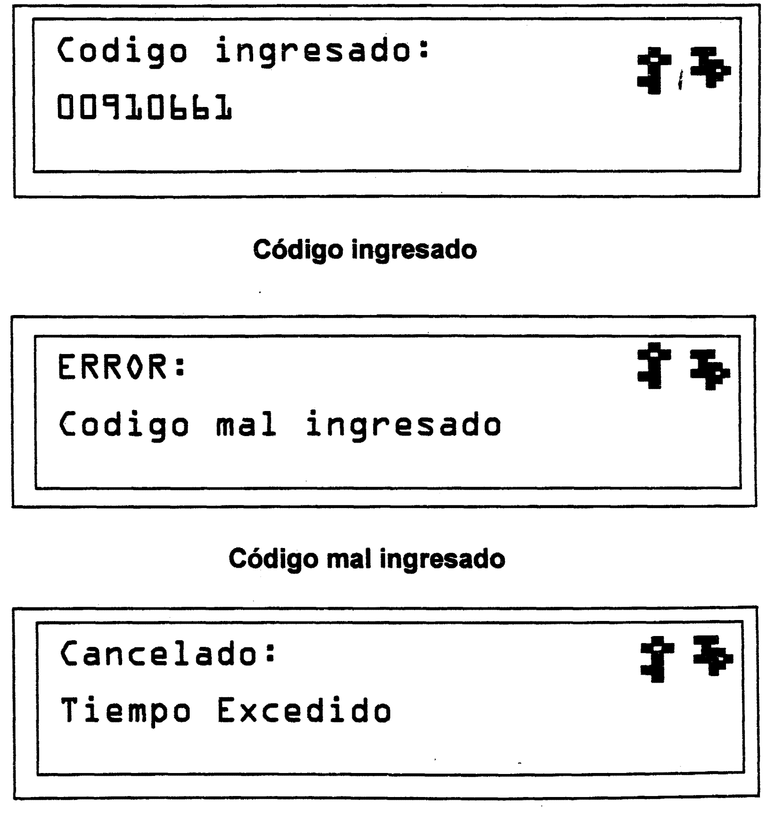
Figura 31: Proceso de ingreso de código

En la Figura 31 podemos apreciar el estado de la pantalla durante el ingreso de un código de identificación. Apenas el sistema detecta que se ha ingresado un carácter, ya sea en el teclado numérico o en la lectora de código de barras, este despliega el mensaje "Ingrese código:" en la línea superior del display, y registra los caracteres ingresados en la línea inferior. El código ingresado no puede tener más de 10 dígitos, y su ingreso debe ser finalizado presionando la tecla ENTER. La lectora de código de barras realiza el ingreso de los dígitos y la adición de la tecla ENTER automáticamente, así que el usuario que la posea no necesita utilizar el teclado numérico para el ingreso de su código personal.

Como dijimos, el código a ingresar no puede tener más de 10 dígitos, así que si el sistema detecta que el usuario ha ingresado ya los diez dígitos correspondientes, este

ignorará cualquier dígito subsecuente que se ingrese, a menos que este sea la tecla ENTER.

La operación de ingreso de código puede finalizar de dos maneras distintas. Si el usuario ha ingresado un código válido (mayor que seis dígitos) y finalizado su ingreso con la tecla ENTER, el sistema recibirá el código y lo enviará al servidor de seguridad para que este realice el proceso de autorización de ingreso. En cambio si por alguna razón el usuario no realizó correctamente el proceso de ingreso de su código, o lo abandonó inconcluso, el sistema descartará el código ingresado.



Código ingresado

Código mal ingresado

Ingreso cancelado por abandono del usuario

Figura 32: Posibles conclusiones del proceso de ingreso de código

La Figura 32 nos muestra los posibles mensajes que recibe el usuario como resultado del ingreso de un código. De estas tres posibles conclusiones al proceso, solo la primera resulta en el envío del código ingresado hacia el servidor de seguridad. La segunda pantalla se muestra cuando el usuario ha ingresado un código de menos de 6 dígitos, mientras que la tercera pantalla se muestra tras un periodo de inactividad de algunos segundos, durante los cuales el usuario no ha ingresado más dígitos de su código. Esto se hace para evitar que el sistema se quede estancado en el proceso de ingreso, si es que el usuario lo abandona de improviso.

5.2.1.3 Procesamiento de respuesta del servidor

Una vez que se ha enviado un pedido de autorización al servidor, el sistema pasa a esperar la respuesta del mismo. Durante el periodo de espera, el sistema ignora cualquier tipo de ingreso de datos de parte del usuario. El proceso de espera puede terminar de varias maneras, las cuales describimos a continuación:

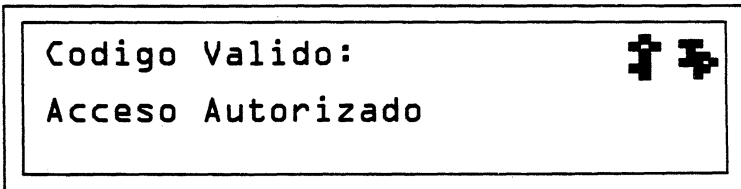


Figura 33: Pantalla de autorización de acceso inmediato

La Figura 33 muestra la pantalla que se presenta cuando el servidor responde que el código ingresado es válido, y que el usuario no necesita confirmar su identidad mediante una clave secreta. Como ya mencionamos, la discriminación entre usuarios

que requieren o no clave se realiza a nivel del servidor de seguridad. En este momento el sistema procede a accionar el mecanismo de apertura de la puerta del espacio bajo su control.

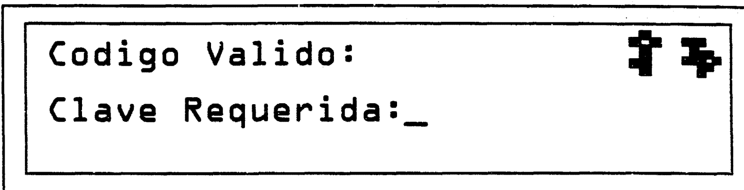


Figura 34: Pantalla de requerimiento de clave secreta

Si el cliente presenta una pantalla como la que se muestra en la Figura 34, el código de identificación del usuario ha sido aceptado por el servidor, pero debido a las políticas de acceso, el servidor requiere que el usuario ingrese su clave secreta de acceso. A partir de este momento el sistema cliente pasará al proceso de ingreso de clave secreta, cuyos particulares analizaremos más adelante.

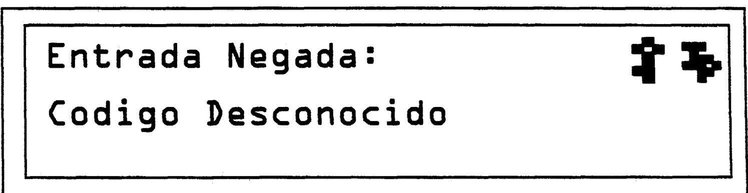


Figura 35: Ejemplo de pantalla de acceso denegado

La Figura 35 muestra una pantalla que indica que el acceso al espacio restringido ha sido denegado. Existen varias razones por las cuales el acceso pueda ser negado al usuario, y todas estas están detalladas en la sección 4.2.4.10, la cual describe la operación del protocolo de seguridad que utilizamos. En todos los casos en que el acceso le sea negado al usuario, la razón para esto será desplegada en la segunda línea

del display del sistema. En el caso presentado en la Figura 35, el usuario ha ingresado un código que no se encuentra registrado en la base de datos del servidor de seguridad.

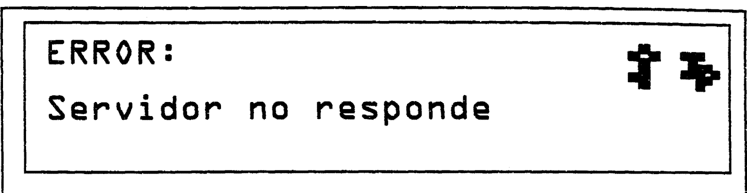


Figura 36: Pantalla de indicación de falta de respuesta del servidor

Finalmente, la última posible conclusión al proceso de respuesta del servidor, es que esta respuesta no se produzca en absoluto, ya sea por que el servidor no existe (o no hay ruta hacia él) o por que este ha sido temporalmente desabilitado. En casos como este, el sistema reintenta tres veces el envío del pedido de autorización, y si no recibe respuesta muestra la pantalla de la Figura 36, y regresa al estado inicial de espera de pedidos de autorización. Notemos que en la pantalla mostrada en la Figura 36, el icono de servidor resuelto sí está presente, lo cual nos es útil para descubrir la causa del problema. Si el icono está presente, podemos asumir que un servidor existe, ya que este ha sido resuelto ya sea mediante BOOTP, o mediante paquetes de broadcast; y por lo tanto sabemos que existe un problema en la comunicación entre cliente y servidor. En cambio si el icono no existe, sabremos inmediatamente que el problema radica ya sea en la falta de un servidor, o en la falta de una ruta IP válida hacia este último.

5.2.1.4 Procesamiento de clave secreta

Si el servidor ha requerido el ingreso de la clave secreta del usuario, el sistema cliente despliega la pantalla de petición de clave, tras lo cual el usuario debe ingresar su clave de cuatro dígitos, utilizando el teclado numérico. Como vemos en la Figura 37, los dígitos de la clave no son mostrados por seguridad. Tras el ingreso del cuarto dígito, el sistema automáticamente pasa a procesar la clave ingresada, sin requerir que el usuario presione la tecla ENTER. Esto es posible porque la longitud de la clave de acceso es fija.

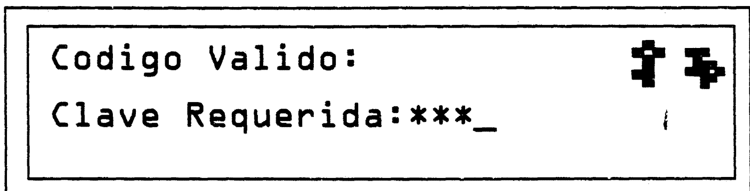


Figura 37: Ingreso de la clave de acceso

Si la clave de acceso es correcta, el sistema despliega la pantalla que se muestra en la Figura 38, y procede a activar el mecanismo de la puerta de ingreso.

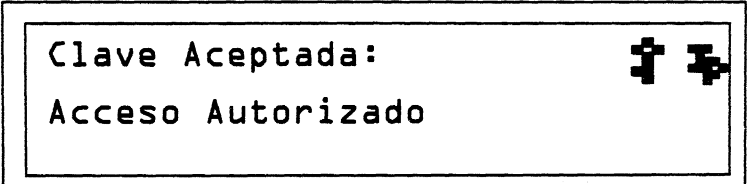


Figura 38: Clave secreta correctamente ingresada

En cambio, si la clave es incorrecta, el sistema despliega una pantalla advirtiendo al usuario que su clave ha sido incorrectamente ingresada y le da otra oportunidad de ingresarla. El usuario tiene tres oportunidades para ingresar correctamente su clave de

acceso, si falla los tres intentos, el acceso es denegado, y se envía una advertencia al servidor para que anote el evento en su registro de seguridad.

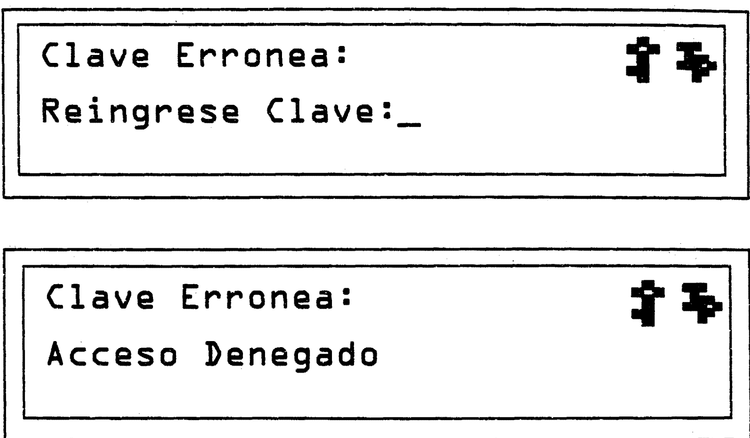


Figura 39: Reintento de ingresar clave secreta y exceso de intentos de ingreso

Al igual que en el proceso de ingreso del código de identificación, si el usuario deja inconcluso el proceso de ingreso de la clave, un contador de tiempo cancela el proceso después de su expiración.

El sistema pasa entonces al estado inicial de espera de códigos de autorización.

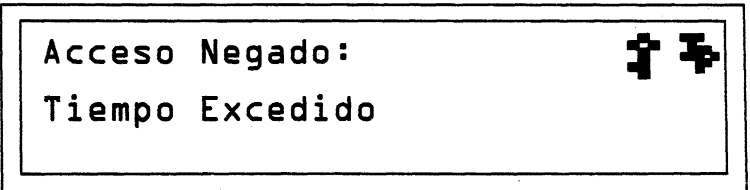


Figura 40: Expiración del proceso de ingreso de clave

Aquí terminamos la descripción del modo de operación de nuestro controlador de acceso. A continuación pasamos a revisar las conclusiones y recomendaciones que hemos desarrollado a la finalización de este trabajo.

6 Conclusiones y Recomendaciones

Siendo el nuestro un trabajo de diseño y construcción, la conclusión del mismo es el equipo cuya manufactura se planteó como objetivo del proyecto. Sin embargo, una vez finalizado el trabajo, debemos evaluar si el resultado del mismo cumple satisfactoriamente los requisitos que se definieron como objetivos del mismo.

Cuando decimos requisitos, no queremos hacer énfasis necesariamente en los objetivos principales del proyecto, el cumplimiento de los cuales esperamos sea evidente para el lector. Más bien queremos hacer una evaluación de la eficiencia general con la que hemos diseñado y construido nuestro controlador de acceso, basándonos en los parámetros que se definieron como deseables para la implementación del mismo. Recordemos los requerimientos que definimos para el diseño del equipo:

Bajo costo: Hay que decir que nuestro equipo no ha sido necesariamente barato de construir, como pudimos observar en el capítulo III. Sin embargo, esto puede esperarse dado que el equipo en cuestión es un prototipo. Lo importante es el potencial de ahorro que puede existir en el proceso de manufactura. Dado el hecho de que todos los componentes usados son bastante comunes (quizás no en el mercado ecuatoriano, pero sí lo son en los mercados de los países industrializados), pensamos que un proceso de producción en mayores cantidades podría abaratar considerablemente el costo por unidad del controlador.

Una de las interrogantes más importantes que se plantea con respecto al costo de nuestro sistema controlador de acceso es si se justifica su implementación frente a la alternativa de implementar controladores en software, basados en plataformas Intel 286 o 386, las cuales sabemos que se pueden obtener a costos muy reducidos en la actualidad. Respecto a esto podemos decir, que aunque es probable que se pueda obtener sistemas 386 a costos comparables o incluso menores al de nuestro equipo, los costos de instalación y mantenimiento de estos equipos y la dificultad de implementarlos en áreas exteriores bien pueden contrarrestar cualquier ahorro monetario que se realice en su adquisición.

Tamaño reducido: A todo lo largo del proceso de diseño y construcción de nuestro sistema, hemos hecho lo posible por mantener sus dimensiones lo más reducidas que sea posible. El diseño inicial de nuestra placa de circuito impreso era un 25% mayor que el diseño que finalmente utilizamos. La extensión del equipo es lo bastante reducida para no resultar molesta su instalación en espacios reducidos. Hay que destacar, como ya lo hemos hecho antes, que el uso de elementos SMD modernos y técnicas avanzadas de manufactura de placas de circuito impreso ofrece el potencial de reducir el tamaño del sistema en al menos un 50% más, a partir del diseño actual.

Facilidad de instalación y mantenimiento: Exceptuando los DIP switches de configuración de la interface de red, todos los parámetros de operación de nuestro sistema están guardados en la memoria EPROM, o son recibidos del servidor BOOTP.

Debido a esto, el proceso de instalación del sistema no requiere ningún conocimiento de sus características y modo de operación.

Facilidad de expansión: Debido al uso de TCP/IP como transporte de datos, y al hecho de que se usa una cadena de identificación de dispositivos de cuatro bytes de extensión, las posibilidades de adición de dispositivos controladores al sistema de seguridad son prácticamente ilimitadas.

Resistencia al abuso y amabilidad hacia el usuario: Todos los componentes del sistema que tienen contacto con el usuario son modulares y reemplazables, por lo que su fallo no comprometerá la integridad del controlador en sí. Se hizo un esfuerzo por mantener la interacción con el usuario lo más libre de ambigüedades que sea posible, mediante el uso de retroalimentación visual y un proceso simple de autorización de acceso.

En el balance final, consideramos que se ha conseguido cumplir con todos los objetivos planteados al inicio del proyecto, hasta donde lo han permitido la tecnología y recursos disponibles para el desarrollo del mismo. Como nosotros mismos lo hemos indicado ya, el diseño y construcción del controlador no son óptimos, y son susceptibles de muchas mejoras, algunas de las cuales trataremos de enumerar a continuación como guía a futuros proyectos que traten de seguir con el desarrollo de este sistema de seguridad.

Rediseño de platina de circuito impreso: Como hemos mencionado a lo largo del desarrollo de este trabajo, nuestro equipo fue diseñado de una manera muy conservadora, con la idea de obtener un sistema cuya operación pueda ser fácilmente validada. Sin embargo, para futuras iteraciones de diseño, se puede utilizar técnicas más innovadoras que permitan una mayor eficiencia y facilidad de construcción del controlador. A la lista de problemas de diseño que presentamos en el capítulo III podemos añadir la consideración de que sería deseable que todos los conectores para periféricos del sistema se encuentren en un mismo lado del controlador, lo cual tendería a ayudar a su instalación en espacios reducidos.

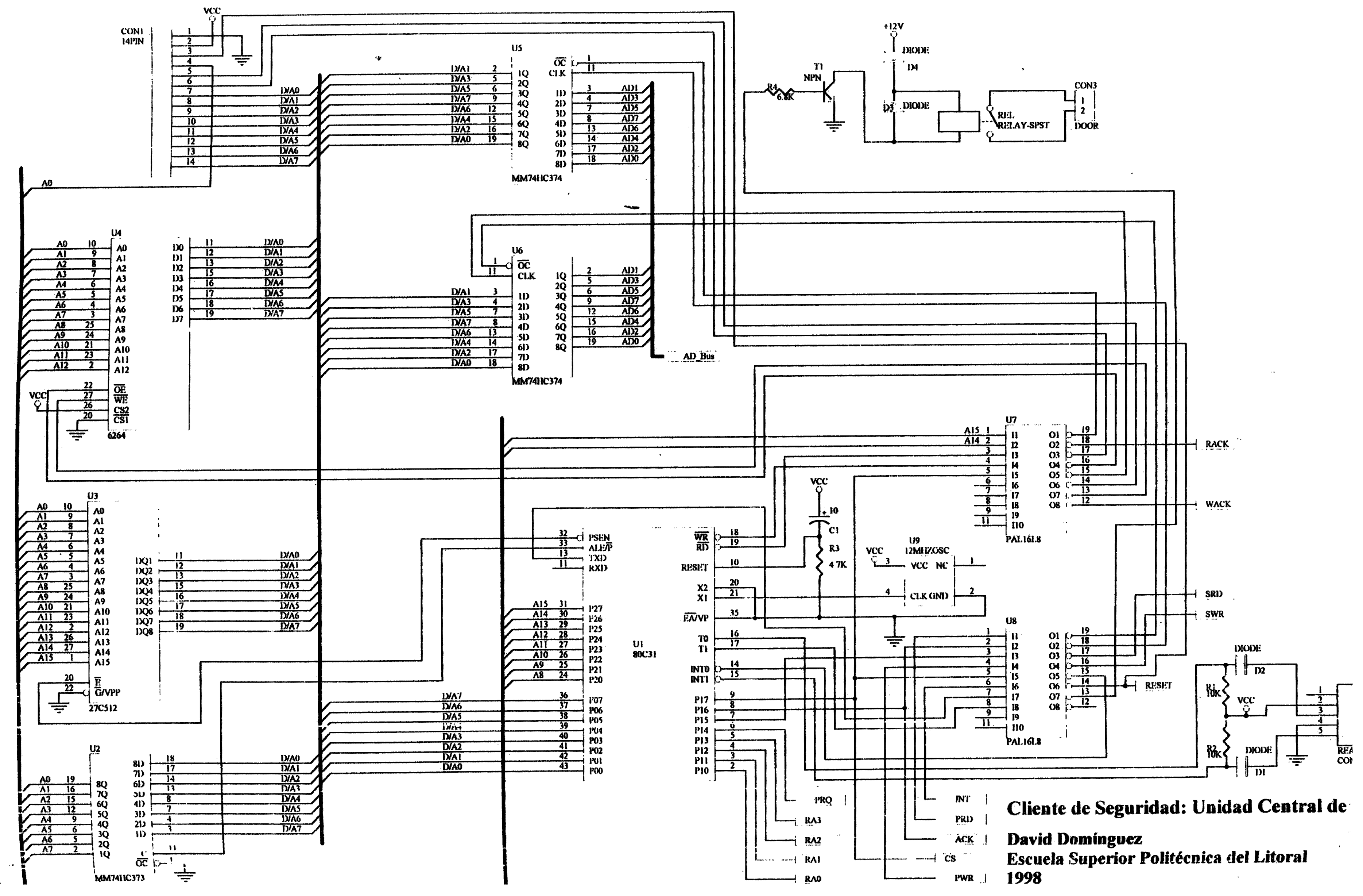
Adición de encripción: Dada la creciente cantidad de personas que tienen conocimientos avanzados de arquitecturas de red (especialmente TCP/IP), y el nivel de seguridad requerido para el transporte de códigos y claves de autorización, es bastante deseable la adición de algún tipo de encripción de datos a nuestro esquema de seguridad. Aunque se han hecho significativos esfuerzos para hacer que la comunicación de datos entre nuestro cliente de seguridad y el servidor remoto sea lo más segura posible, estas precauciones no son suficientes dada la sofisticación de las amenazas de seguridad existentes en la actualidad.

Anexo A

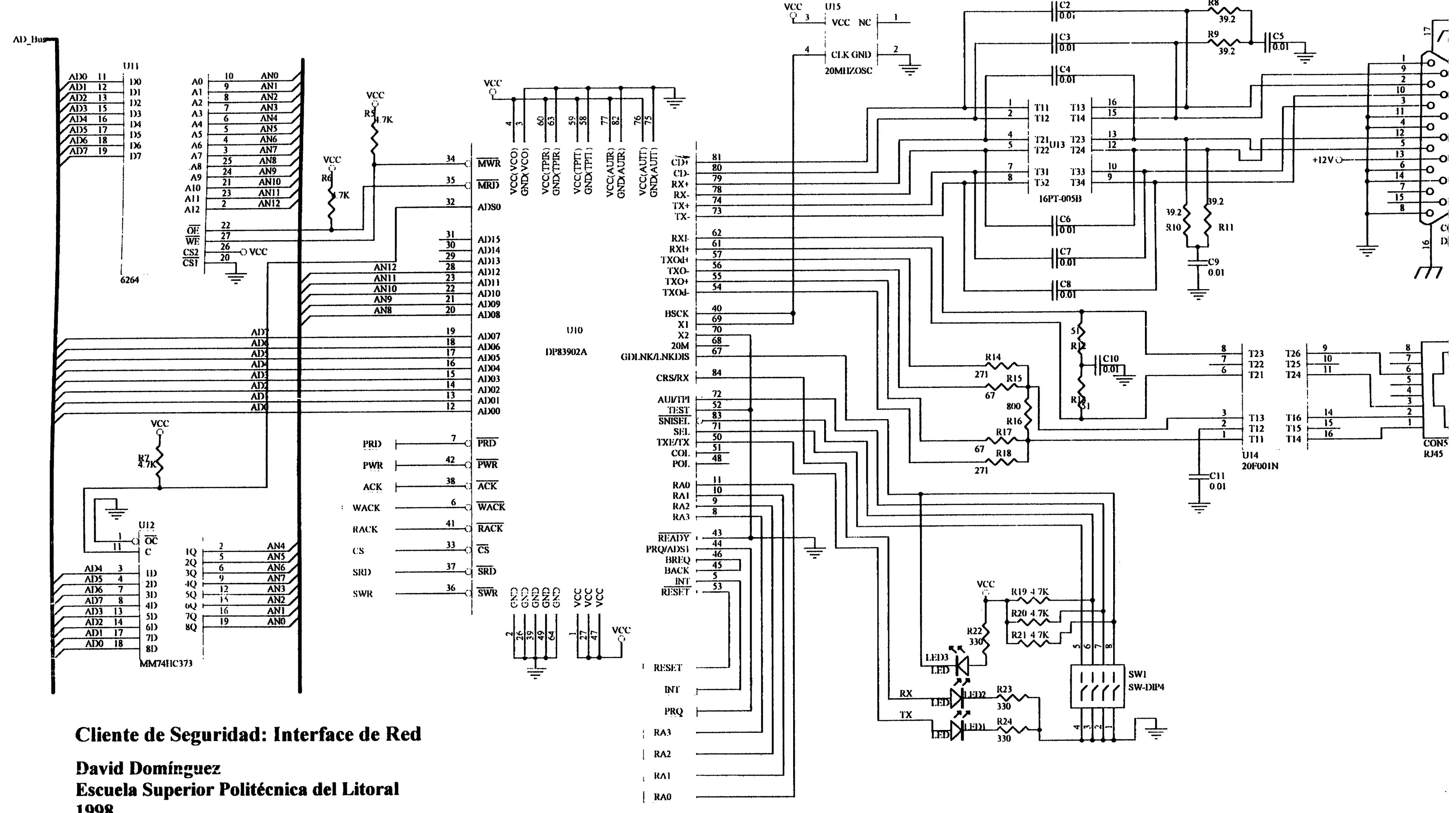
7 Diagramas eléctricos del cliente de seguridad

En este anexo hemos incluido los diagramas eléctricos detallados del cliente de seguridad desarrollado en este trabajo. Los diagramas están divididos de la siguiente manera:

- Diagrama esquemático de la unidad central de proceso
- Diagrama esquemático de la interface de red Ethernet
- Diagrama esquemático de la red de alimentación del cliente de seguridad

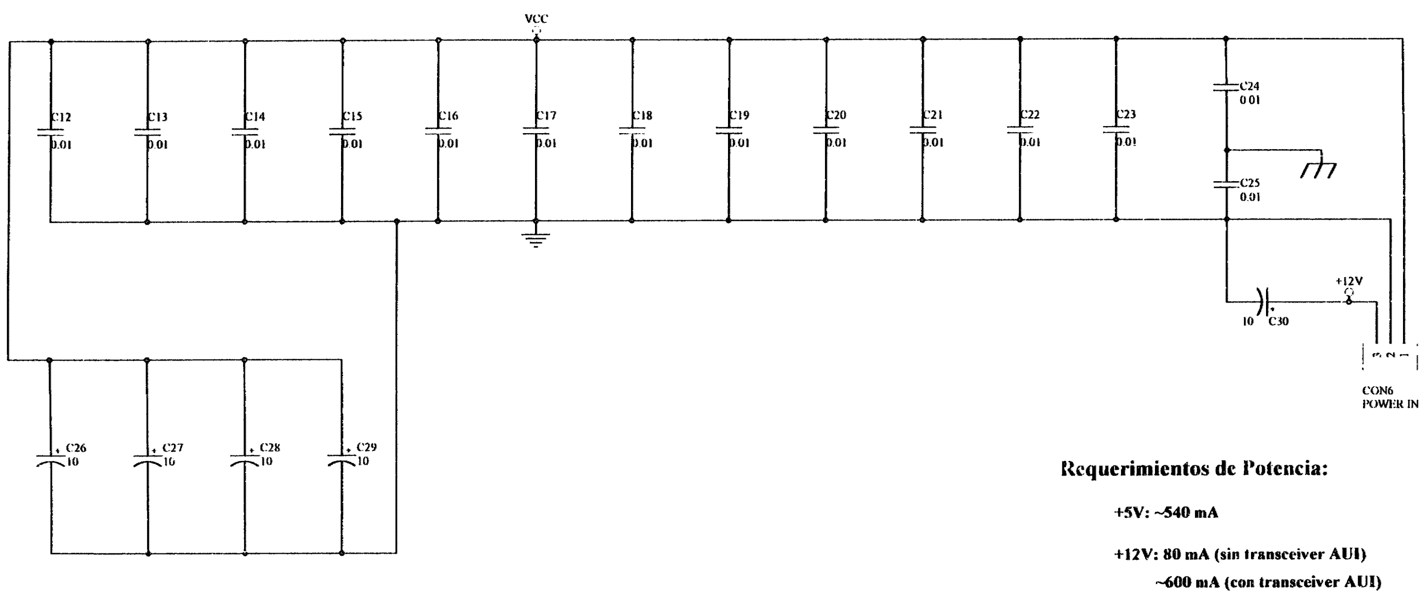


Cliente de Seguridad: Unidad Central de
David Domínguez
Escuela Superior Politécnica del Litoral
1998
CON1
14PIN
U5
MM74HC374
U6
MM74HC374
U4
6264
U3
27C512
U2
MM74HC373
U1
80C31
U7
PAL16L8
U8
PAL16L8
U9
12MHZGSC
CON3
DOOR
REL
RELAY-SPST
T1
NPN
R4
6.8K
R3
4 7K
C1
R1
10K
R2
10K
D1
D2
D3
D4
DIODE
+12V
VCC
AD_Bus
RACK
WACK
SRD
SWR
RESET
PRQ
RA3
RA2
RA1
RA0
INT
PRD
ACK
CS
PWR



Cliente de Seguridad: Interface de Red

David Domínguez
Escuela Superior Politécnica del Litoral
1998



Cliente de Seguridad: Red de alimentación

David Domínguez
Escuela Superior Politécnica del Litoral
1998

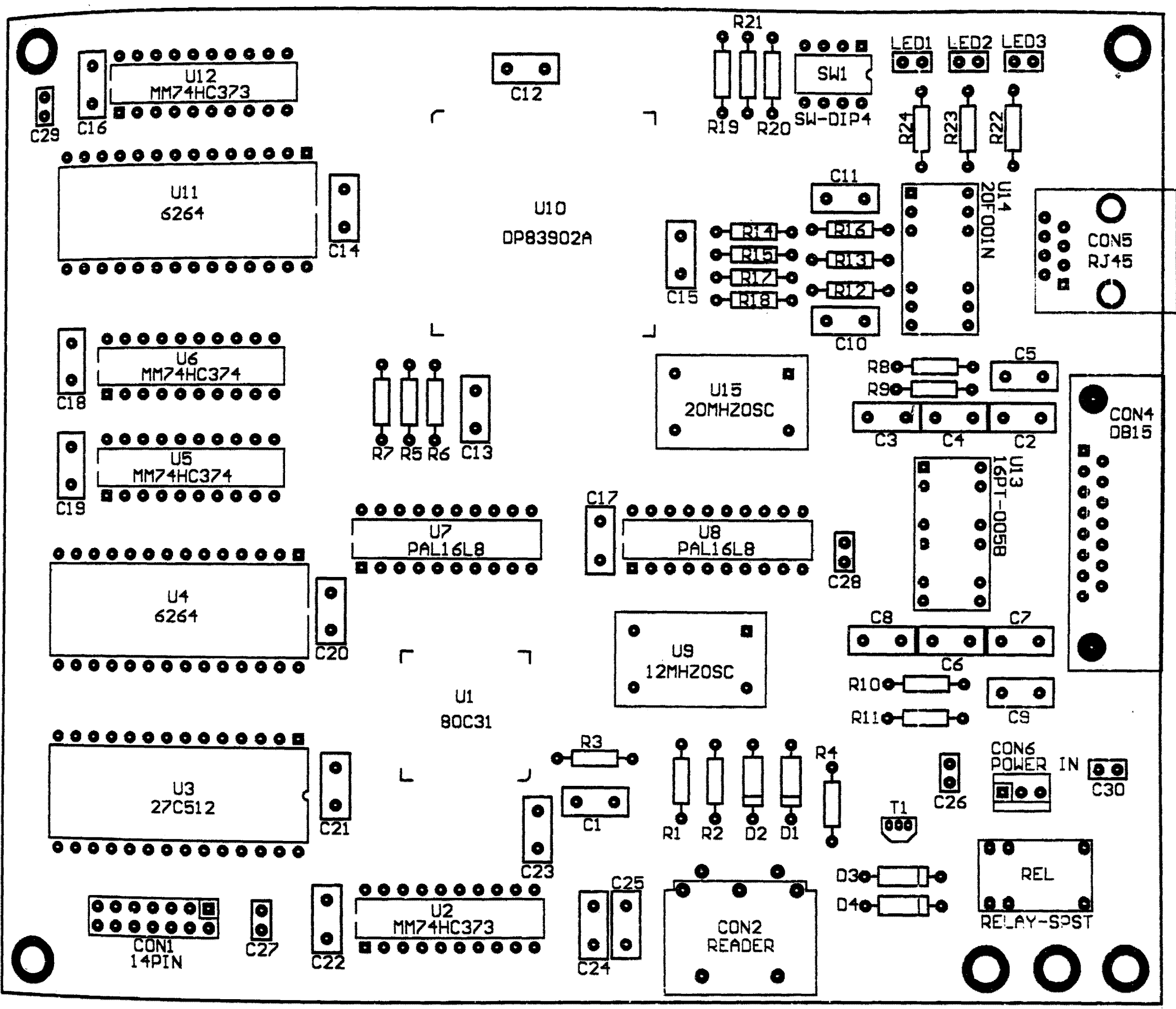
ANEXO B

8 Placa de circuito impreso del cliente de seguridad

En este anexo hemos incluido los diagramas de la última revisión de la placa de circuito impreso del cliente de seguridad. Los diagramas están divididos de la siguiente manera:

- Diagrama de colocación de componentes
- Diagrama de la capa superior de la placa
- Diagrama de la capa inferior de la placa

Esquema de colocacion de componentes

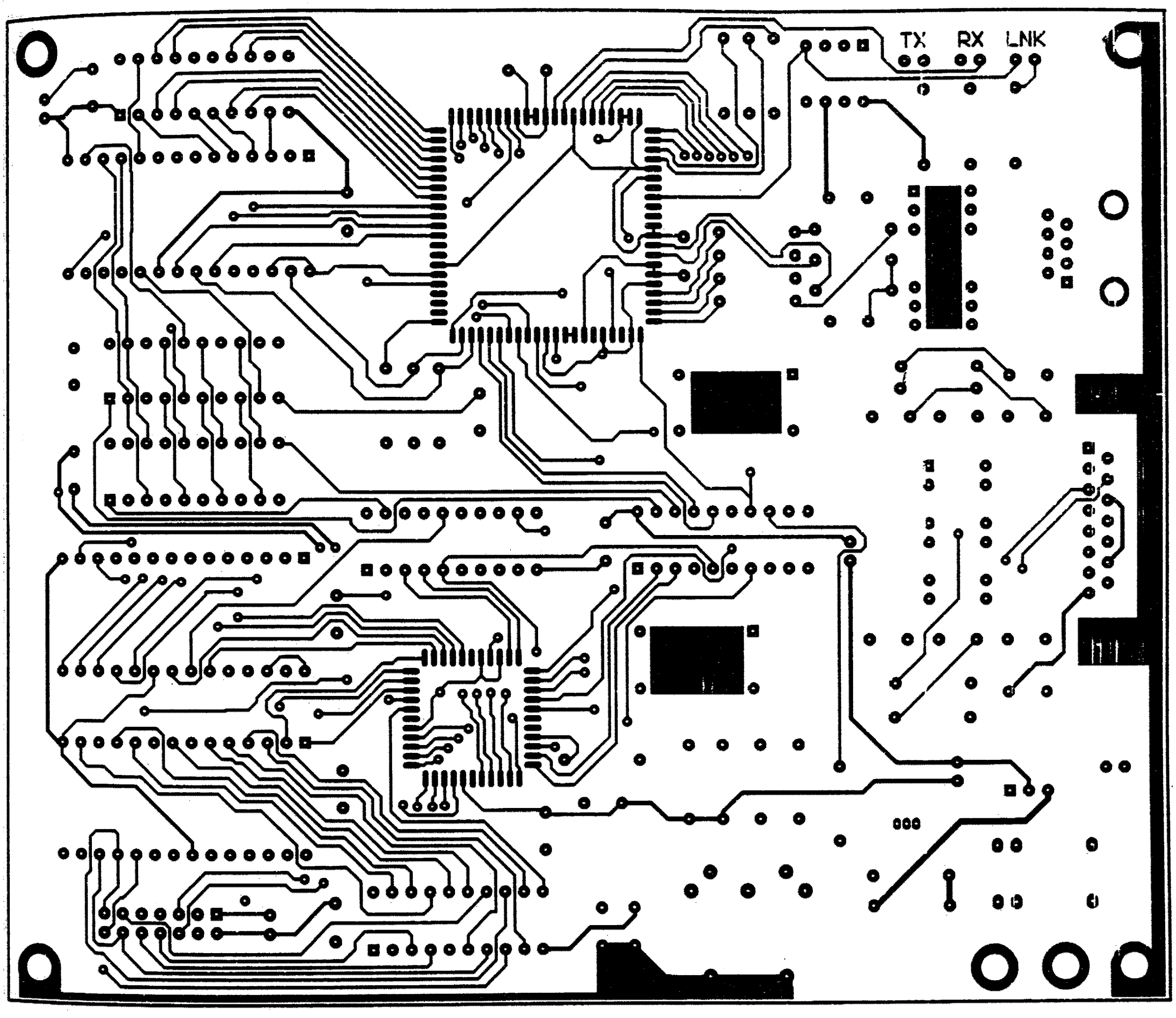


David Dominguez

Escuela Superior Politecnica del Litoral

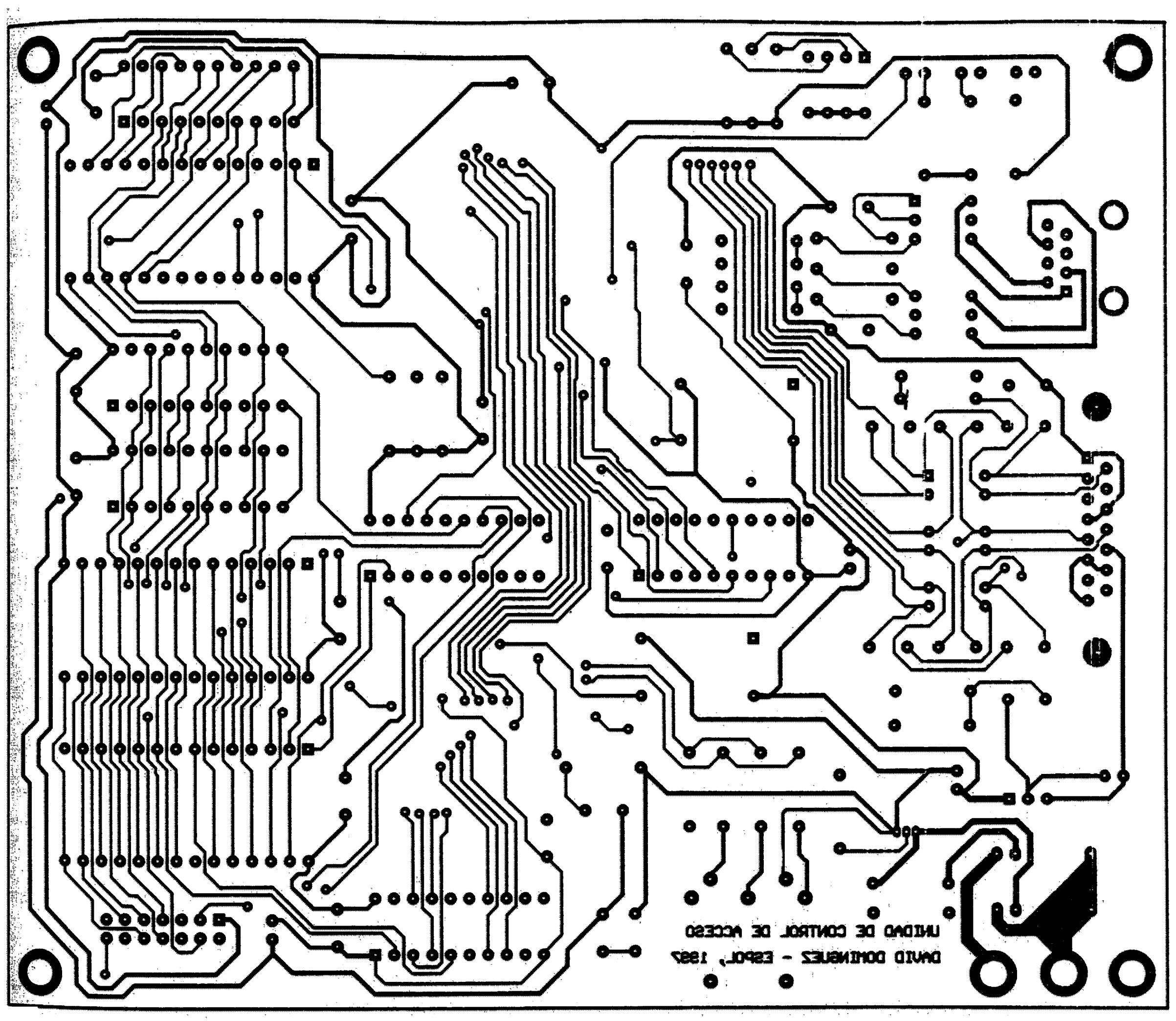
1998

Esquema de capa superior de conexiones



David Dominguez
Escuela Superior Politecnica del Litoral
1998

Esquema de capa inferior de conexiones



David Dominguez

Escuela Superior Politecnica del Litoral

1998

ANEXO C

9 Fuentes de software del cliente de seguridad

9.1 Archivo sec_cli.h

```
/*************************************************/
/*                                               */
/*              sec_cli.h                        */
/*                                               */
/* Encabezado principal del cliente de seguridad */
/*                                               */
/*        David Dominguez. ESPOL, 1998           */
/*                                               */
/*************************************************/


/****************************************************/
/* Direcciones de memoria de Periféricos de Hardware */
/****************************************************/

static unsigned char STNIC_IO         @0x8000; /* Interface de red */
static unsigned char DISPLAY_IO  @0x4000; /* Display de datos, con RS desactivado */
static unsigned char RS_DISPLAY_IO       @0x4001; /* Display de datos, con RS activado */


/************************************/
/* Variables y constantes globales  */
/************************************/

unsigned char ethadd[6] = {8,0,0x17,0,0,1}; /* Dirección Ethernet del sistema (Constante)*/

unsigned char ipadd[4];              /* Dirección IP del sistema */
unsigned char netmask[4];             /* Mascara de subred del sistema */
unsigned char gateway[4];             /* Gateway del sistema */
unsigned char server[4];             /* Servidor de seguridad */
unsigned char client_id[4];           /* Cadena de identificación del sistema */
unsigned char broadcast[4] = {255,255,255,255}; /* Dirección IP broadcast */
unsigned int  udp_port;                          /* Puerto UDP del sistema */

unsigned char program_status;

#define PACKET_RECEIVED   1
#define RESEND         2
#define NETWORK_DOWN     4
#define PASSWORD_PENDING  8
#define RELAY_ACTIVE     16
#define RESPONSE_EXPECTED 32
#define DISPLAY_FLAG     64
```

```
#define CODE_PENDING    128

/* La variable program_status variable contiene informacion de estado de varios procesos */
/*                                                                                        */
/* Bit 0: Paquete recibido de la red                                    */
/* Bit 1: Variable de reenvio de la rutina de overflow de la red    */
/* Bit 2: Indica que la interface de red esta desactivada          */
/* Bit 3: Indica que sistema espera ingreso de clave de usuario          */
/* Bit 4: Indica que relay de portero esta activado          */
/* Bit 5: Indica que sistema esta esperando respuesta del servidor      */
/* Bit 6: Indica que se han escrito datos momentaneos al display      */
/* Bit 7: Indica que el sistema esta recibiendo el código del usuario      */

unsigned char res_status;

#define IP_RESOLVED    1
#define GATEWAY_RESOLVED 2
#define NETMASK_RESOLVED 4
#define SERVER_RESOLVED  8
#define CLIENT_RESOLVED 16
#define PORT_RESOLVED        32
#define IS_LOCAL     64
#define DISPLAY_ACTIVE 128

/* res_status contiene mayormente banderas de resolución de configuración   */
/*                                                                          */
/* Bit 0: Dirección IP resuelta                                             */
/* Bit 1: Gateway resuelto                                       */
/* Bit 2: Mascara de subred resuelta                             */
/* Bit 3: Servidor de seguridad resuelto                         */
/* Bit 4: Codigo de identificacion de cliente resuelto        */
/* Bit 5: Puerto UDP del Protocolo de Seguridad resuelto        */
/* Bit 6: Indica que un paquete IP es para red local          */
/* Bit 7: Indica que el display flourescente esta activado     */

unsigned int interval;    /* Intervalo de procesos de mantenimiento */
unsigned int relay_out;    /* Contador de tiempo de activación del relay (Provisional)*/
unsigned char display_out; /* Contador de borrado de display */
unsigned char display_deact; /* Contador de desactivacion del display */

#define PASS_TIMEOUT      10 /* Tiempo de expiración de datos ingresados por usuario */
#define REPLY_TIMEOUT      2 /* Tiempo de expiración de pedidos al servidor */
#define DISPLAY_TIMEOUT    2 /* Tiempo de expiración de datos momentaneos en display */
#define DISP_ON_TIMEOUT   40 /* Tiempo de expiracion del encendido del display */


/* Variables de la rutina de interrupción de teclado */

#define KEYBUFFSIZE  11    /* Tamaño del buffer de la interface de teclado */

near unsigned char last_key;       /* Ultimo caracter recibido */
near unsigned char bit_count;      /* Bits recibidos de caracter actual */
near unsigned char key_temp;       /* Almacenamiento de caracter actual */
```

```
near unsigned char key_count;       /* Caracteres recibidos */
unsigned char key_buff[KEYBUFFSIZE]; /* Buffer de caracteres */


/************************************/
/* Estructuras de datos del sistema         */
/************************************/

/* Paquete Ethernet */

struct etherp {
  unsigned int len;
  unsigned char d_add[6];
  unsigned char s_add[6];
  unsigned int p_type;
  unsigned char data[1];
};


/* Paquete ARP */

/* Códigos de operación de ARP */

#define ARP_REQUEST  1
#define ARP_REPLY    2
#define RARP_REQUEST 3
#define RARP_REPLY   4

struct arp_packet {
  int hw_type;
  int pr_type;
  char hwa_len;
  char pra_len;
  int arp_op;
  char add_fields[1];
  };


/* Miembros del cache de ARP */


#define AE_FREE 0           /* Miembro libre */
#define AE_PENDING 1         /* Pedido pendiente */
#define AE_RESOLVED 2        /* Pedido resuelto */
#define ARP_TIMEOUT 0xFFFF  /* Tiempo de expiracion de miembro del cache */
#define ARP_RESEND 0x0FFF   /* Tiempo de expiracion de pedido ARP */
#define ARP_MAXRETRY 3       /* Maximo numero de pedidos ARP por paquete */
#define ARP_CACHE_SIZE 3     /* Maximo numero de miembros del cache */
#define MAX_ARP_QSIZE 2      /* Maximo numero de paquetes encolados por miembro */

struct arp_entry {
  unsigned char ae_state;    /* Estado del miembro (libre, pendiente, resuelto) */
  unsigned char ae_attemps;  /* Intentos de resolucion */
```

```
 unsigned int  ae_ttl;     /* timepo de vida restante del miembro */
 unsigned char n_packets;  /* Paquetes encolados */
 struct etherp *packets[MAX_ARP_QSIZE]; /* Puntero a la cola de paquetes */
 unsigned char ae_hwa[6];           /* Direccion Ethernet */
 unsigned char ae_pra[4];           /* Direccion IP */
 };

/* Cache de ARP */

struct arp_entry arp_cache[ARP_CACHE_SIZE];  /* Arp cache */


/* Paquete IP */

struct ip_packet {
  unsigned char ip_verlen;
  unsigned char ip_tos;
  unsigned int  ip_len;
  unsigned int  ip_id;
  unsigned int  ip_fragoff;
  unsigned char ip_ttl;
  unsigned char ip_proto;
        short ip_cksum;
  unsigned char ip_src[4];
  unsigned char ip_dst[4];
  unsigned char ip_data[1];
};

/* Paquete ICMP  */

struct icmp_packet {
  unsigned char ic_type;
  unsigned char ic_code;
  unsigned int  ic_cksum;
  unsigned char data[1];
};

/* Paquete UDP  */

#define UDP_ECHO          7  /* Puerto del servicio de Eco de UDP */
#define BOOTP_CLIENT_PORT 68  /* Puerto de cliente de BOOTP */
#define BOOTP_SERVER_PORT 67  /* Puerto de servidor de BOOTP */
#define SECURITY_SERVICE_PORT 515  /* Puerto de servicio de seguridad */

struct udp_packet {
  unsigned short u_src;
  unsigned short u_dst;
  unsigned short u_len;
  unsigned short u_cksum;
  unsigned char  u_data[1];
```

```
};

/* Paquete BOOTP */

#define NETMASK 1      /* Codigo de mascara de subred en paquete bootp */
#define GATEWAY 3      /* Codigo de gateway en paquete bootp */
#define SEC_SERVER 150 /* Codigo de servidor de seguridad en paquete bootp */
#define CLIENT_ID  151 /* Codigo de cliente en paquete bootp */
#define SEC_PORT   152

struct bootp {
  unsigned char op;
  unsigned char htype;
  unsigned char hlen;
  unsigned char hops;
  unsigned char t_id[4];
  unsigned int  sec;
  unsigned int  unused;
  unsigned char c_ip[4];
  unsigned char y_ip[4];
  unsigned char s_ip[4];
  unsigned char g_ip[4];
  unsigned char c_hadd[16];
  unsigned char server_name[64];
  unsigned char boot_file[128];
  unsigned char vendor_area[64];
  };

/* Miembros del area de fabricantes de BOOTP (vendor area) */

struct v_area {
  unsigned char codechar;
  unsigned char len;
  unsigned char data[1];
  };


/* Paquete de la aplicacion de seguridad */

#define IDENT_STRING_LENGTH 10    /* Longitud de codigo de indentificacion */
#define PASSWORD_STRING_LENGTH 4  /* Longitud de clave */

/* p_code es el tipo de paquete. Valores posibles:  */

/* 0 - Pedido de autorizacion                 */
/* 1 - Respuesta de autorizacion              */
/* 2 - Advertencia de exceso de equivocaciones en clave  */

#define AUTH_REQUEST        0
#define AUTH_REPLY          1
#define PASSWORD_WARNING    2

/* p_status el calificador de respuestas del servidor. */
```

```
/* 0 - Autorizado - Entrada inmediata            */
/* 1 - Autorizado - Ingrese clave                */
/* 2 - No autorizado - usuario desconocido       */
/* 3 - No autorizado - No hay supervisor         */
/* 4 - No autorizado - fuera de horario          */
/* 5 - No utilizado                              */
/* 6 - No autorizado - usuario revocado          */
/* 7 - No autorizado - no tiene privilegios de ingreso */
/* 8 - No autorizado - Sistema cliente desconocido */
/* 9 - No autorizado - usuario repetido          */
/*10 - No autorizado - error en servidor         */

#define IDENT_OK            0
#define PASSWORD_REQUIRED   1
#define IDENT_UNKNOWN       2
#define NO_SUPERVISOR       3
#define OUT_OF_SCHEDULE     4
#define DENIED_BY_ADMIN     6
#define NO_ACCESS           7
#define CLIENT_UNKNOWN      8
#define ALREADY_INSIDE      9
#define SERVER_ERROR        10

struct sec_app {
  unsigned char p_code;
  unsigned char p_status;
  unsigned  int trans_id;
  unsigned char ident[IDENT_STRING_LENGTH];
  unsigned char pwd[PASSWORD_STRING_LENGTH];
  unsigned char client[4];
  };

#define MAX_PASS_RETRIES 2  /* Maximo numero de reintentos de ingreso de clave */

/* Almacenamiento de datos de ultimo codigo ingresado */

struct last_passwd {
  unsigned char passwd[PASSWORD_STRING_LENGTH];
  unsigned char id[IDENT_STRING_LENGTH];
  unsigned int  t_id;
  unsigned char p_timeout;
  unsigned char retries;
  }l_passwd;
```

9.2 Archivo sec_cli.c

```
/***************************************************/
/*                                                  */
/*          sec_cli.c                               */
/*                                                  */
/* Rutina principal del cliente de seguridad        */
/*                                                  */
/*        David Dominguez. ESPOL, 1998              */
/*                                                  */
/***************************************************/

/*************************************/
/* Archivos incluidos en el sistema  */
/*************************************/

#include <8051.h>
#include <intrpt.h>
#include <stdlib.h>
#include <string.h>
#include "sec_cli.h"
#include "io_rout.c"
#include "display.c"
#include "h_init.c"
#include "ethernet.c"
#include "arp.c"
#include "ip.c"
#include "udp.c"
#include "bootp.c"
#include "sec_app.c"

/********************/
/* Rutina principal */
/********************/

void main(void)
{
unsigned char pass_count;

/* Macro para añadir puntero de interrupción de teclado y watchdog timer */

set_vector(EXTI1, isr_keyb);

set_vector(TIMER0, isr_watchdog);

/* Rutina de inicialización del sistema */

hardware_init();

pass_count = 0;
```

```
/* Lazo infinito de procesos del sistema */

for(;;){

 /*******Activacion del watchdog timer******/

 TH0 = 0;
 TL0 = 0;
 TH1 = 0;
 TR0 = 1;

 /******* Manejo de eventos de red *********/

 /* Deteccion de interrupcion de red */

 if(!INT0){
  poll_net();
  }

 /* Procesamiento de paquetes recibidos */

 if(program_status & PACKET_RECEIVED){
  ethernet_in();
  }

 /********* Manejo de eventos de teclado ***********/

 /* Deteccion de ingreso de caracteres */

 if(key_count){

  /* Se esta esperando una clave */

  if(program_status & PASSWORD_PENDING){

   /* Procesar ingreso de digito y mostrar asterisco en pantalla */

   if(pass_count < key_count){
    l_passwd.p_timeout = PASS_TIMEOUT;
    set_cursor2(16);
    for(pass_count = 0; pass_count < key_count; pass_count++){
     dis_rw(0x2A,16);
     }
    }

   /* Se termino de ingresar clave */

   else if(key_count >= PASSWORD_STRING_LENGTH){
    int_off();

    /* Comparar clave ingresada con clave almacenada, abrir si es correcta */

    if(memcmp(&(l_passwd.passwd[0]),&key_buff[0],PASSWORD_STRING_LENGTH) == 0){
```

```
      display_clear();
      display("Clave Aceptada:");
      set_cursor2(0);
      display("Acceso Autorizado");
      icon_display();
      program_status |= DISPLAY_FLAG;
      display_out = DISPLAY_TIMEOUT;
      program_status |= RELAY_ACTIVE;
      T1 = 0;
      relay_out = 0x7FFF;
      program_status &= ~PASSWORD_PENDING;
      }

    /* Si la clave es incorrecta, reintentar */

    else{
     if(l_passwd.retries < MAX_PASS_RETRIES){
       display_clear();
       display("Clave Erronea:");
       icon_display();
       set_cursor2(0);
       display("Reingrese Clave:");
       dis_rw(15,0);
       l_passwd.retries += 1;
       }

     /* Si se ha excedido el numero de reintentos */
     /* cancelar proceso y enviar advertencia     */

     else{
       display_clear();
       display("Clave Erronea:");
       icon_display();
       set_cursor2(0);
       display("Acceso Denegado");
       program_status |= DISPLAY_FLAG;
       display_out = DISPLAY_TIMEOUT;
       sec_out();
       program_status &= ~PASSWORD_PENDING;
       }
     }
   pass_count = 0;
   key_count = 0;
   }
  int_on();
  }

/* Procesamiento de ingreso de codigo de usuario */

else if((key_buff[key_count - 1] == 13)){  /* ENTER detectado */

 /* Si longitud de código es mayor al minimo */

 if(key_count > 6){
```

```
      int_off();
      l_passwd.retries = 0;
      l_passwd.p_timeout = REPLY_TIMEOUT;
      program_status &= ~CODE_PENDING;
      sec_out();              /* Llamar aplicacion de seguridad */
      key_count = 0;
      pass_count = 0;
      }

    /* Si codigo es muy corto, cancelar */

    else{
      key_count = 0;
      pass_count = 0;
      display_clear();
      display("ERROR:");
      set_cursor2(0);
      display("Codigo mal ingresado");
      icon_display();
      program_status &= ~CODE_PENDING;
      program_status |= DISPLAY_FLAG;
      display_out = DISPLAY_TIMEOUT;
      }
    }

  /* Si buffer se agota, no recibir mas caracteres, excepto ENTER */

  else if(key_count == KEYBUFFSIZE){
    key_count = KEYBUFFSIZE - 1;
    int_on();
    }

 /* Proceso normal de ingreso de caracter */

  else if (pass_count < key_count){
    if(!(program_status & CODE_PENDING)){
      display_clear();
      display("Ingrese Codigo:");
      icon_display();
      }
    set_cursor2(0);
    dis_rw(0x3E,16);
    dis_rw(15,0);
    for(pass_count = 0; pass_count < key_count; pass_count++){
      dis_rw(key_buff[pass_count],16);
      }
    program_status |= CODE_PENDING;
    l_passwd.p_timeout = PASS_TIMEOUT;
    }
  }

/********** Lazo de mantenimiento del sistema **********/

if (!interval){
```

```
      int_off();
      l_passwd.retries = 0;
      l_passwd.p_timeout = REPLY_TIMEOUT;
      program_status &= ~CODE_PENDING;
      sec_out();                /* Llamar aplicacion de seguridad */
      key_count = 0;
      pass_count = 0;
      }

    /* Si codigo es muy corto, cancelar */

    else{
      key_count = 0;
      pass_count = 0;
      display_clear();
      display("ERROR:");
      set_cursor2(0);
      display("Codigo mal ingresado");
      icon_display();
      program_status &= ~CODE_PENDING;
      program_status |= DISPLAY_FLAG;
      display_out = DISPLAY_TIMEOUT;
      }
    }

  /* Si buffer se agota, no recibir mas caracteres, excepto ENTER */

  else if(key_count == KEYBUFFSIZE){
    key_count = KEYBUFFSIZE - 1;
    int_on();
    }

 /* Proceso normal de ingreso de caracter */

  else if (pass_count < key_count){
    if(!(program_status & CODE_PENDING)){
      display_clear();
      display("Ingrese Codigo:");
      icon_display();
      }
    set_cursor2(0);
    dis_rw(0x3E,16);
    dis_rw(15,0);
    for(pass_count = 0; pass_count < key_count; pass_count++){
      dis_rw(key_buff[pass_count],16);
      }
    program_status |= CODE_PENDING;
    l_passwd.p_timeout = PASS_TIMEOUT;
    }
  }

/********** Lazo de mantenimiento del sistema **********/

if (!interval){
```

```
/* Si no se ha resuelto IP, enviar pedido de configuracion */

if(!(res_status & IP_RESOLVED)){
 bootp_out();
 }

/* Proceso de mantenimiento del cache de ARP */

arp_timer();

/* Si se han enviado pedidos, y ha expirado */
/* su tiempo de vida, reenviarlos          */

if (program_status & RESPONSE_EXPECTED){
 l_passwd.p_timeout--;
 if(!(l_passwd.p_timeout)){
  l_passwd.p_timeout = REPLY_TIMEOUT;
  sec_out();
  }
 }

 /* Si se espera ingreso de clave y expira el tiempo */
 /* de espera de ingreso, abortar recepcion de clave */

 else if (program_status & PASSWORD_PENDING){
  l_passwd.p_timeout--;
  if(!(l_passwd.p_timeout)){
   display_clear();
   display("Acceso Negado:");
   set_cursor2(0);
   display("Tiempo Excedido");
   icon_display();
   program_status |= DISPLAY_FLAG;
   display_out = DISPLAY_TIMEOUT;
   program_status &= ~PASSWORD_PENDING;
   pass_count = 0;
   key_count = 0;
   }
  }

 /* Si se espera ingreso de codigo de usuario y expira el tiempo */
 /* de espera de ingreso, abortar recepcion de clave             */

 else if (program_status & CODE_PENDING){
  l_passwd.p_timeout--;
  if(!(l_passwd.p_timeout)){
   display_clear();
   display("Cancelado:");
   set_cursor2(0);
   display("Tiempo Excedido");
   icon_display();
   program_status |= DISPLAY_FLAG;
   display_out = DISPLAY_TIMEOUT;
```

```
        program_status &= ~CODE_PENDING;
        pass_count = 0;
        key_count = 0;
        }
      }

    /* Si se han enviado datos transitorios al display, y expira */
    /* el tiempo de vida de los datos, borrarlos del display    */

    else if (program_status & DISPLAY_FLAG){
      display_out--;
      if(!display_out){
        program_status &= ~DISPLAY_FLAG;
        display_clear();
        icon_display();
        }
      int_on();
      }
    else if ((res_status & DISPLAY_ACTIVE) && (res_status & IP_RESOLVED)){
      display_deact--;
      if(!display_deact){
        res_status &= ~DISPLAY_ACTIVE;
        display_off();
        }
      }
    }

    /* Detección de activación de relay de portero electrico     */
    /* y desactivación del mismo despues de tiempo determinado    */
    /* por relay_out. NOTA: relay_out no esta basado en interval, */
    /* ya que el tiempo de activación del relay debe ser corto    */


  if (program_status & RELAY_ACTIVE){
    relay_out--;
    if(!relay_out){
      program_status &= ~RELAY_ACTIVE;
      T1 = 1;
      }
    }

  interval++;

  /*******Desactivacion del watchdog timer******/

  TR0 = 0;
  TH0 = 0;
  TL0 = 0;
  TH1 = 0;
  }

/* Ejecución del código del sistema nunca debe llegar a este punto */
}
```

9.3 Archivo io_rout.c

```
/*************************************************/
/*                                               */
/*              io_rout.h                        */
/*                                               */
/*    Rutinas de control de hardware             */
/*                                               */
/*         David Dominguez. ESPOL, 1998          */
/*                                               */
/*************************************************/

/****************************************************/
/* Rutina de interrupción de interface de teclado   */
/****************************************************/


/* Tabla de indexación para convertir código de tecla a formato ASCII */

unsigned char keymap[128]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,'1',0,0,0,0,0,0,0,'2',0,0,0,0,0,0,'4','3',0,0,0,0,0,
0,0,0,'5',0,0,0,0,0,0,0,'6',0,0,0,0,0,0,'7','8',
0,0,0,0,0,0,'0','9',0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,13,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,'1',0,'4','7',0,0,0,'0',0,
'2','5','6','8',0,0,0,0,'3',0,0,'9',0,0};


void interrupt isr_keyb(void)

{

/* Contar bits entrantes, y desplazar bits 1-8 hacia MCU */

switch(bit_count){
        case 0:         /* Bit de inicio (start bit) ignorado */
                bit_count++;
                break;
        case 9:
/* El noveno bit es de paridad, y se efectua el correspondiente calculo */
/* Si la paridad es incorrecta, se reemplaza el byte ingresado por 0xFF */

#asm
        mov     c,0B4h
        mov     a,_key_temp
        jc      cset
        orl     c,0D0h
        jc      ok
        jmp     fail
cset:
        anl     c,0D0h
```

```
		jnc	ok
fail:
		mov	a,#0FFh
		mov	_key_temp,a
ok:
#endasm
			bit_count++;
			break;
		case 10:

/* Al recibir el 10mo bit, se procesa el caracter ingresado */

#asm
/* Chequeamos si el caracter es un codigo de finalizacion */
/* si lo es, lo ignoramos y almacenamos                    */

		mov	a,_key_temp
		cjne	a,#0F0h,notbreak
		jmp	exit
notbreak:

/* Chequeamos si la paridad esta correcta */
/* si no lo esta, descartamos el caracter */

		cjne	a,#0FFh,parity_ok
		clr	a
exit:
		mov	_last_key,a
		jmp	tidy

/* Si el caracter previo es un codigo de finalizacion */
/* procedemos a convertir a ASCII              */

parity_ok:
		mov	a,_last_key
		cjne	a,#0F0h,tidy
		clr	a
		mov	_last_key,a
		mov	a,_key_temp
		mov	dptr,#_keymap
		add	a,dpl
		mov	dpl,a
		clr	a
		addc	a,dph
		mov	dph,a
		movx	a,@dptr
		jz	tidy
		mov	_key_temp,a
		mov	dptr,#_key_buff
		mov	a,dpl
		add	a,_key_count
		mov	dpl,a
		clr	a
		addc	a,dph
```

```
        mov     dph,a
        mov     a,_key_temp
        movx    @dptr,a
        inc     _key_count
tidy:
        mov     _bit_count,#0
#endasm

/* Si el buffer se llena, se desactiva interrupciones */

                if(key_count == KEYBUFFSIZE){
                 EX1 = 0;
           }
                break;

        default:

/*Bits del 1 al 8 son desplazados a key_temp */

#asm
        mov     c,0B4h
        mov     a,_key_temp
        rrc     a
        mov     _key_temp,a
#endasm
                bit_count++;
                break;
        }

return;
}


/***************************************************/
/* Activacion Contador de Seguridad (Watchdog Timer            */
/***************************************************/


void interrupt isr_watchdog(void)
{

/* Incrementar y chequear contador de espera */

TH1++;

if (TH1){
 return;
 }


/* Inicializar registros */

IE = 0;
IP = 0;
```

```
TMOD = 0;
PCON = 0;
TCON = 0;
TH0 = 0;
TL0 = 0;
P0 = 0xFF;
P1 = 0xFF;
P2 = 0xFF;
P3 = 0xFF;

/* Inicializar memoria externa e interna  */
/* Saltar a direccion inicial de ejecucion */

#asm

        mov     dptr,#0000h
nextx:
        mov     a,#00h
        movx    @dptr,a
        inc     dptr
        mov     a,dph
        cjne    a,#01Fh,nextx
        mov     a,dpl
        cjne    a,#0FFh,nextx
        mov     a,#00h
        movx    @dptr,a
        mov     dptr,#0000h
        mov     0D0h,#00h
        mov     r0,#01h
nexti:
        mov     @r0,#00h
        inc     00h
        cjne    r0,#07Fh,nexti
        mov     @r0,#00h
        mov     r0,#00h
        mov     081h,#07h
        mov     0F0h,#00h
        reti

#endasm

return;
}


/*****************************************************/
/* Rutina de lectura/escritura a buffers de red     */
/*****************************************************/

unsigned char netb_rw( unsigned char c, unsigned char sel)


/* sel selecciona lectura/escritura lectura=1 escritura=0  */
```

```
{

while(!P1_BITS.B4);  /* Esperar qur se active PRQ */

(sel) ? (c = STNIC_IO):(STNIC_IO = c); /* Leer/escribir en puerto E/S */

return c;
}


/*****************************************************/
/* Rutina de lectura/escritura a registros del ST-NIC            */
/*****************************************************/

unsigned char netr_rw(unsigned char c, unsigned char r)

/* r contiene direccion de registro (4 bits bajos) Bits altos  */
/* seleccionan lectura/escritura lectura=1 escritura=0         */

{
P1 &= 0xF0;      /* Inicializa RA0-RA3 */
P1 |= (r & 0x0F); /* Ingresa direccion de registro */

if(r & 0xF0){
   P1_BITS.B5 = 1;  /* Lectura: Sr/w = 1 */
   }
  else{
   P1_BITS.B5 = 0;  /* Escritura: Sr/w = 0 */
   }

P1_BITS.B7 = 0;    /* Baja CS, ST-NIC a modo esclavo */


while (P1_BITS.B6){}  /* Espera que ACK caiga */



if(r & 0xF0){            /* Lectura */
   P1_BITS.B7=1;         /* Activa CS para que ST-NIC escriba al puerto */


   while(!P1_BITS.B6){}  /*Espera que ACK suba */


   P1_BITS.B7=0;         /* Baja  CS nuevamente para evitar activacion de RACK */
   c = STNIC_IO;         /* Ciclo de lectura*/
   P1_BITS.B7=1;         /*Sube CS */
   }
  else{             /* Escritura */
   STNIC_IO = c;         /* Ciclo de escritura */
   P1_BITS.B7=1;         /* Sube CS, lo que ingresa dato al ST-NIC */
```

```
  }

while (!P1_BITS.B6);     /*Confirma que ACK ha subido */

return c;
}

/*****************************************************/
/* Rutina de lectura/escritura al display de datos          */
/*****************************************************/

unsigned char dis_rw(unsigned char c, unsigned char sel)

/* 4 bits bajos de sel seleccionan lectura/escritura lectura=1 escritura=0 */
/* 4 bits altos dan el valor de la linea RS                    */
{

while (DISPLAY_IO & 128); /* Esperar que display este listo */

if(sel & 0x0F){
 (sel & 0xF0) ? (c = RS_DISPLAY_IO) : (c = DISPLAY_IO);  /* Lectura */
 }
else{
 (sel & 0xF0) ? (RS_DISPLAY_IO = c) : (DISPLAY_IO = c);  /* Escritura */
 }

return c;
}
```

9.4 Archivo display.c

```
/**************************************************/
/*                                                */
/*              display.c                         */
/*                                                */
/*      Rutinas de manejo de display              */
/*                                                */
/*          David Dominguez. ESPOL, 1998          */
/*                                                */
/**************************************************/

/**************************************************/
/* Macros de automatizacion de manejo de display  */
/**************************************************/

/* Inicializa el display, borra pantalla */

#define display_clear() dis_rw(1,0)

/* Inicializa el cursor a cero */

#define cursor_home() dis_rw(2,0)

/* Apaga el display */

#define display_off() dis_rw(8,0)

/* Enciende el display */

#define display_on() dis_rw(12,0)

/* Coloca el cursor en la posicion x de la linea 1 */

#define set_cursor1(x) dis_rw(0x80 + x,0)

/* Coloca el cursor en la posicion x de la linea 2 */

#define set_cursor2(x) dis_rw(0xC0 + x,0)

/* Coloca el cursor en la posicion x de CGRAM */

#define set_cgram(x) dis_rw(0x40 + x,0)

/********************************************************/
/* Rutina para enviar cadenas de caracteres al display  */
/********************************************************/

void display(char *a)
{
```

```
display_on();
res_status |= DISPLAY_ACTIVE;
display_deact = DISP_ON_TIMEOUT;

while (*a != '\0'){
 dis_rw(*(a++),0x10);
 }

return;
}

/******************************************/
/* Rutina de control de iconos de sistema            */
/******************************************/

void icon_display(void)
{

/* Muestra icono de resolucion de IP, si */
/* bandera de resolucion esta activa     */

if(res_status & IP_RESOLVED){
   set_cursor1(19);
   dis_rw(1,16);
   }
else{

/* Si no, muestra icono de IP no resuelto */

  set_cursor1(19);
  dis_rw(0,16);
  }

/* Muestra icono de resolucion de servidor */
/* si su bandera esta activa               */

if(res_status & SERVER_RESOLVED){
  set_cursor1(18);
  dis_rw(2,16);
  }

/* Enciende display, e inicializa el cursor */

display_on();
res_status |= DISPLAY_ACTIVE;
display_deact = DISP_ON_TIMEOUT;

cursor_home();

return;
}
```

9.5 Archivo h_init.c

```
/**************************************************/
/*                                                */
/*          h_init.c                              */
/*                                                */
/*   Rutinas de inicializacion                    */
/*                                                */
/*        David Dominguez. ESPOL, 1998            */
/*                                                */
/**************************************************/

extern struct arp_entry arp_cache[ARP_CACHE_SIZE];
extern unsigned char netr_rw(unsigned char , unsigned char );
extern unsigned char dis_rw(unsigned char , unsigned char );

/**************************************************/
/* Rutina de inicializacion de software y hardware      */
/**************************************************/

void hardware_init(void)
{

unsigned char i;

/* Inicializacion de lineas de MCU e interrupciones */

IE = 0;     /* Borrar registro de activacion de interrupciones */
IP = 0;     /* Borrar registro de prioridad de interrupciones */
TCON = 0;    /* Borrar registro de control de contadores */
TMOD = 0x31;  /* Desabilitar contador 1, habilitar contador 0 en modo 16 bits*/

/* Activar lineas de perifericos, para que funcionen con entradas */
T0 = 1;
INT1 = 1;
INT0 = 1;

/* Activar linea del relay para que mantenerlo apagado (redundante) */
T1 = 1;

/* Configuracion de interrupciones */
/* Habilitador principal de interrupciones EA desactivado */
/* Sera activado posteriormente                    */

EA =1;  /* Habilitar interrupciones */
IT1=1;  /* Configurar interrupcion externa 1 para deteccion de transientes */
PX1=1;  /* Configurar interrupcion externa 1 para alta prioridad*/
ET0=1;  /* Habilitar interrupcion del timer 0 */
TH0=0;
TL0=0;
TH1=0;
```

```
TR0=1;  /* Activar timer 0 (watchdog timer) */

/* Ciclo de RESET de hardware del ST-NIC y display */

TXD = 1;
TXD = 0;
while (++i){}
TXD = 1;
while (++i){}
TXD = 0;

/* Inicializar variables globales */

last_key = 0;
bit_count = 0;
key_temp = 0;
key_count = 0;
program_status = 0;
res_status = 0;
interval = 0;
relay_out = 0;

for (i=0;i<4;i++){
  ipadd[i]=0;
  gateway[i]=0;
  server[i]=0;
  netmask[i]=0;
  }

/* Inicializacion de display */

dis_rw(8,0); /* Apagar pantalla */
dis_rw(1,0); /* Incializar display */
dis_rw(2,0); /* Inicializar cursor */
dis_rw(6,0); /* Configurar modo de entrada de datos */
dis_rw(0x30,0); /* Configurar bus de datos a 8 bits */
dis_rw(2,0x10); /* Intensidad de pantalla al 50% */
dis_rw(12,0); /* Activar pantalla */

/* Programar iconos de estado */

set_cgram(0);

dis_rw(3,16);          /* Icono de IP no resuelto (Char 00)*/
dis_rw(23,16);
dis_rw(17,16);
dis_rw(2,16);
dis_rw(25,16);
dis_rw(27,16);
dis_rw(31,16);
dis_rw(0,16);
```

```
dis_rw(28,16);          /* Icono de IP resuelto (Char 01)*/
dis_rw(8,16);
dis_rw(14,16);
dis_rw(29,16);
dis_rw(6,16);
dis_rw(4,16);
dis_rw(0,16);
dis_rw(0,16);

dis_rw(4,16);          /* Icono de servidor resuelto (Char 02)*/
dis_rw(10,16);
dis_rw(4,16);
dis_rw(4,16);
dis_rw(12,16);
dis_rw(4,16);
dis_rw(0,16);
dis_rw(0,16);


/* Mensaje de inicializacion */

cursor_home();
display("Cliente Ver. 1.1");
set_cursor2(0);
display("Iniciando Sistema");

icon_display(); /* Mostrar iconos de estado */

/* Inicializacion del ST-NIC */


netr_rw(0x21,0);       /* Pagina 0 de registros */
netr_rw(0x10,0x0E);  /* Registro de configuración de datos */
netr_rw(0,0x0A);      /* Registros de conteo del DMA remoto */
netr_rw(0,0x0B);
netr_rw(4,0x0C);      /* Registro de configuracion de recepcion */
netr_rw(4,0x0D);      /* Registro de configuracion de transmision */
netr_rw(6,3);        /* Puntero de frontera */
netr_rw(6,1);        /* Inicio de Pagina */
netr_rw(0x1F,2);      /* Final de Pagina */
netr_rw(0xFF,7);      /* Borrar registro de status de interrupcion */

netr_rw(0x11,0x0F); /* Registro de mascara de interrupcion A = Activado */
                    /* D0 Paquete recibido  (A)    */
                    /* D1 Paquete transmitido      */
                    /* D2 Error de recepcion       */
                    /* D3 Error de transmision     */
                    /* D4 Buffer lleno (A)         */
                    /* D5 Contadores llenos        */
                    /* D6 DMA Completo             */
                    /* D7 Reservado                */
```

```
netr_rw(0x61,0);       /* Pagina 1 de registros */

for(i=0;i<6;i++){
 netr_rw(ethadd[i],i+1);        /* Programar direccion Ethernet */
 }

netr_rw(0,8);        /* Programar direcciones multicast ( no usadas) */
netr_rw(0,9);
netr_rw(0,0x0A);
netr_rw(0,0x0B);
netr_rw(0,0x0C);
netr_rw(0,0x0D);
netr_rw(0,0x0E);
netr_rw(0,0x0F);

/* Ciclo de espera, para que mensaje de incializacion sea visible */
/* se hace antes de activar ST-NIC para que no ingresen paquetes */

while (++interval){}
display_clear();
icon_display();

netr_rw(6,7);        /* Registro de pagina actual */
netr_rw(0x22,0);       /* Activar ST-NIC */
netr_rw(0,0x0D);       /* Modo normal de operación */


/* Inicializar cache de ARP */

for (i=0; i < ARP_CACHE_SIZE;i++){
 arp_cache[i].ae_state = AE_FREE;
 arp_cache[i].n_packets = MAX_ARP_QSIZE;
 while(arp_cache[i].n_packets){
  arp_cache[i].packets[--(arp_cache[i].n_packets)] = NULL;
  }
 }

 /* Desactivar watchdog timer */

 TR0 = 0;
 TH0 = 0;
 TL0 = 0;
 TH1 = 0;

return;
}
```

```
/****************************************************/
/* Rutina de activacion de interrupcion de teclado  */
/****************************************************/

void int_on(void)
{

/* Setear TCON para borrar interrupciones generadas */
/* durante pausa de interrupcion                    */

TCON = 4;

INT1 = 1;  /* Inicializar lineas de entreda */
T0 = 1;
EX1 = 1;   /* Activar interrupcion */
return;
}


/*******************************************************/
/* Rutina de desactivacion de interrupcion de teclado  */
/*******************************************************/

void int_off(void)
{
EX1 = 0; /* Desactivar interrupcion*/
return;
}
```

9.6 Archivo ethernet.c

```
/***********************************************/
/*                                             */
/*          ethernet.c                         */
/*                                             */
/*     Rutinas de entrada/salida Ethernet      */
/*                                             */
/*       David Dominguez. ESPOL, 1998          */
/*                                             */
/***********************************************/

extern void arp_in(struct etherp *);
extern void ip_in(struct etherp *);


/*****************************/
/* Rutina de salida Ethernet */
/*****************************/

void ethernet_out(struct etherp *pep)
{

unsigned int t_count;

unsigned char *ep = (unsigned char *) pep;

/* Abortar transmision si la interface de red esta desactivada */

if(program_status & NETWORK_DOWN){
 return;
 }

ep += 2;  /* Alinear puntero de transmision con datos a enviar */

netr_rw(0x22,0);

/* Ejecutar proceso de prelectura para asegurar que suba PRQ */

/* Inicializar registros del DMA remoto */

netr_rw(1,0x0A); /* RBCR0 */
netr_rw(0,0x0B); /* RBCR1 */

netr_rw(0,8); /* RSAR0 */
netr_rw(0,9); /* RSAR1 */

netr_rw(0x0A,0); /* Ciclo de prelectura */

while(!netr_rw(0,0x18)){} /* El registro RSAR0 debe incrementarse */
```

```
/* PRQ debe estar activo ahora, empezar ciclo de escritura */

/* Chequear si se necesita relleno */

if ((pep->len) < 60){
 t_count=(pep->len) + (60 - (pep->len));
 }
else{
 t_count=(pep->len);
 }

/* Configurar registros de transmision ahora */
/* antes de decerementar el contador de bytes */

netr_rw(0,8); /* Reinicializar RSAR0 */
netr_rw(0,9); /* RSAR1 */

netr_rw(t_count%256,0x0A); /* RBCR0 */
netr_rw(t_count/256,0x0B); /* RBCR1 */

netr_rw(0,4);          /* Inicializar Registro de Inicio de Pagina de Transmision */
netr_rw(t_count%256,5); /* Transmit Byte Count 0 */
netr_rw(t_count/256,6); /* Transmit Byte Count 1 */

/* Transferir paquete al buffer de red */

netr_rw(0x12,0); /* Comando de escritura del DMA remoto */

while (t_count--){
 if((pep->len)--){
  netb_rw(*(ep++),0);  /* Transferir datos */
  }
 else{
  netb_rw(0,0);      /* Enviar relleno si se necesita */
  }
 }


/* Paquete transferido al buffer de red, ahora se ordena transmision */

netr_rw(0x26,0); /* Comando de transmision de paquete */

return;

}


/******************************/
/* Rutina de entrada Ethernet        */
/******************************/

void ethernet_in (void)
{
```

```
unsigned char next, *ep;
unsigned int byte_count;

struct etherp *pep;

netr_rw(0x0F,0x0B); /* RBCR1 cargado con 0x0F para activar el modo de envio de paquete*/

netr_rw(0x1A,0); /* Comandod de envio de paquete */

/* Primeros cuatro bytes son encabezado añadido por ST-NIC */

netb_rw(0,1);                    /* Ignorar byte de Status */
next = netb_rw(0,1);             /* Puntero del siguiente paquete */
byte_count  = netb_rw(0,1);      /* Byte bajo del contador de longitud */
byte_count += (netb_rw(0,1))*256;  /* Byte alto del contador de longitud */

/* Asignamos memoria para el paquete */

ep = (unsigned char *) calloc(1,byte_count + (byte_count & 1) + 2);

/* Estructura Ethernet apuntada a la memoria asignada */

pep = (struct etherp *) ep;


if (ep == NULL){
  while (byte_count--){
   netb_rw(0,1);        /* Si no hay memoria, se recibe y descarta paquete */
   }
  }
else{
  pep->len = byte_count; /* Guardar largo del paquete */
  ep +=2;
  while (byte_count--){
   *(ep++) = netb_rw(0,1); /* Recibir paquete */
   }

/* Aqui discriminamos el tipo de paquete */
/* segun el campo de tipo de Ethernet   */

  switch (pep->p_type){
   case 0x0806:          /* Paquete ARP */
    arp_in(pep);
    break;
   case 0x0800:          /* Paquete IP */
    ip_in(pep);
    break;
   default:
    free(pep);
    break;
   }
  }
```

```
netr_rw(0x62,0);          /* Pagina de registros 2 */
                    /* Leemos registro de pagina actual */

if (netr_rw(0,0x17) == next ){
 program_status &= ~PACKET_RECEIVED;  /* si Frontera = Actual, no mas paquetes */
 }

netr_rw(0x22,0);          /* Pagina de registros 0 */

return;
}


/**********************************************/
/* Rutina de identificacion de eventos de red     */
/**********************************************/

void poll_net(void)
{

unsigned char i;

i = netr_rw(0,0x17); /* Leer registro de estado de interrupciones de ST-NIC */


/* Paquete recibido */

if (i & 1){
 program_status |= PACKET_RECEIVED; /* Activar bandera */
 }

/* Buffer lleno */

if (i & 0x10){
 i = netr_rw(0,0x10) & 2;   /* Chequear bit de transmision de registro de comando */
 if(i){
  program_status |= RESEND; /* Usar RESEND como almacenamiento temporal del bit */
  }
 for(i=0;i < 64; i++){}      /* Esperar para que ST-NIC pueda termianr transmisiones */

 netr_rw(0x21,0);              /* Desactivar ST-NIC y */
 program_status |= NETWORK_DOWN;

 netr_rw(0,0x0A);              /* Inicializar registros de DMA remoto */
 netr_rw(0,0x0B);

 if (program_status & RESEND){  /* Chequear bit de transmision para reenviar */
  if(netr_rw(0,0x17) & 10){   /* transmisiones truncadas              */
   program_status &= ~RESEND;
   }
  }
 netr_rw(4,0x0D);                    /*Setear ST-NIC en loopback  */
 netr_rw(0x22,0);                   /* Activar STNIC          */
 while (program_status & PACKET_RECEIVED){  /* Vaciar anillo de entrada  */
```

```
  ethernet_in();
  }
 netr_rw(0x10,7);                /* Borrar bandera de buffer lleno */
 netr_rw(0,0x0D);                /* Setear ST-NIC a transmision normal */
 program_status &= ~NETWORK_DOWN;
 if(program_status & RESEND){
  netr_rw(0x26,0);               /* Transmitir datos pendientes */
  program_status &= ~RESEND;
  }
 }

netr_rw(0xFF,7); /* Inicializar registro de estado de interrupciones */

return;
}
```

9.7 Archivo arp.c

```
/**************************************************/
/*                                                */
/*              arp.c                             */
/*                                                */
/*  Rutinas de manejo del protocolo ARP           */
/*                                                */
/*       David Dominguez. ESPOL, 1998             */
/*                                                */
/**************************************************/

extern struct arp_entry *arp_find(unsigned char *);
extern struct arp_entry *arp_add(struct arp_packet *);
extern void arpq_send(struct arp_entry *);

/*************************/
/* Rutina de entrada ARP    */
/*************************/

void arp_in(struct etherp *pep )
{

struct arp_packet *ap = (struct arp_packet *) &pep->data[0];
struct arp_entry *pae;

/* Chequear que el paquete ARP sea valido para Ethernet e IP */

if(((ap->hw_type)!=1) || ((ap->pr_type)!=0x0800)){
  free(pep);
  return;
  }

/* Chequear que el paquete vaya dirigido a esta máquina...(semi-standard) */

if(memcmp(&ap->add_fields[16],ipadd,4) != 0){
  free(pep);
  return;
  }

/* Buscar miembros equivalentes del cache y revalidarlos */

if ((pae = arp_find((unsigned char *)&ap->add_fields[6])) !=NULL ){
  memcpy(&pae->ae_hwa[0],&ap->add_fields[0],6);
  pae->ae_ttl = ARP_TIMEOUT;
  }

/* Si no hay miembros equivalentes añadimos uno */

if ( pae == NULL){
  pae = arp_add(ap);
```

```
}

/* Si hay miembros equivalentes pendientes, los marcamos como resueltos */

if (pae->ae_state == AE_PENDING){
 pae->ae_state = AE_RESOLVED;
 arpq_send(pae);
 }

/* Ai el paquete es un pedido, reusar espacio para enviar respuesta */

if (ap->arp_op == ARP_REQUEST ){

/* Reorganizamos paquete para ser respuesta, */
/* llenamos los campos del paquete          */

  ap->arp_op = ARP_REPLY;
  memcpy(&ap->add_fields[10],&ap->add_fields[0],6);
  memcpy(&ap->add_fields[16],&ap->add_fields[6],4);
  memcpy(&pep->d_add[0],&ap->add_fields[10],6);
  memcpy(&pep->s_add[0],ethadd,6);
  memcpy(&ap->add_fields[0],ethadd,6);
  memcpy(&ap->add_fields[6],ipadd,4);
  pep->p_type = 0x806;

  ethernet_out(pep);  /* transmitimos el paquete */
  free(pep);
  }
 else{

  free(pep);
  }

 return;
 }


 /*************************************/
 /* Rutina de busqueda del cache ARP          */
 /*************************************/

 struct arp_entry *arp_find(unsigned char *pra)
 {

 struct arp_entry *pae;
 unsigned char i;

 /* Buscar miembros con coindican con dirección IP de argumento */

 for (i=0; i < ARP_CACHE_SIZE ; i++){
   pae = &arp_cache[i];
   if (pae->ae_state == AE_FREE){
     continue;
     }
```

```
  if (memcmp(&pae->ae_pra[0], pra, 4) == 0){
   return pae;
   }
  }
return NULL;
}


/*********************************************/
/* Rutina de adicion de miembros al cache ARP            */
/*********************************************/


extern struct arp_entry *arp_alloc(void);

struct arp_entry *arp_add(struct arp_packet *ap)
{


struct arp_entry *pae;

/* Usamos arp_alloc() para buscar espacio en el cache */
/* para nuestro nuevo miembro                         */

pae = arp_alloc();

/* inicializamos el miembro, y llenamos los campos de dirección */

pae->n_packets = 0;
memcpy(&pae->ae_hwa[0],&ap->add_fields[0],6);
memcpy(&pae->ae_pra[0],&ap->add_fields[6],4);
pae->ae_ttl = ARP_TIMEOUT;
pae->ae_state = AE_RESOLVED;

return pae;
}


/************************************************************/
/* Rutina de envio de paquetes encolados de miembros resueltos        */
/************************************************************/

void arpq_send(struct arp_entry *pae)
{


struct etherp *pep;
unsigned char i;


 /* Si no hay paquetes encolados, salimos */

if(pae->n_packets == 0){
 return;
```

```
  }

/* Si existen, transmitimos todos los paquetes encolados */

for (i = 0; i < (pae->n_packets) ;i++){
  pep = pae->packets[i];
  pae->packets[i] = NULL;
  memcpy(&pep->d_add[0],&pae->ae_hwa[0],6);
  ethernet_out(pep);
  free(pep);
  }

/* Reinicializamos la cola de paquetes del miembro */

pae->n_packets = 0;

return;
}

/*********************************************************/
/* Rutina de eliminación de colas de miembros expirados         */
/*********************************************************/

void arp_dq(struct arp_entry *pae)
{
struct etherp *pep;
unsigned char i;

if (pae->n_packets == 0){
  return;
  }

for (i = 0; i < (pae->n_packets) ;i++){ /* Descartamos paquetes */
  pep = pae->packets[i];
  pae->packets[i]=NULL;
  free(pep);
  }

pae->n_packets = 0;

return;

}


/****************************************************************/
/* Rutina asignación de espacio para miembros nuevos del cache         */
/****************************************************************/

struct arp_entry *arp_alloc(void)
{

static far unsigned char ae_next = 0;
unsigned char i;
```

```
struct arp_entry *pae;

/* Primero buscamos miembros libres */

for (i=0; i < ARP_CACHE_SIZE; i++){
  if(arp_cache[ae_next].ae_state == AE_FREE){
   break;
   }
  ae_next = (ae_next +1) % ARP_CACHE_SIZE;
  }

/* Si no hay miembros libres, usamos el siguiente miembro */
/* apuntado por la variable estatica ae_next            */
/* Si el miembro tiene pauqtes encolados, los descartamos */

pae = &arp_cache[ae_next];
ae_next = (ae_next +1) % ARP_CACHE_SIZE;

if ((pae->ae_state == AE_PENDING) && (pae->n_packets != 0)){
  arp_dq(pae);
  }
pae->ae_state = AE_PENDING;
return pae;
}


/**********************************/
/* Rutina de envio de pedidos ARP        */
/**********************************/

void arp_send(struct arp_entry *pae)
{


struct arp_packet *ap;
struct etherp *pep;

pep = (struct etherp *) calloc(1,44);  /* Asignar memoria para el pedido */

if (pep == NULL){      /* Si no hay memoria, retornar, ARP reintentará mas tarde */
  return;
  }

/* Crear el paquete ARP */

ap= (struct arp_packet *) &pep->data[0];
ap->arp_op = ARP_REQUEST;                /* Setear tipo de paquete ARP */
ap->hw_type = 1;                     /* Setear hardware a Ethernet */
ap->pr_type = 0x0800;                  /* Setear protocolo a IP */
ap->hwa_len = 6;                        /* Setear largo de direccion fisica */
ap->pra_len = 4;                        /* Setear largo de direccion lógica */
memset(&ap->add_fields[10],0,6);            /* Encerar direccion fisica de destino */
```

```
memcpy(&ap->add_fields[16],&pae->ae_pra[0],4); /* Setear direccion IP de destino */
memcpy(&ap->add_fields[0],ethadd,6);        /* Setear direccion fisica de origen */
memcpy(&ap->add_fields[6],ipadd,4);         /* Setear direccion IP de origen */

memset(&pep->d_add[0],0xFF,6);       /* Setear destino Ethernet a broadcast*/
memcpy(&pep->s_add[0],ethadd,6);     /* Setear origen Ethernet */
pep->p_type = 0x806;                 /* Setear tipo de protocolo Ethernet a ARP */
pep->len = 42;                       /* Setear tamaño de paquete Ethernet */

ethernet_out(pep);        /* Transmitir paquete Ethernet */
free(pep);

return;
}

/*********************************************/
/* Rutina de administracion del cache de ARP         */
/*********************************************/

void arp_timer(void)
{

struct arp_entry *pae;
unsigned int i;

/* Revismos todos los miembros del cache */

for (i=0; i <ARP_CACHE_SIZE; i++){
  if((pae = &arp_cache[i])->ae_state == AE_FREE){ /* Ignoramos los miembros libres */
   continue;
   }
  if((pae->ae_ttl -= 1) <= 0){        /* Los miembros resueltos que han expirado */
   if (pae->ae_state == AE_RESOLVED){  /* son liberados                        */
    pae->ae_state = AE_FREE;
    }

   /* Los miembros pendientes que han expirado y han enviado el maximo de pedidos */
   /* son liberados y sus colas descartadas                    */

   else if (++(pae->ae_attemps) > ARP_MAXRETRY){
    pae->ae_state = AE_FREE;
    arp_dq(pae);
    }

   /* Si el nuemro de pedidos enviados no es maximo, se envia un nuevo pedido */

   else{
    pae->ae_ttl = ARP_RESEND;
    arp_send(pae);
    }
   }
  }
 return;
 }
```

9.8 Archivo ip.c

```
/***********************************************/
/*                                             */
/*             ip.c                            */
/*                                             */
/*  Rutinas de manejo del protocolo IP         */
/*                                             */
/*       David Dominguez. ESPOL, 1998          */
/*                                             */
/***********************************************/

extern void icmp_in(struct etherp *);
extern void udp_in(struct etherp *);
extern short ck_sum(unsigned short *, unsigned int);

/************************/
/* Rutina de entrada IP  */
/************************/

void ip_in( struct etherp *pep )
{
struct ip_packet *pip = (struct ip_packet *) &pep->data[0];

/* Chequear version del paquete IP */
/* y longitud del encabezado      */

if (pip->ip_verlen != 0x45){
  free(pep);
  return;
  }

/* Chequear dirección de destino */

if ((memcmp(ipadd,&pip->ip_dst[0],4) != 0) && (memcmp(broadcast,&pip->ip_dst[0],4) != 0)){
  free(pep);
  return;
  }

/* Chequear si paquete esta fragmentado */

if (pip->ip_fragoff & 0x3FFF){
  free(pep);
  return;
  }

switch(pip->ip_proto){ /* Discrimimar protocolo superior */

  case 1:        /* Datagrama ICMP */
   icmp_in(pep);
   break;
```

```
 case 17:        /* Datagrama UDP  */
  udp_in(pep);
  break;

 default:
  free(pep);     /* Desconocido, descartar */
  break;
 }

return;
}

/************************/
/* Rutina de salida IP      */
/************************/

void ip_out( struct etherp *pep)
{

/* El protocolo que envia el paquete debe proveer los campos de */
/* destino de IP, origen de IP,tipo de protocolo de IP,        */
/* fragmentacion de IP y el largo del paquete                  */


struct arp_entry *pae;
struct ip_packet *pip = (struct ip_packet *) &pep->data[0];
unsigned char i;


/* Llenamos los campos del paquete */

pip->ip_verlen = 0x45;        /* Version y longitud */
pip->ip_ttl = 255;           /* TTL */
pip->ip_tos = 0;             /* Tipo de servicio normal */
pip->ip_id = interval;        /* Identificacion de IP se llena con variable interval */
pip->ip_cksum = 0;            /* Borramos sumatoria */
pip->ip_cksum = ck_sum((unsigned short *) pip, 20); /* Calculamos sumatorio */

memcpy(&pep->s_add[0],ethadd,6); /* Setear origen Ethernet */
pep->p_type=0x0800;            /* Setear protocolo Ethernet a IP */
pep->len = (pip->ip_len) + 14;   /* Setear largo de paquete Ethernet */


/* Chequeamos si el paquete es para la red local */

if (res_status & IP_RESOLVED){
 res_status |= IS_LOCAL;
 for (i=0;i<4;i++){
  if((ipadd[i] & netmask[i])  != ((pip->ip_dst[i]) & netmask[i])){
   res_status &= ~IS_LOCAL;
   }
  }
 }
```

```
if(res_status & IS_LOCAL){
  pae=arp_find(&pip->ip_dst[0]); /* Si es local, usamos arp_find() para      */
  }                              /* resolver la dirección Ethernet de destino */

/* Si el paquete no es para la red local, chequear si es un broadcast*/

else if (memcmp(&pip->ip_dst[0],broadcast,4)==0){
  memset(&pep->d_add[0],0xFF,6);          /* Setear destino de Ethernet a */
  ethernet_out(pep);                     /* broadcast y enviar paquete   */
  free(pep);
  return;
  }

/* Si no es broadcast revisamos si hay un gateway definido */

else if (res_status & GATEWAY_RESOLVED){
  pae=arp_find(gateway);
  }

else{
  free(pep);
  return;       /* Si no hay gateway, paquete no tiene ruta */
  }

/* Si se encuentra un enlace de ARP, */
/* se llena el destino Ethernet y    */
/* se envia el paquete               */

if((pae != NULL) && (pae->ae_state == AE_RESOLVED) ){
  memcpy(&pep->d_add[0],&pae->ae_hwa[0],6);
  ethernet_out(pep);
  free(pep);
  return;
  }

/* Si no existe un enlace de ARP, */
/* creamos un pedido de ARP       */

if(pae == NULL){
  pae = arp_alloc();
  pae->n_packets = 0;
  memcpy(&pae->ae_pra[0],&pip->ip_dst[0],4);
  pae->ae_attemps = 0;
  pae->ae_ttl = ARP_RESEND;
  arp_send(pae);
  }

/* Si existe un enlace no resuelto, y la cola esta llena */
/* descartamos el paquete                                */

if (pae->n_packets < MAX_ARP_QSIZE ){
  pae->packets[pae->n_packets] = (struct etherp *) pep;
```

```
 pae->n_packets++;
 }
else{
 free(pep);
 }

return;
}

/***********************************/
/* Rutina calculo de sumatoria IP          */
/***********************************/

short ck_sum(unsigned short *buf, unsigned int nchars)
{
 unsigned long sum;
 unsigned int nwords;

/* Convertimos numero de bytes a enteros de 16 bits, */
/* redondeando al inmediato superior              */

 nwords = (nchars>>1) + (nchars & 1);

/* Almacenamos la sumatoria de enteros de 16 bits */
/* en un entero de 32 bits                       */

  for (sum=0; nwords > 0;nwords--){
   sum += (unsigned long) *buf++;
   }

/* Reducimos la sumatoria de 32 a 16 bits, con carry */

  sum = (sum >>16) + (sum & 0xFFFF);
  sum += (sum >>16);

/* Retornamos el complemento a uno del resultado */

  return ~sum;
}

/**************************/
/* Rutina de entrada ICMP       */
/**************************/

void icmp_in(struct etherp *pep)
{
struct ip_packet *pip;
struct icmp_packet *ic;
unsigned int len;

pip = (struct ip_packet *) &pep->data[0];
ic = (struct icmp_packet *) &pip->ip_data[0];

len = (pip->ip_len)-20;
```

```
/* Discriminamos el tipo de paquete */

/* Pedido de Eco, reusamos el espacio del pedido para la respuesta */

if(ic->ic_type == 8){

   /* si el largo del mensaje es impar, añadimos un cero para que */
   /* la sumatoria del datagrama funcione                         */

   if(len & 1){
     pip->ip_data[len] = 0;
     }

   /*  Configuramos respuesta y la enviamos a IP */

   ic->ic_type=0;
   ic->ic_cksum = 0;
   ic->ic_cksum = ck_sum((unsigned short *) ic,len);
   memcpy(&pip->ip_dst[0],&pip->ip_src[0],4);
   memcpy(&pip->ip_src[0],ipadd,4);
   ip_out(pep);
   return;
   }

/* Paquete es un Respuesta a un pedido de mascara de subred */

if(ic->ic_type == 18){

  /* Comparamos identificador de paquete */

  if (memcmp(&ic->data[0],ethadd,4) == 0){

   /* si no hay una mascara resuelta */
   /* usamos la mascara recibida     */

   if (!(res_status & NETMASK_RESOLVED)){
     memcpy(netmask,&ic->data[4],4);
     }

   /* Si no hay un gateway, usamos la direccion de origen como gateway */

   if(!(res_status & GATEWAY_RESOLVED)){
     memcpy(gateway,&pip->ip_src[0],4);
     res_status |= GATEWAY_RESOLVED;
     }
   }
  free(pep);
  return;
  }

free(pep);
return;
}
```

```
/*************************/
/* Rutina de salida ICMP      */
/*************************/

void icmp_out(void)

{
struct etherp *pep;
struct ip_packet *pip;
struct icmp_packet *ic;

/* Pedimos asignacion de memoria pra el paquete */

pep = (struct etherp *) calloc(1,48);
if(pep == NULL){
  return;
  }

/* Construimos el paquete y llenamos los campos del mismo */

pip = (struct ip_packet *) &pep->data[0];
ic  = (struct icmp_packet *) &pip->ip_data[0];
ic->ic_cksum = 0;
ic->ic_type = 17;
ic->ic_code = 0;

/* El campo de indentificación del pedido se llena con   */
/* los cuatro primeros bytes de nuestra direccion fisica */

memcpy(&ic->data[0],ethadd,4);

/* Calculamos la sumatoria del pedido */

ic->ic_cksum = ck_sum((unsigned short *)ic,12);

pip->ip_len = 32;
memcpy(&pip->ip_dst[0],gateway,4);

pep->len = 46;

pip->ip_proto = 1;
pip->ip_fragoff = 0;
memcpy(&pip->ip_src[0],ipadd,4);

/* Transmitimos el paquete a IP */

ip_out(pep);

return;
}
```

9.9 Archivo udp.c

```
/************************************************/
/*                                              */
/*              udp.c                           */
/*                                              */
/*  Rutinas de manejo del protocolo UDP         */
/*                                              */
/*       David Dominguez. ESPOL, 1998           */
/*                                              */
/************************************************/

extern unsigned short udp_cksum(struct ip_packet *);
extern void sec_in(struct etherp *);
extern void bootp_in(struct etherp *);
extern void udp_out(struct etherp *);

/*************************/
/* Rutina de entrada UDP       */
/*************************/

void udp_in(struct etherp *pep)
{

struct ip_packet *pip = (struct ip_packet *) &pep->data[0];
struct udp_packet *up = (struct udp_packet *) &pip->ip_data[0];


 /* Verificar sumatoria, si existe */

  if(up->u_cksum && udp_cksum(pip)){
  free(pep);
  return;
  }

 /* Ahora discriminamos paquetes, segun el puerto de destino */

 /* paquete de la aplicacion de seguridad */

 if(up->u_dst == SECURITY_SERVICE_PORT){
  sec_in(pep);
  return;
  }

 /* Paquete de BOOTP */

 else if(up->u_dst == BOOTP_CLIENT_PORT){
  bootp_in(pep);
  return;
  }
```

```
/* Paquete del servicio de eco de UDP */

else if(up->u_dst == UDP_ECHO){
  up->u_dst = up->u_src;
  up->u_src = UDP_ECHO;
  memcpy(&pip->ip_dst[0],&pip->ip_src[0],4);
  udp_out(pep);
  return;
  }

/* Protocolo Desconocido */

else{
  free(pep);
  }

return;
}


/****************************************/
/* Rutina de calculo de sumatoria UDP          */
/****************************************/

unsigned short udp_cksum(struct ip_packet *pip)
{

struct udp_packet *up;
unsigned short *psh;
unsigned long sum;
unsigned short len;
unsigned short i;

up = (struct udp_packet *) &pip->ip_data[0];

sum =0;

len = up->u_len;

psh = (unsigned short *) &pip->ip_src[0];

/* Empezamos a sumar los campos del pseudo-encabezado UDP */

for(i=0; i < 4; i++){
  sum += (unsigned long) *psh++;
  }

psh = (unsigned short *) up;
sum += (unsigned long)( (pip->ip_proto) + len);

/* Ahora añadimos el propio paquete UDP, verificando si necesita relleno */

if(len & 1){
  ((unsigned char *)up)[len] = 0;
```

```
 len += 1;
 }

len =len>>1;     /* Convertimos longitud a 16-bits */

for (i=0; i<len; i++){
 sum += (unsigned long) *psh++;
 }

/* Devolvemos resultado a formato de 16 bits */

sum = (sum >>16) + (sum & 0xFFFF);
sum += (sum >>16);

/* retornamos complemento del resultado */

return (short) (~(sum & 0xFFFF));
}


/************************/
/* Rutina de salida UDP        */
/************************/

void udp_out(struct etherp *pep)
{

/* El software de capas superiores debe proveer el valor del IP de */
/* destino, puertos de origen y destino, y longitud del paquete    */

struct ip_packet *pip = (struct ip_packet *) &pep->data[0];
struct udp_packet *up = (struct udp_packet *) &pip->ip_data[0];

up->u_cksum = 0;                  /* Borrar sumatoria  */

memcpy(&pip->ip_src[0],ipadd,4);  /* Setear IP de origen */

pip->ip_len = 20 + up->u_len;     /* Setear tamaño de paquete IP */
pip->ip_proto = 17;

up->u_cksum = udp_cksum(pip);     /* Calcular sumatoria  */

if(up->u_cksum == 0){         /* Si sumatoria es cero, llenarla de unos,*/
 up->u_cksum = 0xFFFF;        /* para distingirla de falta de sumatoria */
 }


ip_out(pep);              /* Enviar paquete a IP */

return;
}
```

9.10 Archivo bootp.c

```
/*********************************************/
/*                                           */
/*            bootp.c                        */
/*                                           */
/* Rutinas de manejo del protocolo BOOTP     */
/*                                           */
/*      David Dominguez. ESPOL, 1998         */
/*                                           */
/*********************************************/


/***************************/
/* Rutina de salida BOOTP  */
/***************************/

void bootp_out(void)
{

struct etherp *pep;
struct ip_packet *pip;
struct udp_packet *up;
struct bootp *bp;

/* Asignamos espacio para el paquete */

pep = (struct etherp *) calloc(1, sizeof(struct bootp) + 44);

if(pep == NULL ){
  return;
  }

/* Empezamos la construcción del paquete */

pip = (struct ip_packet *) &pep->data[0];
up = (struct udp_packet *) &pip->ip_data[0];
bp = (struct bootp *) &up->u_data[0];

bp->op = 1;         /* Tipo de Paquete: Pedido BOOTP */
bp->htype = 1;      /* Tipo de Hardware: Ethernet */
bp->hlen = 6;       /* Longitud de direccion Ethernet */
bp->sec =5;         /* Tiempo desde inicializacion (valor arbitrario) */
memcpy(&bp->t_id[0], ethadd, 4);  /* Setear identificacion  */
memcpy(&bp->c_hadd[0],ethadd,6);  /* Setear direccion fisica de origen */

up->u_src = BOOTP_CLIENT_PORT;     /* Puertos UDP  */
up->u_dst = BOOTP_SERVER_PORT;

up->u_len = sizeof(struct bootp) + 8;   /* largo del datagrama */
```

```
pip->ip_fragoff = 0x4000;       /* Bit de no fragmentar    */
memset(&pip->ip_dst[0],0xFF,4);  /* Setear IP de destino a broadcast */

udp_out(pep);   /* Enviar paquete a UDP */

return;
}


/***************************/
/* Rutina de entrada BOOTP       */
/***************************/

void bootp_in(struct etherp *pep)
{

struct ip_packet *pip = (struct ip_packet *) &pep->data[0];
struct udp_packet *up = (struct udp_packet *) &pip->ip_data[0];
struct bootp *bp = (struct bootp *) &up->u_data[0];
struct v_area *va;


unsigned char i;
unsigned char cookie[4];   /*  "Magic Cookie" estandar*/

cookie[0] =  99;
cookie[1] = 130;
cookie[2] =  83;
cookie[3] =  99;

/* Chequear identificador con la direccion Ethernet */

if (memcmp(&bp->t_id[0], ethadd, 4) != 0){
  free(pep);
  return;
  }

/* Chequear presencia de direccion IP, copiarla si existe, activar bandera */

if ((memcmp(&bp->y_ip[0], ipadd, 4) != 0)){
  res_status |= IP_RESOLVED;
  memcpy(ipadd, &bp->y_ip[0],4);
  }
else{
  free(pep);
  return;
  }

/* Chequear "magic cookie"  a ver si corresponde al estandar */

if (memcmp(&bp->vendor_area[0], cookie,4) == 0){

  /* Examinar area de fabricantes */
```

```
i=4;
while(i < 64){
  va = (struct v_area *) &bp->vendor_area[i];
  switch (va->codechar){
    case NETMASK :                    /* Mascara de subred resuelta */
      memcpy(netmask,&va->data[0],4);
      res_status |= NETMASK_RESOLVED;
      i += (2 + va->len);
      break;
    case GATEWAY:                     /* Gateway resuelto */
      memcpy(gateway,&va->data[0],4);
      res_status |= GATEWAY_RESOLVED;
      i += (2 + va->len);
      break;
    case SEC_SERVER :                 /* Servidor de seguridad resuelto */
      memcpy(server,&va->data[0],4);
      res_status |= SERVER_RESOLVED;
      i += (2 + va->len);
      break;
    case CLIENT_ID :                  /* Identificacion de sistema resuelta */
      memcpy(client_id,&va->data[0],4);
      res_status |= CLIENT_RESOLVED;
      i += (2 + va->len);
      break;
    case SEC_PORT :                   /* Puerto UDP del sistema resuelto */
         memcpy(&udp_port,&va->data[0],2);
      res_status |= PORT_RESOLVED;
      i += (2 + va->len);
      break;
    case 0 :
      i++;
      break;
    case 255:
      i=64;
      break;
    }
  }
}

/* Si no se resuelve una mascara, enviar pedido ICMP    */
/* de mascara de subred y usar ruteo de clases hasta eso */

if (!(res_status & NETMASK_RESOLVED)){
  memset(netmask,0,4);
 if((ipadd[0] & 0x80) == 0){       /* Setear mascara de acuerdo a clase de IP */
   netmask[0] = 0xFF;               /* Clase A  */
   }
  else if((ipadd[0] & 0xC0) == 0x80){
   memset(netmask,0xFF,2);          /* Clase B  */
   }
  else if((ipadd[0] & 0xE0) == 0xC0){
   memset(netmask,0xFF,3);          /* Clase C */
   }
  res_status |= NETMASK_RESOLVED;
```

```
/* Enviamos el pedido, si hay respuesta la mascara sera actualizada */

  icmp_out();
  }

/* Si no se resuelve un codigo de cliente,       */
/* usar cuatro últimos bytes de dirección física */

if(!(res_status & CLIENT_RESOLVED)){
  memcpy(client_id, &ethadd[2] , 4);
  res_status |= CLIENT_RESOLVED;
  }


/* Si no se encuentra un gateway en el area de fabricante */
/* intentar en el campo de gateway de BOOTP               */

if (bp->g_ip[0] && !(res_status & GATEWAY_RESOLVED)){
  res_status |= GATEWAY_RESOLVED;
  memcpy(gateway, &bp->g_ip[0],4);
  }

/* Si no se especifica un puerto UDP, se utiliza el puerto por defecto */

if(!(res_status & PORT_RESOLVED)){
  udp_port = SECURITY_SERVICE_PORT;
  }

free(pep);

icon_display();

int_on();  /* Activar interrupciones despues de resolver IP */
return;
}
```

9.11 Archivo sec_app.c

```
/***********************************************/
/*                                             */
/*          sec_app.c                          */
/*                                             */
/*  Rutinas de la aplicacion de seguridad      */
/*                                             */
/*        David Dominguez. ESPOL, 1998         */
/*                                             */
/***********************************************/

/***********************************************/
/* Rutina de salida de la aplicacion de seguridad        */
/***********************************************/

void sec_out(void)
{


struct etherp *pep;
struct udp_packet *up;
struct ip_packet *pip;
struct sec_app *sec;

unsigned char i,j;

/* Chequear si la rutina ha sido llamada para enviar un paquete de */
/* autorizacion o un paquete de advertencia                  */

if(!(program_status & PASSWORD_PENDING)){

/* Paquete de autorizacion, aumentamos conteo de reintentos de envio */

  l_passwd.retries += 1;

/* Chequeamos si es el primer intento de envio de pedido */

  if (program_status & RESPONSE_EXPECTED){

    /* Si no es primer intento, chequeamos si es maximo # de intentos */

    if(l_passwd.retries == 4){   /* Abortamos proceso */
      display_clear();
      display("ERROR:");
      set_cursor2(0);
      display("Servidor no responde");
      icon_display();
      program_status |= DISPLAY_FLAG;
      display_out = DISPLAY_TIMEOUT;
      program_status &= ~RESPONSE_EXPECTED;
```

```
    l_passwd.retries = 0;
    int_on();
    return;
    }
   }
  else{            /* Primer intento */
   display_clear();
   display("Codigo Ingresado:");
   program_status |= RESPONSE_EXPECTED;
   }
  }

/* Asignar espacio para el paquete */

pep = (struct etherp *) calloc(1,44 + sizeof(struct sec_app));

if (pep == NULL){
  return;
  }

pip = (struct ip_packet *) &pep->data[0];
up = (struct udp_packet *) &pip->ip_data[0];
sec = (struct sec_app *) &up->u_data[0];

/* Discriminar construcción de paquete segun su tipo */

if(program_status & PASSWORD_PENDING){

  /* Advertencia */

  sec->p_code = PASSWORD_WARNING;
  memcpy(&sec->ident[0],&l_passwd.id[0],IDENT_STRING_LENGTH);
  }
else{

  /* Pedido de autorizacion */

  sec->p_code = AUTH_REQUEST;

  /* Preparamos para mostrar codigo ingresado en pantalla */

  set_cursor2(0);
  dis_rw(0x3E,16);

  /* Borramos códigos anteriores */

  memset(&l_passwd.id[0],'0',IDENT_STRING_LENGTH);
  memset(&sec->ident[0],'0',IDENT_STRING_LENGTH);

  /* Mostramos codigo, y lo almacenamos en paquete y en memoria */

  i=0;
  while (key_buff[i] != 13){
   i++;
```

```
  }
 i = (KEYBUFFSIZE - 1) - i;
 for(j = i; j < (KEYBUFFSIZE-1);){
  sec->ident[j] = l_passwd.id[j] = key_buff[j - i];
  dis_rw(sec->ident[j],16);
  j++;
  }

 icon_display();
 }

/* Llenamos código de transacción */

sec->trans_id = l_passwd.t_id = interval;

/* Llenamos código de cliente */

memcpy(&sec->client[0],client_id,4);

/* configuramos UDP e IP */

up->u_src = udp_port;
up->u_dst = udp_port;
up->u_len = 8 + sizeof(struct sec_app);
pip->ip_fragoff = 0;

/* Si no tenemos IP del servidor, hacemos broadcast del pedido */

if(res_status & SERVER_RESOLVED){
  memcpy(&pip->ip_dst[0],server,4);
  }
else{
  memset(&pip->ip_dst[0],0xFF,4);
  }


udp_out(pep);

return;
}


/****************************************************/
/* Rutina de entrada de la aplicacion de seguridad          */
/****************************************************/

void sec_in(struct etherp *pep)
{

struct ip_packet *pip = (struct ip_packet *) &pep->data[0];
struct udp_packet *up = (struct udp_packet *) &pip->ip_data[0];
struct sec_app *sec = (struct sec_app *) &up->u_data[0];

/* Paquetes provenientes de clientes son descartados */
```

```
if ((sec->p_code == AUTH_REQUEST) || (sec->p_code == PASSWORD_WARNING )){
 free(pep);
 return;
 }

/* Chequear ID de cliente contra nuestro ID */

if(memcmp(&sec->client[0],client_id,4) != 0){
 free(pep);
 return;
 }

/* Si no tenemos IP de servidor, usamos origen de paquete */

if(!(res_status & SERVER_RESOLVED)){
 memcpy(server, &pip->ip_src[0],4);
 res_status |= SERVER_RESOLVED;
 icon_display();
 }

/* Si tenemos servidor definido, descartamos paquetes de otros sistemas */

else if(memcmp(server,&pip->ip_src[0],4) != 0){
 free(pep);
 return;
 }

/* Confirmamos ID de transacción */

if (sec->trans_id != l_passwd.t_id ){
 free(pep);
 return;
 }

/* Si el paquete es de respuesta, lo procesamos */

if (sec->p_code == AUTH_REPLY){

 /* Si no se esta esperando una respuesta, lo descartamos */

 if(!(program_status & RESPONSE_EXPECTED)){
  free(pep);
  return;
  }

 /* Si el código de usuario no corresponde al enviado, lo dscartamos */

 else if(memcmp(&sec->ident[0],&l_passwd.id[0],IDENT_STRING_LENGTH) != 0){
  free(pep);
  return;
  }

 /* Procesamos status de respuesta */
```

```
display_clear();
switch(sec->p_status){
 case IDENT_OK:          /* Autorizado */
  display("Codigo Valido:");
  set_cursor2(0);
  display("Acceso Autorizado");
  icon_display();
  program_status |= DISPLAY_FLAG;
  display_out = DISPLAY_TIMEOUT;
  program_status |= RELAY_ACTIVE;
  relay_out = 0x7FFF;
  T1 = 0;             /* Abrimos la puerta */
  break;
 case PASSWORD_REQUIRED:  /* Debemos pedir clave */
  display("Codigo Valido:");
  icon_display();
  set_cursor2(0);
  display("Clave Requerida:");
  dis_rw(15,0);

  /* Copiamos clave de paquete a estructura de almacenamiento */

  memcpy(&(l_passwd.passwd[0]),&sec->pwd[0],PASSWORD_STRING_LENGTH);
  l_passwd.p_timeout = PASS_TIMEOUT;
  l_passwd.retries = 0;
  program_status |= PASSWORD_PENDING;
  key_count = 0;
  break;

 /* Casos de entrada denegada */

 case IDENT_UNKNOWN:
  display("Entrada Negada:");
  icon_display();
  set_cursor2(0);
  display("Codigo Desconocido");
  program_status |= DISPLAY_FLAG;
  display_out = DISPLAY_TIMEOUT;
  break;
 case OUT_OF_SCHEDULE:
  display("Entrada Negada:");
  icon_display();
  set_cursor2(0);
  display("Fuera de Horario");
  program_status |= DISPLAY_FLAG;
  display_out = DISPLAY_TIMEOUT;
  break;
 case NO_SUPERVISOR:
  display("Entrada Negada:");
  icon_display();
  set_cursor2(0);
  display("No hay supervisor");
  program_status |= DISPLAY_FLAG;
```

```
      display_out = DISPLAY_TIMEOUT;
      break;
    case DENIED_BY_ADMIN:
      display("Entrada Negada:");
      icon_display();
      set_cursor2(0);
      display("Acceso Revocado");
      program_status |= DISPLAY_FLAG;
      display_out = DISPLAY_TIMEOUT;
      break;
    case NO_ACCESS:
      display("Entrada Negada:");
      icon_display();
      set_cursor2(0);
      display("Ingreso Prohibido");
      program_status |= DISPLAY_FLAG;
      display_out = DISPLAY_TIMEOUT;
      break;
    case CLIENT_UNKNOWN:
      display("Entrada Negada:");
      icon_display();
      set_cursor2(0);
      display("Terminal desconocida");
      program_status |= DISPLAY_FLAG;
      display_out = DISPLAY_TIMEOUT;
      break;
    case ALREADY_INSIDE:
      display("Entrada Negada:");
      icon_display();
      set_cursor2(0);
      display("Usuario repetido");
      program_status |= DISPLAY_FLAG;
      display_out = DISPLAY_TIMEOUT;
      break;
    case SERVER_ERROR:
      display("Entrada Negada:");
      icon_display();
      set_cursor2(0);
      display("Error en servidor");
      program_status |= DISPLAY_FLAG;
      display_out = DISPLAY_TIMEOUT;
      break;
    default:
      break;
    }
  program_status &= ~RESPONSE_EXPECTED;
  }
free(pep);
int_on();

return;
}
```

ANEXO D

10 Fuentes del servidor de seguridad

10.1 Archivo server.c

```
/*************************************************************/
/** TOPICO DE GRADUACION: REDES LAN Y WAN                  **/
/** PROGRAMA SERVIDOR                                      **/
/** REALIZADO POR:                                         **/
/**                 DORIS VALENCIA M.                      **/
/**             LETICIA JARA J.                            **/
/**                 TONNY TRIVIÑO A.                       **/
/**                 GERARDO MALLA L.                       **/
/** FECHA: OCTUBRE DE 1996                                 **/
/**                                                        **/
/**                                                        **/
/**                                                        **/
/** MODIFICADO POR DAVID DOMINGUEZ MAYO 1998               **/
/**                                                        **/
/*************************************************************/

#include <windows.h>
#include <malloc.h>
#include <string.h>
#include <stdio.h>
#include <direct.h>
#include "modifies.h"
#include "server.h"
#include <winsock.h>
#include "util.h"
#include <sql.h>


HENV    henv;
HDBC    hdbc;
RETCODE retcode;
UCHAR FAR *  szSqlState;
SDWORD FAR * pfNativeError;
UCHAR FAR *  szErrorMsg;
SWORD FAR * pcbErrorMsg;
BOOL bBroadcast = TRUE;

LPSTR ProgramName="TCP/IP Server",IniFile="server.ini";
                                /** Archivo .ini donde se encuentran **/
                                /** definidos la direccion y puerto **/
                                /** del servidor                **/
int abrir_base_datos();
void cerrar(SOCKET);
```

```
/*****************************************************/
/*****          PROGRAMA PRINCIPAL              *****/
/*****************************************************/
int PASCAL WinMain(HINSTANCE hInst,HINSTANCE hPrev,LPSTR comando,int show)
{
        SOCKET s;
        struct sec_app *sec;
        int res,succesbd,err;
        int sock_addr_len = sizeof(struct sockaddr);
        unsigned short port;
        SOCKADDR_IN local_socket,remote_socket;
        char terror[129],PuertoIP[6],curdir[129],file[129],msg[129];
        WSADATA SockVer;

        err=WSAStartup(SOCKET_VERSION,(LPWSADATA)&SockVer);
        if(err!=0)
        {
          //WSAGetTextError(WSAGetLastError(),terror);
          //sprintf(msg,"Error en WSAStartup().\n%s",terror);
          strcpy(msg,"Error en WSAStartup()");
          MessageBox(NULL,(LPSTR)msg,ProgramName,MB_OK|MB_ICONHAND);
          return 0;
  }

        _getcwd(curdir,128);  /* ==> obtiene directorio actual */
        sprintf((LPSTR)file,"%s\\%s",(LPSTR)curdir,(LPSTR)IniFile);

        /*************** Puerto del Servidor ***************/
        res=GetPrivateProfileString("Servidor","Puerto","",PuertoIP,5,file);
        if(res<=0)
                res=GetPrivateProfileString("Servidor","Puerto","",PuertoIP,5,IniFile);

        if(res>=0)
                PuertoIP[res]=0;

        /** Se pasa al puerto la direccion de host a direccion en formato */
        /** de red                                                         */
        if(atoi(PuertoIP) != 0){
          port = htons((unsigned short) atoi(PuertoIP));
        }
        else{
          port = htons(DEF_SEC_SERVER_PORT);
  }

        /**** Se crea un socket para la comunicacion  ***/
        s = socket(PF_INET, SOCK_DGRAM, 0);
        if(s==INVALID_SOCKET)
        {
          WSAGetTextError(WSAGetLastError(),terror);
          sprintf(msg,"Error en socket().\n%s",terror);
          MessageBox(NULL,(LPSTR)msg,ProgramName,MB_OK|MB_ICONHAND);
                cerrar(s);
                return 0;
        }
```

```
/* Set up the server name */
local_socket.sin_family    = PF_INET;
local_socket.sin_port      = port;
local_socket.sin_addr.s_addr = INADDR_ANY;

/** Se relaciona el socket a una direccion de punto final (maquina) **/
res=bind(s,(LPSOCKADDR)&local_socket,sizeof(local_socket));
if(res==SOCKET_ERROR)
{
   WSAGetTextError(WSAGetLastError(),terror);
   sprintf(msg,"Error en bind().\n%s",terror);
   MessageBox(NULL,(LPSTR)msg,ProgramName,MB_OK|MB_ICONHAND);
         cerrar(s);
         return 0;
}

res = setsockopt(s, SOL_SOCKET, SO_BROADCAST,(LPSTR) &bBroadcast, sizeof(BOOL));
if (res == SOCKET_ERROR){
   WSAGetTextError(WSAGetLastError(),terror);
   sprintf(msg,"Error en bind().\n%s",terror);
   MessageBox(NULL,(LPSTR)msg,ProgramName,MB_OK|MB_ICONHAND);
         cerrar(s);
         return 0;
}

/**** Conectando a Base de Datos *******/
   succesbd=abrir_base_datos();
   if (succesbd==0)
   {
      sprintf(msg,"Error en conectar a Base de Datos");
      MessageBox(NULL,(LPSTR)msg,ProgramName,MB_OK|MB_ICONHAND);
            cerrar(s);
            return 0;
   }

   /******* Lazo infinito *******/
   do
   {

            //Asigna memoria para recibir paquetes
sec = (struct sec_app *) malloc(sizeof(struct sec_app));
      if(sec == NULL){
               return 0;
      }

      /** Lee los datos recibidos y los escribe en buf **/
         res=recvfrom(s,(char *)sec,sizeof(struct sec_app),0,(SOCKADDR *) &remote_sock
&sock_addr_len);
```

```
            if(res==SOCKET_ERROR)
        {
          WSAGetTextError(WSAGetLastError(),terror);
          sprintf(msg,"Error en recvfrom().\n%s",terror);
          MessageBox(NULL,(LPSTR)msg,ProgramName,MB_OK|MB_ICONHAND);
                      free(sec) ;
                      cerrar(s);
                      return 0;
            }


        if(res == sizeof(struct sec_app))
              {
          /** Invoca la funcion que valida los datos y retorna la ***/
          /** respuesta codificada                                ***/

    if(sec->p_code == AUTH_REQUEST){


      valida_acceso(sec);

                /*** Se envia respuesta al cliente ****/
                      sec->p_code = AUTH_REPLY;
               res=sendto(s,(char *)sec,sizeof(struct sec_app),0,(SOCKADDR *) &remote_socket,
sizeof(SOCKADDR));
            if(res==SOCKET_ERROR)
            {
              WSAGetTextError(WSAGetLastError(),terror);
                    sprintf(msg,"Error en sendto().\n%s",terror);
                    MessageBox(NULL,(LPSTR)msg,ProgramName,MB_OK|MB_ICONHAND);
                          cerrar(s);
                                free(sec);
                    return 0;
                  }

                }
    else if(sec->p_code == PASSWORD_WARNING){
      valida_acceso(sec);
                }
            }
            free(sec);


      }while(1);

      /** Se desconecta y libera todo los recursos utilizados **/
      cerrar(s);
  return 1;
}


/***********************************************************************/
```

```
/** Realiza la apertura de la base de datos para mantenerla asi durante          */
/** todas la operaciones que se realicen sobre ella                              */
/*****************************************************************************/
int abrir_base_datos()
{
 /** Se crea el manejador de ambiente  ***/
 retcode = SQLAllocEnv(&henv);          // Manejador de ambiente
 if (retcode == SQL_SUCCESS)
 {
         /** Se crea el manejador para la connecion  **/
   retcode = SQLAllocConnect(henv, &hdbc); // Manejador de Coneccion
   if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
   {

  /* Se setea para esperar un timeout de 5 seconds para conectarse. */
        retcode=SQLSetConnectOption(hdbc, SQL_LOGIN_TIMEOUT, 5);
        if(retcode==SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
     {
      // Conectamos a la Base de datos
      retcode = SQLConnect(hdbc, "siscal", SQL_NTS,
                              "sa", SQL_NTS,
                                 "", SQL_NTS);
         if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
           return 1;
      else
           return 0;
        }
        else
      return 0;
   }
   else
    return 0;
 }
 else
   return 0;
}

/******funcion cerrar recursos*******/
void cerrar(SOCKET s)
{
  SQLDisconnect(hdbc);
  SQLFreeConnect(hdbc);
  SQLFreeEnv(henv);
  closesocket(s);
  WSACleanup();
}
```

10.2 Archivo server.h

```
#define CIERTO 1
#define FALSO  0

int valida_acceso(struct sec_app *);
```

10.3 Archivo Valiacce.c

```
/************************************************************************/
/** TOPICO DE GRADUACION: REDES LAN Y WAN                             **/
/** PROGRAMA VALIDACION PARA EL ACCESO A LOS LABORATORIOS            **/
/** REALIZADO POR:                                                    **/
/**                 DORIS VALENCIA M.                                 **/
/**     LETICIA JARA J.                                               **/
/**                 TONNY TRIVIÑO A.                                  **/
/**                 GERARDO MALLA L.                                  **/
/**                                                                   **/
/**                                                                   **/
/**  MODIFICADO POR DAVID DOMINGUEZ. MAYO 1998                        **/
/**                                                                   **/
/************************************************************************/

#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <sql.h>
#include "modifies.h"


HSTMT   hstmt;
HDBC    hdbc;
RETCODE retcode;
int politica[20];

/**   DECLARACION DE LA FUNCIONES INTERNAS DEL PROGRAMA **/
int existe_usuario(char *, char *);
int DIAD(char *);
int HORAD(char *,char *);
int existe_ayudante(char *);
int registra_acceso_lab(char *,char *);
int registra_log_users(char *,char *,char *);
int FECHAD(char *);
```

```
int FECHAD2(char *);
int verifica_horario(int num_p);
int verifica_persona_politica(char xmatricula[11],char xcod_lab[5],int *num_p);
int obtener_cond_acceso(char *,char *,char *);
int verifica_registro_acceso(char *,char *);
void increm_num_ayudante(char *);
void decrem_num_ayudante(char *);
/**********************************************/
/*******     PROGRAMA PRINCIPAL        ********/
/**********************************************/
int valida_acceso(struct sec_app *sec)
{
 unsigned char xclave_lab_ing[5],xmatric_ing[IDENT_STRING_LENGTH+1],
 xclave_ing[PASSWORD_STRING_LENGTH+1];
 unsigned char clave[5],categoria[2],cod[3],no_clave[5] = {'0','0','0','0',0};
 int band,codigo_mensaje=0,w_num_ayudante=0;
 int cond_acceso,ayudante, cumple_horario,existe_politica;
 int num_polit=0,acc=0;

 memcpy(xmatric_ing,sec->ident,IDENT_STRING_LENGTH);
 xmatric_ing[IDENT_STRING_LENGTH]='\0';
 memcpy(xclave_ing,sec->pwd,PASSWORD_STRING_LENGTH);
 xclave_ing[PASSWORD_STRING_LENGTH]='\0';
 memcpy(xclave_lab_ing,sec->client,4);
 xclave_lab_ing[4]='\0';


 if(sec->p_code == PASSWORD_WARNING)
 {
  strcpy(cod,"05");  /* clave mal ingresada tres veces */
  registra_log_users(xmatric_ing,xclave_lab_ing,cod);
  return 1;
 }


 band=existe_usuario(clave,xmatric_ing);
 clave[4]='\0';

 if(band==0)
 {
   strcpy(cod,"02");  /* matricula no existe */
          sec->p_status = 2;
          registra_log_users(xmatric_ing,xclave_lab_ing,cod);
 }
 else
 {
        cond_acceso=obtener_cond_acceso(xmatric_ing,xclave_lab_ing,categoria);
        switch(cond_acceso)
   {
         case 0: strcpy(cod,"07");/*usuario no registrado en este laboratorio*/
                  sec->p_status = 7;
              registra_log_users(xmatric_ing,xclave_lab_ing,cod);
                        break;
   case 1: if(strcmp(categoria,"A")==0)
```

```
                    increm_num_ayudante(xclave_lab_ing);

              strcpy(cod,"01");   /* se le abre la puerta */
                    sec->p_status = 1;
     acc=registra_acceso_lab(xmatric_ing,xclave_lab_ing);
          if (acc==0){
                            strcpy(cod,"09");
                      sec->p_status = 9;
                    }
                    if (acc==0 || acc==11)
                            decrem_num_ayudante(xclave_lab_ing);
                    registra_log_users(xmatric_ing,xclave_lab_ing,cod);
          break;
case 2: ayudante=existe_ayudante(xclave_lab_ing);
              if(ayudante==11){
       strcpy(cod,"08");/*no hay registro en tabla para el laboratorio*/
                      sec->p_status = 8;
                    }
                    else
               if(ayudante==10){
                      strcpy(cod,"10");/*problema con sentencia de base de datos*/
                      sec->p_status = 10;
           }
           else
                            if(ayudante==1)
        {
                              if(strcmp(categoria,"A")==0)
                          increm_num_ayudante(xclave_lab_ing);

                  strcpy(cod,"01"); /* se abre puerta */
                                 sec->p_status = 1;
                  acc=registra_acceso_lab(xmatric_ing,xclave_lab_ing);
                        if (acc==0){
                                          strcpy(cod,"09");
                                    sec->p_status = 9;
                                 }
        }
            else{
                  strcpy(cod,"03"); /*No hay ayudante */
                              sec->p_status = 3;
                            }

          registra_log_users(xmatric_ing,xclave_lab_ing,cod);
          break;
case 3: existe_politica=verifica_persona_politica(xmatric_ing,xclave_lab_ing,&num_polit);
                    if(existe_politica==10){
                            strcpy(cod,"10");/*problema con sentencia de base de datos*/
                           sec->p_status = 10;
                    }
                    else
                           if(existe_politica)
                      {
                            cumple_horario=verifica_horario(num_polit);
                            if(cumple_horario==10){
```

```
                                        strcpy(cod,"10");/*problema con sentencia de base de datos*/
                                        sec->p_status = 10;
                                  }
                                       else
                                        if(cumple_horario)
                                        {
                if(strcmp(categoria,"A")==0){
                                               increm_num_ayudante(xclave_lab_ing);
                                                    }
                                              strcpy(cod,"01");/*se le permite el ingreso*/
                                                    sec->p_status = 1;
                                              acc=registra_acceso_lab(xmatric_ing,xclave_lab_ing);
                                          if (acc==0){
                                                         strcpy(cod,"09");
                                                         sec->p_status = 9;
                                                    }
                                        }
                                        else{
                                              strcpy(cod,"04");/*no cumple con horario*/
                                                    sec->p_status = 4;
                                             }
                                  }
                                  else{
                                            strcpy(cod,"06");/*politica expiro,inactiva o no existe
politica*/
                                            sec->p_status = 6;
                                      }

           registra_log_users(xmatric_ing,xclave_lab_ing,cod);
                 break;
       case 4: ayudante=existe_ayudante(xclave_lab_ing);
           if(ayudante==11){
             strcpy(cod,"08");/*no hay registro en tabla para el laboratorio*/
                                       sec->p_status = 8;
                                 }
                                 else
            if(ayudante==10){
                                        strcpy(cod,"10");/*problema con sentencia de base de datos*/
                                              sec->p_status = 10;
                                  }
                 else
                         if(ayudante==1)
                         {

existe_politica=verifica_persona_politica(xmatric_ing,xclave_lab_ing,&num_polit);
                                       if(existe_politica==10){
                                         strcpy(cod,"10");/*problema con sentencia de base de datos*/
                                         sec->p_status = 10;
                                         }
                                       else
                                        if(existe_politica)
                                          {
                                         cumple_horario=verifica_horario(num_polit);
                                         if(cumple_horario==10){
```

```
                                        strcpy(cod,"10");/*problema con sentencia de base de datos*/
                                                  sec->p_status = 10;
                                                        }
                                                    else
                                                      if(cumple_horario)
                                                      {
                                                              if(strcmp(categoria,"A")==0){
                                                    increm_num_ayudante(xclave_lab_ing);
                                                                        }
                                                      strcpy(cod,"01");/*se le permite el ingreso*/
                                                      sec->p_status = 1;

acc=registra_acceso_lab(xmatric_ing,xclave_lab_ing);
                                                          f (acc==0){
                                                            strcpy(cod,"09");
                                                            sec->p_status = 9;
                                                          }
                                                      }
                                                      else{
                                                  strcpy(cod,"04");/*no cumple con horario*/
                                                  sec->p_status = 4;
                                                              }
                                              }/*(existe_politica)*/
                                              else{
                                       strcpy(cod,"06");/*politica expiro,inactiva o no existe politica*/
                                      sec->p_status = 6;
                                                  }
                                  }/*(ayudante==1)*/
         else{
                            strcpy(cod,"03");  /* No hay ayudante */
                                         sec->p_status = 3;
                                     }

            registra_log_users(xmatric_ing,xclave_lab_ing,cod);
            break;
         case 10: strcpy(cod,"10");/*problema con sentencia de base de datos*/
                        sec->p_status = 10;
             registra_log_users(xmatric_ing,xclave_lab_ing,cod);
                            break;
    }/*switch*/
    if(sec->p_status == 1){
         if(strcmp(clave,no_clave)!=0)
      {
              memcpy(&sec->pwd[0],clave,4);
      }
         else{
              sec->p_status = 0;
         }
       }

  }
 return 1;
}
```

```
/*********************************************************************/
/*  CONVIERTE EL DIA DE ENTERO A CARACTER (LU, MA, MI, JU, ....) */
/*********************************************************************/
int DIAD(diac)
char diac[3];
{
 struct tm *newtime;

 time_t osBinaryTime;   // C run-time time (defined in <time.h>)
 time( &osBinaryTime ); // Get the current time from the
                        // operating system.
 newtime = localtime( &osBinaryTime);

 switch(newtime->tm_wday)
 {

  case 0: strcpy(diac,"DO");
       break;
  case 1: strcpy(diac,"LU");
       break;
  case 2: strcpy(diac,"MA");
       break;
  case 3: strcpy(diac,"MI");
       break;
  case 4: strcpy(diac,"JU");
       break;
  case 5: strcpy(diac,"VI");
       break;
  case 6: strcpy(diac,"SA");
       break;
 }
 diac[2]='\0';
 return 1;
}

/*************************************************************************/
/* CONVIERTE LA HORA Y MINUTOS DE ENTERO A CHAR CON FORMATO  */
/* HH:MM y HHMM                                               */
/*************************************************************************/
int HORAD(horac,horai)
char horac[6],horai[5];
{
 int i;
 char minutoc[3];
 struct tm *newtime;

 time_t osBinaryTime;   // C run-time time (defined in <time.h>)
 time( &osBinaryTime ); // Get the current time from the
                        // operating system.
 newtime = localtime( &osBinaryTime);

 for(i=0;i<=5;i++)
   horac[i]='\0';
 for(i=0;i<=2;i++)
```

```
   minutoc[i]='\0';

_itoa(newtime->tm_hour,horac,10);
_itoa(newtime->tm_hour,horai,10);
_itoa(newtime->tm_min,minutoc,10);

 if(newtime->tm_hour<=9)
 {
  horac[1]=horac[0];
  horai[1]=horai[0];
  horac[0]='0';
  horai[0]='0';
 }
 if(newtime->tm_min<=9)
 {
  minutoc[1]=minutoc[0];
  minutoc[0]='0';
 }

 horac[2]=':';
 strcat(horac,minutoc);
 strcat(horai,minutoc);
 horac[5]='\0';
 horai[4]='\0';
 return 1;
}

/*************************************************************************/
/** Valida que el usuario especificado por la matricula exista en la   **/
/** tabla Usuario retornando 1 y la clave, sino retorna 0.             **/
/*************************************************************************/
int existe_usuario(char xclave[5],char xmatricula[11])
{
  UCHAR  szclave[5];
  SDWORD     cbclave = SQL_NTS;
  char temp[100]="";

  retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
  if (retcode == SQL_ERROR)
  {
    SQLFreeStmt(hstmt, SQL_DROP);
    return 10;
  }
  memset(temp,'\0',strlen(temp));
  strcat(temp,"SELECT clave FROM Usuario WHERE matricula ='");
  strcat(temp,xmatricula);
  strcat(temp,"'");

  retcode = SQLExecDirect(hstmt,temp,SQL_NTS);
  if(retcode==SQL_ERROR)
  {
    SQLFreeStmt(hstmt, SQL_DROP);
    return 10;
  }
```

```
if(retcode==SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
  retcode=SQLBindCol(hstmt, 1, SQL_C_CHAR, szclave, 5, &cbclave);
        if(retcode==SQL_ERROR)
  {
   SQLFreeStmt(hstmt, SQL_DROP);
   return 10;
  }

  retcode = SQLFetch(hstmt);
        if(retcode==SQL_ERROR)
  {
   SQLFreeStmt(hstmt, SQL_DROP);
   return 10;
  }

  if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
        {
                strcpy(xclave,szclave);
                SQLFreeStmt(hstmt, SQL_DROP);
                return 1;
  }

        if(retcode==SQL_NO_DATA_FOUND)
        {
                SQLFreeStmt(hstmt, SQL_DROP);
                return 0;
  }
 }
}
/***********************************************************************/
/** Verifica si el campo num_ayudante de la tabla Laboratorio es mayor      **/
/** que cero (existe ayudante en el laborat.), para el laboratorio          **/
/** especifico; si es asi retorna 1 sino retorna 0 y retorna 11 cuando      **/
/** no encontro registro alguno.                                            **/
/***********************************************************************/
int existe_ayudante(char xcod_lab[5])
{
 SWORD      snum_ayudante;
 SDWORD     cnum_ayudante = 0;
 char temp[100]="";

 retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
 if (retcode == SQL_ERROR)
 {
  SQLFreeStmt(hstmt, SQL_DROP);
        return 10;
 }

 memset(temp,'\0',strlen(temp));
 strcat(temp,"SELECT num_ayudante FROM Laboratorio WHERE cod_lab =");
 strcat(temp,xcod_lab);
```

```
retcode = SQLExecDirect(hstmt,temp,SQL_NTS);
if(retcode==SQL_ERROR)
{
  SQLFreeStmt(hstmt, SQL_DROP);
        return 10;
}

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{

   retcode=SQLBindCol(hstmt, 1, SQL_C_SSHORT, &snum_ayudante, 0, &cnum_ayudante);
         if(retcode==SQL_ERROR)
         {
                 SQLFreeStmt(hstmt, SQL_DROP);
           return 10;
         }

   retcode = SQLFetch(hstmt);
   if (retcode == SQL_ERROR)
         {
     SQLFreeStmt(hstmt, SQL_DROP);
                 return 10;
         }

   if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
         if(snum_ayudante>0)
                 {
        SQLFreeStmt(hstmt, SQL_DROP);
                         return 1;
                 }
                 else
                 {
                         SQLFreeStmt(hstmt, SQL_DROP);
                         return 0;
                 }

         if(retcode==SQL_NO_DATA_FOUND)
         {
    SQLFreeStmt(hstmt, SQL_DROP);
           return 11;
         }
         else
         {
     SQLFreeStmt(hstmt, SQL_DROP);
            return 10;
         }
 }
}

/****************************************************************************/
/** Inserta o actualiza un registro en la tabla Acceso_Lab con los datos   */
/** del usuario que ingreso al laboratorio. Antes se realiza un conteo de  */
/** cuantos registros existen para poder incrementarlo en 1(para insertar) */
/****************************************************************************/
```

```
int registra_acceso_lab(char xmatricula[11],char xcod_lab[5])
{
        char xfecha[20]="\0", xhora[6]="\0", yhora[5]="\0";
 UCHAR
szmatricula[11]="\0",szhora_inicio[6]="\0",szhora_fin[6]="\0",szdia[20]="\0",szhora_equipo[5]="\0",szst
atus[2]="\0";
 SWORD      num_acceso,cod_lab,cod_equipo;
 SDWORD      cbmatricula = SQL_NTS,cbhora_equipo = SQL_NTS,cbhora_inicio = SQL_NTS,cbdia =
SQL_NTS;
 SDWORD      cbhora_fin = SQL_NTS, ccod_equipo = 0 , ccod_lab = 0, cnum_acceso = 0, cbstatus =
SQL_NTS;
 SWORD      conta;
 SDWORD      cconta=0;
 int bande=0,existe_registro=0;

 existe_registro=verifica_registro_acceso(xmatricula,xcod_lab);

 if(existe_registro==10)
 {
         return 10;
 }
 if(existe_registro==11)
 {
         return 0; //asignado equipo
 }
 if (existe_registro==1)
 {
         return 11;
 }
 if(existe_registro==0)
 {
  retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
  if (retcode == SQL_ERROR)
  {
   SQLFreeStmt(hstmt, SQL_DROP);
   return 10;
  }

/* Cuenta el número de registros que contiene la tabla para actualizar el siguiente*/
 retcode = SQLExecDirect(hstmt,
        "SELECT Count(*) AS cont FROM Acceso_Lab",
        SQL_NTS);

 if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
 {
         retcode=SQLBindCol(hstmt, 1, SQL_C_SSHORT, &conta, 0, &cconta);
         if(retcode==SQL_ERROR)
         {
    SQLFreeStmt(hstmt, SQL_DROP);
    return 10;
         }

   retcode = SQLFetch(hstmt);
   if (retcode == SQL_ERROR)
```

```
        {
    SQLFreeStmt(hstmt, SQL_DROP);
    return 10;
        }

   if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
        {
    bande=1;
   }
        else
         if(retcode==SQL_NO_DATA_FOUND)
         {
                conta=0;
                bande=1;
         }
         else
    {
            SQLFreeStmt(hstmt, SQL_DROP);
            return 10;
         }
}
else
{
        SQLFreeStmt(hstmt, SQL_DROP);
        return 10;
}

if(bande)
{
 retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
 if (retcode == SQL_ERROR)
 {
   SQLFreeStmt(hstmt, SQL_DROP);
        return 10;
 }

 retcode = SQLPrepare(hstmt,
       "INSERT INTO Acceso_Lab (num_acceso, matricula, cod_equipo, dia, hora_inicio, hora_equipo,
hora_fin, cod_lab, status) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)",
       SQL_NTS);

 if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
 {
   SQLSetParam(hstmt, 1, SQL_C_SSHORT,
             SQL_SMALLINT, 0, 0, &num_acceso, &cnum_acceso);
   SQLSetParam(hstmt, 2, SQL_C_CHAR,
             SQL_CHAR, 10, 0, &szmatricula, &cbmatricula);
   SQLSetParam(hstmt, 3, SQL_C_SSHORT,
             SQL_SMALLINT, 0, 0, &cod_equipo, &ccod_equipo);
   SQLSetParam(hstmt, 4, SQL_C_CHAR,
             SQL_CHAR, 19, 0, &szdia, &cbdia);
        SQLSetParam(hstmt, 5, SQL_C_CHAR,
             SQL_CHAR, 5, 0, &szhora_inicio, &cbhora_inicio);
        SQLSetParam(hstmt, 6, SQL_C_CHAR,
```

```
            SQL_CHAR, 4, 0, &szhora_equipo, &cbhora_equipo);
 SQLSetParam(hstmt, 7, SQL_C_CHAR,
            SQL_CHAR, 5, 0, &szhora_fin, &cbhora_fin);
        SQLSetParam(hstmt, 8, SQL_C_SSHORT,
            SQL_SMALLINT, 0, 0, &cod_lab, &ccod_lab);
        SQLSetParam(hstmt, 9, SQL_C_CHAR,
            SQL_CHAR, 1, 0, &szstatus, &cbstatus);

        num_acceso=conta+1;
 strcpy(szmatricula, xmatricula);
        szmatricula[10]='\0';
        cod_equipo = 0;  /* parameter data        */
        FECHAD(xfecha);
        strcpy(szdia, xfecha);
        szdia[19]='\0';
 HORAD(xhora,yhora);
        strcpy(szhora_inicio, xhora);
        szhora_inicio[5]='\0';
        szhora_equipo[0]='\0';
        szhora_fin[0]='\0';
        cod_lab=atoi(xcod_lab);
        strcpy(szstatus, "N");


 retcode = SQLExecute(hstmt);        /* Execute statement with */
                                  /* first row  */
        if(retcode==SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
 {
   SQLFreeStmt(hstmt, SQL_DROP);
          return 1;
        }
 else
        {
                SQLFreeStmt(hstmt, SQL_DROP);
                return 10;
        }

}
else
{
  SQLFreeStmt(hstmt, SQL_DROP);
        return 10;
}

 }
}
else
{
        return 1;
}
}
```

```
/*************************************************************************/
/** Inserta un registro en la tabla log_users con los datos del usuario    */
/** que ingreso o intento ingresar al laboratorio. Antes se hace un        */
/** conteo de cuantos registros existen para poder aumentarlo en 1.        */
/*************************************************************************/
int registra_log_users(char xmatricula[11],char xcod_lab[5],char cod[2])
{
char xfecha[18], xhora[6], yhora[5];
UCHAR      szmatricula[11],szfecha[20],szhora[5];
SWORD      cod_acceso,cod_lab,cod_error;
SDWORD      cbmatricula = SQL_NTS,cbfecha = SQL_NTS,cbhora = SQL_NTS ;
SDWORD      ccod_error = 0 , ccod_lab = 0, ccod_acceso = 0;
SWORD      conta;
SDWORD      cconta=0;
int bande=0;

retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
if (retcode == SQL_ERROR)
{
  SQLFreeStmt(hstmt, SQL_DROP);
        return 10;
}
/* Cuenta el número de registros que contiene la tabla para actualizar el siguiente*/
retcode = SQLExecDirect(hstmt,
        "SELECT Count(*) AS cont FROM log_users",
        SQL_NTS);

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
  retcode=SQLBindCol(hstmt, 1, SQL_C_SSHORT, &conta, 0, &cconta);
        if(retcode==SQL_ERROR)
        {
    SQLFreeStmt(hstmt, SQL_DROP);
          return 10;
        }

  retcode = SQLFetch(hstmt);
  if (retcode == SQL_ERROR)
        {
    SQLFreeStmt(hstmt, SQL_DROP);
          return 10;
        }

  if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
        {
    bande=1;
        }
        else
         if(retcode==SQL_NO_DATA_FOUND)
         {
                conta=0;
                bande=1;
         }
         else
```

```
            {
              SQLFreeStmt(hstmt, SQL_DROP);
              return 10;
            }
}
else
{
 SQLFreeStmt(hstmt, SQL_DROP);
 return 10;
}

if(bande)
{
 retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
 if (retcode == SQL_ERROR)
 {
   SQLFreeStmt(hstmt, SQL_DROP);
   return 10;
 }

 retcode = SQLPrepare(hstmt,
      "INSERT INTO log_users (cod_acceso, matricula, cod_lab, fecha, hora, cod_error) VALUES (?, ?,
?, ?, ?, ?)",
      SQL_NTS);

 if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
 {
   SQLSetParam(hstmt, 1, SQL_C_SSHORT,
            SQL_SMALLINT, 0, 0, &cod_acceso, &ccod_acceso);
   SQLSetParam(hstmt, 2, SQL_C_CHAR,
            SQL_CHAR, 10, 0, &szmatricula, &cbmatricula);
   SQLSetParam(hstmt, 3, SQL_C_SSHORT,
            SQL_SMALLINT, 0, 0, &cod_lab, &ccod_lab);
   SQLSetParam(hstmt, 4, SQL_C_CHAR,
            SQL_CHAR, 19, 0, &szfecha, &cbfecha);
          SQLSetParam(hstmt, 5, SQL_C_CHAR,
            SQL_CHAR, 4, 0, &szhora, &cbhora);
          SQLSetParam(hstmt, 6, SQL_C_SSHORT,
            SQL_SMALLINT, 0, 0, &cod_error, &ccod_error);

          cod_acceso=conta+1;
   strcpy(szmatricula, xmatricula);
          szmatricula[10]='\0';
          cod_lab=atoi(xcod_lab);
          FECHAD(xfecha);
          strcpy(szfecha, xfecha);
          szfecha[19]='\0';
          HORAD(xhora,yhora);
          strcpy(szhora, yhora);
          szhora[4]='\0';
   cod_error=atoi(cod);

   retcode = SQLExecute(hstmt);
          if(retcode==SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
```

```
  {
   SQLFreeStmt(hstmt, SQL_DROP);
               return 1;
        }
  else
        {
               SQLFreeStmt(hstmt, SQL_DROP);
               return 10;
        }
 }
 else
 {
        SQLFreeStmt(hstmt, SQL_DROP);
        return 10;
 }
 }/* bande */
}

/**********************************************************/
/* CONVIERTE LA FECHA DE SISTEMA EN CHAR CON FORMATO */
/* DATETIME (MM-DD-AAAA HH:MM:SS)                     */
/**********************************************************/
int FECHAD(fecha)
char fecha[20];
{
 int i;
 char minutoc[3],segundoc[3],horac[9],diac[4],mesc[4],anioc[11];
 struct tm *newtime;

 time_t osBinaryTime;   // C run-time time (defined in <time.h>)

 time( &osBinaryTime ); // Get the current time from the
               // operating system.
 newtime = localtime( &osBinaryTime);


  for(i=0;i<=8;i++)
    horac[i]='\0';
  for(i=0;i<=2;i++)
  {
        minutoc[i]='\0';
        segundoc[i]='\0';
  }
  for(i=0;i<=3;i++)
  {
   anioc[i]='\0';
   mesc[i]='\0';
   diac[i]='\0';
  }

 _itoa(newtime->tm_hour,horac,10);
 _itoa(newtime->tm_min,minutoc,10);
 _itoa(newtime->tm_sec,segundoc,10);
 _itoa(newtime->tm_mon+1,mesc,10);
```

```
 _itoa(newtime->tm_mday,diac,10);
 _itoa(newtime->tm_year+1900,anioc,10);

 if(newtime->tm_hour<=9)
 {
  horac[1]=horac[0];
  horac[0]='0';
 }
 if(newtime->tm_min<=9)
 {
  minutoc[1]=minutoc[0];
  minutoc[0]='0';
 }
 if(newtime->tm_sec<=9)
 {
  segundoc[1]=segundoc[0];
  segundoc[0]='0';
 }
 if(newtime->tm_mon+1<=9)
 {
  mesc[1]=mesc[0];
  mesc[0]='0';
 }
 if(newtime->tm_mday<=9)
 {
  diac[1]=diac[0];
  diac[0]='0';
 }

 horac[2]=':';
 strcat(horac,minutoc);
 horac[5]=':';
 strcat(horac,segundoc);
 horac[8]='\0';

 strcpy(fecha,mesc);
 strcat(fecha,"-");
 strcat(fecha,diac);
 strcat(fecha,"-");
 strcat(fecha,anioc);

 strcat(fecha," ");
 strcat(fecha,horac);
 fecha[19]='\0';
 return 1;
}

/***********************************************************/
/* CONVIERTE LA FECHA DE SISTEMA EN CHAR CON FORMATO  */
/* (AAAA-MM-DD HH:MM:SS)                                    */
/***********************************************************/
int FECHAD2(fecha)
char fecha[20];
{
```

```
int i;
char minutoc[3],segundoc[3],horac[9],diac[4],mesc[4],anioc[11];
struct tm *newtime;

time_t osBinaryTime;   // C run-time time (defined in <time.h>)

time( &osBinaryTime ); // Get the current time from the
                       // operating system.
newtime = localtime( &osBinaryTime);


 for(i=0;i<=8;i++)
   horac[i]='\0';
 for(i=0;i<=2;i++)
 {
           minutoc[i]='\0';
           segundoc[i]='\0';
 }
 for(i=0;i<=3;i++)
 {
  anioc[i]='\0';
  mesc[i]='\0';
  diac[i]='\0';
 }

_itoa(newtime->tm_hour,horac,10);
_itoa(newtime->tm_min,minutoc,10);
_itoa(newtime->tm_sec,segundoc,10);
_itoa(newtime->tm_mon+1,mesc,10);
_itoa(newtime->tm_mday,diac,10);
_itoa(newtime->tm_year+1900,anioc,10);

 if(newtime->tm_hour<=9)
{
 horac[1]=horac[0];
 horac[0]='0';
}
if(newtime->tm_min<=9)
{
 minutoc[1]=minutoc[0];
 minutoc[0]='0';
}
if(newtime->tm_sec<=9)
{
 segundoc[1]=segundoc[0];
 segundoc[0]='0';
}
if(newtime->tm_mon+1<=9)
{
 mesc[1]=mesc[0];
 mesc[0]='0';
}
if(newtime->tm_mday<=9)
{
```

```
  diac[1]=diac[0];
  diac[0]='0';
 }

 horac[2]=':';
 strcat(horac,minutoc);
 horac[5]=':';
 strcat(horac,segundoc);
 horac[8]='\0';

 strcpy(fecha,anioc);
 strcat(fecha,"-");
 strcat(fecha,mesc);
 strcat(fecha,"-");
 strcat(fecha,diac);

 strcat(fecha," ");
 strcat(fecha,horac);
 fecha[19]='\0';
 return 1;
}

/*****************************************************************/
/** Incrementa en 1 el campo num_ayudante de la tabla Laboratorio */
/** para el laboratorio especifico                              */
/*****************************************************************/
void increm_num_ayudante(char xcod_lab[5])
{
 char temp[100]="";
 retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
 if (retcode == SQL_ERROR)
 {
   SQLFreeStmt(hstmt, SQL_DROP);
          return ;
 }

 memset(temp,'\0',strlen(temp));
 strcat(temp,"UPDATE Laboratorio SET num_ayudante=num_ayudante+1 WHERE cod_lab=");
 strcat(temp,xcod_lab);

 retcode = SQLExecDirect(hstmt,temp,SQL_NTS);

 if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
 {
          return ;
 }
 else
 {
          return ;
 }
}
/**/
```

```
/*******************************************************************/
/** Decrementa en 1 el campo num_ayudante de la tabla Laboratorio        */
/** para el laboratorio especifico                                        */
/*******************************************************************/
void decrem_num_ayudante(char xcod_lab[5])
{
 char temp[100]="";
 retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
 if (retcode == SQL_ERROR)
 {
   SQLFreeStmt(hstmt, SQL_DROP);
          return ;
 }

 memset(temp,'\0',strlen(temp));
 strcat(temp,"UPDATE Laboratorio SET num_ayudante=num_ayudante-1 WHERE cod_lab=");
 strcat(temp,xcod_lab);

 retcode = SQLExecDirect(hstmt,temp,SQL_NTS);

 if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
 {
          return ;
 }
 else
 {
          return ;
 }
}


/*******************************************************************/
/** Selecciona y guarda los codigos de politica en la estructura          */
/** politica; para ser selecionada la politica debe estar activa y        */
/** no debe estar vencida para la matricula y codigo de laboratoirio      */
/** especifico                                                            */
/*******************************************************************/
int verifica_persona_politica(xmatricula,xcod_lab,num_p)
char xmatricula[11],xcod_lab[5];
int *num_p;
{
 char xfecha[18]="";
 char temp[200]="";

 SWORD      cod_politica;
 SDWORD     ccod_politica = 0 ;

 retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
 if (retcode == SQL_ERROR)
 {
  SQLFreeStmt(hstmt, SQL_DROP);
  return 10;
 }
```

```
memset(temp,'\0',strlen(temp));
FECHAD2(xfecha);
strcat(temp,"SELECT cod_politica FROM Persona_Politica WHERE matricula='");
strcat(temp,xmatricula);
strcat(temp,"' AND cod_lab=");
strcat(temp,xcod_lab);
strcat(temp," AND politica_activa='S' AND fecha_expiracion>'");
strcat(temp,xfecha);
strcat(temp,"'");

retcode = SQLExecDirect(hstmt, temp, SQL_NTS);
if(retcode==SQL_ERROR)
{
  SQLFreeStmt(hstmt, SQL_DROP);
  return 10;
}
if(retcode==SQL_SUCCESS || retcode==SQL_SUCCESS_WITH_INFO)
{
  retcode=SQLBindCol(hstmt, 1, SQL_C_SSHORT, &cod_politica, 0, &ccod_politica);
  if(retcode==SQL_ERROR)
  {
   SQLFreeStmt(hstmt, SQL_DROP);
   return 10;
  }

  *num_p=0;
  while (TRUE)
  {
   retcode = SQLFetch(hstmt);
   if (retcode == SQL_ERROR)
   {
         SQLFreeStmt(hstmt, SQL_DROP);
    return 10;
   }
   if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
   {
         politica[*num_p]=cod_politica;
         *num_p=*num_p+1;
   }
   if(retcode==SQL_NO_DATA_FOUND)
        if(*num_p==0)
         {
          SQLFreeStmt(hstmt, SQL_DROP);
          return 0;
    }
         else
         {
          SQLFreeStmt(hstmt, SQL_DROP);
          return 1;
    }
  }
}
}
```

```
/*************************************************************************/
/** Verifica que el dia y la hora de alguna de las politicas guardadas        */
/** en la estructura politica, este dentro del dia y hora actual. Esta        */
/** verificacion se la realiza en la tabla Detalle_Politica                   */
/*************************************************************************/
int verifica_horario(num_p)
int num_p;
{
  SWORD      ssecuencia;
  SDWORD     csecuencia = 0;
  char temp[200]="",string[4]="";
  char xdia[3],xhora[6],yhora[5];
  int politi;

  DIAD(xdia);
  xdia[2]='\0';

  HORAD(xhora,yhora);
  xhora[5]='\0';

  num_p--;
  while(num_p>=0)
  {
   retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
   if (retcode == SQL_ERROR)
   {
    SQLFreeStmt(hstmt, SQL_DROP);
    return 10;
   }
         memset(temp,'\0',strlen(temp));
         politi=politica[num_p];
   itoa(politi, string, 10);
   strcat(temp,"SELECT secuencia FROM Detalle_Politica WHERE cod_politica = ");
   strcat(temp,string);
   strcat(temp," AND dia = '");
   strcat(temp,xdia);
   strcat(temp,"' AND '");
   strcat(temp,xhora);
   strcat(temp,"' BETWEEN hora_inicio AND hora_fin");

   retcode = SQLExecDirect(hstmt,temp,SQL_NTS);
   if (retcode == SQL_ERROR)
   {
    SQLFreeStmt(hstmt, SQL_DROP);
    return 10;
   }
   if(retcode==SQL_SUCCESS || retcode==SQL_SUCCESS_WITH_INFO)
   {
    retcode=SQLBindCol(hstmt, 1, SQL_C_SSHORT, &ssecuencia, 0, &csecuencia);
    if (retcode == SQL_ERROR)
    {
     SQLFreeStmt(hstmt, SQL_DROP);
     return 10;
    }
```

```
/**********************************************************************/
/** Verifica que el dia y la hora de alguna de las politicas guardadas    */
/** en la estructura politica, este dentro del dia y hora actual. Esta    */
/** verificacion se la realiza en la tabla Detalle_Politica               */
/**********************************************************************/
int verifica_horario(num_p)
int num_p;
{
  SWORD      ssecuencia;
  SDWORD     csecuencia = 0;
  char temp[200]="",string[4]="";
  char xdia[3],xhora[6],yhora[5];
  int politi;

  DIAD(xdia);
  xdia[2]='\0';

  HORAD(xhora,yhora);
  xhora[5]='\0';

  num_p--;
  while(num_p>=0)
  {
   retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
   if (retcode == SQL_ERROR)
   {
    SQLFreeStmt(hstmt, SQL_DROP);
    return 10;
   }
         memset(temp,'\0',strlen(temp));
         politi=politica[num_p];
   itoa(politi, string, 10);
   strcat(temp,"SELECT secuencia FROM Detalle_Politica WHERE cod_politica = ");
   strcat(temp,string);
   strcat(temp," AND dia = '");
   strcat(temp,xdia);
   strcat(temp,"' AND '");
   strcat(temp,xhora);
   strcat(temp,"' BETWEEN hora_inicio AND hora_fin");

   retcode = SQLExecDirect(hstmt,temp,SQL_NTS);
   if (retcode == SQL_ERROR)
   {
     SQLFreeStmt(hstmt, SQL_DROP);
     return 10;
   }
   if(retcode==SQL_SUCCESS || retcode==SQL_SUCCESS_WITH_INFO)
   {
     retcode=SQLBindCol(hstmt, 1, SQL_C_SSHORT, &ssecuencia, 0, &csecuencia);
     if (retcode == SQL_ERROR)
     {
      SQLFreeStmt(hstmt, SQL_DROP);
      return 10;
     }
```

```
retcode = SQLFetch(hstmt);
if (retcode == SQL_ERROR)
{
     SQLFreeStmt(hstmt, SQL_DROP);
 return 10;
}
if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
        SQLFreeStmt(hstmt, SQL_DROP);
        return 1;
}
if(retcode==SQL_NO_DATA_FOUND)
        if(num_p==0)
        {
         SQLFreeStmt(hstmt, SQL_DROP);
         return 0;
        }
num_p--;
}
}
}
/***********************************************************************/
/** Le asigna a la categoria P o S o A una condicion de acceso 1, y          **/
/** a cualquier otra categoria una condicion de acceso 4, retornando         **/
/** esta cond. acceso y la categoria                                          **/
/***********************************************************************/
int obtener_cond_acceso(xmatricula,xcod_lab,xcategoria)
char xmatricula[11],xcod_lab[5],xcategoria[2];
{
 UCHAR  szcategoria[2];
 SDWORD     cbcategoria = SQL_NTS;
 char temp[100]="";

 retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
 if (retcode == SQL_ERROR)
 {
   SQLFreeStmt(hstmt, SQL_DROP);
   return 10;
 }
 memset(temp,'\0',strlen(temp));
 strcat(temp,"SELECT categoria FROM Usuario_Lab WHERE matricula ='");
 strcat(temp,xmatricula);
 strcat(temp,"' AND cod_lab=");
 strcat(temp,xcod_lab);

 retcode = SQLExecDirect(hstmt,temp,SQL_NTS);
 if(retcode==SQL_ERROR)
 {
   SQLFreeStmt(hstmt, SQL_DROP);
   return 10;
 }

 if(retcode==SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
 {
```

```
    retcode=SQLBindCol(hstmt, 1, SQL_C_CHAR, szcategoria, 2, &cbcategoria);
    if(retcode==SQL_ERROR)
    {
     SQLFreeStmt(hstmt, SQL_DROP);
     return 10;
    }

    retcode = SQLFetch(hstmt);
    if(retcode==SQL_ERROR)
    {
     SQLFreeStmt(hstmt, SQL_DROP);
     return 10;
    }

    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
    {
            if(strcmp(szcategoria,"P")==0 || strcmp(szcategoria,"S")==0 || strcmp(szcategoria,"A")==0)
            {
              strcpy(xcategoria,szcategoria);
                    xcategoria[1]='\0';
              SQLFreeStmt(hstmt, SQL_DROP);
              return 1;
      }
            else
            {
              strcpy(xcategoria,szcategoria);
              xcategoria[1]='\0';
              SQLFreeStmt(hstmt, SQL_DROP);
              return 4;
      }
    }

    if(retcode==SQL_NO_DATA_FOUND)
    {
            SQLFreeStmt(hstmt, SQL_DROP);
            return 0;
    }
  }
}
/***********************************************************************/
/** Verifica en la tabla Acceso_Lab si existe algun registro del dia con      */
/** status N (no atendido) y actualiza la hora_inicio; si no existe          */
/** retorna codigo 0; si existe algun registro con status E retorna          */
/** codigo 11 (usuario ya esta en el laboratorio.                            */
/***********************************************************************/
int verifica_registro_acceso(char xmatricula[11],char xcod_lab[5])
{
 UCHAR      szhora_inicio[6]="\0",szdia[20]="\0",szstatus[2]="\0";
 SDWORD     cbhora_inicio = SQL_NTS,cbdia = SQL_NTS, cbstatus = SQL_NTS;
 char temp[200]="\0",fecha[20]="\0",xhora[6]="\0", yhora[5]="\0", fechac[11]="\0";

  retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
  if (retcode == SQL_ERROR)
  {
```

```
  SQLFreeStmt(hstmt, SQL_DROP);
  return 10;
}

memset(temp,'\0',strlen(temp));
memset(fecha,'\0',strlen(fecha));
FECHAD(fecha);
strncpy(fechac,fecha,10);
fechac[10]='\0';
strcat(temp,"SELECT dia,hora_inicio,status FROM Acceso_Lab WHERE matricula ='");
strcat(temp,xmatricula);
strcat(temp,"' AND cod_lab=");
strcat(temp,xcod_lab);
strcat(temp," AND status<>'A' AND CONVERT(char(12),dia,110)='");
strcat(temp,fechac);
strcat(temp,"'");

retcode = SQLExecDirect(hstmt,temp,SQL_NTS);
if(retcode==SQL_ERROR)
{
  SQLFreeStmt(hstmt, SQL_DROP);
  return 10;
}

if(retcode==SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
  retcode=SQLBindCol(hstmt, 1, SQL_C_CHAR, szdia, 20, &cbdia);
        retcode=SQLBindCol(hstmt, 2, SQL_C_CHAR, szhora_inicio, 6, &cbhora_inicio);
        retcode=SQLBindCol(hstmt, 3, SQL_C_CHAR, szstatus, 2, &cbstatus);
  if(retcode==SQL_ERROR)
  {
   SQLFreeStmt(hstmt, SQL_DROP);
   return (10);
  }

        while(1)
        {
          retcode = SQLFetch(hstmt);
    if(retcode==SQL_ERROR)
    {
      SQLFreeStmt(hstmt, SQL_DROP);
      return 10;
    }

    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
    {
     if (strcmp(szstatus,"E")==0)
                {
                        SQLFreeStmt(hstmt, SQL_DROP);
                        return 11; //Asignado
                }
                else
                {
                                SQLFreeStmt(hstmt, SQL_DROP);
```

```
                                        memset(temp,'\0',200);
                                        memset(fecha,'\0',20);
                                        memset(xhora,'\0',6);
                                        memset(yhora,'\0',5);
                                        FECHAD(fecha);
                                        HORAD(xhora,yhora);
                                        strcat(temp,"UPDATE Acceso_Lab SET dia='");
                                        strcat(temp,fecha);
                                        strcat(temp,"', hora_inicio='");
                                        strcat(temp,xhora);
                                        strcat(temp,"' WHERE matricula='");
                                        strcat(temp,xmatricula);
                                        strcat(temp,"' AND cod_lab=");
                                        strcat(temp,xcod_lab);
                                        strcat(temp," AND dia='");
                                        strcat(temp,szdia);
                                        strcat(temp,"'");

                                        retcode = SQLAllocStmt(hdbc, &hstmt); // Statement handle
                                        if (retcode == SQL_ERROR)
                                        {
                                         SQLFreeStmt(hstmt, SQL_DROP);
                                         return 10;
                                        }

                                        retcode = SQLExecDirect(hstmt,temp,SQL_NTS);
                                        if(retcode==SQL_ERROR)
                                        {
                                                SQLFreeStmt(hstmt, SQL_DROP);
                                                return 10;
                                        }

                                        if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
                                        {
                                                SQLFreeStmt(hstmt, SQL_DROP);
                                                return 1;
                                        }
                        }
                }

        if(retcode==SQL_NO_DATA_FOUND)
        {
                SQLFreeStmt(hstmt, SQL_DROP);
                        return 0;
        }
    }/*while*/
  }
}
```

10.4 Archivo modifies.h

```
#define DEF_SEC_SERVER_PORT  515
#define IDENT_STRING_LENGTH   10
#define PASSWORD_STRING_LENGTH 4
#define CLIENT_NAME_LENGTH    4

#define AUTH_REQUEST        0
#define AUTH_REPLY          1
#define PASSWORD_WARNING    2

/* p_code is the packet type. Possible values are:  */
/* 0 - Client Discovery                             */
/* 1 - Client Discovery Reply                       */
/* 2 - Auth Request                                 */
/* 3 - Auth Reply                                   */
/* 4 - Password Warning                             */

/* p_status is the server reply qualifier. Values are: */
/* 0 - Authentified - Inmediate entry              */
/* 1 - Authentified - Enter Password               */
/* 2 - Not authentified - User unknown             */
/* 3 - Not authentified - No supervisor in lab     */
/* 4 - Not authentified - Out of schedule          */
/* 5 - Not used                     */
/* 6 - Not authentified - Administratively denied    */
/* 7 - Not authentified - User can't access this lab  */
/* 8 - Not authentified - Unknown Client           */
/* 9 - Not authentified - User already inside      */
/*10 - Not authentified - Database Error           */
struct sec_app {
 unsigned char p_code;
 unsigned char p_status;
       int s_id;
 char ident[IDENT_STRING_LENGTH];
 char pwd[PASSWORD_STRING_LENGTH];
 char client[CLIENT_NAME_LENGTH];
 };
```

ANEXO E

11 Fuentes de la lógica programable

11.1 Archivo pal1.pld

```
Name     PAL1;
Date     27/08/97;
Revision  2;
Designer  D. Dominguez;
Company   ESPOL;
Device    P16L8;

/*****************************************************************/
/* Allowable Target Device Types:PAL16L8                         */
/*****************************************************************/

/** Inputs **/

Pin 1    = ADD15  ;   /*Address line 15 (High)      */
Pin 2    = ADD14  ;   /*Address line 14 (High)      */
Pin 3    = RD     ;   /*MCU RD line (Low)           */
Pin 4    = WR     ;   /*MCU WR line (Low)           */
Pin 5    = CS     ;   /*CS Line (Low)               */

/** Outputs **/

Pin 19   =OE1     ;   /*I/O port Output Enable 1 line */
Pin 18   =RACK    ;   /*ST-NIC Read Acknowledge line  */
Pin 17   =RDlcd   ;   /*Display Read line             */
Pin 16   =RDram   ;   /*System RAM Read line          */
Pin 15   =CLK2    ;   /*I/O Port Clock 2 line         */
Pin 14   =WRlcd   ;   /*Display Write line            */
Pin 13   =WRram   ;   /*System RAM Write line         */
Pin 12   =WACK    ;   /*ST-NIC Write Acknowledge line */

/** Logic Equations **/


OE1= !(ADD15 & !ADD14 & !RD)     ;
RACK= !(ADD15 & !ADD14 & !RD & CS) ;
RDlcd= !(!ADD15 & ADD14 & !RD)   ;
RDram= !(!ADD15 & !ADD14 & !RD)  ;
CLK2= !(ADD15 & !ADD14 & !WR)    ;
WRlcd= !(!ADD15 & ADD14 & !WR)   ;
WRram= !(!ADD15 & !ADD14 & !WR)  ;
WACK= !(ADD15 & !ADD14 & !WR & CS) ;
```

11.2 Archivo pal2.pld

```
Name     PAL2;
Date     27/08/97;
Revision  2;
Designer  D. Dominguez;
Company   ESPOL;
Device    P16L8;

/********************************************************************/
/* Allowable Target Device Types:PAL16L8                            */
/********************************************************************/

/** Inputs **/

Pin 1    = PRD    ;  /*ST-NIC Port Read line (Low)    */
Pin 2    = ACK    ;  /*ST-NIC ACK line (Low)          */
Pin 3    = SRw    ;  /*Select Read/Write line (High)  */
Pin 4    = PWR    ;  /*ST-NIC Port Write line (Low)   */
Pin 5    = CS     ;  /*CS Line (Low)                  */
Pin 6    = INT    ;  /*ST-NIC Interrupt line (Low)    */
Pin 7    = RELAY  ;  /*Inverter for Relay Trigger     */
Pin 8    = RESET  ;  /* Reset Line                    */

/**Outputs **/

Pin 19   =OE2     ;  /*I/O port Output Enable 2 line  */
Pin 18   =CLK1    ;  /*I/O Port Clock 1 line          */
Pin 17   =SRD     ;  /*ST-NIC Slave Read line         */
Pin 16   =SWR     ;  /*ST-NIC Slave Write line        */
Pin 15   =INVINT  ;  /*MCU External Interrupt 0 line  */
Pin 14   =INVREL  ;  /*Inverter for Relay Trigger     */
Pin 13   =INVRES  ;  /*Inverter for Reset Line        */

/** Logic Equations **/

OE2= !(!PRD # (!ACK & !SRw))    ;
CLK1= !(!PWR # (!CS & SRw))     ;
SRD= !(!CS & SRw)               ;
SWR= !(!CS & !SRw)              ;
INVINT= !INT                    ;
INVREL= !RELAY                  ;
INVRES= !RESET                  ;
```

11.3 Archivo pal1.jed

```
CUPL(WM)      4.7a  Serial# MW-50777772
Device        p16l8  Library DLIB-h-36-8
Created       Fri Oct 31 23:55:00 1997
Name          PAL1
Partno        00001
Revision      2
Date          27/08/97
Designer      D. Dominguez
Company       ESPOL

*QP20
*QF2048
*G0
*F0
*L00000 11111111111111111111111111111111
*L00032 10011011111111111111111111111111
*L00256 11111111111111111111111111111111
*L00288 10011011111101111111111111111111
*L00512 11111111111111111111111111111111
*L00544 01101011111111111111111111111111
*L00768 11111111111111111111111111111111
*L00800 10101011111111111111111111111111
*L01024 11111111111111111111111111111111
*L01056 10011111011111111111111111111111
*L01280 11111111111111111111111111111111
*L01312 01101111011111111111111111111111
*L01536 11111111111111111111111111111111
*L01568 10101111011111111111111111111111
*L01792 11111111111111111111111111111111
*L01824 10011111011011111111111111111111
*C3EDA
* CDA9
```

11.4 Archivo pal2.jed

```
CUPL(WM)      4.8a  Serial# MW-66999998
Device       p16l8  Library DLIB-h-37-8
Created      Tue Nov 11 10:25:05 1997
Name         PAL2
Partno       00002
Revision      2
Date         27/08/97
Designer      D. Dominguez
Company        ESPOL

*QP20
*QF2048
*G0
*F0
*L00000 11111111111111111111111111111111
*L00032 11101111111111111111111111111111
*L00064 10111011111111111111111111111111
*L00256 11111111111111111111111111111111
*L00288 11111111101111111111111111111111
*L00320 11110111111110111111111111111111
*L00512 11111111111111111111111111111111
*L00544 11110111111110111111111111111111
*L00768 11111111111111111111111111111111
*L00800 11111011111110111111111111111111
*L01024 11111111111111111111111111111111
*L01056 11111111111111110111111111111111
*L01280 11111111111111111111111111111111
*L01312 11111111111111111111011111111111
*L01536 11111111111111111111111111111111
*L01568 11111111111111111111111101111111
*C3EE2
* CDC6
```

ANEXO F

12 Fuentes adicionales de información y referencia

En esta sección detallamos la manera de encontrar la información de referencia necesaria para el estudio de esta tesis. La mayoría de la información y software mencionados aquí han sido entregados a la FIEC junto con este trabajo; sin embargo, para referencia futura del lector, detallamos a continuación la información necesaria para que el lector investigue más a fondo los temas tratados durante esta tesis.

- **Display fluorescente Noritake ITRON CU200225ECPB-U1J:** Una copia del manual de usuario de este display ha sido entregada a la FIEC junto con este trabajo, para referencia futura. Este display fluorescente es compatible a nivel de programación con la popular familia HD44780 de controladores LCD, así que la información disponible en el Web en referencia al HD44780 será de utilidad para el lector. Se puede encontrar especificaciones del equipo en la página Web del fabricante: http://www.itron-ise.co.jp

- **Controlador Ethernet National DP83902A ST-NIC:** El manual de desarrollo, además de varias notas de aplicación y programación de este dispositivo están disponibles en formato Adobe Acrobat en la página Web del fabricante: http://www.national.com

- **Microcontrolador Intel 80C31:** El manual de desarrollo, información adicional, notas de aplicación y referencias a sistemas y herramientas de desarrollo están disponibles en formato Adobe Acrobat en la página Web del fabricante: http://developer.intel.com Además existe una gran cantidad de información adicional en el Web sobre esta familia de controladores, disponible tanto de otras fabricantes (Philips, Atmel, Matra, etc.) así como de desarrolladores y usuarios.

- **Unidades de lógica programable Texas Instruments TIBPAL16L8:** Las especificaciones de uso, en formato Adobe Acrobat, están disponibles en la página Web del fabricante: http://www.ti.com/sc/pld/tipld.htm

- **Filtros de red YCL 20F001N y 16PT-005B:** Los diagramas esquemáticos y tablas de equivalencia con otros fabricantes se pueden encontrar en la página Web del fabricante: http://www.ycl.com

- **Programas de diseño electrónico Protel Advanced Schematics y Protel Advanced PCB:** El manual de usuario(en formato Adobe Acrobat), información de soporte, librerías adicionales y demos están disponibles en la página Web del fabricante: http://www.protel.com

- **Programa de compilación de PLDs CUPL:** Una versión reducida del producto, el manual de usuario, actualizaciones, tutoriales y soporte están disponibles en la página Web del fabricante: http://www.logicaldevices.com

- **Sistema de desarrollo Avocet AVCASE51 para el 80C31/51:** Existe una limitada cantidad de información, apoyo técnico y demos en la página Web del fabricante del software: http://www.avocetsystems.com

- **Sistema de desarrollo Microsoft Visual C++ 4.2:** Información sobre el producto, actualizaciones y soporte en línea pueden obtenerse en la página Web del fabricante: http://www.microsoft.com

- **Aplicación monitor PAULMON1 para el 80C31/51:** El código, esquemas de diseño, e información de uso se pueden encontrar en la página Web del autor de la aplicación: http://www.ece.ortst.edu/~paul/8051-goodies/paulmon1.html

13 Bibliografía

1. COMER, DOUGLAS E. Internetworking with TCP/IP Volúmenes I y II, segunda Edición. Prentice Hall, 1991.

2. QUINN, BOB; SHUTE, DAVE. Windows Sockets Network Programming. Addison Wesley, 1995

3. FERRERO, ALEXIS. The evolving Ethernet. Addison Wesley, 1996

4. PHILIPS SEMICONDUCTOR. Connecting a PC keyboard to the I^2-C bus. Application Note AN434. Philips Semiconductor, 1992

5. PHILIPS SEMICONDUCTOR. 80C51 Family Architecture. Philips Semiconductor, 1995

6. PHILIPS SEMICONDUCTOR. 80C51 Family Programmer´s Guide and Instruction Set. Philips Semiconductor, 1995

7. PHILIPS SEMICONDUCTOR. 80C51 Family Hardware Description. Philips Semiconductor, 1995

8. WAKEMAN, LARRY. The Design and Operation of a Low Cost, 8-bit PC-X Compatible Ethernet Adapter using the DP83902. Application Note 842. Nation Semiconductor Corp. 1993

9. SCHUE, RICK. Interfacing the Densitron LCD to the 8051. Application Brief AI 39. Intel Corp. 1996

10. WILLIAMSON, TOM. Designing Microcontroller Systems for Electrically Noi Enviroments. Application Note AP-125. Intel Corp. 1993

11. NATIONAL SEMICONDUCTOR CORP. DP83902A ST-NIC Serial Netwc Interface Controller for Twisted Pair. National Semiconductor Corp. 1995

12. VALENCIA, DORIS; JARA, LETICIA; TRIVIÑO, TONNY; MALL GERARDO. Código fuente del programa "server.exe". Tópico de Graduaci "REDES LAN Y WAN". ESPOL. 1996