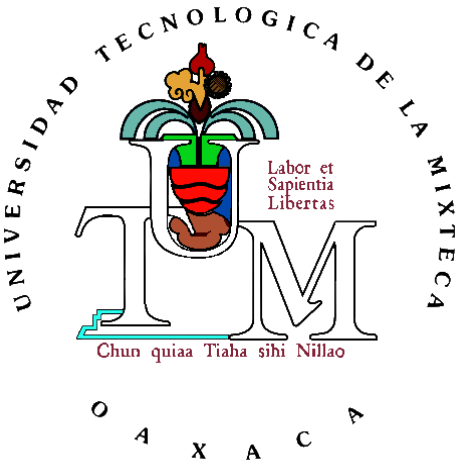


UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

Sistema mínimo de propósito general basado en el microcontrolador DS80C400 con operación en un sistema de red

TESIS

Para obtener el título de**:**
Ingeniero en Electrónica

Presenta:
Rubén Alberto Oviedo Edison

Director de Tesis:
M. C. Felipe Santiago Espinosa

Huajuapan de León, Oaxaca; Julio de 2007

Tesis presentada el 05 de Julio de 2007 ante los siguientes sinodales:

M. C. Mónica Edith García García.
M. C. Alejandro E. Ramírez González.
M. C. Jacob J. Vásquez Sanjuán.

Bajo la dirección de:

M. C. Felipe Santiago Espinosa.

ÍNDICE GENERAL

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

La evolución tecnológica ha llevado al acceso instantáneo de la información y la comunicación inmediata entre las personas. Las tecnologías de la comunicación y de la información triunfan en la medida en que ayudan a compartir saberes entre instituciones y ciudadanos, entre grupos de ciudadanos o entre culturas [URL 1].

Hoy, la población mundial conectada a Internet usa la red para las comunicaciones interpersonales. Pero este uso típico de las computadoras e Internet representa tan sólo una fracción del potencial de estas tecnologías [URL 2].

El reto es que la tecnología sea la base sobre la cual implantar aplicaciones útiles. En este sentido, cobran gran relevancia las actividades de investigación y desarrollo en el área de redes. Por otro lado, los microcontroladores han evolucionado a tal grado que incluyen hardware para soportar protocolos de comunicación a través de una red. La próxima meta es la comunicación a distancia entre personas y dispositivos, y/o dispositivos y otros. Debido a que una casa típica tiene sistemas electrónicos candidatos a la conexión integrada a Internet, hay muchas oportunidades para utilizar Internet en tareas cotidianas, incluyendo sistemas de seguridad, dispositivos médicos, control de fábricas, etc.

Con esta premisa, en el presente trabajo se plantea el diseño y la construcción de un sistema mínimo con trabajo en un sistema de red con las características necesarias para el desarrollo de aplicaciones. Un sistema de propósito general que sea la plataforma para satisfacer necesidades específicas que involucren Internet y en un futuro se adapte en tareas cotidianas.

PLANTEAMIENTO DEL PROBLEMA

Actualmente Internet es el mayor medio de comunicación en el mundo. Esto explica el interés de instituciones, empresas y particulares por hacerse un espacio en este ámbito mediante diversas aplicaciones que se han desarrollado y se siguen haciendo.

La elaboración de aplicaciones, ya sea a través de software o dispositivos físicos, que interactúen por medio de una red son de uso común y hay diversos programas y dispositivos que existen para tal propósito. Haciendo uso de esta disponibilidad, y teniendo como base la comunicación a través de la red, en este trabajo se plantea el desarrollo de un sistema mínimo de propósito general basado en el microcontrolador DS80C400 que tiene las características precisas

para trabajo en red. La selección de este microcontrolador se debe a las características propias del dispositivo, porque conforma un sistema mínimo por sí solo que permite realizar diseños de dimensiones reducidas [URL 3].

El desarrollo del trabajo involucra el estudio de las características del DS80C400, sus recursos, su repertorio de instrucciones y el acondicionamiento para su puesta en marcha.

El sistema a realizar tiene como finalidad proporcionar una base sólida para el desarrollo de aplicaciones que permitan utilizar los beneficios de un sistema de red, además del apoyo académico para la comprensión del funcionamiento de comunicación en red por medio de la implementación de prácticas.

En consideración de lo anterior, se pretende realizar un sistema mínimo que opere conectado a la red. Contando principalmente con los módulos siguientes: unidad de control y proceso, unidad de memoria, unidad de entrada/salida y unidad de comunicación a Internet.

Unidad de control y proceso

Es el núcleo del sistema, generalmente conocida como unidad central de proceso (CPU, *Central Process Unit*). Su principal función es la de interpretar y ejecutar las instrucciones, realizando las operaciones aritméticas y lógicas indicadas por las instrucciones del programa. En el sistema, el microcontrolador DS80C400 cumplirá con este papel.

Unidad de memoria

Es el recurso donde se almacenan las instrucciones de los programas (memoria de sólo lectura), y los datos y resultados que se procesan (memoria de lectura y escritura). Se emplearán memorias basadas en semiconductores.

- Memoria de acceso aleatorio (RAM, *Random Access Memory*), memorias de lectura/escritura para almacenamiento de variables y datos del programa. Son volátiles.
- Memoria de sólo lectura (ROM, *Read Only Memory*), memoria para almacenamiento de programas fijos (aplicaciones, rutinas básicas de sistemas operativos, etc.) y variables constantes. Son no volátiles.

Unidad de entrada/salida

Permite la comunicación del sistema con el mundo exterior. Los dispositivos de E/S se denominan habitualmente periféricos y son los encargados de recibir y entregar información del exterior siendo la unión entre el usuario y el sistema.

Unidad de comunicación a Internet

Es el enlace de comunicación del sistema con Internet, que funcionará como interfaz entre las señales físicas transmitidas sobre la red y el entorno digital del sistema. Además del reloj que se encarga de sincronizar todo el sistema compensando los retardos de los diferentes módulos.

En la figura 1 se muestran los módulos que conforman al sistema.

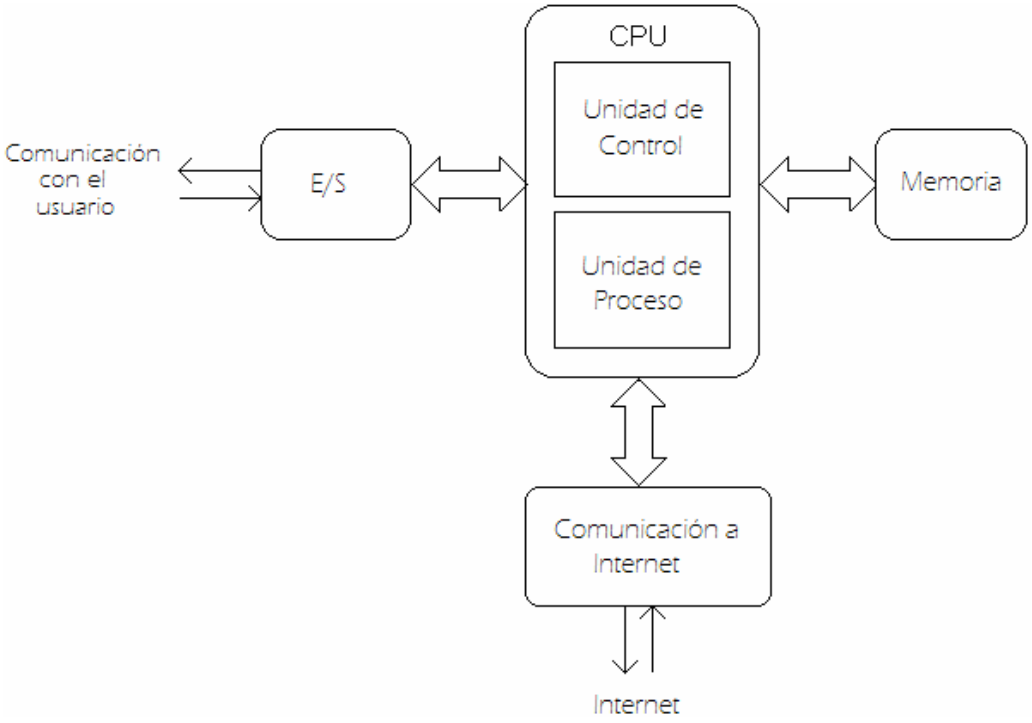


Figura 1. Módulos que integran al sistema.

JUSTIFICACIÓN

Parte esencial de la formación académica de los ingenieros en Electrónica dentro de la Universidad Tecnológica de la Mixteca (UTM), es el manejo de dispositivos programables como los microcontroladores, cuya utilidad se puede constatar con el continuo desarrollo de prácticas, pero el interés de usarlos como parte de una aplicación final y conectado a Internet, le da un valor agregado a este trabajo. Además, el presente trabajo proveerá de una herramienta que apoye la formación académica de futuras generaciones en la implementación de prácticas sobre redes, así como en la creación de diversas aplicaciones a futuro, como control y

monitoreo a distancia que pueda ser de utilidad en la institución o para particulares.

El sistema cuenta con las siguientes características:

- Trabaja en un sistema de red, contando con una dirección que lo identifica.
- Incluye un sistema básico de entrada/salida (BIOS, *Basic Input-Output System*) en la unidad de memoria.
- También dispone de puertos para entrada y salida, lo que permite la generación de aplicaciones, y
- cuenta con un espacio para el almacenamiento de datos del usuario y para las aplicaciones que lo requieran.

El proyecto está basado en el microcontrolador DS80C400 porque cuenta con recursos que lo convierten en un dispositivo versátil y manejable, destacándose sobre otros circuitos que también trabajan en red como los de la familia MSP430 de Texas Instruments, de la cual se puede usar el MSP430F1222, o el AT90CAN64 Automotive de ATMEL Corporation y también el MC9S08Rx16/8 de Freescale Semiconductors. Además se pueden conseguir muestras de evaluación del DS80C400 de forma gratuita.

OBJETIVOS

El objetivo general del trabajo es el diseño y construcción de un sistema mínimo de propósito general que se pueda conectar a una red de computadoras y la implementación de un conjunto de rutinas de soporte para facilitar el desarrollo de aplicaciones.

Para lograrlo se persiguen como objetivos particulares:

- Proporcionar al sistema mínimo la capacidad de ejecutar código realizado por el usuario.
- Incorporar puertos de entrada y salida que faciliten el intercambio de información durante el desarrollo de aplicaciones.
- Proveer capacidad de almacenamiento para datos de usuario.
- Incluir el manejo de periféricos para interactuar con el usuario, tales como teclado, una pantalla alfanumérica, entre otros.
- Manejar algún medio no volátil para el almacenamiento de datos.
- Desarrollar un ejemplo de aplicación sobre el protocolo de control de transmisión/protocolo de Internet (TCP/IP, *Transmission Control Protocol/ Internet Protocol*) para mostrar el funcionamiento del sistema.
- Documentar los resultados.

La figura 2 muestra la manera en que el microcontrolador interactúa con los diversos dispositivos que conforman el sistema mínimo, necesarios para satisfacer los objetivos planteados.

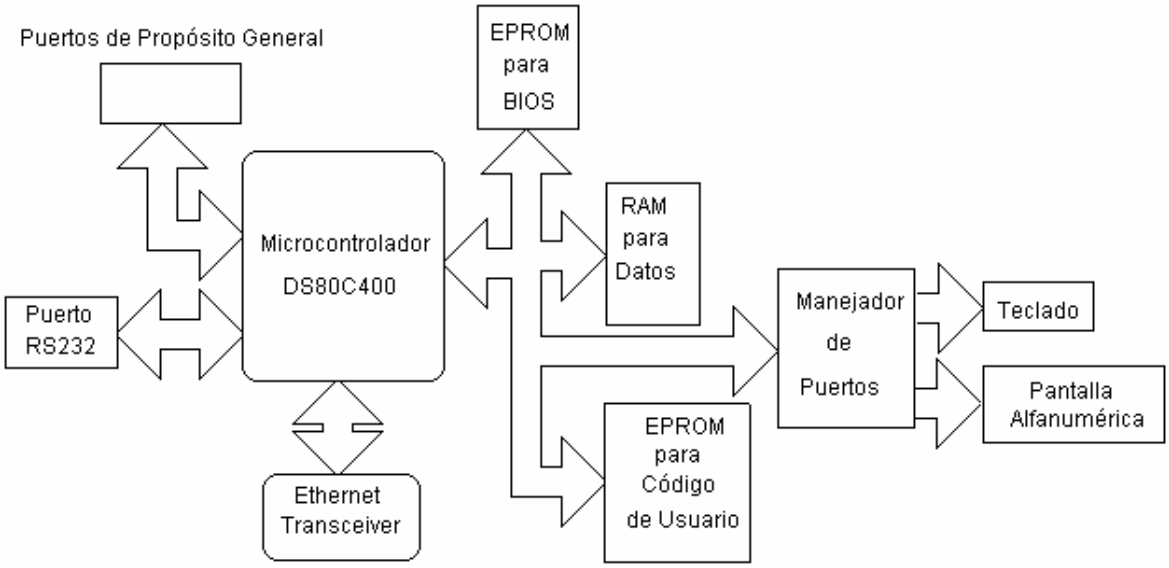


Figura 2. Diagrama a bloques que muestra la manera en que interactúa el microcontrolador DS80C400 con los dispositivos dentro del sistema mínimo.

La mayor parte de la literatura referente al microcontrolador DS80C400 se encuentra en la página Web de MAXIM/Dallas Semiconductor [URL 4]. La recopilación de la información utilizada para poner la interfaz de la red en ejecución se han encontrado en libros, revistas, artículos y en Internet.

Una vez desarrollada la caracterización teórica y la experimentación del comportamiento de las etapas del sistema ante la variación de parámetros, se deberán corroborar las predicciones de forma experimental, creando diseños y poniéndolos en ejecución con hardware.

CONTENIDO

El presente documento se divide en una introducción y cinco capítulos, donde se explican los procedimientos realizados para cumplir con los objetivos propuestos.

Esta sección, Introducción, permite visualizar el planteamiento del problema, la justificación e importancia, y los objetivos planteados.

El capítulo I, Antecedentes y Fundamentos, presenta un marco teórico donde se describen los antecedentes de la investigación y los conceptos necesarios en el desarrollo de este trabajo, desde lo que es una red y su funcionamiento, la explicación de que es un sistema mínimo, componentes y las características del circuito en el que se basó la tesis.

El capítulo II, Desarrollo del Hardware, presenta el marco de desarrollo de la parte física y explica las etapas que componen el diseño del sistema mínimo.

El capítulo III, Plataforma de Diseño, se ocupa de la descripción del software, explicando el desarrollo de los códigos de programa necesarios en el diseño del sistema propuesto.

El capítulo IV, Ejemplo de Aplicación, presenta un marco diagnóstico (pruebas y correcciones) de la situación. Este diagnóstico tiene como finalidad desarrollar las bases para aplicaciones futuras del sistema.

El capítulo V, Conclusiones, muestra los resultados de esta investigación, las recomendaciones y trabajos futuros.

1

ANTECEDENTES Y FUNDAMENTOS

1.1 ANTECEDENTES

Internet es la interconexión de muchas redes. Sus orígenes se remontan a los años 60s, cuando Estados Unidos crea una red exclusivamente militar: la red de la agencia de investigación de proyectos avanzados (ARPANET, *Advanced Research Projects Agency Network*), que legó el trazado de una red inicial de comunicaciones a la que fueron integrándose otras instituciones gubernamentales y redes académicas. A inicios de los 90s, con la introducción de nuevas facilidades de interconexión y herramientas gráficas simples para el uso de la red, su desarrollo fue abismal, creando nuevas redes de libre acceso [URL 5] [URL 6].

Internet hoy día es parte de la vida cotidiana y por sus características ofrece la posibilidad de que la alta tecnología sea puesta al servicio de las mayorías y de los procesos de cambio. Este entorno informático se está haciendo realidad, los aparatos empiezan a incorporar la conexión a Internet creando así una nueva categoría de sistemas: los sistemas con conexión a Internet integrada.

1.1.1 Redes de Computadoras

Una red de computadoras es un conjunto de computadoras conectadas entre sí en una base permanente. Puede significar dos computadoras conectadas sobre un mismo escritorio o miles alrededor del mundo a través de Internet. Los usuarios de una red pueden compartir archivos, impresoras y otros recursos, enviar mensajes electrónicos y ejecutar programas en otras computadoras [URL 7].

1.1.2 Componentes de una red

Una red tiene tres niveles de componentes: software de aplicaciones, software de red y hardware de red [URL 8].

El software de aplicaciones son programas informáticos a través de los cuales se comunican los usuarios de la red y les permiten compartir información (como archivos, gráficos o vídeos) y recursos (como impresoras o unidades de disco).

El software de red consiste en rutinas que establecen protocolos o normas para que las computadoras se comuniquen entre sí. Indican cómo efectuar conexiones lógicas entre las aplicaciones de la red, dirigir el movimiento de paquetes en la red física y minimizar las posibilidades de colisión entre paquetes enviados simultáneamente.

El hardware de red son los componentes físicos que unen las computadoras. Dos componentes importantes son los medios de transmisión que transportan las señales de las computadoras (típicamente cables, fibras ópticas o microondas) y el adaptador de red que permite acceder al medio físico que conecta a las computadoras, recibir paquetes desde el software de red y transmitir instrucciones y/o peticiones a otras computadoras.

1.2 LA RED ETHERNET

Internet y Ethernet suelen confundirse, pero representan diferentes conceptos.

Internet se define como la red de redes o la red que une todas las redes, mientras que Ethernet es la capa física para las redes, y puede usarse para trabajar con diferentes tipos de protocolos. Esta tecnología se usa para intercambiar datos entre computadoras y dispositivos. Y por su capacidad de integración a Internet muchos dispositivos ya lo integran, como en el caso del microcontrolador utilizado para el desarrollo de este trabajo.

1.2.1 Funcionamiento de Ethernet

En Ethernet una computadora sólo presta atención a los paquetes de información que van dirigidos a ella [URL 9]. Concepto que se ilustra con la figura 1.1.

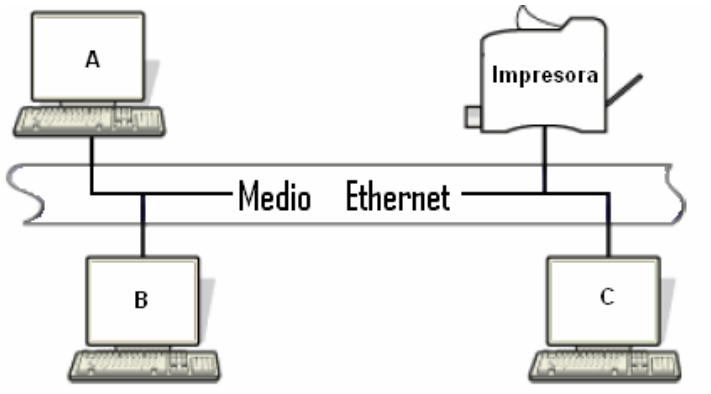


Figura 1.1. Una pequeña red de trabajo Ethernet

Cuando el medio se utiliza por un nodo, ningún otro puede intentar transmitir en él y debe esperar hasta que el medio este libre. Cada nodo conectado tiene una dirección única que lo identifica de forma individual y una dirección de difusión que todos los nodos escuchan. Todos los datos transmitidos en una red Ethernet se encierran en un *frame* que incluye al destino y fuente de la trama.

La red Ethernet funciona como un bus de difusión conocido como 'entrega con el mejor esfuerzo' y un control de acceso distribuido. Es un bus porque todas las estaciones comparten un sólo canal de comunicación y es de difusión porque todos los transceptores reciben todas las transmisiones. Su mecanismo de operación de entrega con el mejor esfuerzo es porque el hardware no provee información al emisor si el paquete se recibió. Por ejemplo, si la máquina destino se apaga, los paquetes enviados se perderán y el emisor no lo sabrá.

El control de acceso se define como distribuido porque no tiene la autoridad central para garantizarlo. El esquema de acceso se conoce como acceso múltiple sensible a portadora con detección de colisión (CSMA/CD, *Carrier Sense Multiple Access with Collision Detect*) ya que varias máquinas pueden acceder a la red simultáneamente y cada una determina si el medio físico está disponible verificando si se presenta una onda portadora.

1.2.2 Direccionamiento de hardware Ethernet

Las direcciones ocupadas por Ethernet no guardan relación con las de Internet. Las direcciones IP de Internet las asigna el usuario, las direcciones Ethernet se fijan "de fábrica".

Cada computadora conectada a una red Ethernet tiene un número único de 48 bits conocido como dirección Ethernet. Para asignar una dirección, los fabricantes adquieren bloques de direcciones y las ordenan en secuencia conforme fabrican el hardware de interfaz Ethernet.

Como este direccionamiento se realiza entre dispositivos físicos también se le llama direccionamiento o dirección física. Conociendo esta dirección física se pueden hacer cambios con facilidad porque los niveles superiores del software de red están diseñados para adaptarse a estos cambios.

1.2.3 Formato de la trama Ethernet

Como en todas las redes de conmutación de paquetes, cada trama Ethernet contiene un campo con la dirección destino, como lo muestra la figura 1.2.

Ethernet

Preámbulo	Dirección Destino	Dirección Fuente	Tipo	Datos	Verificación de Redundancia Cíclica
8 bytes	6 bytes	6 bytes	2 bytes	46–1500 bytes	4 bytes

Figura 1.2. Trama Ethernet.

Además de la información para identificar la fuente y el destino, cada trama transmitida contiene un preámbulo, un campo de tipo, un campo de datos y una comprobación de redundancia cíclica (CRC, *Cyclic Redundancy Check*). El preámbulo ayuda a la sincronización de los nodos de recepción, mientras que la CRC ayuda a la interfaz a detectar los errores de transmisión.

El campo de tipo de trama define el tipo de datos que se están transfiriendo. Desde el punto de vista de Internet este campo es esencial porque significa que las tramas de Ethernet se autoidentifican, permitiendo que múltiples protocolos se manejen juntos en una sola máquina y sea posible entremezclar diferentes protocolos en una sola red física sin interferencia. Cuando una trama llega a una máquina, el sistema operativo utiliza el tipo de trama para determinar qué módulo de software de protocolo se usará para procesarla.

1.3 PROTOCOLOS EN REDES

Los protocolos son reglas y procedimientos técnicos establecidos entre dos dispositivos que dictan su comunicación e interacción.

En una red, tienen que trabajar juntos varios protocolos, para asegurar que los datos se preparen correctamente, se transfieran al destino correspondiente y se reciban de forma apropiada. Sus trabajos se tienen que coordinar para evitar conflictos o que se realicen tareas incompletas. Los resultados de esta coordinación se conocen como trabajo en niveles y un ejemplo se ilustra en la

figura 1.3 que representa al modelo de interconexión de sistemas abiertos (OSI, *Open Systems Interconnections*).

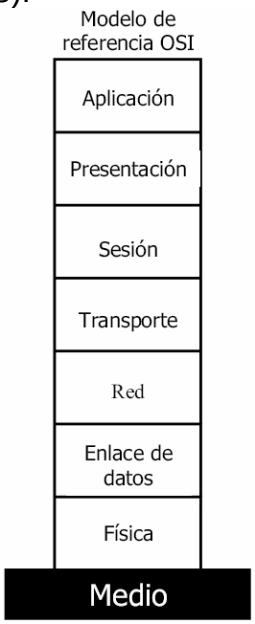


Figura 1.3. Niveles del Modelo OSI, tomada de [8].

Una pila de protocolos es una combinación de protocolos, donde cada nivel de la pila especifica un protocolo diferente para la gestión de una función o de un subsistema del proceso de comunicación. El modelo OSI es de los más usados, se compone de siete niveles y muchos protocolos se adaptan a él. En la tabla 1.1 se describen los diferentes niveles de este modelo.

Tabla 1.1. Descripción de niveles en el modelo OSI.

CAPA	FUNCIÓN	DESCRIPCIÓN
Nivel de aplicación	Inicia o acepta una petición	Transmite los datos sobre la red
Nivel de presentación	Añade información de formato, presentación y cifrado al paquete de datos	Transfiere paquetes al otro extremo de la línea o enlace físico
Nivel de sesión	Añade información del flujo de tráfico para determinar cuándo se envía el paquete	Organiza el trayecto de los mensajes a través de la red
Nivel de transporte	Añade información para el control de errores	Vigila la integridad de la información transmitida
Nivel de red	Se añade información de dirección y secuencia al paquete	Supervisa y coordina los cambios entre procesos
Nivel de enlace de datos	Añade información de comprobación de envío y prepara los datos para que vayan a la conexión física	Conversión de datos y códigos al formato del destinatario
Nivel físico	El paquete se envía como una secuencia de bits	Selección de servicios apropiados a cada aplicación

1.4 Familia de protocolos TCP/IP

Este no es un protocolo, sino un conjunto de ellos que toma su nombre de los dos más conocidos: TCP e IP. Es el conjunto de protocolos más usado por las computadoras conectadas a Internet.

Como en Internet hay conectadas computadoras de clases muy diferentes y con hardware y software incompatibles en muchos casos, además de todos los medios y formas posibles de conexión, una de las grandes ventajas de la familia TCP/IP consiste en hacer que la comunicación entre todos sea posible.

La familia de protocolos TCP/IP no se corresponde exactamente con el modelo OSI como se muestra en la figura 1.4.

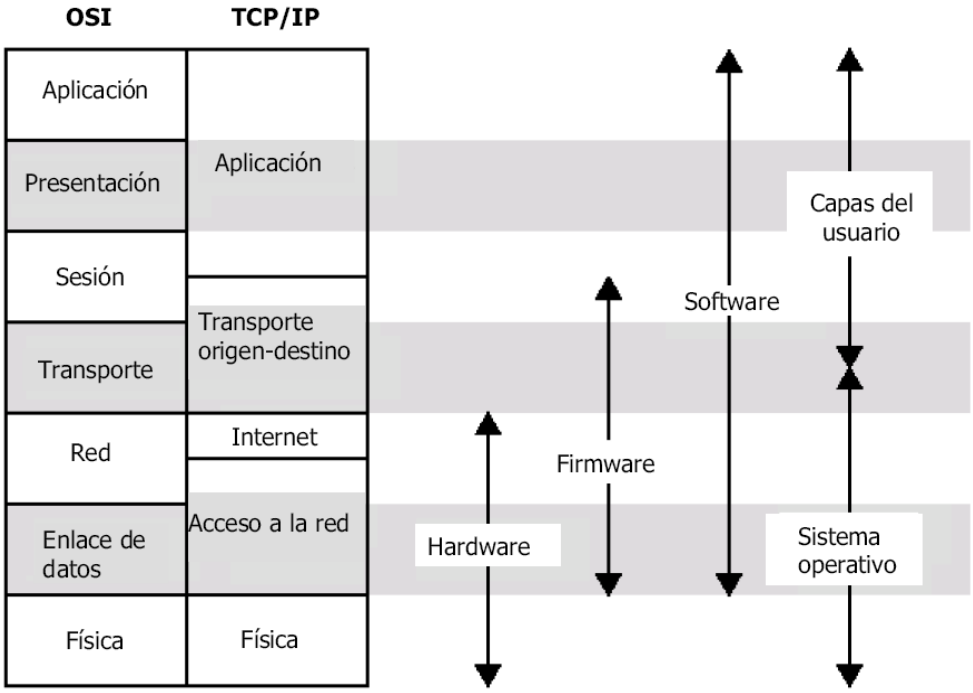


Figura 1.4. Una comparación entre las arquitecturas de protocolo TCP/IP y OSI, tomada de [8].

En vez de tener siete niveles, utiliza cinco, normalmente conocidos como conjunto de protocolos de Internet y que son:

- ***Aplicación:*** Correspondiente con los niveles OSI de aplicación, presentación y sesión. Incluye protocolos destinados a proporcionar servicios como el del protocolo sencillo de transferencia de correo electrónico (SMTP, *Simple Mail Transfer Protocol*), del protocolo de transferencia de archivos (FTP, *File Transfer Protocol*), de conexión

remota (TELNET, *TCP/IP Terminal Emulation Protocol*) y el protocolo de intercambio de hipertexto (HTTP, *Hypertext Transfer Protocol*).

- ***Transporte:*** Los protocolos de este nivel, TCP y el protocolo datagrama de usuario (UDP, *User Datagram Protocol*), se encargan de manejar los datos y proporcionar la fiabilidad necesaria en la transferencia de los mismos.
- ***Internet***: Corresponde al nivel de red del modelo OSI. Incluye al protocolo IP que se encarga de enviar los paquetes de información a sus destinos correspondientes. Lo usan los protocolos del nivel de transporte.
- ***Acceso a la red:*** Es responsable del acceso y encaminamiento de los datos a través de la red.
- ***Física***: Define las características del medio de transmisión.

1.4.1 Funcionamiento de la familia de protocolos TCP/IP

Para transmitir información con TCP/IP, ésta se divide en unidades que reciben el nombre de "datagramas" y son conjuntos de datos que se envían como mensajes o paquetes independientes. TCP/IP transfiere los paquetes comenzando con una cabecera con información de control, como la dirección de destino, seguido de los datos.

IP es un protocolo de la capa de red que permite a las aplicaciones ejecutarse transparentemente sobre redes interconectadas. No es necesario conocer que hardware se usa, por tanto la aplicación corre en una red de área local.

TCP es un protocolo de la capa de transporte, asegura que los datos sean entregados, que lo que se recibe sea lo que se pretendía enviar y que los paquetes se reciban en el orden en que fueron enviados. Se termina una conexión si ocurre un error que haga imposible la transmisión fiable.

1.5 SISTEMA MÍNIMO

Un sistema mínimo basado en microprocesador o microcontrolador es una microcomputadora de propósito específico, equipada con el mínimo de componentes (memoria RAM, ROM, puertos, sensores actuadores, etc.) para realizar sus funciones.

Los propósitos para los que puede diseñarse pueden caer en una infinidad de campos como: instrumentación, control, monitoreo, señalización, secuenciamiento, autorización, comunicaciones, procesamiento de señales, etc. En la figura 1.5 se muestran los bloques básicos de un sistema mínimo.

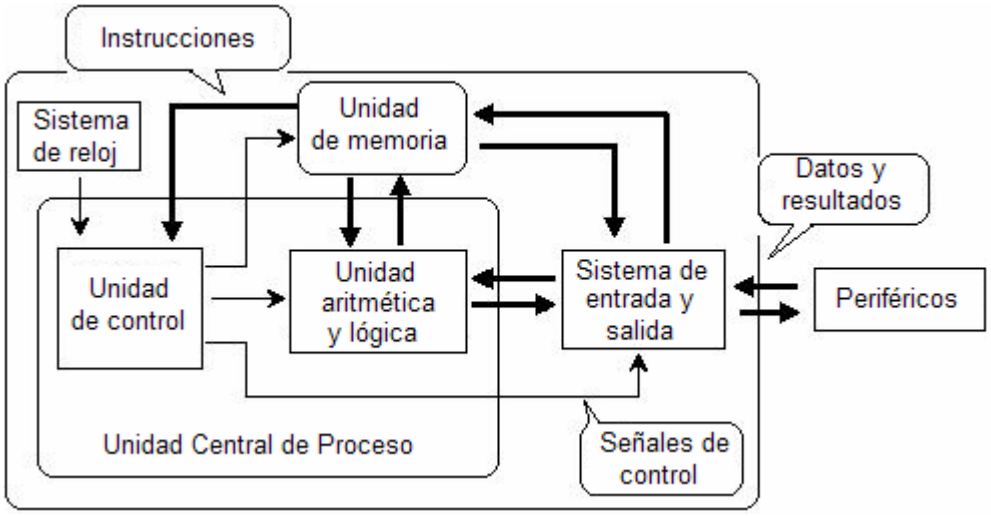


Figura 1.5. Diagrama a bloques básico de un sistema mínimo.

Por lo general, para el desarrollo de sistemas mínimos, se emplean microcontroladores que son dispositivos que contienen: una CPU, basado principalmente en un microprocesador de 4, 8 ó 16 bits, puertos paralelos de entrada y salida, puerto serie, temporizadores, contadores, memorias, y en algunos casos hasta convertidores analógicos digitales, todo dentro de un solo chip. Y se emplean en una multitud de sistemas presentes en la vida diaria, generalmente empotrados en sistemas donde controlan características o acciones, como lavaplatos, aparatos de Tv o coches, entre otros.

Una aplicación típica podría emplear varios microcontroladores para controlar pequeñas partes del sistema. Estos podrían comunicarse entre sí y con un procesador central, probablemente más potente, compartir la información y coordinar sus acciones, como ocurre en cualquier computadora.

1.5.1 El microcontrolador de red DS80C400

Las capacidades de trabajo en red del DS80C400 lo hacen una elección natural para diseños donde se habilita Ethernet. Incluye una pila TCP/IP en la memoria ROM del procesador.

A continuación, se describen algunas de las principales características del microcontrolador [1].

1.5.1.1 Características Generales

El DS80C400 es un microcontrolador de red de 100 terminales, como se ve en la figura 1.6, que ofrece la más alta integración disponible a partir de una arquitectura 8051. Para habilitar el acceso a red se utiliza la pila TCP/IP v4/6

provista en una memoria ROM interna. El controlador de acceso a medios (MAC, *Media Access Controller*) de Ethernet permite al DS80C400 el acceso y comunicación sobre Internet soportando conexiones TCP simultáneas y transferencias hasta de 5 Mbps. Su máxima frecuencia de reloj es de 75 MHz que resulta en un ciclo de instrucción de 54 ns. Incluye hardware acelerador de operaciones aritméticas con el que incrementa la velocidad de las operaciones de multiplicación y división.

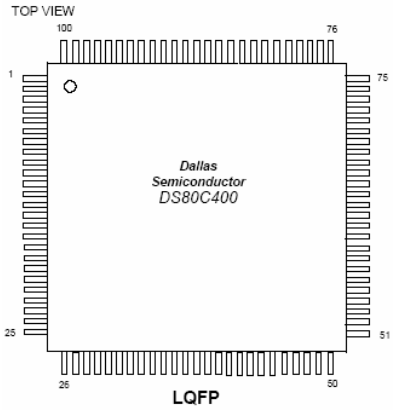


Figura 1.6. Aspecto externo del microcontrolador DS80C400.

Con la conexión a una red y capacidades de entrada/salida (I/O, *Input/Output*), el DS80C400 es adecuado para servir como una unidad central de proceso en la red. Puede controlar activamente redes de menor jerarquía con hardware dedicado como un controlador de área de red 2.0B (CAN, *Controller Area Network*), un controlador de red 1-Wire, tres puertos seriales full-duplex, y ocho puertos de 8-bits (hasta 64 terminales digitales de I/O).

La conectividad inmediata y el soporte de red se tienen a través de una ROM empotrada de 64 KBytes, que contiene la asistencia para realizar un enlace en la red sobre una conexión de Ethernet usando el protocolo de configuración dinámica de host (DHCP, *Dynamic Host Configuration Protocol*) conjuntamente con el protocolo trivial de transferencia de archivos (TFTP, *Trivial File Transfer Protocol*).

1.5.1.2 Comparación del rendimiento con un 8051

El alto rendimiento del DS80C400 no viene solo de aumentar la frecuencia de reloj sino de un diseño más eficiente. Su núcleo actualizado quita los ciclos simulados de memoria presentes en un ciclo de máquina estándar correspondiente a 12 ciclos de reloj en 8051. En este circuito un ciclo de máquina requiere sólo 4 ciclos. Así, las instrucciones se ejecutan de forma más rápida. La mejora de programas individuales depende de la mezcla real de las instrucciones usadas.

1.5.1.3 El regulador de Ethernet

El DS80C400 incorpora un controlador de Ethernet 10/100Mbps, que soporta los requisitos del protocolo de funcionamiento para un dispositivo en la capa física (PHY, *Physical Layer*) y cumple con el estándar Ethernet/IEEE 802.3. Provisto de recepción, transmisión y los mecanismos de control de flujo a través de una interfaz de medio independiente (MII, *Media Independent Interface*) con un bus serial para configurar los dispositivos externos de la capa física.

Al operar en modo half-duplex, el DS80C400 comparte el medio físico de Ethernet con otras estaciones en la red. El DS80C400 hace uso del medio IEEE 802.3 con CSMA/CD. El MAC antes de transmitir espera hasta que el enlace físico esté desocupado.

En operación en modo full-duplex, el medio físico conecta el DS80C400 directamente con otra estación, permitiendo transmisión y recepción simultánea entre los dos sin el riesgo de la colisión.

1.5.1.4 El control del reloj y el manejo de la energía

El DS80C400 incluye características que permiten la flexibilidad para seleccionar diversas fuentes de reloj para el sistema y frecuencias de funcionamiento. Para que pueda trabajar con cristales de bajo costo y permita una operación rápida incluye un multiplicador en su circuito de reloj.

Además de los modos de operación estándar, el DS80C400 proporciona un modo para el control de energía (PMM, *Power-Management Mode*) que permite al microcontrolador continuar la ejecución de la instrucción a una velocidad muy baja para reducir perceptiblemente el consumo de energía (debajo incluso del modo en reposo).

2

DESARROLLO DEL HARDWARE

2.1 INTRODUCCIÓN

Esta sección presenta una descripción del hardware de este proyecto y examina los componentes principales. Los detalles de programación de algunos de estos circuitos se conocerán en la sección siguiente.

El diseño y construcción del sistema se desarrolló eligiendo los componentes que se adecuen al funcionamiento que se necesita. La primera etapa de la implementación consistió en la investigación de los subsistemas desarrollados, su compatibilidad y los requerimientos para su interconexión.

El proyecto intenta conectarse a Internet con un sistema que cuente con: un segmento en el que funcione un sistema operativo y una pila de protocolos TCP/IP, capacidad de interconexión con una capa física de red. La posibilidad de adición extra de elementos y su respectivo intercambio de información. Además de suficiente espacio para los datos necesarios en los procesos y los generados durante la ejecución.

2.2 EL DISEÑO

El diagrama a bloques de la figura 2.1 es una abstracción del hardware requerido. Se indican las unidades funcionales de las que se conforma y el microcontrolador como unidad principal.

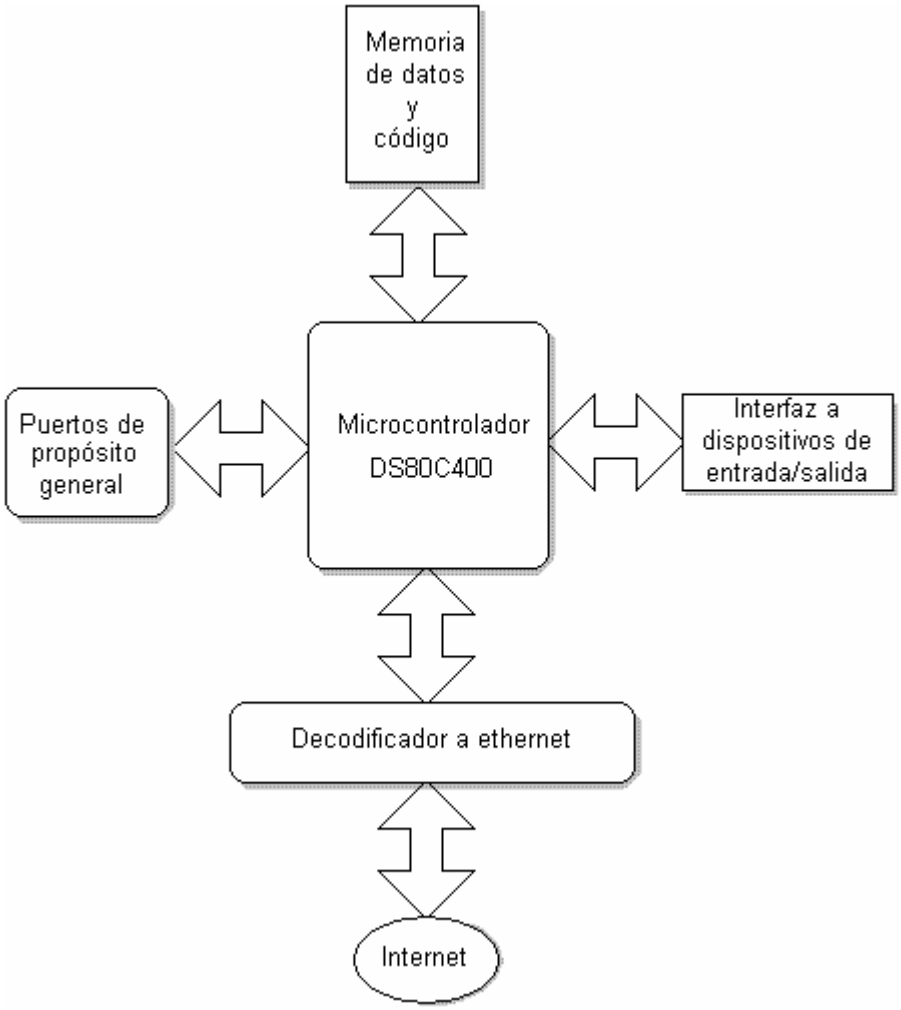


Figura 2.1. Diagrama a bloques del sistema mínimo.

2.2.1 Descripción del sistema mínimo

El diseño es una plataforma con medios simples y flexibles para lograr que una amplia variedad de dispositivos se puedan conectar directamente con la red.

El microcontrolador es el corazón del diseño porque ejecuta directamente el código nativo del entorno de ejecución (*runtime environment*), proporciona el control y la comunicación entre dispositivos y las capacidades para el establecimiento de una red.

Para el manejo de datos y programa se usan dos tipos de memorias. Una que almacena el entorno de ejecución y otra para el área de datos del sistema.

El dispositivo decodificador de Ethernet auxilia al microcontrolador para enviar y recibir mensajes a través de la red.

Los elementos periféricos, con excepción de la memoria, también se pueden interconectar directamente a los buses de dirección y de datos del microcontrolador.

Los puertos quedan disponibles para el manejo de periféricos externos como un teclado, botones, visualizadores, etcétera.

En las siguientes secciones se detallan los bloques que integran al sistema, los dispositivos elegidos y sus características para una operación coordinada con el microcontrolador DS80C400.

2.3 EL MICROCONTROLADOR

Este proyecto se basa en el microcontrolador DS80C400 de MAXIM/Dallas Semiconductor que trabaja en la línea de procesadores 8051 y proporciona las bibliotecas de apilado de TCP/IP provistas en la ROM del procesador proporcionando una conectividad rápida y simple a Ethernet/Internet en un solo chip (hardware y software).

2.3.1 DS80C400

Este microcontrolador de dimensiones reducidas, con ayuda incorporada para comunicación de entrada/salida, posee varios puertos con terminales de propósito general para realizar tareas simples de control, además de la posibilidad de uso de Ethernet. Su aspecto físico se muestra en la figura 2.2.

Figura 2.2. Microcontrolador DS80C400 de MAXIM/ Dallas Semiconductor.

Sus principales características son:

- Arquitectura de alto rendimiento.
  - Ciclo de instrucción de 54 ns.
  - Frecuencia de operación de 75 Mhz.
  - Espacio de direccionamiento externo de 16 MBytes.

  - Cuatro apuntadores de datos con auto incremento/decremento.
  - Acelerador matemático de 16/32 Bits.

- Trabajo de red multitarea e I/O.
  - Controlador de Acceso a Medios 10/100 Mbps.
  - CAN 2.0 B.
  - Controlador de red 1-Wire.
  - Tres puertos seriales.
  - Hasta 8 puertos bidireccionales de 8 bits (64 terminales I/O).

- ROM Firmware.
  - Soporte de cargador de red sobre Ethernet usando DHCP y TFTP.
  - Pila de red TCP/IP accesible.
  - Soporta IPv4 e IPv6.
  - Implementa UDP,TCP,DHCP,ICMP (*Internet Control Message Protocol*) e IGMP (*nternet Group Management Protocol*).

- 10/100 Ethernet MAC.
  - IEEE 802.3 flexible para las interfaces MII (10/100 Mbps) y ENDEC (*Encode/Decode*) (10Mbps) que permite selección de operación en baja potencia de la PHY.
  - 8 KBytes de memoria para Rx/Tx de paquete de datos con unidad de control de buffer.
  - Operación half o full-duplex con control de flujo.

- Sistema lógico primario integrado.
  - 16 fuentes de interrupción total con 6 externas.
  - Cuatro temporizadores/contadores de 16 bits.
  - *Watchdog Timer* programable.
  - Multiplicador de reloj 2x/4x reduce la interferencia electromagnética (EMI, *Electromagnetic Interference*).

2.3.2 Sistema de control de Reloj

El DS80C400 cuenta con un circuito especial de control de reloj que se refleja en la rápida ejecución de las instrucciones y aporta máxima flexibilidad de temporización, disponibilidad y economía en la selección de un cristal. La figura 2.3 muestra el diagrama del sistema del control.

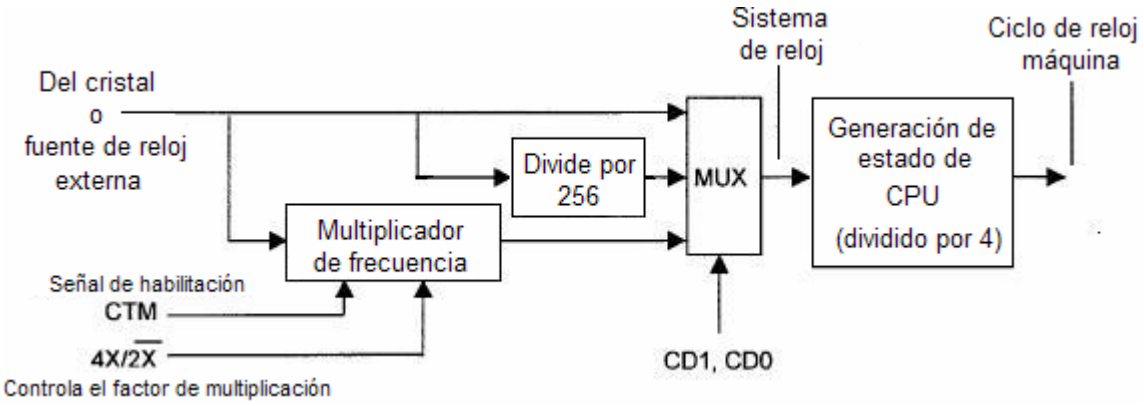


Figura 2.3. Diagrama a bloques del sistema de control del reloj.

El circuito genera dos señales de reloj que usa el microcontrolador. El *reloj de sistema interno* que proporciona el tiempo base para los temporizadores y los periféricos internos. Opera con un divisor por 4 para generar el *reloj de ciclo máquina* que proporciona el tiempo base para las operaciones de la CPU. Todas las instrucciones se ejecutan entre uno y cinco ciclos de máquina.

2.3.3 Software del DS80C400

El microcontrolador tiene integrada una plataforma de software desarrollada por MAXIM/Dallas Semiconductor para interactuar con otros módulos del sistema, además de realizar las funciones con los protocolos de comunicación, interfaces físicas y funciones de propósito general. Esta plataforma se describe en el capítulo 3.

2.4 LA MEMORIA DE DATOS Y PROGRAMA

Para elegir la memoria se debe considerar la capacidad de direccionamiento del microcontrolador y el tiempo de acceso.

2.4.1 El microcontrolador y la memoria

El DS80C400 incorpora cuatro áreas de memoria interna:

- 256 Bytes de RAM de acceso directo.
- 9 KBytes de memoria estática de acceso aleatorio (SRAM, *Static Random Access Memory*) configurable.
- 256 Bytes de RAM reservada para mensajes CAN.
- 64 KBytes de ROM para firmware.

Puede direccionar hasta 16 MBytes de memoria externa de la siguiente manera: incluye un bus multiplexado de 16 bits para datos y direcciones, similar a un 8051, con esto se pueden direccionar hasta 64 Kbytes. Mas sin embargo, 6 líneas de diferentes puertos pueden configurarse para incrementar el bus de direcciones, alcanzando un direccionamiento de 2 MBytes. Además, otras 8 líneas de los puertos se pueden configurar como habilitadores para referenciar a 8 espacios diferentes de 2 MBytes cada uno. Este espacio externo puede combinarse para código o datos, teniendo un máximo de 4 MBytes para los datos.

2.4.1.1 El mapa de memoria

El mapa de memoria especifica dónde se ubican los dispositivos periféricos y la memoria en el espacio de direcciones del microcontrolador. Y consiste en los segmentos que se pueden ver en la figura 2.4.

El espacio de dirección que ocupe el periférico puede ser menor a los indicados en la figura, porque depende de los requerimientos de cada aplicación. El mínimo de memoria para los segmentos de programa y datos es de 512 kilobytes para cada uno.

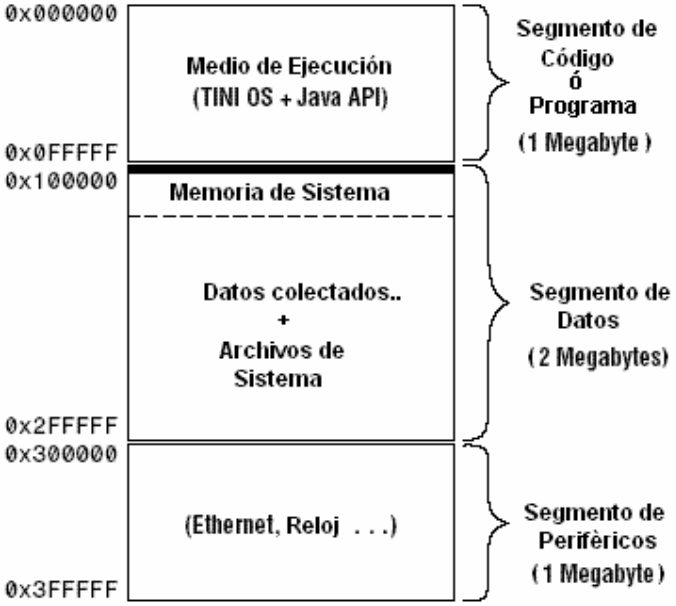


Figura 2.4. Mapa de memoria.

Los segmentos de programa y de datos se ocupan por los circuitos de memoria, y el segmento de periféricos por otros componentes de hardware como el regulador de Ethernet y el reloj en tiempo real, que ocupan los siguientes rangos de dirección:

- Controlador de Ethernet - [0x300000 - 0x307FFF].
- Reloj en tiempo real - 0x310000.

Por ello, en el diseño del sistema se evitan estos rangos para interconectar cualquier otro dispositivo. El resto del rango de direcciones está disponible.

El DS80C400 contiene 9 KBytes de SRAM interna que físicamente se divide en un bloque de 1 KBytes y otro de 8 KBytes. El primero se puede utilizar para el manejo de la pila TCP/IP o para el uso de variables. El bloque de 8 KBytes lo usa el MAC de Ethernet como estructura de memoria intermediario para los datos entrantes o salientes del paquete de datos y se puede, al mismo tiempo, acceder por el DS80C400 como memoria de datos.

También tiene una memoria ROM interna diseñada específicamente para recibir el ambiente Tiny InterNet Interface (TINI®), una plataforma desarrollada para los microcontroladores con conexión a red de MAXIM/Dallas Semiconductor. El soporte lógico inalterable de la ROM se compone de tres unidades en ejecución importantes: una pila completa TCP/IPv4/6, un planificador de tareas con derecho preferente y la funcionalidad automática de cargado de la red (*NetBoot*).

2.4.2 La memoria externa de datos

En esta sección se describe a la memoria no volátil usada para el manejo de los datos.

2.4.2.1 SRAM No Volátil bq4017/bq4017Y

Como características generales tiene:

- Retención de datos en ausencia de la energía.
- Protección automática de escritura durante los ciclos de encendido/apagado.
- Operación convencional de SRAM y no requiere de circuitos externos; número ilimitado de ciclos de escritura.
- Retención mínima de cinco años de datos en ausencia de energía.
- Batería interna aislada hasta que se requiera.

El diagrama a bloques del circuito de memoria bq4017 se muestra en la figura 2.5.

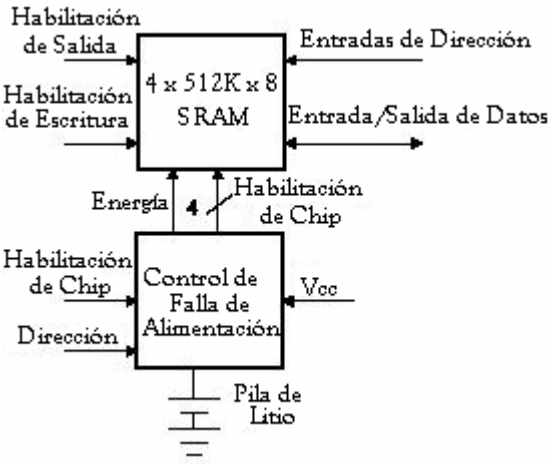


Figura 2.5. Diagrama a bloques SRAM bq4017.

2.4.2.2 Direccionamiento de memoria de datos externa

El DS80C400 permite hasta 4 MBytes de acceso de memoria de datos habilitando chips periféricos. Los registros de control del puerto 5 (P5CNT; A2h) y puerto 6 (P6CNT; B2h) señalan el número de chips permitidos y la cantidad máxima de memoria de datos direccionable por chip.

2.4.3 Memoria externa de programa

En esta sección se presentan los requisitos de la memoria flash elegida para el sistema.

2.4.3.1 Direccionamiento de memoria de programa externa

El microcontrolador DS80C400 dispone de registros de función especial para configurar ciertas terminales de puerto como líneas de dirección y habilitación del chip. Los registros de control de los puertos 4 (P4CNT; 92h) y 6 (P6CNT; B2h) controlan el número de chips permitidos y la cantidad máxima de memoria de programa que se puede alcanzar por chip.

2.4.3.2 Los requisitos

Para trabajar con el DS80C400, un chip de memoria flash debe ser eléctricamente compatible y cumplir con los requisitos de tiempo definidos (es decir, que sea lo bastante rápido). El DS80C400 tiene una ROM incorporada que soporta la carga de una SRAM y flash a través del puerto serial y de la red. La velocidad máxima del puerto serial del cargador se limita por la velocidad del cristal, es decir, 115200 bps requiere por lo menos 20MHz. La tabla 2.1 presenta una selección de memorias flash compatible con el microcontrolador.

Tabla 2.1. Selección de las memorias de destello 3V (x 8)

Vendedor (ID)	Dispositivo (ID)	Tamaño (Byte)	Soporte de ROM
Atmel	AT49BV001A AT49LV008A AT49LL080	128K 1M 1M	Prog No No
Intel	28F320J3	4M	No
Macronix	MX29LV081 MX29LV033A	1M 4M	Completo Completo
Micron	MT28F004B3 MT28F008B3	512K 1M	No No
Sharp	LH28F016SC	2M	No
Spansion (AMD/Fujitsu)	AM29LV004B AM29LV081B AM29LV160B AM29LV017D	512K 1M 2M 2M	Prog Completo Prog Completo
ST Microelectronics	M29W004B M29W800D	512K 1M	Prog No

Esto demuestra la flexibilidad del microcontrolador para adaptarse a dispositivos externos.

2.4.3.3 Memoria Flash AM29LV017D

El AM29LV017D es una memoria flash. Para realizar las operaciones de lectura, programación y borrado se puede hacer con un solo voltaje de alimentación (3 volts) y se puede programar con programadores estándares de EPROM (*Erasable Programmable Read-Only Memory*).

Ofrece tiempos de acceso de 70, 90, y 120 ns, permitiendo alta velocidad sin estados de espera. Para eliminar el entorpecimiento del bus, el dispositivo tiene separados los controles de habilitación del chip, permiso de escritura y salida. Además, posee una arquitectura de sector uniforme, soportando borrado completo y no hay necesidad de un programa adicional para su programación porque es compatible con el cargador del microcontrolador.

2.5 EL DECODIFICADOR A ETHERNET

La comunicación de Ethernet y de Internet se construye en las capas donde cada una tiene una tarea específica, construyendo el hardware de la misma manera.

El trabajo en red se auxilia con un controlador externo de Ethernet para mandar y recibir mensajes de red. La PHY constituye el enlace entre las señales físicas transmitidas en la red de trabajo y el ambiente digital del microcontrolador.

El DS80C400 tiene integrado un MAC para Ethernet con una MII estándar, rutinas para transmitir/recibir paquetes Ethernet que se pueden encontrar en línea [URL 10] y estar comunicado con un circuito externo para el manejo de la PHY. La interfaz con la PHY se implementa con el circuito LXT972A de Intel.

2.5.1 Interfaz de Medios Independiente (MII)

El DS80C400 contiene una interfaz MII–PHY con dos bloques básicos como se muestra en el diagrama de la figura 2.6, en el cual también se indican las señales asociadas con la MII.

El bloque de entrada/salida proporciona una transmisión y recepción de datos independiente y las señales de entrada a la PHY para establecer el estado de la red. Y el bloque de control implementa una comunicación de bus serial de 2-líneas para facilitar el acceso a registros de la PHY.

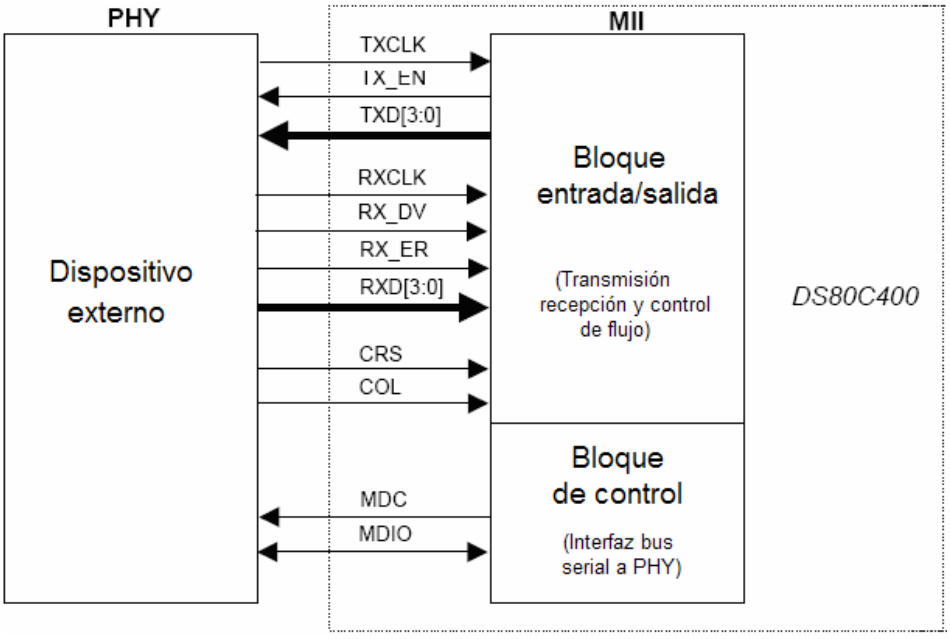


Figura 2.6. Diagrama a bloques de la MII.

El propósito del bloque de entrada/salida es transmitir y recibir las tramas Ethernet entre el MAC DS80C400 y un dispositivo externo para el manejo de la PHY, también la supervisión de las señales del estado de la red proporcionadas por la PHY.

2.5.2 Intel® LXT972A Dual-Speed Fast Ethernet Transceiver

EL LXT972A es un transmisor-receptor PHY Fast Ethernet que cumple con las normas del IEEE. Soporta redes de 10 Mbps y 100 Mbps, y su condición de funcionamiento se puede fijar usando auto negociación, la detección paralela o control manual.

Las características más importantes de operación de este dispositivo son:

- Bajo consumo de potencia (300 mW típico).
- Operación en 10BASE-T y 100BASE-TX con una conexión RJ45.
- Configurable vía puerto serial o terminales de control de hardware.
- Necesita una referencia de reloj para generar las señales de transmisión, recepción y recuperación, conectando un cristal en las terminales de oscilador o un reloj externo.

Tiene otras características que determinan las condiciones de operación del circuito [2], pero que dependen del tipo de transmisión (10/100 Mbps).

2.5.2.1 Funcionamiento

El circuito integrado LXT972A maneja la recuperación, codificación y el descifrado de los datos así como el reloj. Se comunica con la capa del MAC a través de la MII, como se muestra en la figura 2.7. Consiste de una interfaz de datos y un administrador de interfaz pasando los datos hacia la red a través de un transformador que sirve de protección y acoplamiento, para por último usar un conector. La MII de datos traspasa los datos entre el LXT972A y el MAC con buses paralelos separados para recepción y transmisión. La velocidad de operación (10/100 Mbps) es automática, lo que significa que el diseño no necesita considerar si el enlace físico consiste en 10 Mbps o par trenzado de 100 Mbps o una red de fibra óptica.

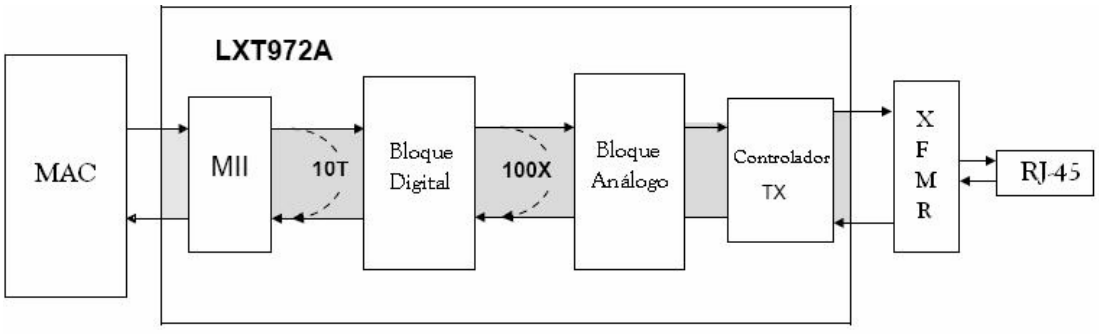


Figura 2.7. MII típica.

Para la transmisión y recepción de datos, la MII trabaja como un bus de datos paralelo básico, con una señal de reloj para sincronizar los datos.

Para mandar un paquete de datos se debe poner en el buffer de envío de Ethernet, si hay un paquete previo en la misma dirección, se espera a la transmisión completa antes de modificar la memoria de buffer. La operación es la misma al recibir un paquete con el buffer de recepción del DS80C400.

El MAC atiende la recepción de datos de la PHY juntando la trama completa de Ethernet y comprueba el campo adecuado para asegurar que los datos no posean errores. Al completar la recepción de la trama debe transmitírselo a la capa IP, donde se ponen en ejecución los protocolos IP y el protocolo de resolución de direcciones (ARP, *Address Resolution Protocol*).

2.6 INTERFAZ A DISPOSITIVOS DE ENTRADA/SALIDA

En esta sección se describe la capacidad que tiene el microcontrolador para comunicarse con otros dispositivos por los protocolos de comunicación que tiene integrados.

Los dispositivos periféricos se pueden interconectar a los buses de dirección y de datos del microcontrolador, que tiene integrado el soporte para los siguientes protocolos seriales de bajo nivel (figura 2.8):

- *Comunicación serial.* Basada en el estándar RS232-C.
- *1-Wire net.* Es una red de sensores, actuadores y elementos pequeños que comparten el mismo conductor para la comunicación y la energía.
- *CAN 2.0.* Un robusto protocolo de comunicación de alto rendimiento para comunicaciones seriales.
- *TTL I/O. (Transistor-Transistor Logic, Lógica Transistor-Transistor)* Terminales bidireccionales del microcontrolador de propósito general.

Figura 2.8. Entradas/Salidas integradas.

Es posible utilizar las capacidades integradas de entrada/salida del microcontrolador para el manejo de dispositivos en vez de mapearlos en memoria.

2.6.1 El Bus 1-Wire

El microcontrolador incluye soporte de comunicación con dispositivos externos 1-Wire a través de un bus que provee control completo y coordina las actividades de transmisión/recepción (Tx/Rx) con mínima supervisión.

Una red 1-Wire es una colección de uno o más dispositivos unidireccionales que comparten un solo conductor para la comunicación y la energía. Características que permiten construir una red de trabajo con dispositivos unidireccionales y una disminución en los costos de los sistemas con una importante simplificación de los diseños por el manejo de un protocolo con una interfaz que proporciona control, señal y alimentación sobre un sistema de cableado simplificado como se muestra en el ejemplo de la figura 2.9.

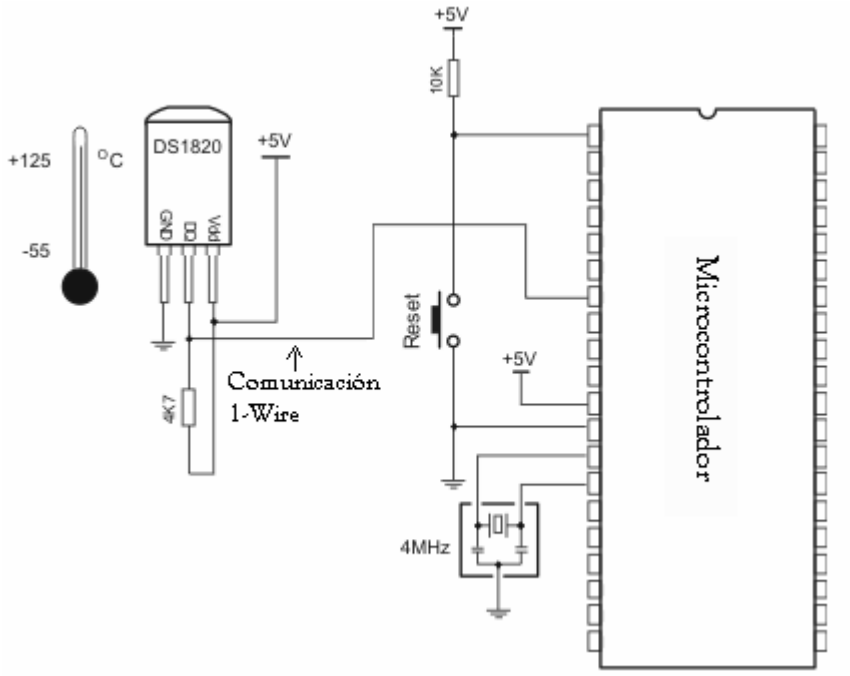


Figura 2.9. Interfaz típica de un dispositivo 1-Wire.

2.6.1.1 El bus principal 1-Wire del DS80C400

El DS80C400 tiene dos formas de acceso a una red 1-Wire. Con un adaptador interno, nombrado así porque su interfaz física es una terminal de puerto del microcontrolador. El número de dispositivos unidos y la longitud de línea conducida por el adaptador interno se limitan por las características eléctricas de la terminal usada.

La otra forma es con un convertidor externo 1-Wire unido al puerto serial auxiliar del microcontrolador. Este circuito es un adaptador equipado y capaz de controlar las redes 1-Wire para cubrir un área grande y potencialmente se podrían conectar muchos dispositivos, por lo que es preferible.

El sistema aprovecha el adaptador incluido en el microcontrolador para la identificación de dispositivos 1-Wire conectados en el bus maestro 1-Wire.

2.6.1.2 El DS2502

El dispositivo DS2502 es un dispositivo 1-Wire que mantiene la identificación MAC y almacena la información relevante sobre el producto al que se asocia y dicha información se puede leer con la interfaz mínima, por ejemplo una sola terminal del microcontrolador.

El DS2502 tiene una identificación de familia y contiene 1 KBytes de memoria EPROM programable. En este trabajo se encargará de proporcionar la dirección MAC de Ethernet al sistema, dirección que se le ha asignado durante su fabricación y es la que permitirá identificar al dispositivo asociado dentro de una red.

2.7 BUSES DE EXPANSIÓN

Para cualquier sistema que será la base para el desarrollo de aplicaciones es importante que cuente con los mecanismos de comunicación con otros dispositivos. Esto le proporciona flexibilidad al sistema para su crecimiento y comunicación con el mundo exterior.

Se entiende por dispositivos de entrada/salida a aquellos que se puedan interconectar al sistema y en cualquier momento separar, cambiar o adicionar. Muchas aplicaciones usan pines de propósito general para monitorear y controlar hardware externo.

La manera en que se comunican es usando el bus paralelo del microcontrolador. Y en el sistema es de importancia que tenga comunicación con el mundo exterior para el desarrollo de muchas aplicaciones.

2.7.1 Bus paralelo de entradas/salidas

Se usa el término entradas/salidas paralelas para referirse a la comunicación con los dispositivos interconectados a los buses de dirección y de datos del microcontrolador. Bajo este contexto, en el sistema se realizó la comunicación con el reloj de tiempo real y el controlador de Ethernet sobre el bus del microcontrolador.

El bus paralelo de entrada/salida es muy rápido y flexible, se utiliza como mínimo, para interconectar con los circuitos de memoria externas de programa y de datos. Por ejemplo, el diagrama a bloques de la figura 2.10 presenta una configuración genérica para interconectar dispositivos externos al bus.

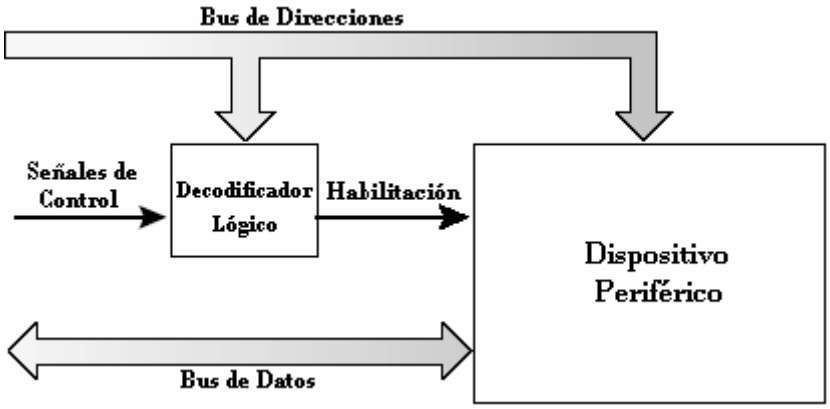


Figura 2.10. Interfaz al bus del controlador.

Como sus nombres sugieren, el bus de direcciones especifica la dirección destino para las operaciones de lectura o escritura, mientras que el de datos los transfiere a/desde el dispositivo. La combinación de ciertas señales de control y líneas de dirección se pueden utilizar conjuntamente con decodificación lógica para actuar como señal de habilitación para el periférico. Algunos dispositivos se pueden interconectar directamente al bus sin usar ninguna decodificación lógica.

2.8 INTEGRACIÓN DE LOS MÓDULOS DEL SISTEMA

Para elaborar el prototipo del sistema es importante tener conocimiento de cómo se deben integrar los módulos que se han descrito. En esta sección se presentan los diagramas a bloques de los módulos y sus señales de comunicación.

La figura 2.11 muestra al módulo que corresponde al microcontrolador, el núcleo del sistema, al cual se conectan cada uno de los bloques descritos.

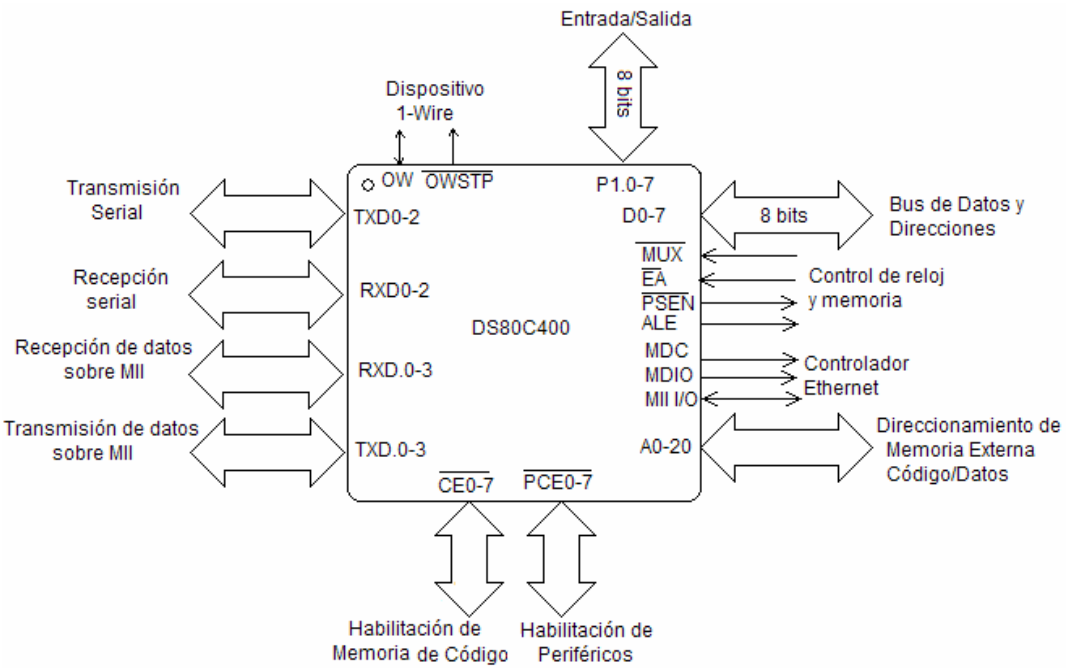


Figura 2.11. Diagrama a bloques del microcontrolador DS80C400.

En la figura 2.11 se muestran las señales con etiquetas que describen su función, lo que permite identificar cómo se realiza la comunicación con las otras secciones del sistema.

Para el intercambio de información con las memorias de datos y programa se usa el bus de datos y direcciones. Además de terminales específicas de control para la habilitación y sincronización, tal como se muestra en la figura 2.12.

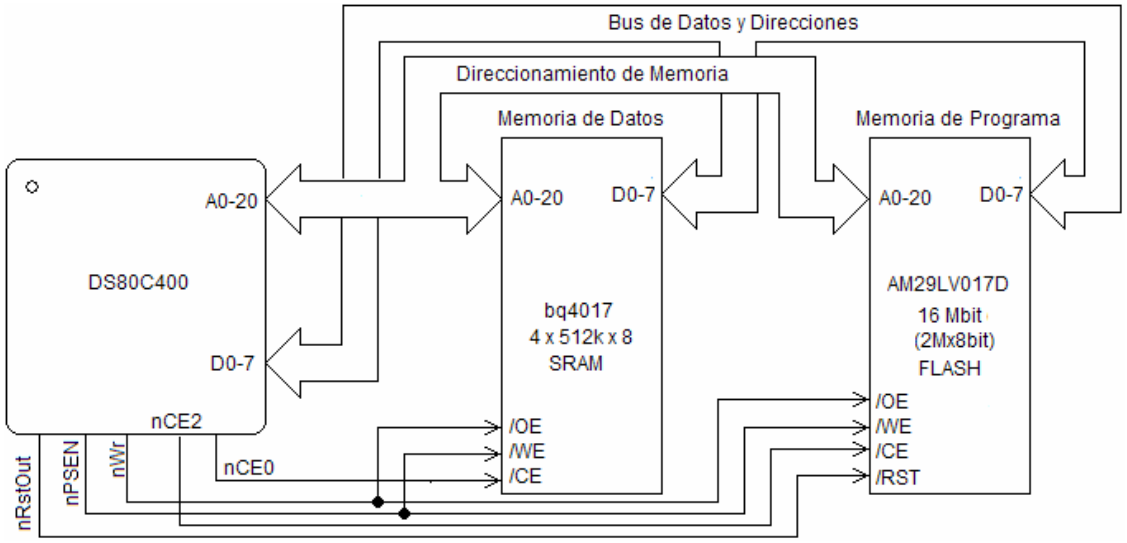


Figura 2.12. Diagrama a bloques de la memoria de datos y programa.

Para la comunicación con el módulo del decodificador Ethernet, el DS80C400 usa las terminales de transmisión y recepción, las primeras identificadas por la simbología TX mientras que las otras por RX.

La figura 2.13 muestra las conexiones del decodificador Ethernet con el microcontrolador, también las que corresponden a la comunicación al exterior a través de un conector RJ45.

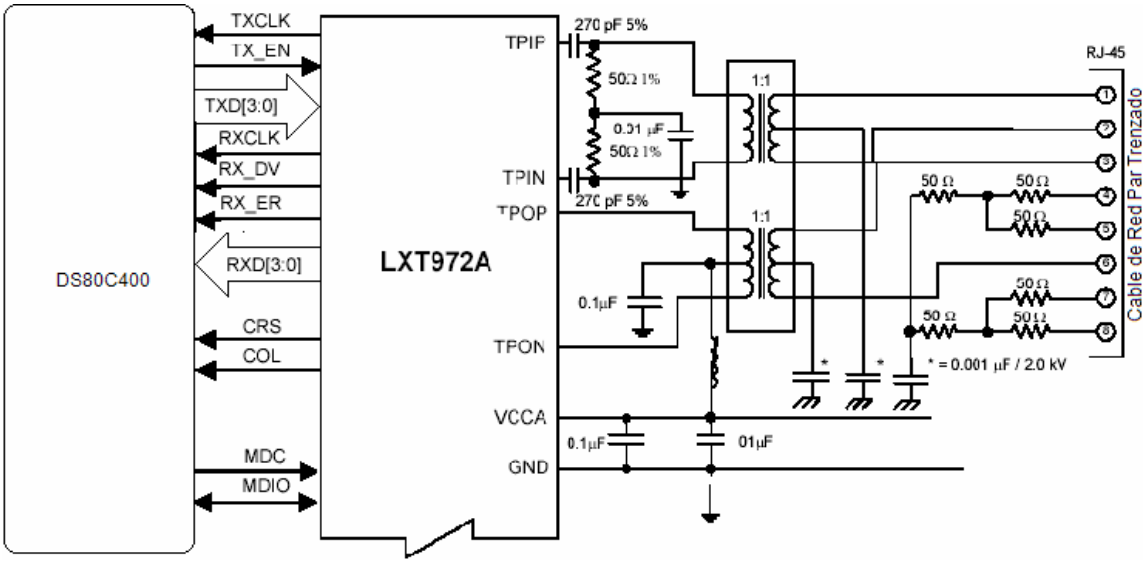


Figura 2.13. Diagrama a bloques del decodificador a Ethernet.

El sistema cuenta con diferentes interfaces a dispositivos externos. Para la comunicación serial se utilizan las señales de TX y RX (terminales 49 y 50 del microcontrolador) y con un adaptador de niveles TTL-RS232 se comunica al exterior a través de un conector serial como se muestra en la figura 2.14.

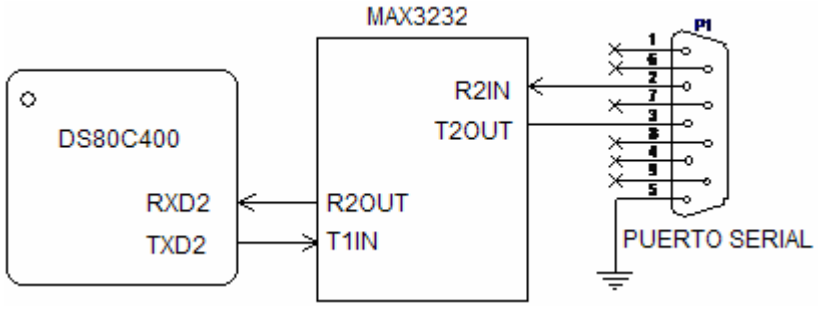


Figura 2.14. Diagrama a bloques de la comunicación serial.

Para la comunicación con dispositivos 1-Wire, el microcontrolador usa la terminal 99 ó *One Wire* (OW) para el acceso al bus 1-Wire en el intercambio de información junto con la terminal 100 de habilitación *One Wire* (OWSTP, *One Wire Strong Pull Up Enable*) para la habilitación de más dispositivos. Como se muestra en la figura 2.15, además de tener un conector disponible para dispositivos 1-Wire (J32) en el sistema se usa un identificador MAC de una línea que esta conectado a las mismas terminales OW y OWSTP para su comunicación con el microcontrolador.

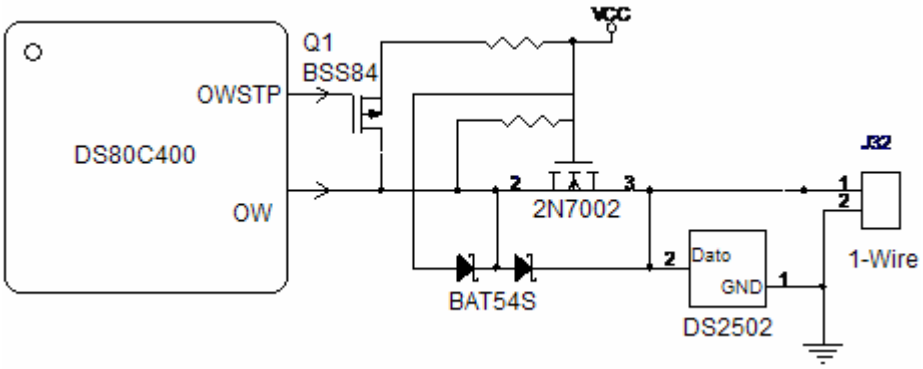


Figura 2.15. Diagrama a bloques de la comunicación 1-Wire.

Para la expansión y posibilidad de conexión con más dispositivos I/O, el sistema cuenta con: a) conexión al bus de datos y direcciones a través del puerto 0, b) señales de direccionamiento de datos (A0, A1, A19, A20) y programa (nCE3 a nCE6); c) alimentación a través de un conector y d) habilitación de periféricos como se muestra en la figura 2.16.

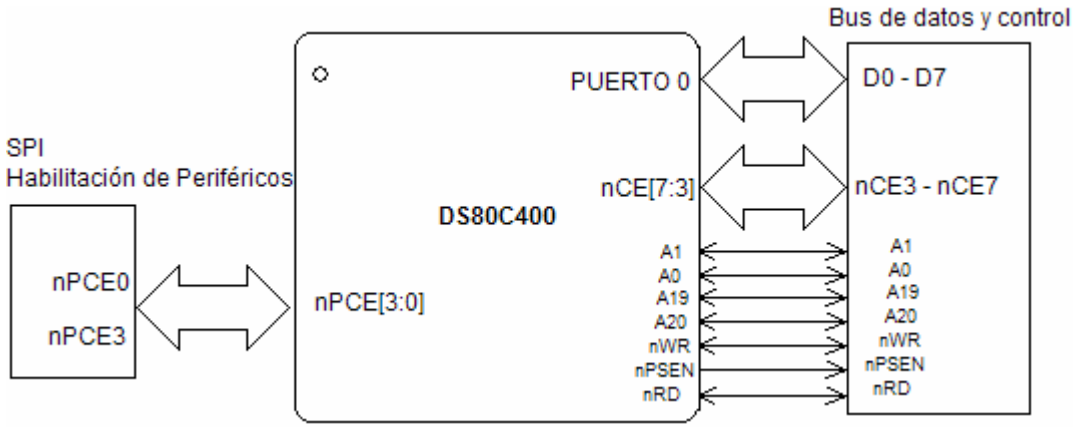


Figura 2.16. Diagrama a bloques del conector al bus de datos y direcciones y de la habilitación de periféricos.

Además, se integró una posibilidad más para la conexión de dispositivos externos a través de un dispositivo lógico programable complejo (CPLD, *Complex Programmable Logic Device*), que utiliza el bus de datos y direcciones,

dando la posibilidad de más conexiones I/O a través de otro conector como se muestra la figura 2.17.

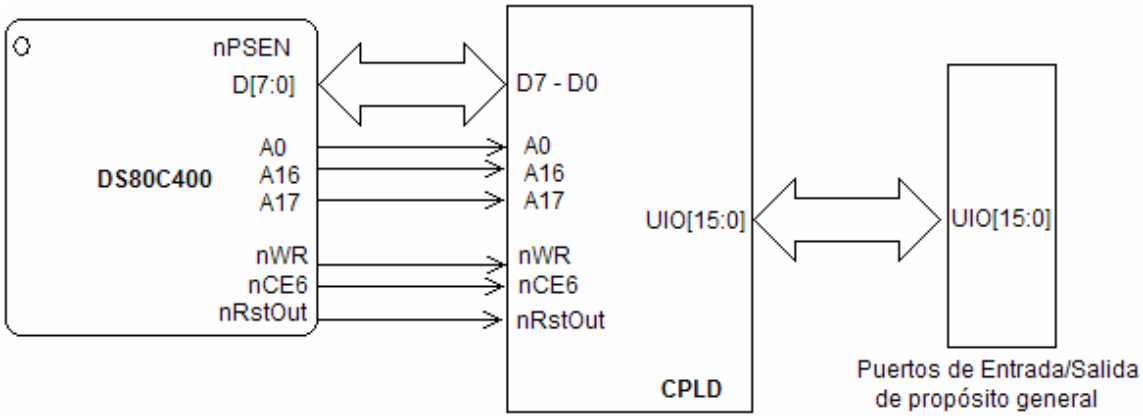


Figura 2.17. Diagrama a bloques del CPLD y las conexiones I/O a partir del CPLD.

En el Anexo A se muestran los esquemáticos de las secciones complementarias que permiten el funcionamiento del sistema, como la alimentación.

El sistema físico se muestra en la figura 2.18. En el Anexo B se indican cada una de las partes mencionadas en este capítulo.

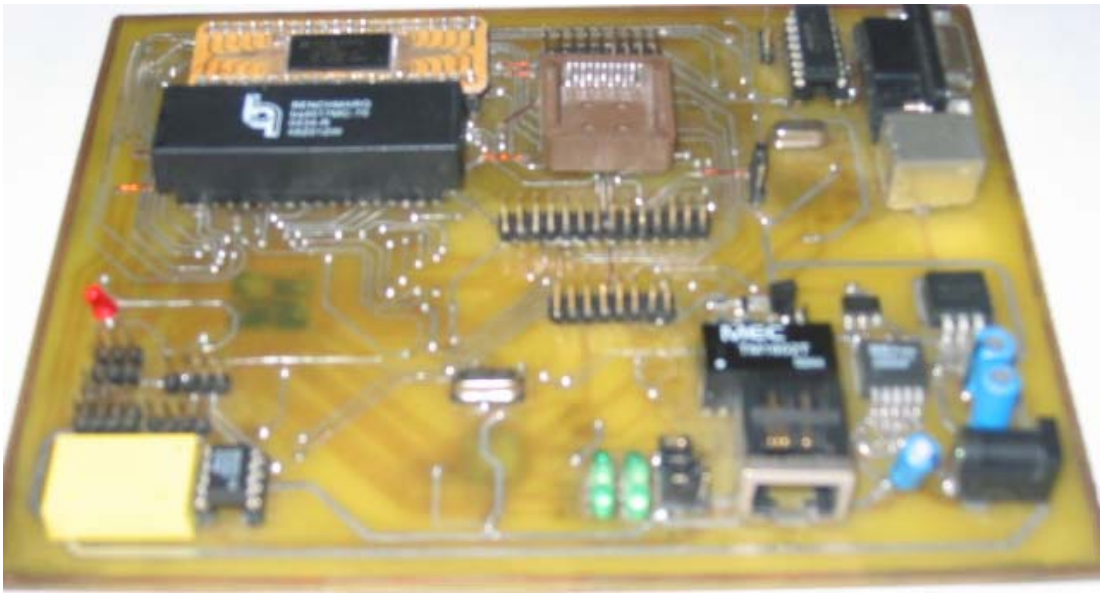


Figura 2.18. Tarjeta desarrollada (vista superior).

3

PLATAFORMA DE DISEÑO

3.1 INTRODUCCIÓN

Cuando inició este proyecto era comprensible cómo iba a funcionar y las tareas a realizar. Se conocían ciertos procesos como el manejo de datos o decodificación, pero se desconocían los mecanismos para interactuar con una red.

Una vez establecidos los elementos de hardware, en este capítulo se revisarán los aspectos de programación necesarios para la interacción entre los elementos y la comunicación con la red.

Similar a escoger los dispositivos apropiados para cumplir con los objetivos del trabajo, en la parte de programación existe la posibilidad de probar y elegir el software adecuado. Fue necesario estudiar e investigar qué programas se podían usar y si han sido realizados, los parámetros para su desarrollo y la forma en que se deben programar estos componentes.

3.2 SELECCIÓN DEL SOFTWARE

La elección del hardware esencial para llevar a cabo el sistema fue solamente la mitad del trabajo. También se requiere una gran cantidad de software que brinde la ayuda para ejecutar tareas múltiples, pilas de protocolo de red y una interfaz de programación. Es decir, un entorno de ejecución bien definido que proporcione todas estas características y permita que el usuario se centre en la aplicación.

En la búsqueda de software se consideró la adaptación a las características de los dispositivos y que forme una plataforma flexible.

3.2.1 Programación de la plataforma

La importancia del microcontrolador DS80C400 no solo es en el aspecto de hardware, también a nivel software ya que en la memoria que tiene incorporada hay incluido un entorno de ejecución hecho para este tipo de microcontroladores y la interpretación de aplicaciones.

Para acceder a la información contenida en la memoria interna y/o programar el microcontrolador puede escribirse código en lenguaje Java, lenguaje C o en ensamblador 8051.

Al usar lenguaje C o ensamblador, cada vez que se quiere actualizar o cargar una aplicación, es necesario programar nuevamente el microcontrolador, deteniendo completamente el funcionamiento del sistema, además de que la implementación de los protocolos de Internet ocuparía una gran cantidad del recurso de memoria, por ello se eligió usar Java como se describe a continuación.

3.3 LA PLATAFORMA TINI®

Para el desarrollo de este trabajo se optó por la puesta en marcha de un entorno de ejecución desarrollada por MAXIM/Dallas Semiconductor. A este ambiente de desarrollo se le denomina interfaz pequeña de Internet (TINI®, *Tiny Internet Interface*) y es un sistema operativo específico basado en el lenguaje de programación Java que incluye rutinas para conexión a Internet en el DS80C400. Es una plataforma que provee a los diseñadores de sistemas y realizadores de software una herramienta simple, flexible y de bajo costo para diseñar una amplia variedad de dispositivos hardware que puedan conectarse a una red de trabajo.

Para su uso no se requiere ser un experto en equipo o programas de Ethernet. La programación en Java permite tomar ventaja de las extensas librerías disponibles para esta plataforma y que están integradas en el microcontrolador a las que se puede acceder automáticamente cuando se programa vía TINI®.

3.3.1 Requisitos de la plataforma de desarrollo

Se utiliza el término "plataforma de desarrollo" para referirse a la computadora usada para crear, construir, y cargar las aplicaciones. Ésta es la máquina en la que funciona el kit de desarrollo Java (JDK, *Java Developer's Kit*) y el ambiente

de prueba que está conectado con el sistema usando Ethernet y/o un cable serial.

Ya que las herramientas empleadas se han escrito en Java, las aplicaciones de TINI® se pueden realizar en cualquiera de los sistemas operativos siguientes:

- MAC OSX.
- Solaris.
- Linux.
- OS Win32 (Windows 95, 98, NT, 2000, XP).

La instalación del entorno de ejecución de TINI® se puede hacer a través de un puerto USB, requisito que satisface cualquier computadora o estación de trabajo.

3.3.2 Componentes de Software

Se refiere a los programas necesarios para que la plataforma TINI® opere y ayude en la creación de aplicaciones. MAXIM/Dallas Semiconductor proporciona dos herramientas para cargar los programas. La primera es el kit de herramientas del microcontrolador (MTK, *Microcontroller Tools Kit*), recomendable para usuarios nuevos de TINI®, y por el momento solo disponible para la plataforma Windows. Básicamente es un archivo ejecutable que sirve para comunicarse con el cargador de inicialización/ROM de la mayoría de los microcontroladores de Dallas Semiconductor. Simplifica la configuración de funciones especiales y la carga y descarga de código, no es necesario software adicional para hacerlo funcionar.

La segunda herramienta es el JDK o JavaKit, un programa con una interfaz para el cargador. El JDK requiere el siguiente software para operar:

- Java Development Environment (JDE).
- Java Communications Application Programing Interface (API).
- TINI® Software Development Kit (TINI SDK).

JDE ofrece el ambiente de compilación, el segundo proporciona la infraestructura de comunicación serial RS-232 y colabora con el ambiente de operación, ambos se usan para el desarrollo de aplicaciones en Java. Entre el JDE y el TINI SDK ofrecen las herramientas de Java, el compilador y la máquina virtual de Java (JVM, *Java Virtual Machine*). Estos programas también se pueden obtener de manera libre en Internet [URL 12].

Una característica más del JDK es la inclusión de un convertidor TINI®, una herramienta con interfaz de línea de comandos para convertir archivos de clase Java a aplicaciones TINI®.

3.4 EL ENTORNO DE EJECUCIÓN TINI®

El sistema operativo, apropiadamente referido como el entorno de ejecución TINI®, se divide en dos categorías: código nativo ejecutado directamente por el microcontrolador, y una API interpretada por la JVM. El código de aplicación se escribe en Java y se usa la API para explotar las capacidades del entorno de ejecución nativo en el microcontrolador y los recursos de hardware subyacentes.

Los programas Java que se ejecutan en TINI® son aplicaciones con privilegios completos y acceso a todos los recursos del sistema, lo que significa que la aplicación es responsable de la configuración y del control de todo el sistema.

El entorno de ejecución se compone de las siguientes partes: instrucciones agrupadas en lo que se denomina API, un ejecutor denominado máquina virtual, métodos nativos que son procedimientos auxiliares de los protocolos de red y del sistema operativo que se encarga de los recursos.

3.4.1 Descripción de la API

La API son las puestas en práctica de los códigos a ocupar, es decir, programas construidos que se ejecutan cuando se requieren. Combina clases específicas de TINI®, que se encuentran como subpaquetes en la raíz *com.dalsemi,* con los definidos en el JDK de la versión 1.4.2_X usada en este trabajo, como se muestra en la tabla 3.1.

Tabla 3.1. Paquetes propios de Java y paquetes específicos de TINI®.

Paquetes Java	Paquetes com.dalsemi
Java.lang	Com.dalsemi.system
Java.io	Com.dalsemi.tininet
Java.net	Com.dalsemi.shell
Java.util	Com.dalsemi.com
	Com.dalsemi.onewire

En el Anexo C se enlistan algunas clases definidas. Las clases que se incluyen en el ambiente se conocen como la porción incorporada de la API.

3.4.2 La máquina virtual de Java

El espacio de memoria de la JVM de TINI® es pequeño (40 kilobytes) y a pesar de eso apoya en la funcionalidad de implementaciones completas, incluyendo:

- Soporte completo para hilos.
- Soporte para todos los tipos primitivos.
- Caracteres.

3.4.3 Los métodos nativos

Representa la colección de los métodos que apoyan a la API enseñando la infraestructura proporcionada por TINI®. Incluyen el acceso a la capa de red así como los controladores de dispositivos sin trabajo de red. También los métodos para configurar y tener acceso a recursos de sistema como el contador de tiempo del *watchdog timer* y el reloj en tiempo real.

3.4.4 El sistema operativo TINI®

Es la capa más baja del entorno de ejecución, responsable de manejar los recursos del sistema incluyendo el acceso a la memoria, a los procesos múltiples de programación, los hilos de ejecución y de la interacción recíproca con los componentes de hardware internos y externos. El sistema operativo es un cuerpo complejo de código que realiza muchas tareas independientes pero bien representadas por la suma de los tres componentes principales siguientes.

- ***Programación de procesos e hilos****,* se encargan de la ejecución del código al nivel de aplicación y de la sincronización permitiendo la ejecución de procesos múltiples.
- ***Administración del subsistema de memoria***, que realiza las siguientes tareas: asigna espacio para los procesos de Java y del sistema, automáticamente recolecta la basura generada por los procesos de Java y administra los archivos de sistema.
- ***Administración del subsistema de entrada/salida***, se divide en dos componentes principales: entrada/salida sin red y con red, y la aplicación Java es responsable de los detalles de comunicación de bajo nivel con los dispositivos.

3.4.5 El funcionamiento

Para entender la secuencia de eventos que ocurren al cargarse el sistema, se necesita tomar un segmento del área de programa de la figura 2.4, y dividirla en tres piezas, como se muestra en la figura 3.1.

- El cargador.
- El entorno de ejecución.
- La aplicación primaria de Java.

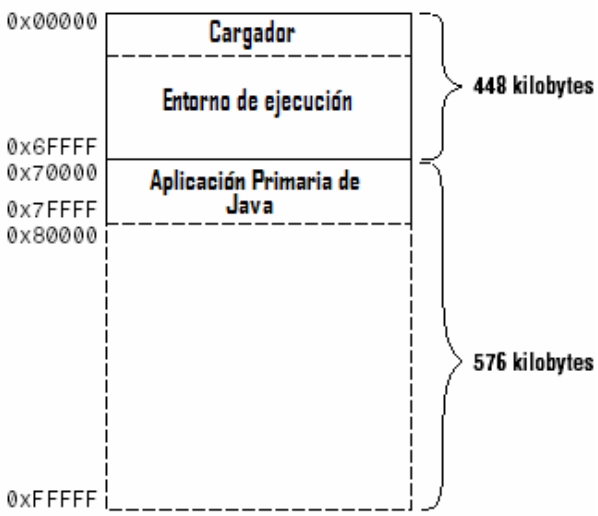


Figura 3.1. Área de programa expandida.

La secuencia de carga se puede describir en términos muy simples. El cargador es el primer código ejecutado por el microcontrolador, que en condiciones normales de inicio, transfiere rápidamente el control al entorno de ejecución. Después de ejecutar algunas rutinas de inicialización del sistema, el entorno de ejecución lanza la aplicación primaria de Java.

El proceso desde la carga hasta la puesta en marcha de la aplicación se divide en 3 pasos:

Paso 1: Ejecución del cargador

Es un programa autónomo, muy pequeño, que controla la carga del entorno de ejecución y la aplicación primaria de Java en la ROM. Su comportamiento depende del origen del reinicio que precede a la ejecución de la primera instrucción máquina del microcontrolador y puede ser reinicio por encendido o externo.

Si se trata de un reinicio por encendido, el cargador inmediatamente transfiere el control al código de inicialización del entorno de ejecución. Mientras que con un reinicio externo, espera recibir un patrón de datos especifico en el puerto serial del microcontrolador para después continuar con el inicio normal del sistema.

Paso 2: Inicializando el entorno de ejecución

Después de que el cargador cede el control se ejecutan rutinas de inicialización con las siguientes tareas.

- Revisar la integridad de la pila TCP/IP.
- Revisar la integridad de los archivos de sistema.
- Inicializar controladores de dispositivos.
- Crear procesos iniciales.

Los dos primeros procesos revisan las inconsistencias de los datos para prevenir su pérdida o corrupción por alguna interrupción de alimentación y/o operaciones incompletas. Después se ejecuta la inicialización para el puerto serial, controladores Ethernet y de otros módulos de operación. Finalmente los procesos iniciales.

Paso 3: Inicio de la aplicación primaria de Java

La aplicación primaria de Java es en un sentido análogo el hilo principal de cualquier aplicación Java. Siempre es la primera aplicación cargada automáticamente por el entorno de ejecución.

Como en todos los procesos Java, la JVM ejecuta primero la clase de los métodos inicializadores en las clases API, seguida por las clases de aplicaciones. Típicamente la aplicación primaria asume el control del sistema entero y es responsable de alguna configuración e inicialización de dispositivo de hardware que sea requerido. También puede cargar otros procesos Java o en su caso cargar procesos independientes como una línea de comandos.

A continuación se cubren los aspectos de programación y consideraciones con el hardware para desarrollar aplicaciones de comunicación con otros dispositivos.

3.5 COMUNICACIÓN CON OTROS DISPOSITIVOS

Esta sección describe como se realiza la comunicación serial y paralela haciendo uso de los paquetes API de comunicación Java.

3.5.1 Comunicación serial

Muchos dispositivos usan el puerto serial para comunicarse con otros, de hecho, para la mayoría de dispositivos es el único mecanismo de comunicación con el exterior. En estos casos no tienen un medio directo para participar en una red.

3.5.1.1 Las comunicaciones API de Java

La API se ha definido como una extensión de la plataforma Java. Está integrada por paquetes, de los cuales *javax.comm* y *com.dalsemi.comm* son los que se usan para la comunicación con dispositivos y sus clases se usarán dependiendo de la aplicación. Para el manejo de la comunicación serial, al instalar la API se debe incluir un archivo llamado *javax.comm.properties* que contiene las especificaciones para la comunicación serial. Aunque con la plataforma TINI® siempre está instalado y disponible para el entorno de ejecución. En el Anexo F se indica como se instala la plataforma TINI®.

Específicamente para la comunicación serial se requiere de objetos, porciones de código compuesto de variables y métodos, de la clase *SerialPort*, que es una subclase de la clase *CommPort*. Estos objetos proporcionan los métodos para configurar el puerto y permitir la lectura y escritura de datos al puerto físico. Aunque los objetos se crean con el método *open* sobre un objeto *CommPortIdentifier*.

La clase *CommPortIdentifier* administra el acceso a los puertos, proporciona un mecanismo para notificar cuándo el estado del puerto cambia, útil cuando hay usos múltiples que necesitan compartir un solo puerto. Los objetos *CommPortIdentifier* se pueden crear a través de los métodos *getPortIdentifier*.

La clase *SerialPort* proporciona los métodos para configurar individualmente los parámetros de tasa de transferencia, número de bits de datos, bits de paro, tipo de paridad y control de flujo, así como para obtener sus valores actuales.

En TINI®, los valores por defecto son 115,200 bps, 8 bits de datos, 1 bit de paro, sin paridad ni flujo de control, que se definen como enteros constantes públicos en la clase *SerialPort*. El método común para configurar el puerto serial es el *setSerialPortParams*.

Un ejemplo de configuración del puerto serial (representado por el objeto *SerialPort sp*) para transmitir y recibir datos a 115,200 bps con 8 bits de datos seguidos por un bit de paro y sin verificación de paridad es:

```
try {
     ...
     sp.setSerialPortParams(115200, SerialPort.DATABITS_8,
     SerialPort.STOPBITS_1, SerialPort.PARITY_NONE);
     ...
     } catch (UnsupportedCommOperationException usc) {
     ...
     }
```

Si algún parámetro es inválido, se genera un valor de excepción con *UnsupportedCommOperationException* y los parámetros regresan a los valores anteriores a invocar *setSerialPortParams*.

3.5.2 Comunicación paralela

Esta sección describe la interfaz del bus paralelo de TINI®, considerando las señales relevantes, el mapa de memoria usado para acceder al espacio entero de dirección del microcontrolador, seguido de una descripción de cómo una aplicación Java puede comunicarse con los dispositivos con interfaz al bus paralelo.

3.5.2.1 El bus paralelo de TINI®

La API de TINI® a través de la clase *DataPort* permite la comunicación con dispositivos unidos al bus. Es de uso obligado para la interfaz con dispositivos de memoria externa para almacenamiento de datos y programa. También dispositivos periféricos como el controlador Ethernet y el reloj de tiempo real.

La conexión por este medio permite que los programas TINI® accedan al dispositivo conectado para escribir y leer direcciones específicas. La combinación de señales de control y algunas líneas de dirección en conjunto con un decodificador lógico pueden servir para habilitar un periférico, como se mostró en la figura 2.10.

Las señales que comprenden el bus paralelo del microcontrolador pueden agruparse en las siguientes categorías[1]:

- Datos: bus bidireccional.
- Direcciones: bus unidireccional, controlado por el microcontrolador.
- Control: permite distinguir entre operaciones de lectura y escritura, así como la selección del dispositivo.

3.5.2.2 La clase DataPort

El acceso al bus paralelo para entrada/salida se realiza usando la clase *DataPort* definida en el paquete *com.dalsemi.system*. Un objeto de esta clase provee un delgado, pero eficiente, encapsulado del bus paralelo. Permite el control de los tiempos del bus y la lectura o escritura de datos. La manera en que se invoca a esta clase es:

```
public DataPort(int address)
```

[1]. En el Anexo D se enlistan las señales con una breve descripción.

donde, el parámetro address especifica la dirección inicial para las operaciones de entrada/salida. Un objeto de *DataPort* puede iniciar con cualquier dirección y la puede cambiar usando el método *setAddress*.

```
public void setAddress (int address)
```

3.5.3 Manejo de puertos

Los programadores de software empotrado acostumbran tener acceso directo a las terminales de puerto del microcontrolador. Esto es importante para aplicaciones empotradas porque esas terminales son de uso frecuente y proporcionan una base de interconexión simple para dispositivos como LEDs, relevadores o motores de paso.

El microcontrolador cuenta con 6 puertos, cada uno con grupos de 8 terminales disponibles, de los cuales se destinan para tareas de propósito general de entrada/salida los puertos 5 y 8[2]. Para el manejo de estos puertos a través de software también se dispone de una clase Java: la clase BitPort.

3.5.3.1 Clase BitPort

El acceso de forma individual a las terminales de los puertos se hace a través de la clase *BitPort* contenida en el paquete *com.dalsemi.system.* Usando el siguiente constructor se une un objeto a una terminal específica de puerto.

```
public BitPort (byte bitname)
```

La terminal de puerto se especifica con el parámetro *bitname*. Como se muestra a continuación, al unir un nuevo objeto a la terminal 3 del puerto 5 del microcontrolador.

```
BitPort bp = new BitPort(BitPort.Port5Bit3);
```

Los métodos *set* y *clear* se usan para controlar el estado de la terminal.

```
public void set( )
public void clear( )
```

Al invocar el método *clear* en la terminal se obtiene un nivel bajo de voltaje (un 0 lógico), con el método *set* se genera un nivel de voltaje alto (un 1 lógico). El método *set* se debe invocar antes de la primera llamada del método *read*.

[2]. En el Anexo D se muestra la tabla D.2 con los puertos accesibles a Java.

```
public int read( )
public int readLatch( )
```

El método *read* devuelve el valor de la terminal asociada, mientras *readLatch* el estado de la ultima operación de "escritura" que sería una invocación de los métodos *set (0*) o *clear (1).*

A continuación se describe un ejemplo para controlar una terminal de puerto del microcontrolador usando la clase BITPORT.

3.5.3.2 Ejemplo de manejo de una terminal

El diagrama de la figura 3.2 muestra un LED conectado al bit 2 del puerto 5 del microcontrolador (p5.2).

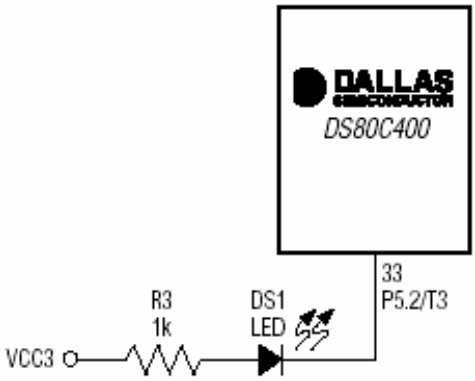


Figura 3.2. LED conectado a la terminal 2 del puerto 5 del microcontrolador.

El LED está conectado con una terminal compartida con la red interna 1-Wire, pero al no usar este recurso, la terminal está libre para utilizarse como entrada/salida de propósito general. Aprovechando el recurso de las clases se puede sensar el estado de la terminal, lo cual permite a través de código, manejar el comportamiento del dispositivo conectado, en este caso el LED.

El programa de la figura 3.3 utiliza la clase *BitPort* del paquete *com.dalsemi.system* para el acceso a p5.2. Una vez que se tenga un objeto de la clase BitPort, se pueden invocar los métodos *set* y *clear* para encender y apagar el LED respectivamente.

La creación del objeto que manejará a la terminal del puerto se especifica en la línea:

```
BitPort bp = new BitPort (BitPort.Port5Bit2);
```

Ya que *BitPort* es una clase que no es de la API normal de Java, al compilar con el programa *javac* producirá un error al no encontrarla. Se necesita usar el

indicador *"-bootclasspath"* para decirle al compilador dónde se definen las clases API de TINI®.

javac – bootclasspath c:\tini1.12\bin\tiniclasses.jar NombrePrograma.java

Es importante no poner el archivo *'tiniclasses.jar'* en la dirección de clases (o en la carpeta *jre\lib\ext*) ya que se tendrán problemas para que funcione cualquier programa de Java. Pero se debe incluir el argumento *"bootclasspath"* al compilar cualquier aplicación para TINI®. En el Anexo G se describe el proceso para crear aplicaciones TINI®.

```
import com.dalsemi.system.BitPort;
class Blinky
{
      public static void main(String[] args)
      {
            BitPort bp = new BitPort(BitPort.Port5Bit2);
            for (;;)
            {
                  // Enciende el LED
                  bp.clear();
                  // Lo deja por un 1/4 de segundo
                  try
                  {
                        Thread.sleep(250);
                  }
                  catch (InterruptedException ie)
                  {
                  }
                  // Apaga el LED
                  bp.set();
                  // Lo deja apagado por 1/4 de segundo
                  try
                  {
                        Thread.sleep(250);
                  }
                  catch (InterruptedException ie)
                  {
                  }
            }
      }
}
```

Figura 3.3. Código de programa para apagar/encender un LED conectado a una terminal.

Al hacer funcionar el programa, el LED conmutará a una frecuencia de 2 Hz por tiempo indefinido, si el programa se ejecuta en primer plano no se mostrará como activo en la línea de comandos de una sesión TELNET y se tendría que iniciar una nueva sesión para parar el programa, apagar o reiniciar el sistema. Para evitar este tipo de situaciones, las aplicaciones pueden ejecutarse en segundo plano agregando el argumento *&.*

```
TINI/> java NombrePrograma.tini &
TINI/>
```

Y para terminar un proceso se usa el comando *kill* especificando el identificador del proceso. Los procesos activos se pueden conocer con el comando *ps*.

```
TINI/> ps
3 processes
1: Java GC (Owner root)
2: init (Owner root)
4: NombrePrograma.tini (Owner root)
```

Ya mostrado el total de procesos y la identificación de cada uno, sólo se termina la ejecución del proceso deseado, en este caso el proceso 4 que corresponde al de la aplicación, con

```
TINI /> kill 4
```

Se continuará dentro del sistema y se puede ejecutar otra aplicación, si se requiere salir se escribe la instrucción *bye*.

Ahora se cubren los aspectos de programación para el entorno de red de TINI®.

3.6 RED TCP/IP

El principal objetivo de TINI® es proveer una plataforma para desarrollar aplicaciones que conecten a la red dispositivos sin tal capacidad. Java facilita la escritura de aplicaciones usando clases de acceso básico a red, que se reúnen en un paquete llamado *java.net*.

3.6.1 El entorno de red TINI®

El diagrama de la figura 3.4 muestra el entorno de red y despliega los protocolos de la capa de aplicación que soporta la API.

- HTTP.
- Sistema de nombres de dominio (DNS, *Domain Name System*).
- DHCP.
- TELNET.
- FTP.
- Ping (*ICMP echo request/reply, respuesta/replica eco de ICMP*).

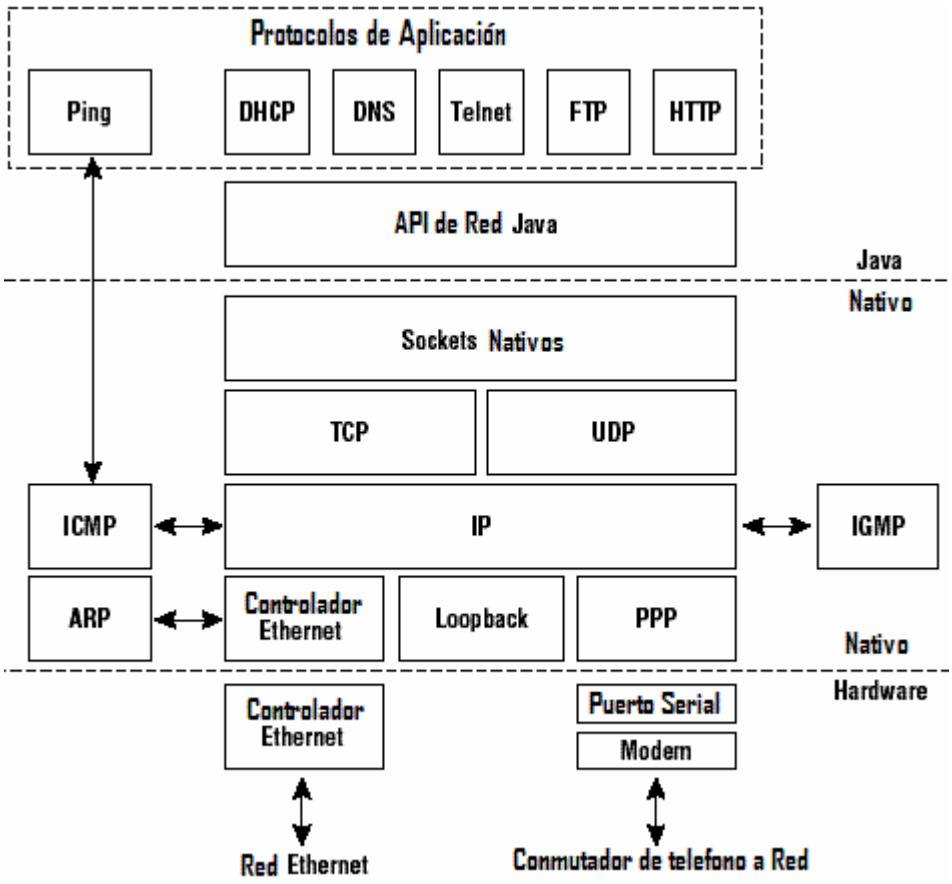


Figura 3.4. Pila de Red.

A excepción del Ping, el resto de protocolos se implementan usando los sockets del paquete java.net como el mecanismo de transporte de red. El soporte para más protocolos de la capa de aplicación lo proveen los subpaquetes de *com.dalsemi.tininet* pertenecientes a la clase TININet, que también sirve para configurar o averiguar parámetros específicos de las diferentes interfaces de red.

3.6.2 La interfaz de red

TINI® soporta Ethernet como interfaz de red, además del protocolo punto a punto (PPP, *Point-to-Point Protocol*) serial y *Loopback*. Para verificar la configuración de interfaz, se les enumera. La interfaz 0 es siempre Ethernet (eth0), 1 para la interfaz loopback, 2 y 3 para más de dos conexiones simultáneas PPP.

La interfaz de red puede añadirse o removerse usando los métodos *addInterfaceEntry( )* y *removeInterfaceEntry( )* de TININet.

3.6.2.1 Ethernet

En este diseño el trabajo de red Ethernet lo soporta TINI® con un controlador Ethernet para mandar y recibir mensajes de red a través de su dirección Ethernet o MAC, que se lee de otro dispositivo asociado.

Los siguientes métodos *getEthernetAddress( ),* definidos en *TININet,* regresan la dirección Ethernet ambos como un arreglo de bytes o de caracteres.

```
public static void getEthernetAddress (byte [] address)
public static void String getEthernetAddress ()
```

La interfaz Ethernet es la que típicamente se configura porque proporciona el medio más eficiente para la transferencia de datos. Por supuesto, las otras dos interfaces también cuentan con sus respectivas librerías. Para conexiones *PPP* se usan las clases de *com.dalsemi.tininet.ppp* mientras que *loopback* es una interfaz que siempre existe porque se configura automáticamente en el proceso de cargado del entorno y sirve para comunicarse con el anfitrión de la misma red, por lo que es suficiente usar un comando de línea para ejecutarlo: "ping" o "ping localhost".

Con estos programas se puede definir en qué red se trabajará. Un aspecto importante es como se configura la red.

3.6.3 Los parámetros de red

La clase TININet en el paquete *com.dalsemi.tininet* provee métodos estáticos, que son métodos que no se cargan en tiempo de ejecución pero se han compilado y realizan su función dentro del archivo ejecutable, para guardar y reparar toda la información de configuración usada por las diferentes interfaces de red y la pila TCP/IP, e incorporar los protocolos de aplicación.

La información de configuración de una red puede obtenerse usando el comando *ipconfig* que proporciona algunas opciones que permiten un control completo de parámetros importantes de la red.

Una muestra de salida de una configuración típica de red es la siguiente:

```
"ipconfig -x."

Interface 0 is active.
Name : eth0 (default)
Type : Ethernet
IP Address : 192.168.0.15
Subnet Mask : 255.255.255.0
Gateway : 192.168.0.1

Interface 1 is active.
Name : lo
Type : Local Loopback
IP Address : 127.0.0.1
Subnet Mask : 255.0.0.0
Gateway : 0.0.0.0

Interface 2 is not active.
Interface 3 is not active.
```

La clase *TININe*t en el paquete *com.dalsemi.tininet* provee métodos estáticos para la información de la configuración usada por las diferentes interfaces de red. Los métodos que utiliza son:

```
public static boolean setIPAddress (byte[ ] localIP)
public static boolean setIPAddress (String localIP)
```

El primer método necesita un arreglo de bytes de longitud 4 con la dirección IP en formato *big-endian byte*[3]. Este ordenamiento de byte se aplica para todas las puestas de *TININet* que requieren un arreglo de bytes para especificar alguna dirección IP o máscara de subred. El segundo método setIPAddress toma una cadena con una dirección IP en notación punto decimal- por ejemplo, "192.168.1.1". Todos los parámetros que especifican a una interfaz, como la dirección IP y la máscara de subred, tienen un método adicional que permite especificar la interfaz a la que se le aplicarían los nuevos valores.

```
public static boolean setIPAddress(String interfaceName, byte [ ] localIP)
```

Después de revisar el ambiente de software necesario para desarrollar y ejecutar aplicaciones TINI® escritos en Java, ahora se describe el proceso desde la creación a la ejecución en el sistema de cualquier aplicación TINI®.

3. Consiste en representar los bytes en orden natural: así el valor IP 192.168.6.165 se almacenaría en memoria en la secuencia {192, 168, 6, 165}.

3.7 CREACIÓN DE APLICACIONES

La figura 3.5 muestra un diagrama que describe el proceso para la creación de una aplicación ejecutable para el sistema.

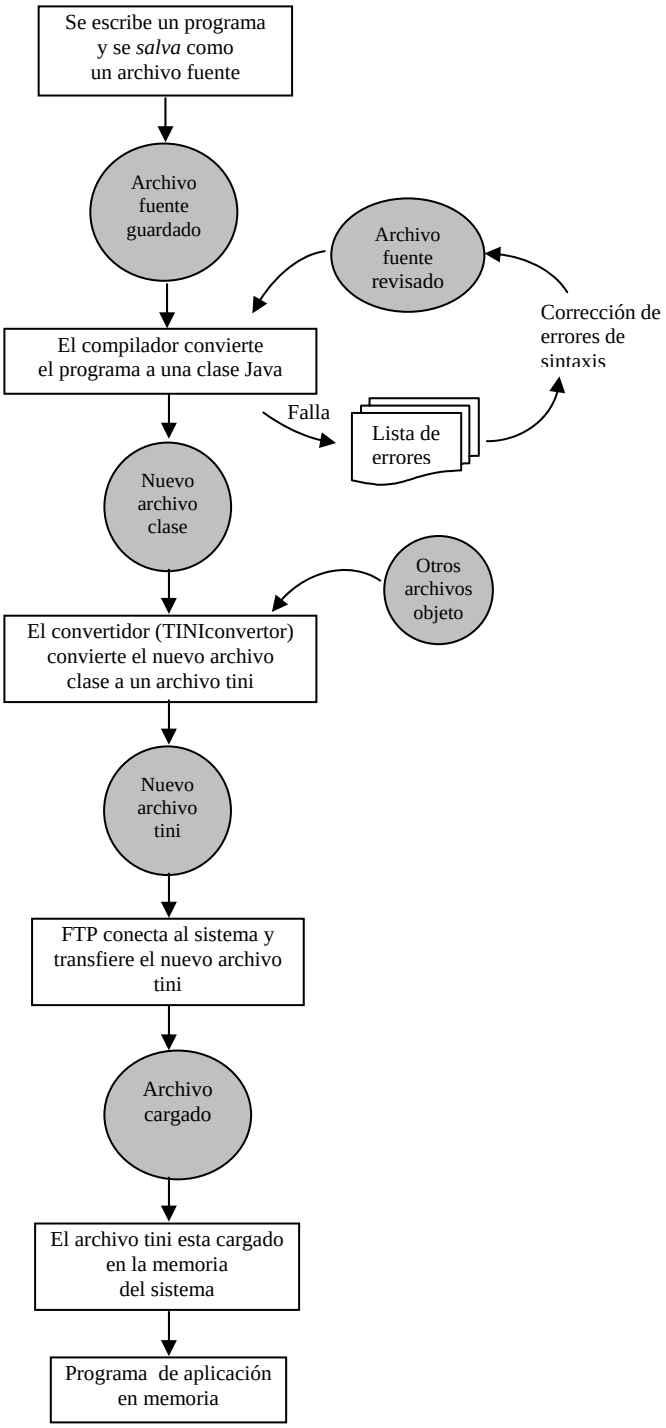


Figura 3.5. Creación de un archivo aplicación TINI.

Como se puede ver, se definen cinco pasos para realizar una aplicación para el sistema: a) crear un archivo fuente, b) compilar el archivo fuente con un compilador del lenguaje al que se desea pasar, c) convertir este archivo a uno del tipo .tini que son los que identifica el sistema como aplicaciones, d) luego se carga el archivo al sistema de archivos tini residente en memoria y e) por último, el archivo que se ha colocado en el sistema, está en espera de ser ejecutado.

Este proceso se detalla en el anexo G a través del programa “Hola Mundo”.

4

EJEMPLO DE APLICACIÓN

Partiendo de los objetivos del proyecto, se realizaron pruebas que demuestran el comportamiento del sistema. Este capítulo describe los resultados experimentales alcanzados a partir de la puesta en marcha de un ejemplo de caracterización del sistema.

4.1 INTRODUCCIÓN

El propósito de poner en ejecución al sistema es demostrar su capacidad de operación y cómo se puede utilizar. La meta es que el hardware sea una interfaz de red para una aplicación que trabaja con el sistema empotrado en el microcontrolador y exponer la capacidad de conexión con la red.

4.2 IMPLEMENTACIÓN DE LA APLICACIÓN

Con el fin de mostrar cómo funciona el sistema y cómo se pueden aprovechar sus características, se desarrolló una aplicación que consiste en un sistema remoto de adquisición de datos visible en una red de área local.

4.2.1 Descripción

La aplicación consiste en un sistema simple para la medición y control de temperatura. El sistema se acondiciona para que a través de un sensor de temperatura conectado a la tarjeta, se pueda consultar este parámetro de forma local por medio de una pantalla de cristal líquido (LCD, *Liquid Crystal Display*) o

de forma remota, en una computadora conectada a la red de área local. Se incluye la opción de control para mostrar cómo se puede variar un rango de temperatura, a través de un teclado o con una aplicación por TELNET. El sistema compara que la temperatura medida esté dentro del rango especificado, de lo contrario activará uno de los actuadores cuando el valor de la temperatura rebase o este por debajo de los límites de temperatura definidos. Una representación ilustrativa de esta idea se muestra en la figura 4.1.

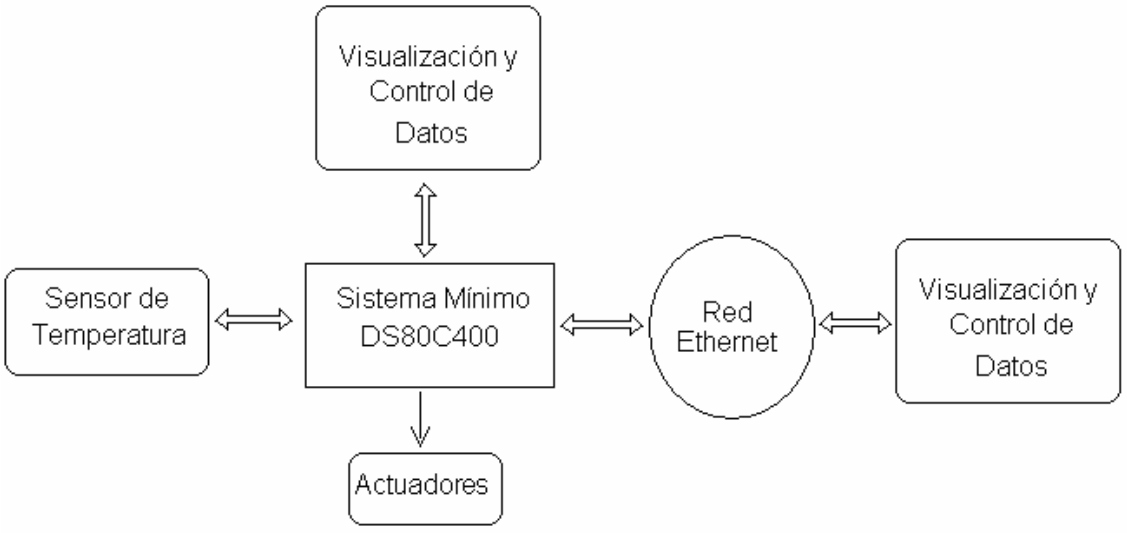


Figura 4.1. Diagrama general de la aplicación.

4.2.2 Desarrollo de la aplicación

El primer paso del ejercicio fue determinar los elementos que interactuarán con el sistema y la forma en que lo harán.

4.2.2.1 La tarjeta

En este ejemplo se trabaja con las opciones de conectividad que posee la tarjeta y se proporciona una solución al problema de conectar dispositivos pequeños a la red. Aprovechando los módulos de comunicación con los que cuenta como la Ethernet MAC, los puertos de I/O y los terminales de propósito general, se ejemplifica la versatilidad y capacidad de la tarjeta construida.

La tarjeta es el enlace entre la interfaz Ethernet y el bus de datos y/o direcciones al que están conectados los dispositivos controlados a través de los puertos.

Para la solución se aprovecha el sistema TINI® embebido en el microcontrolador de la tarjeta, que permite la conexión a Internet. También se

manejan los puertos disponibles del microcontrolador a los que se conectan el sensor, el teclado, el LCD y los actuadores.

En la figura 4.2 se muestra el espacio disponible para la conexión de diferentes periféricos.

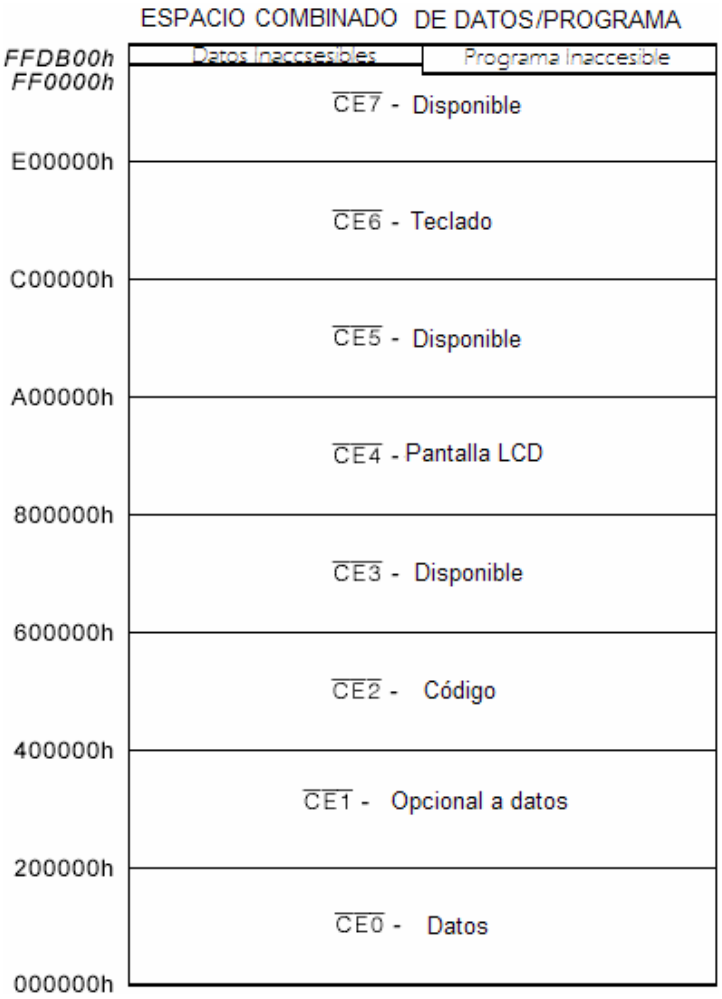


Figura 4.2. Mapa de memoria para programas y datos.

Para usar espacio de memoria disponible se debe considerar el empleado por la memoria para almacén de datos y programa, y el espacio del entorno de ejecución tal como se mostró en la figura 4.2. El espacio a usar por cada periférico se determina a través de terminales disponibles para su habilitación, las cuales se muestran en la tabla 4.1.

Tabla 4.1. Terminales CE disponibles y sus funciones alternas correspondientes.

SEÑAL CE	FUNCIÓN ALTERNA : TERMINAL DE PUERTO
CE1	P4.1
CE3	P4.3
CE4	P6.0
CE5	P6.1
CE6	P6.2
CE7	P6.3

El manejo de estos dispositivos se hace por los puertos disponibles, y para su habilitación se utilizan las terminales P6.2 (CE6) para el teclado y P6.0 (CE4) para la pantalla LCD. El sensor es un caso especial, ya que se trata de un dispositivo 1-Wire y por lo tanto solo requiere de una terminal específica determinada por el 1-Wire maestro en este microcontrolador y corresponde a la terminal OW (99).

4.2.2.2 El sensor

Con la capacidad del DS80C400 que incluye el protocolo de comunicación para dispositivos 1-Wire, se usó el DS18B20 que es un termómetro digital para monitorear la temperatura ambiental.

El sensor se conecta a la tarjeta por medio del bus maestro 1-Wire del microcontrolador. Por medio del programa se sabrá que esta conectado, debido a que se revisa continuamente la existencia de algún dispositivo en la terminal OW.

El sistema actúa como intermediario entre el dispositivo 1-Wire y una PC de control conectada a la red Ethernet. La PC recibe toda la información relativa a la temperatura medida ya que el sistema identifica al sensor a través de un único número serial que tienen estos dispositivos.

4.2.2.3 La pantalla LCD

Este periférico sirve para visualizar de forma local, el valor actual de la temperatura ambiente y los valores del rango en que debe mantenerse la temperatura antes de accionar un actuador.

Se realiza el mapeo en el espacio de memoria para la habilitación de periféricos (PMCE, *Program Memory Chip Enable*), que es un área externa para datos, a la que el microcontrolador accede muy eficientemente y se usa para una interfaz directa al bus de datos y direcciones.

Para este ejercicio se dispondrá como espacio para el LCD el correspondiente a CE4 y por lo tanto se tiene disponible un rango de 2 MBytes a partir de la dirección 600001h.

4.2.2.4 El teclado

El uso de un teclado demuestra la posibilidad que tiene la tarjeta para manejar interrupciones externas como eventos. En este ejemplo, su funcionamiento es para configurar los valores del rango de temperatura de manera local.

Se conecta a la tarjeta a través de las terminales con función alterna similar a la pantalla LCD. También se realiza el mapeo a una dirección de memoria de acuerdo al espacio PMCE disponible como lo muestra la figura 4.3.

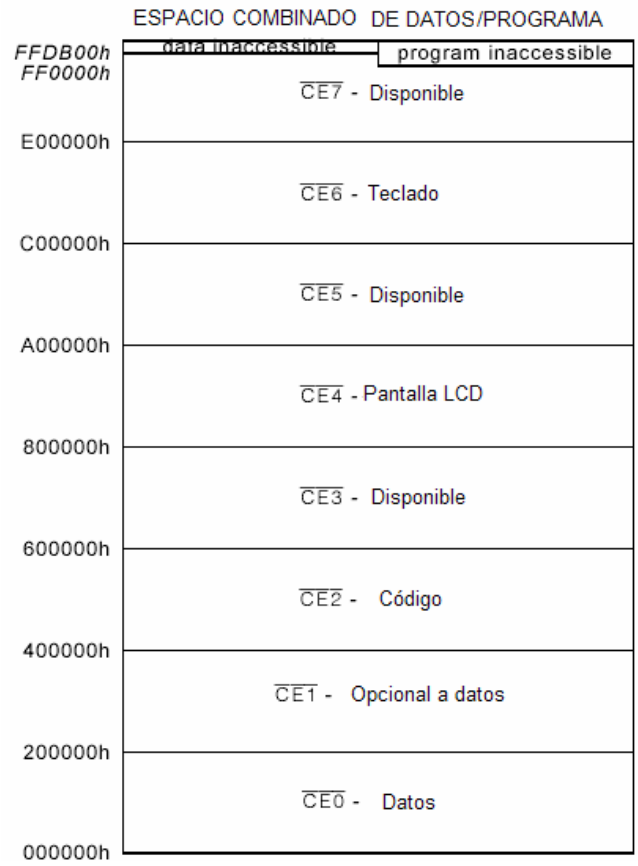


Figura 4.3. Mapa de memoria con los periféricos habilitados.

4.3 DESCRIPCIÓN GENERAL DEL FUNCIONAMIENTO

El sistema es una combinación de alta integración de hardware e implementa un segmento de código funcionalmente importante de diversos protocolos de comunicación bajo un ambiente de operación Java.

El lenguaje de operación Java permite la ejecución de aplicaciones remotas independientemente de la estructura de hardware o software del sistema. Lo que facilita la realización de aplicaciones independientes de la plataforma física.

La aplicación Java ejecutándose sobre el sistema de la tarjeta desempeña la tarea de comunicación con el dispositivo en su lenguaje nativo y presenta los resultados a sistemas remotos alcanzables a través de una red TCP/IP. El programa revisa periódicamente la presencia de una respuesta en el adaptador 1-Wire y se comunica con el programa intermedio para enviarla al servidor y se muestre en una página.

La importancia de la programación en Java es que soporta la ejecución de varios procesos al mismo tiempo, por lo que se puede hacer que el sistema este ofreciendo los datos justo en el momento en que sean obtenidos, a la vez que se realizan otras acciones totalmente independientes unas de otras, como en este caso el del teclado y el programa para la variación de parámetros.

En la figura 4.4 se muestra de manera descriptiva el comportamiento de la aplicación. El proceso *WebTemp* trabaja de manera continua, se encarga de actualizar la página Web y de controlar a otros cuatro procesos que trabajarán en forma concurrente. Los administra de manera que les asigna los datos necesarios para su funcionamiento y los resultados que generan usarlos para realizar un funcionamiento integral. Estos procesos son:

- Web: Responder a las peticiones en el puerto 80 del TCP.
- Temperatura: Revisar la temperatura medida por el DS1920.
- LCD: Actualizar la pantalla LCD y
- Teclado: Manipular al teclado.

Además del programa *ConfiguraSensor* que está en espera de ser activado por el usuario, cuando requiere cambiar el rango de temperatura, por el puerto 23 (TELNET).

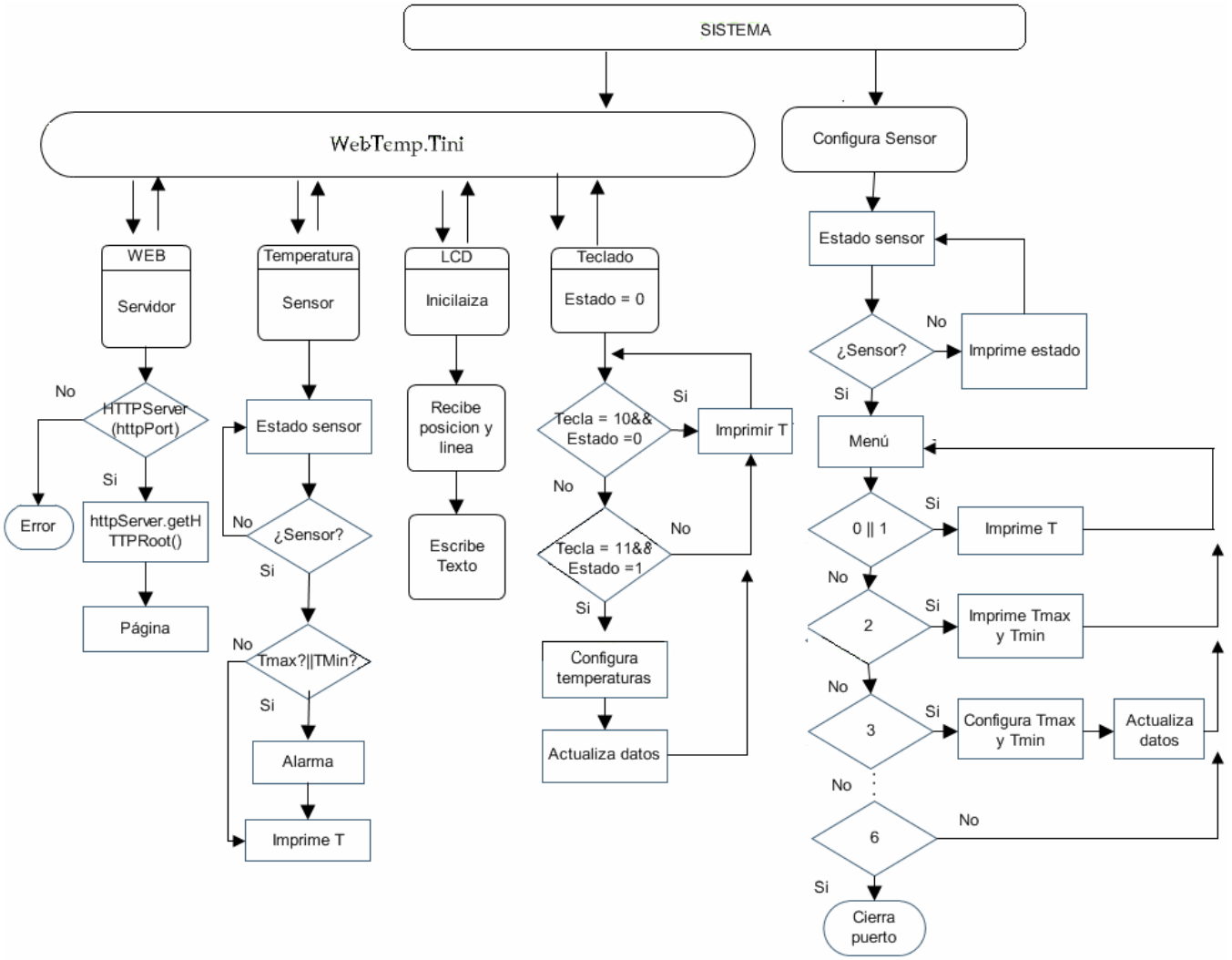


Figura 4.4. Diagrama de flujo de la aplicación.

De acuerdo a la figura 4.4 se observa que *WebTemp* es un hilo compuesto por procesos individuales. *ConfiguraSensor* es un proceso que aunque parece individual tiene que ver con *WebTemp* al modificar los parámetros del sensor que ambos ocupan, solo que tiene que funcionar individualmente porque es de conexión remota y el uso del mismo recurso afecta el intercambio de información.

El sistema se comporta de acuerdo con el programa residente en el microcontrolador y se encarga de inicializar los elementos que operan en el ejercicio. Es un programa realizado en Java y que se carga en el microcontrolador para que sea ejecutado. La carga se realiza con una sesión FTP a través de la dirección IP asignada, siguiendo las siguientes instrucciones.

C:\> ftp 192.168.6.159

Al ingresar al sistema se pide el nombre de usuario y la contraseña que sirve como protección. Iniciada la sesión, se sigue el procedimiento habitual para la transferencia de archivos con FTP.

ftp>
ftp> bin así la imagen binaria no se altera en la transferencia.
ftp> put TempWeb.tini
ftp> bye

Con esto el programa residirá en la memoria del sistema y estará listo para funcionar. Sólo basta conectar la alimentación a la tarjeta y establecer una conexión a Ethernet, para que la aplicación se pueda ver a través de una página.

El programa consiste en la declaración y configuración de los recursos a utilizar, se les asignan los espacios de dirección a ocupar, como se muestra en la figura 4.5, que es parte del código realizado.

```
public LCD( )
{
        this.lines=2;
        this.width=16;
        line = new int[width][lines];
        LCDinit(0x00880000);
}
```

(a)

```
int count,valor=0;

DataPort bus8b= new DataPort(0x00C00004);
```

(b)

Figura 4.5. Especificación de espacio en el programa para (a) el LCD y (b) para el teclado.

También se deben especificar los paquetes a usar para estos dispositivos, que son las clases del paquete de trabajo. En el caso del LCD se implementó la rutina de inicialización, esto es importante para reconocer los comandos y datos que se le enviarán.

Con el sensor se debe usar las clases de TINI® para dispositivos 1-Wire contenidos en el paquete *com.dalsemi.onewire* que se define en un grupo de subclases que permiten al sistema identificar los dispositivos de una via disponibles y la funcionalidad compartida por los dispositivos.

Para los periféricos (LCD y teclado) las clases utilizadas se tienen en el paquete *com.dalsemi.system,* especialmente las clases con acceso al bus paralelo de I/O como el DataPort y el BitPort. El primero para lectura y escritura de datos al bus mientras que el segundo para asignar un objeto a una terminal de puerto especifico.

Este programa reside en la memoria del sistema y opera de forma automática, ya que se declara en el archivo de arranque del sistema operativo (.startup) junto con la inicialización del sistema. Su funcionamiento se describe en la siguiente sección y se muestra lo que el usuario podrá visualizar.

4.3.1 Resultados

El valor actual de la temperatura se puede conocer localmente revisando el valor desplegado en el LCD o en forma remota, desde cualquier punto de la red con la página creada, en la dirección 192.168.6.159 que corresponde a la IP asignada a la tarjeta.

Cuando el termómetro digital está correctamente conectado se puede ver el dato medido como se muestra en la figura 4.6 a través de la página del sistema y en el LCD que se muestra en la figura 4.7.

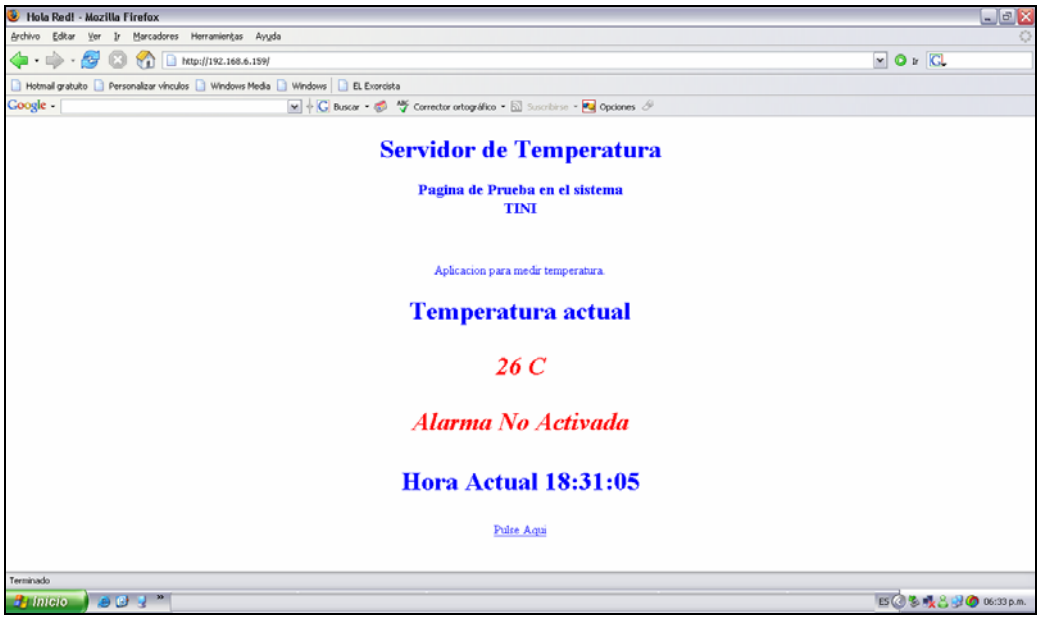


Figura 4.6. Página del sistema que muestra la temperatura actual.

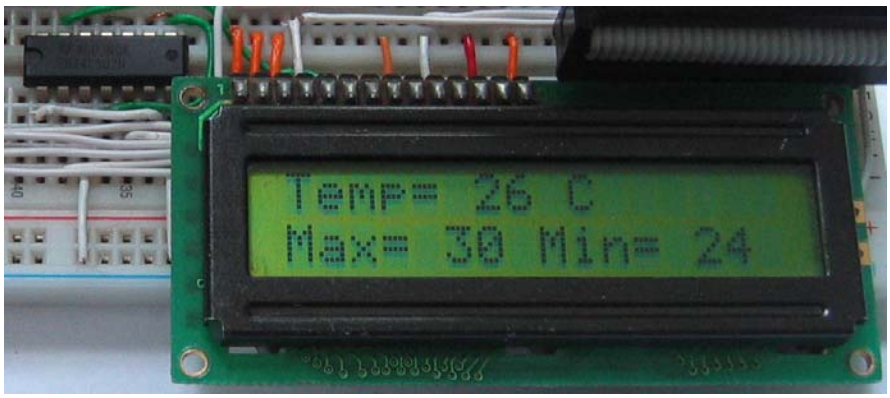


Figura 4.7. LCD con la temperatura medida.

En caso de que no este conectado el dispositivo también se hace la indicación de manera local y remota, como lo muestra la figura 4.8 con la página y la figura 4.9 a través del LCD.

Figura 4.8. Aviso de sensor no disponible a través de la página.

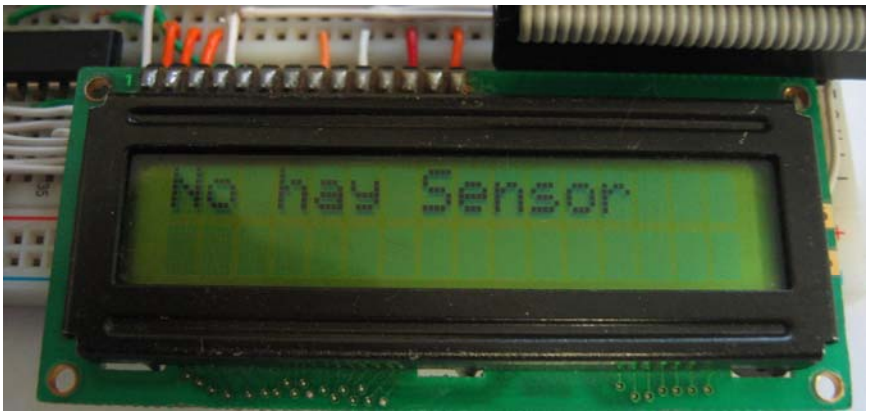


Figura 4.9. LCD con mensaje de aviso de sensor no disponible.

El sistema está monitoreando el valor del sensor continuamente para compararlo con los valores máximo y mínimo que definen el rango permitido. Si el valor medido está fuera del rango se activará un actuador.

Los actuadores se modelan por medio de un par de LEDs. Uno está ubicado en la misma tarjeta, el cual se encenderá si la temperatura medida esta arriba del valor máximo definido. El segundo LED es externo, conectado a una de las terminales de puerto y que se enciende cuando la temperatura esta abajo del rango mínimo.

Al mismo tiempo que estos actuadores se activan, en el LCD se desplegó un mensaje de aviso como lo muestra la figura 4.10.

Figura 4.10. Aviso local de activación de alarma.

También el estado de alarma se presenta a través de la página del sistema con un mensaje como se muestra en la figura 4.11.

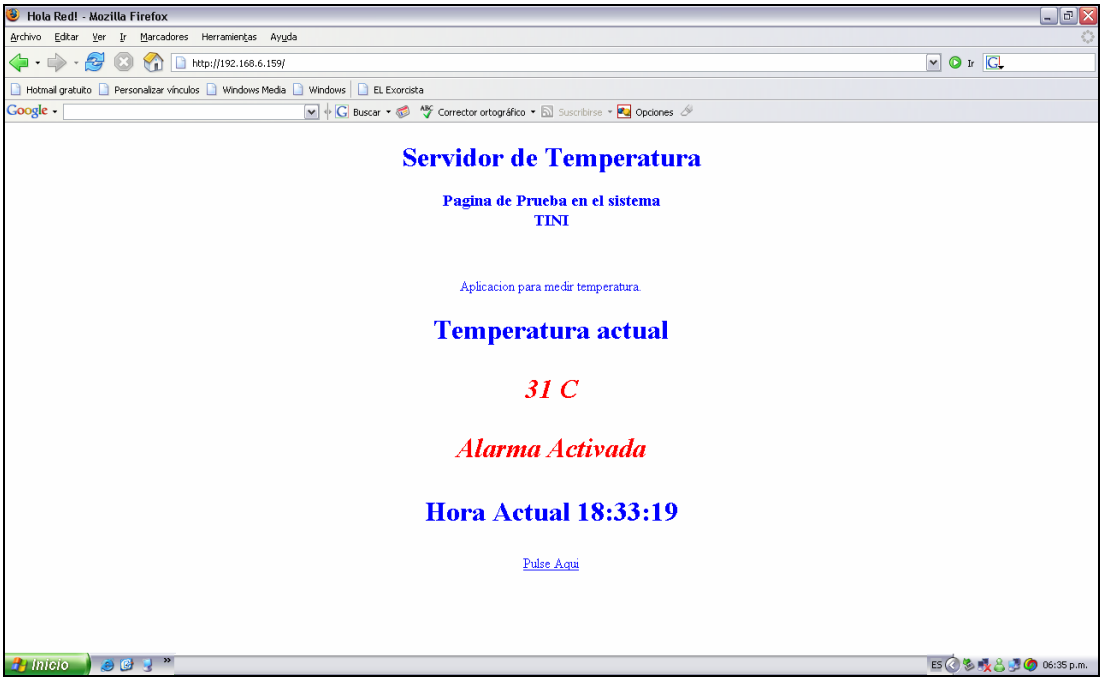


Figura 4.11. Mensaje de activación de alarma en la página del sistema.

La información de la página se actualiza constantemente; esto se realiza a través de la comunicación con los elementos remotos conectados a la tarjeta, en este caso el sensor, enviándole solicitud de lectura.

Java ofrece la posibilidad de ejecutar procesos autónomos, aprovechando esta facultad y con el fin de exponer el buen funcionamiento del sistema, en este ejemplo además de la medición de temperatura se pueden modificar los valores del rango para poner la alarma.

Para definir los valores de temperatura máxima y mínima, hay dos posibilidades, la primera es de acceso local a través del teclado. En el teclado hay dos teclas reservadas para la configuración. La primera tecla con posición 11 permite el ingreso a la configuración del rango, empezando con la temperatura máxima. Con las teclas de los dígitos se define el valor y con la tecla en la posición 12 se acepta. Luego, de manera similar se procede con la temperatura mínima.

La otra forma que el usuario tiene para acceder al sistema es a través de una conexión TELNET por línea de comandos indicando la IP correspondiente y tiene que ejecutar un programa que reside en memoria. En caso de una red protegida con restricción a TELNET la posibilidad es usar una página en lenguaje hipertexto de marcas (HTML, *HiperText Markup Language*) a través de la cual se llame la aplicación residente en memoria. El enlace suministrado por la TINI® es bidireccional, permitiendo a un sistema remoto controlar el dispositivo.

Primero se deben detener los procesos activos para luego proceder con la configuración.

C:\> telnet 192.168.6.159
Tini> Kill #de proceso para detener la ejecución de TempWeb
Tini> java Configurasensor.tini

El programa *ConfiguraSensor* no se ejecuta de manera permanente pero está almacenado en la memoria del sistema y opera con los mismos recursos. Una vez que se pone en funcionamiento, ofrece el menú mostrado en la figura 4.12.

Figura 4.12. Menú que muestra el programa ConfiguraSensor.

Las opciones son:

0.- Leer temperatura, que despliega una vez el valor de la temperatura medida.
1.- Leer la temperatura un número n de veces que permite observar el valor de la temperatura las veces que el usuario escriba, con una diferencia de un par de segundos entre medición.
2.- Leer alarmas muestra los valores máximo y mínimo que corresponden al rango que maneja la aplicación.
3.- La siguiente opción es para fijar esos valores, como se muestra en la figura 4.13 donde pide los valores correspondientes para después indicar que esos valores ya se han tomado.
4.- Estado de alarma es la opción para verificar si la alarma esta o no activada.
5.- El modo de despliegue es para cambiar la escala en la que se mostrará la temperatura, son dos opciones: Celsius (ºC) o Fahrenheit (ºF).
6.- Salir de la aplicación.

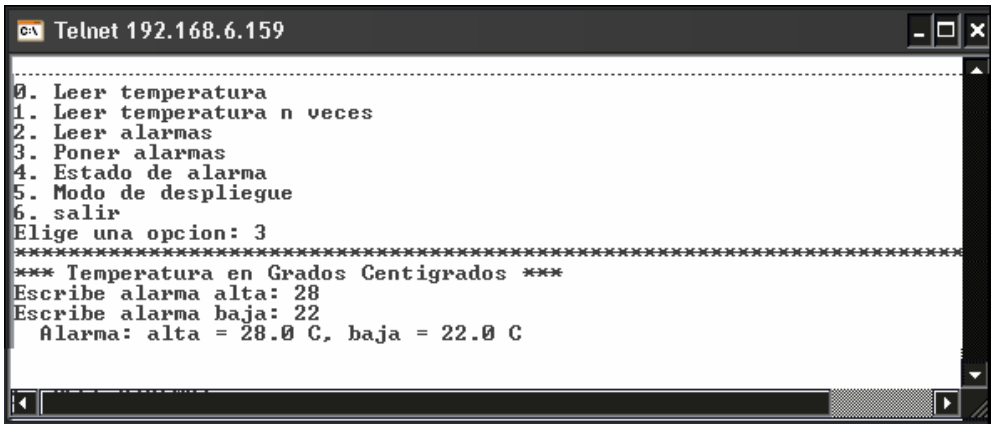


Figura 4.13. Opción 3 para la variación del rango de temperatura.

Al terminar con el proceso de configuración, es necesario volver a ejecutar el programa que soporta la aplicación, con la siguiente orden en la línea de comandos:

Tini> java TempWeb.tini &

Uno de los aspectos fundamentales en el desarrollo de la aplicación, fue el permitir la modificación de variables desde dos ámbitos diferentes, uno local y el otro remoto.

En el apéndice E se describen las clases realizadas para el funcionamiento de la aplicación.

5

CONCLUSIONES

La finalización de este proyecto enfocado a la creación de un sistema mínimo con conectividad a una red, conlleva al planteamiento de conclusiones importantes.

El sistema realizado cumple satisfactoriamente con los objetivos planteados inicialmente.

- Se logró un sistema con los módulos que establecen un sistema mínimo: memoria disponible, comunicación e intercambio de información.
- Se tiene conectividad a Ethernet, que en combinación con las interfaces de comunicación del DS80C400 simplifica el establecimiento de una red.
- El sistema cuenta con una capacidad de comunicación e integración de dispositivos externos al sistema.
- Incluye un sistema operativo basado en Java para la ejecución de aplicaciones.
- En la parte académica, el desarrollo de aplicaciones con este sistema, puede ayudar a la mejor comprensión de materias como: Redes de computadoras, arquitectura de computadoras y microcomputadoras, además permitirá poner en práctica aspectos de programación.

Con la llegada de los dispositivos integrados, los cuales se conectan a Internet, se presenta una gran oportunidad para desarrollar aplicaciones innovadoras. Este tipo de tecnología presenta una elegante puerta de entrada a la futura revolución en el campo de la conectividad de todos los dispositivos electrónicos a la red de redes.

Los resultados obtenidos en la implementación de la aplicación de monitoreo y control de temperatura para probar el sistema fueron bastante satisfactorios. El sistema se comportó correctamente al establecer una comunicación remota y local, al permitir control sobre la aplicación y la ejecución de procesos en paralelo, además de que no hubo problemas al realizar dos procesos o más al mismo tiempo, aún cuando los procesos eran independientes uno del otro. El sistema operativo TINI® ofrece la ventaja de tener independencia de la plataforma física y además la metodología de desarrollo mostró ser muy versátil y flexible.

Con el soporte integrado de conectividad Ethernet, comunicación 1-Wire, entradas/salidas de propósito general e interfaz serial el sistema tiene la capacidad para el desarrollo de aplicaciones en diversos ambientes y con el control de variables en ámbitos diferentes, uno local y el otro remoto.

5.1 PERSPECTIVAS

Usando las capacidades incorporadas de TCP/IP en la ROM del microcontrolador en conjunto con el sistema TINI®, el sistema puede ser ampliamente aprovechable para aplicaciones a través de la Internet e incorporar dispositivos sin este tipo de conexión.

Se cuenta con una plataforma de hardware y software, que puede ser la base para el desarrollo de aplicaciones como: Automatización de servicios para particulares (domótica) e industrial, monitoreo remoto de procesos, etcétera. Aplicaciones que puedan manipularse con un control vía Ethernet TCP/IP o en combinación con protocolos de comunicación inalámbrica, se puede llegar al desarrollo de más aplicaciones o realizar una gran red de comunicación entre sistemas empotrados.

REFERENCIAS

[1] MAXIM: DS80C400 Network Microcontroller. Data Sheet REV: 060805, MAXIM/Dallas Semiconductor. Agosto, 2005.

[2] INTEL: Intel LXT972A Single-Port 10/100 Mbps PHY Transceiver. Data Sheet 24918603, Intel Corp., Oct. 2005.

[3] D. LOOMIS. "*The TINI Specification and Developer's Guide",* Addison-Wesley. Pub Co, 2001.

[4] MAXIM: High-Speed Microcontroller User´s Guide, Rev: 042307 MAXIM/Dallas Semiconductor, 2007.

[5] MAXIM: High-Speed Microcontroller User´s Guide: Network Microcontroller Supplement, Rev: 8, 8/06, MAXIM/Dallas Semiconductor.

[6] SUN DEVELOPER NETWORK: Java 2 SDK, Standard Edition, Version 1.4.2 Release Notes, Sun Microsystems, Inc., 1994-2007.

[7] SUN DEVELOPER NETWORK: Java 2 Platform, Standard Edition, v1.4.2 API Specification, Sun Microsystems, Inc., 2003.

[8] STALLINGS, William: *"Comunicaciones y Redes de Computadores"*, 6ª Ed. Prentice Hall, 2002. ISBN 84-205-2986-9.

[9] TANENBAUM, Andrew S.: *"Redes de Computadoras"*, 3ª Ed. Prentice-Hall Hispanoamericana, S.A., México, 1997. ISBN 968-880-958-6.

[10] BLOCH, J. "*Effective Java Programming Language Guide".* Addison-Wesley Professional, 2001.

[11] ARNOLD, Ken; GOSLING James; HOLVES David. "*El lenguaje de programación JAVA".* Addison-Wesley 2001.

[12] GARCÍA DE JALÓN, J. et al. "*Aprenda Java como si estuviera en primero".* Escuela Superior de Ingenieros Industriales de San Sebastián, Universidad de Navarra, Colección:"Aprenda…, como si estuviera en primero", Febrero 2000

[13] CAMPIONE, M.: *"The Java(TM) Tutorial"*. Addison-Wesley Pub Co, 2000.

[14] NAUGHTON, Patrick. "Manual de JAVA". Osborne-McGraw Hill, 1996.

[15] HOLZNER, Steven Holzner. "La biblia de JAVA 2". Anaya Multimedia, 2000.

[16] MAXIM: DS18B20 Programmable Resolution 1-Wire Digital Thermometer. Data Sheet REV: 030107, MAXIM/Dallas Semiconductor.

[17] MAXIM. 1-Wire Specification, Rev: 081297, MAXIM/Dallas Semiconductor. Diciembre, 2004.

[18] AWTREY, Dan. *"The 1-Wire Weather Station"*, Dallas Semiconductor, Junio, 1998.

Sitios de Internet

[URL 1] http://onu.org.do/instraw/internet/index.html
"Sitio de información de las Naciones Unidas en la República Dominicana". Tema: Información general sobre el Internet, sus usos y definición. Última visita: 15 de junio de 2007.

[URL 2] http://www.iespana.es/mundointernet/index.html
"Buscador español de información en general". Tema: Definición de red. Última visita: 15 de junio de 2007.

[URL 3] http://wwwfie.iepn.edu.ec/microprocesadores/
"Sitio de la Escuela Politécnica Nacional, Quito, Ecuador". Tema: Los microcontroladores y la arquitectura Von Newman. Última visita: junio 2005.

[URL 4] http://www.maxim-ic.com/
"Sitio web de la compañía de desarrollo de circuitos electrónicos MAXIM/Dallas Semiconductor". Tema: Documentación DS80C400. Última visita: 15 de junio de 2007.

[URL 5] http://www.almiron.org/historia.html
"Blog de la Dra. Nuria Almiron con información de tecnología, los medios de comunicación y poder económico". Tema: Evolución de Internet. Última visita: 15 de junio de 2007.

[URL 6] http://www.iespana.es/mundointernet/historia.html
"Buscador español de información en general". Tema: Historia de Internet. Última visita: 15 de junio de 2007.

[URL 7] http://www.educa.aragob.es/cursoryc/redes1/
"Sitio del Departamento de educación del Gobierno de Aragón, España, dedicado a la impartición de cursos de informática". Tema: Información sobre Redes. Última visita: 15 de junio de 2007.

[URL 8] http://es.encarta.msn.com © 1997-2007 Microsoft Corporation.
"Red (informática)," Enciclopedia Microsoft® Encarta® Online 2007.
Tema: Componentes de una red. Última visita: 15 de junio de 2007.

[URL 9] http://www.protocols.com
"Sitio con información acerca de los protocolos de red." Tema: Definición de protocolos TCP/IP. Última visita: 15 de junio de 2007.

[URL 10] ftp://ftp.dalsemi.com/pub/tini/ds80c400
"Espacio en el servidor de MAXIM/Dallas Semiconductor para descarga de archivos relacionados con el microcontrolador DS80C400." Tema: Consulta y revisión de código. Última visita: 15 de junio de 2007.

[URL11] http://www.maxim-ic.com/products/microcontrollers/micro_ethernet.cfm
"Sitio web de la compañía de desarrollo de circuitos electrónicos MAXIM/Dallas Semiconductor". Tema: Microcontroladores con red. Última visita: 15 de junio de 2007.

[URL 12] http://java.sun.com/
"Sitio de la compañía SUN Microsystems para desarrolladores de JAVA". Tema: Descarga de software y revisión de clases API v1.4.2. Última visita: 15 de junio de 2007.

ANEXO A. Esquemáticos del sistema

Después de la descripción de las partes que componen al sistema por medio de diagramas a bloque que se presentan en el Capítulo 2, en este anexo se muestran los diagramas esquemáticos que conforman al sistema.

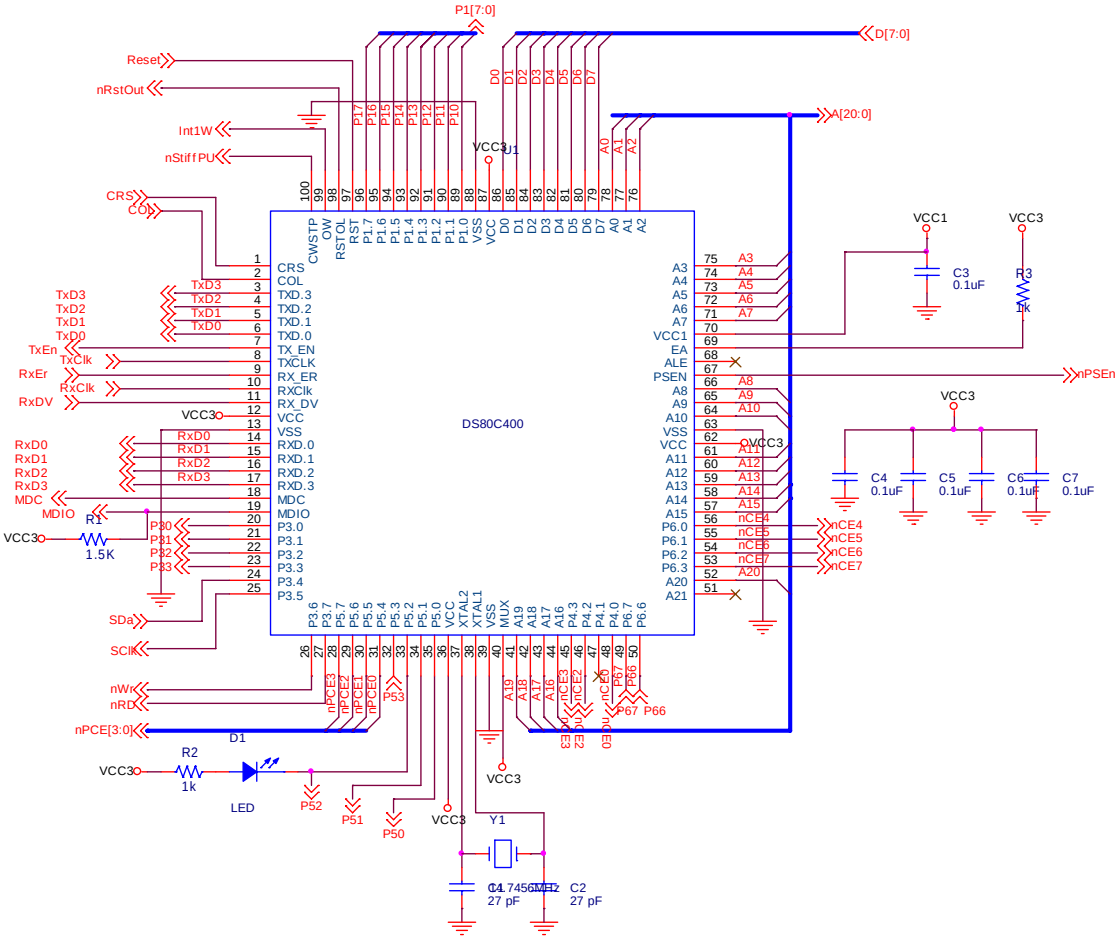


Figura A.1. Diagrama esquemático del microcontrolador.

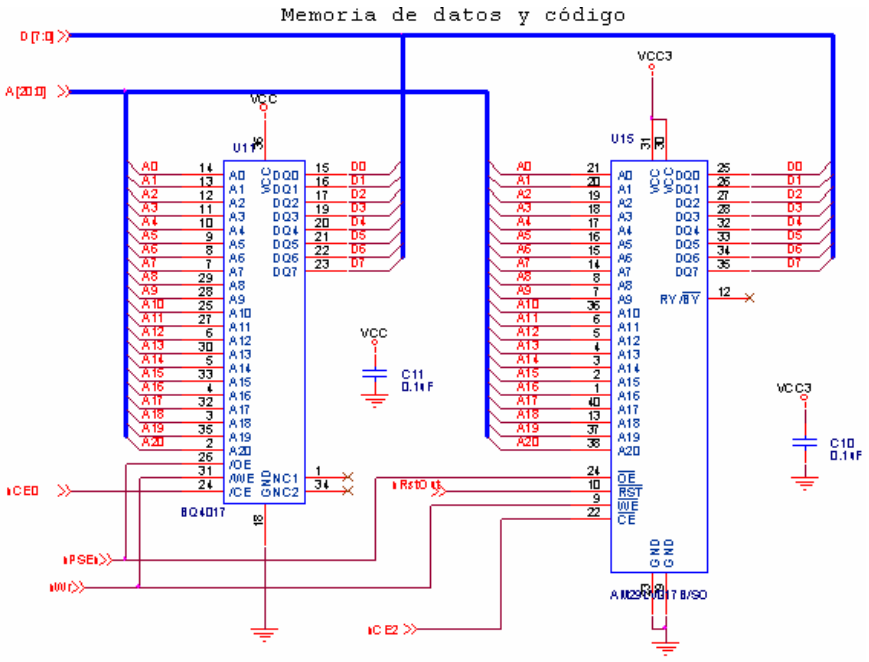


Figura A.2. Diagrama esquemático del bloque de memoria de datos y programa.

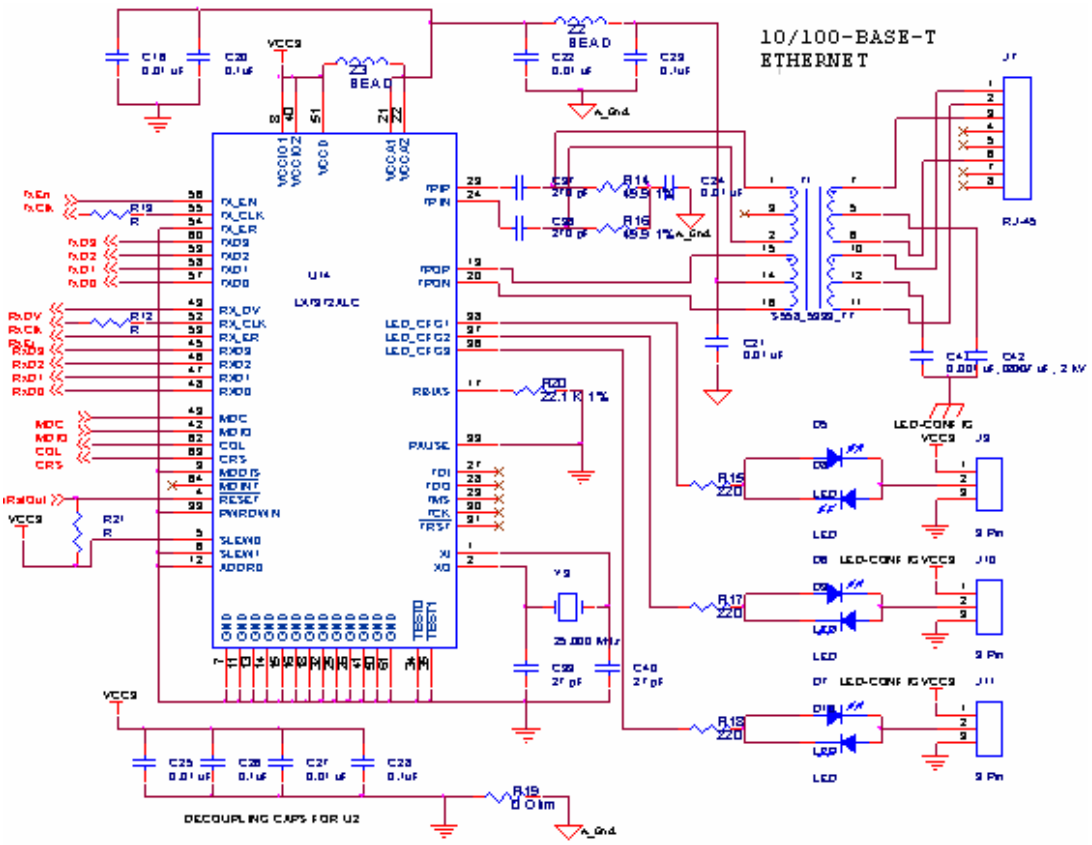


Figura A.3. Diagrama esquemático del bloque del decodificador a Ethernet.

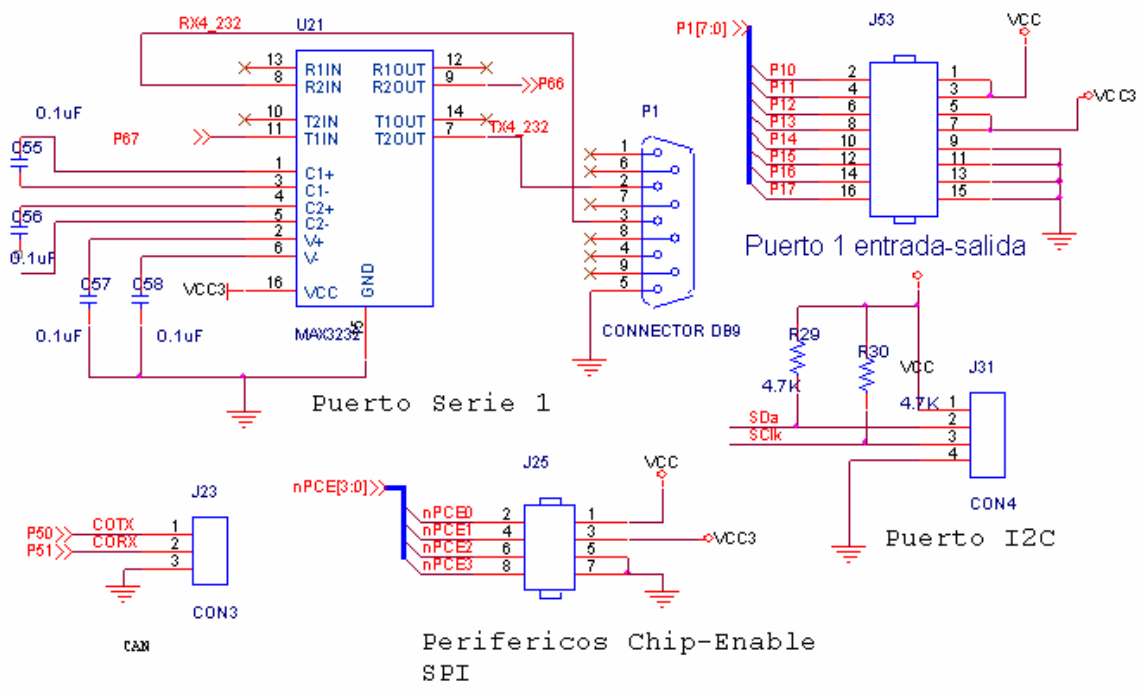


Figura A.4. Diagrama esquemático del bloque de interfaz a dispositivos de entrada/salida.

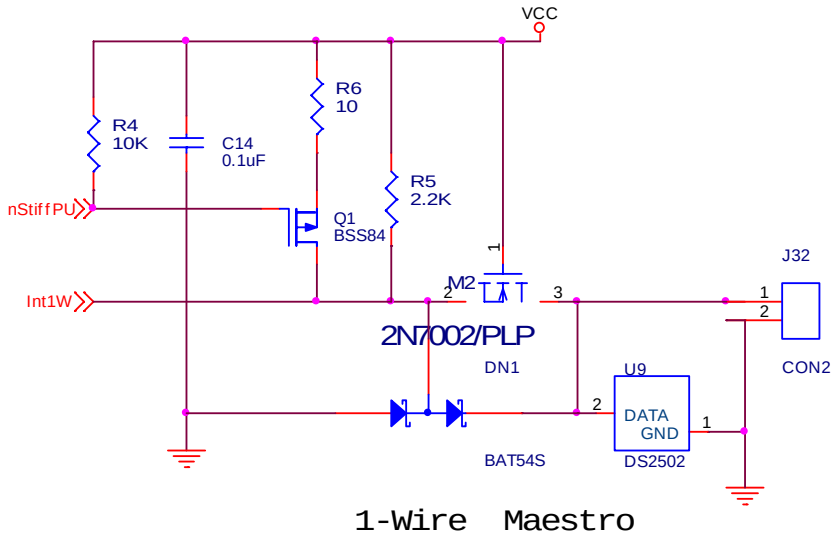


Figura A.5. Diagrama esquemático del bloque de interfaz a dispositivos de entrada/salida 1-Wire.

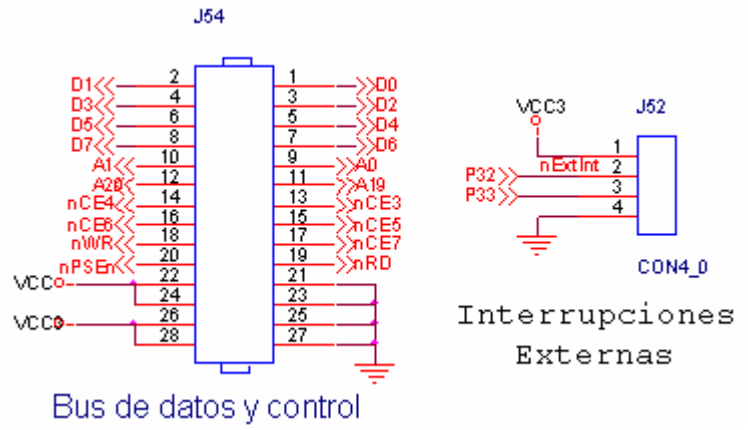


Figura A.6. Diagrama esquemático del bloque de buses de expansión.

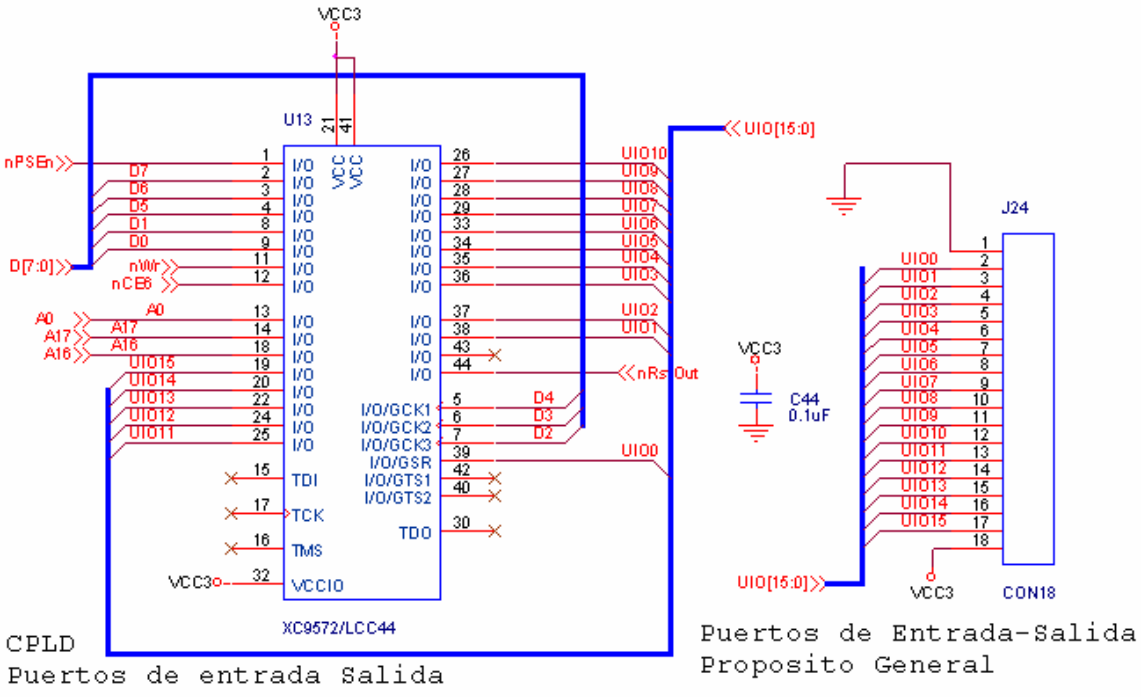
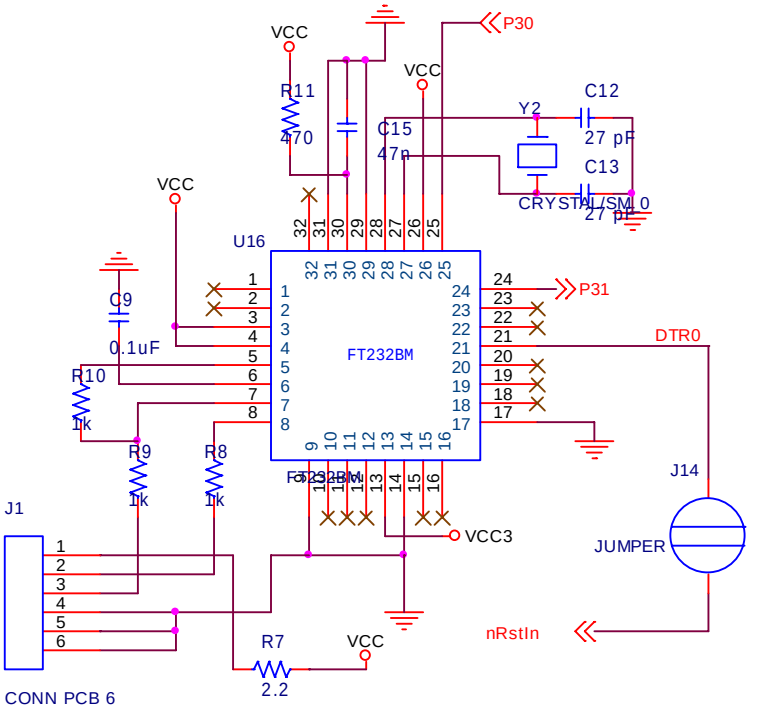
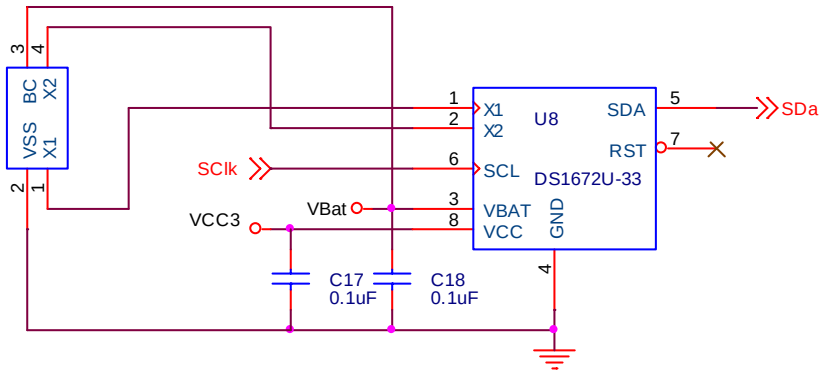


Figura A.7. Diagrama esquemático del bloque de buses de expansión a través de CPLD.



Puerto de Programacion

Figura A.8. Diagrama esquemático del bloque del convertidor serial a USB usado para programación.



Reloj Interno

Figura A.9. Diagrama esquemático del reloj interno del sistema.

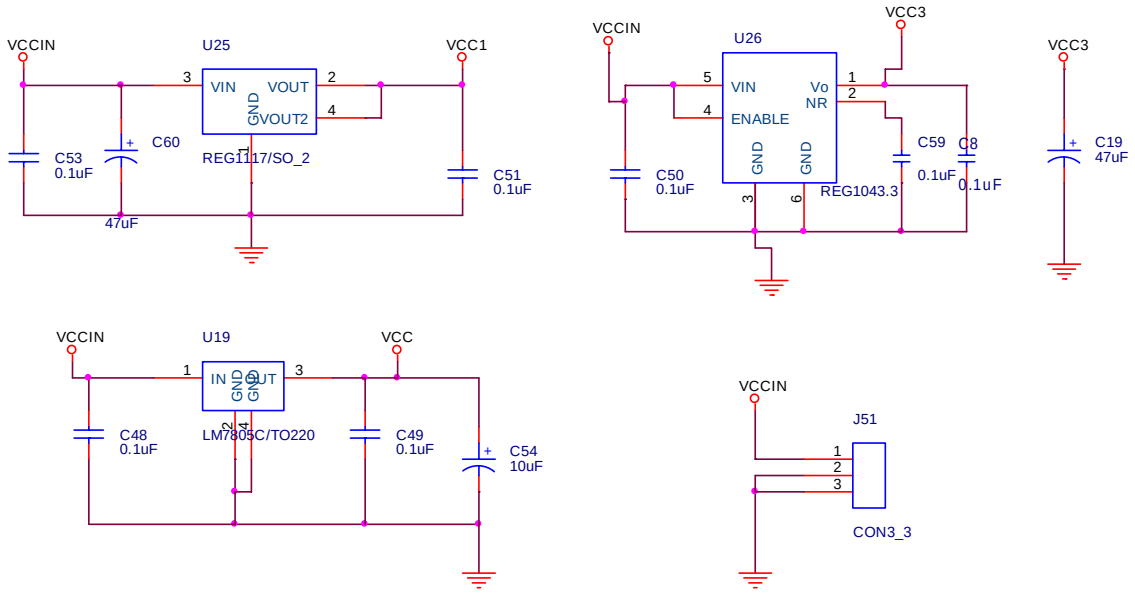


Figura A.10. Diagrama esquemático de la sección de alimentación del sistema.

ANEXO B. Prototipo del Sistema

La figura B.1 muestra una fotografía del sistema realizado.

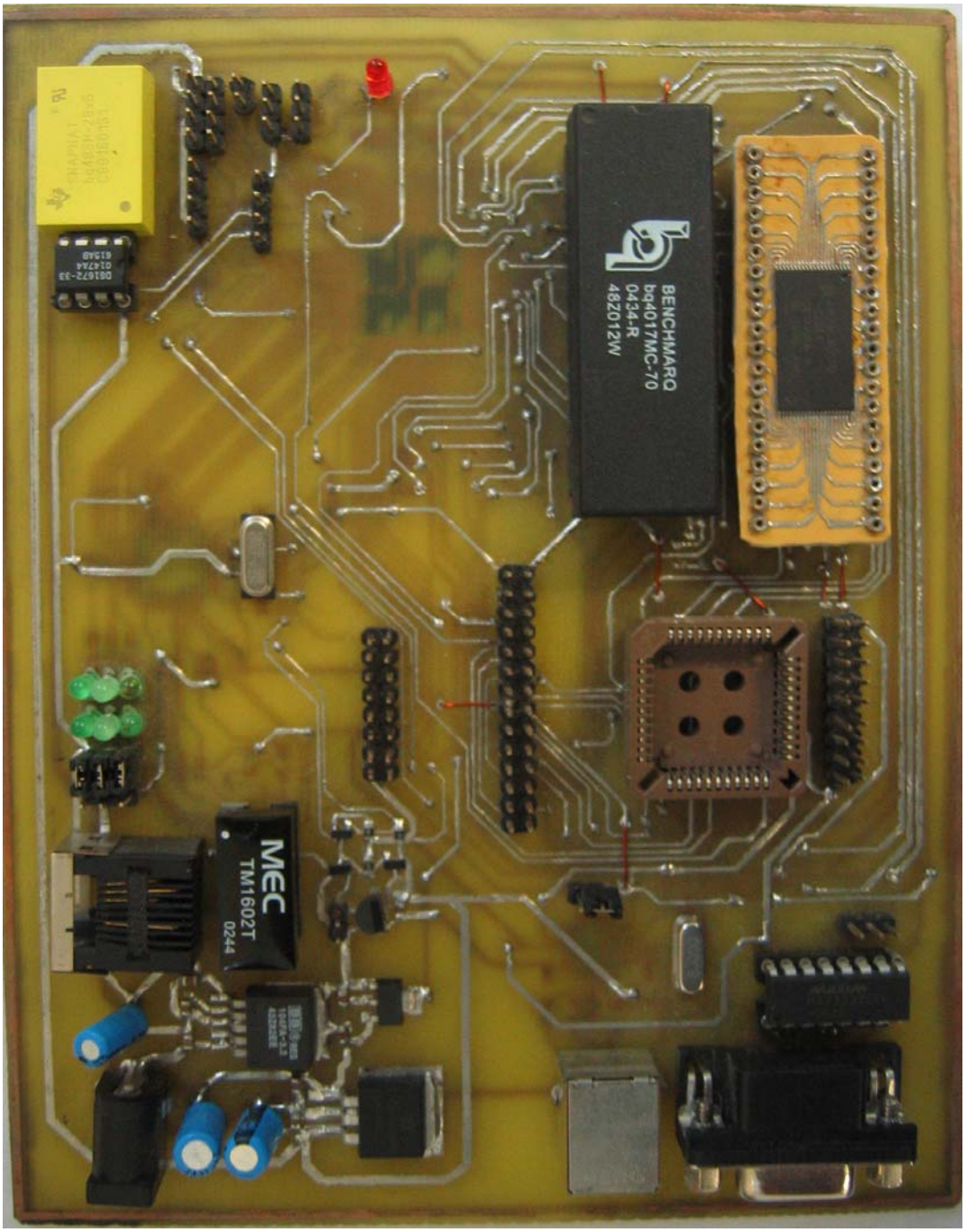


Figura B.1. El sistema realizado (vista superior).

En las figuras B.2 y B.3 se muestran las partes que componen al sistema identificadas con una etiqueta.

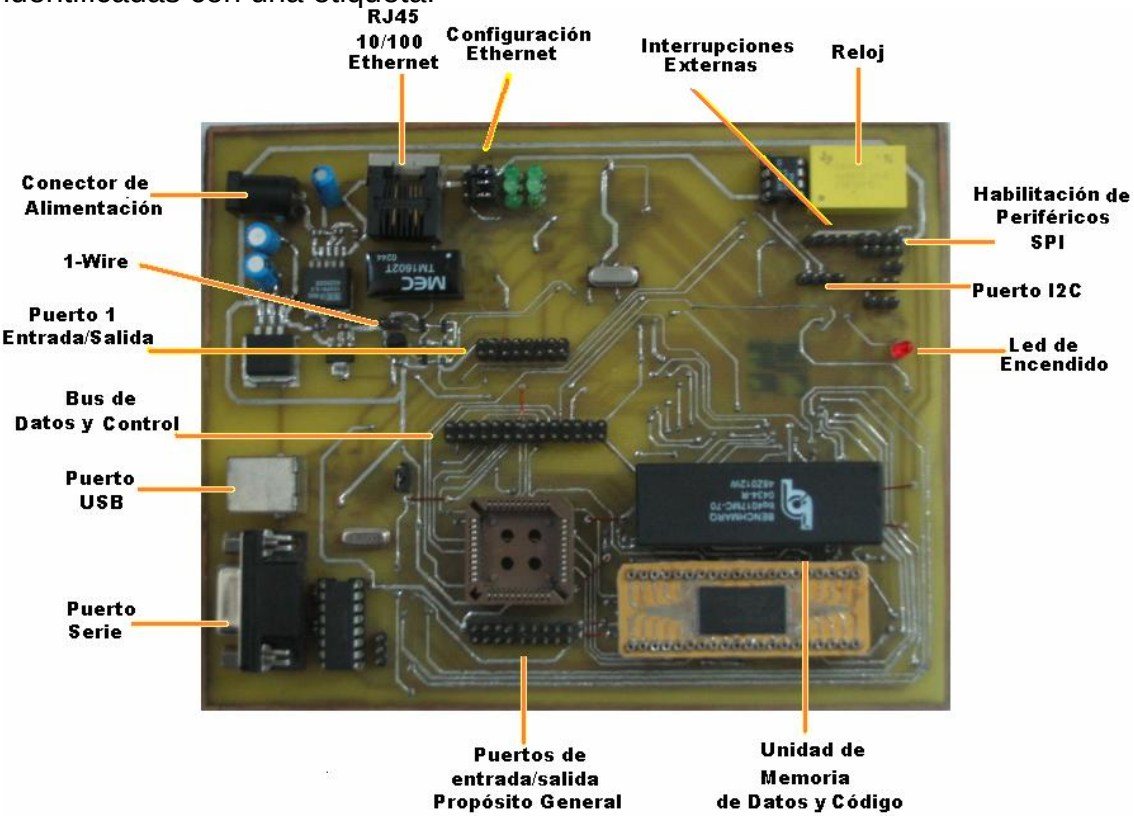


Figura B.2. Identificación de los bloques en el sistema (vista superior).

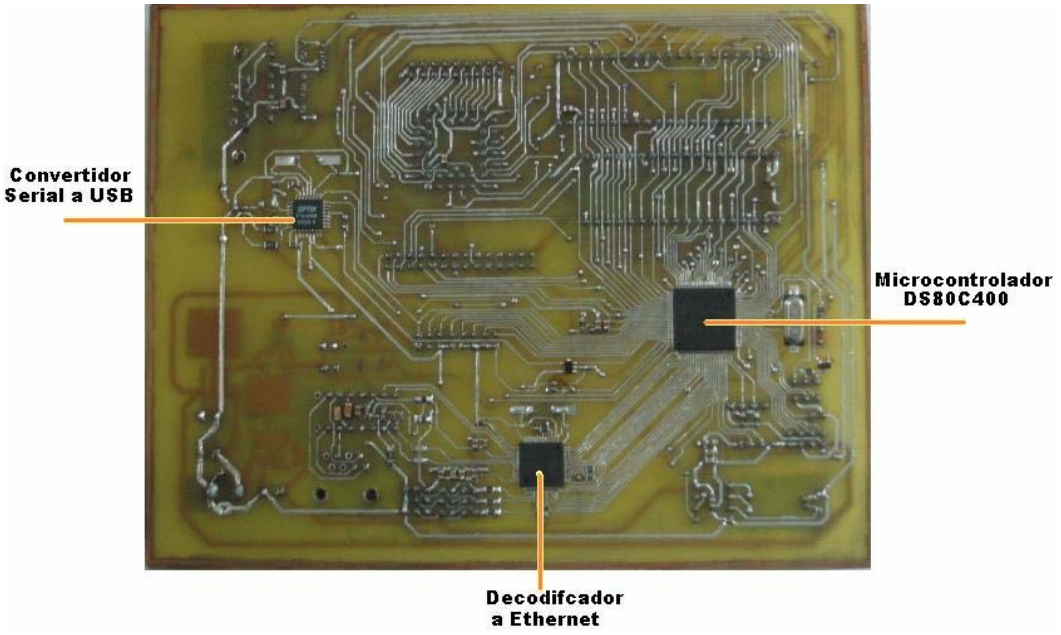
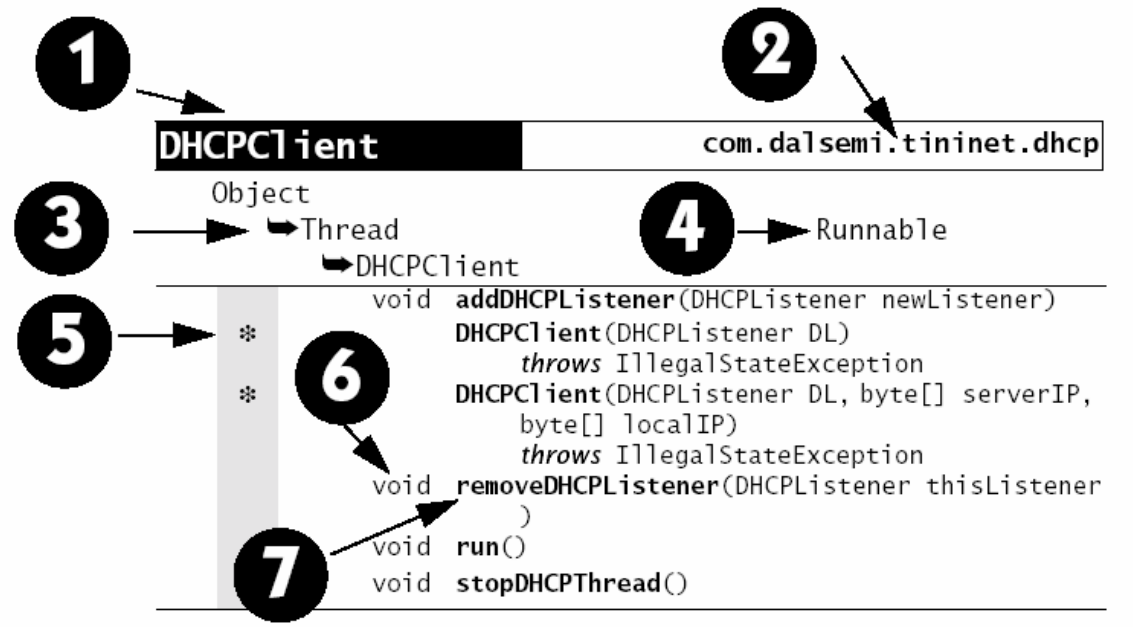


Figura B.3. Identificación de los bloques en el sistema (vista inferior).

ANEXO C. Documentación API de JAVA versión 1.4.2

Lo siguiente es un sumario condensado de las clases definidas en la API de TINI®, listadas alfabéticamente.



1.- El nombre de la clase
2.- El nombre del paquete que contiene la clase
3.- La cadena de superclases. Cada clase es una subclase de la que le precede.
4.- Los nombres de las interfaces implementadas por cada clase.
5.- Un constructor: Otros iconos que pueden aparecer en esta columna.

- ○ clase abstracta
- ● final
- □ estático
- ■ estático final
- © protegido
- ✍ campo

6.- El tipo de variable que regresa un método o el tipo de declaración de una variable.
7.- El nombre de la clase miembro. Si es un método, la lista de parámetros y la cláusula opcional. Los miembros se ordenan alfabéticamente.

Esta clase tiene utilidades para traducir y verificar la dirección de red 1-Wire.

Address	com.dalsemi.onewire.utils
Object	
↳ **Address**	
// Comprueba la CRC8 calculada de la dirección de red 1-Wire.	
⇨ boolean	**isValid**(byte[] address)
// Comprueba la CRC8 calculada de la dirección de red 1-Wire.	
⇨ boolean	**isValid**(long address)
// Comprueba la CRC8 calculada de la dirección de red 1-Wire.	
⇨ boolean	**isValid**(String address)
// Convierte una dirección endian de iButton o de dispositivo 1-Wire en un arreglo de bytes.	
⇨ byte[]	**toByteArray**(long address)
// Convierte una dirección cadena (big endian) 1-Wire a un arreglo byte (little endian).	
⇨ byte[]	**toByteArray**(String address)
// Convierte una dirección de red 1-Wire a un arreglo byte (little endian).	
⇨ long	**toLong**(byte[] address)
// Convierte una dirección de red 1-Wire a un arreglo byte (little endian).	
⇨ long	**toString**(byte[] address)
// Convierte una dirección de arreglo byte (little endian) a una cadena hexadecimal (big endian).	
⇨ String	**toString**(byte[] address)
// Convierte una dirección tipo long (little endian) a una cadena hexadecimal (big endian).	
⇨ String	**toString**(long address)

Esta clase permite leer y escribir bytes a dispositivos de memoria mapeados.

DataPort	com.dalsemi.system
Object	
↳ **DataPort**	
// Inicializa la dirección de memoria para mapear el acceso IO.	
int	**address**
// Crea un objeto DataPort con dirección 0x000000.	
□	**DataPort()**
// Crea un objeto DataPort con la dirección especificada.	
□	**DataPort**(int address)
// Obtiene la dirección de inicio para lectura/escritura.	
int	**get address** ()
// Recobra el modo FIFO	
boolean	**getFIFOMode**()
// Obtiene el número de ciclos de memoria a usar en el acceso a memoria	
int	**getStretchCycles**()
// Lee un solo byte de la dirección.	
int	**read()**
// Escribe un solo byte de la dirección.	
void	**write**(int value)
// Pone la direccion de inicio para lectura/escritura	
void	**setAddress**( )
// Se usa para poner el número de ciclos del bus de memoria a 2,3,4,...12.	
□ byte	**STRETCH0 a STRETCH10**

BitPort es una clase que permite la manipulación bit de los pines de puerto disponibles, los pines de puerto del controlador Ethernet y provee de métodos abstractos de manipulación de bits en direcciones de memoria mapeada a través de la clase DataPort.

BitPort	com.dalsemi.system
Object	
↳ **BitPort**	
	// Crea una nueva instancia de BitPort usando el bit specificado.
□	**BitPort**(byte bitname)
	// Crea una nueva instancia de BitPort usando el objeto DataPort specificado.
□	**BitPort**(DataPort port)
	// Limpia el pin de puerto en bajo
	void **clear**( )
	// Limpia el pin en bajo especificado
	void **clear**(int bitpos)
	// Lee el pin de puerto.
	int **read()**
	// Lee el pin de puerto especificado.
	int **readBit(int bitpos)**
	// Lee el valor actual de latch.
	int **readLatch()**
	// Lee el valor actual de latch del pin especificado.
	int **readLatch**(int bitpos)
	// Pone el pin en alto
	void **set**( )
	// Pone el bit especificado en alto
	void **set**(int bitpos)
	// SMC EEDO, salida solamente, pin 4.
□	byte **ETH_EEDO**()
	// SMC EESK, salida solamente, pin 7.
□	byte **ETH_EESK**()
	// P3.2, /INT0, /SMCINT (Ethernet interrupt), pin 6
□	byte **Port3Bit2**
	// P3.3, /INT1, /EXTINT (External interrupt), pin 7.
□	byte **Port3Bit3**
	// P5.2, C1RX, RXD1, DS2480 RX, "serial 1", pin 19
□	byte **Port5Bit2**

La clase HTTPServer implementa un servidor HTTP. A través de sus utilidades permite el acceso de peticiones HTTP, proporcionarle servicios, sincronizar los accesos y es útil si el servidor actualiza dinámicamente una página.

HTTPServer	com.dalsemi.tininet.http

Object
↳ HTTPServer

// Número de puerto por default (puerto 80).	
int	**DEFAULT_HTTP_PORT**
// Borra la petición.	
int	**DELETE**
// Obtiene la petición.	
int	**GET**
// Respuesta HTTP_BAD_REQUEST estándar.	
int	**HTTP_BAD_REQUEST**
// Respuesta HTTP_CREATED estándar.	
int	**HTTP_CREATED**
// Constructor por default.	
	HTTPServer()
// Crea un HTTPServer usando el puerto httpPort.	
	HTTPServer(int httpPort)
// Crea un HTTPServer usando el puerto httpPort.	
	HTTPServer(int httpPort, boolean logEnabled)
// Regresa la ruta HTTP del servidor.	
String	**getHTTPRoot**()
// Regresa la pagina índice por default del servidor.	
String	**getIndexPage()**
// Obtiene el número de puerto del servidor actual.	
int	**getPortNumber()**
// Comprueba la entrada de petición del cliente HTTP y los servicios suportados.	
int	**serviceRequests()**

La implementación del TINIShell. A través de sus utilidades permite una interacción con los usuarios asignando privilegios para operar con el sistema.

DefaultTINIShell — com.dalsemi.shell

Object
↳ TINIShell
↳ **DefaultTINIShell**

// Crea el ambiente inicial para un shell.	**DefaultTINIShell()**
// Ejecuta un comando en el Shell	void **execute(**java.lang.Object[] commandLine, SystemInputStream in, SystemPrintStream out, SystemPrintStream err, java.util.Hashtable env)
// Regresa una copia del ambiente actual.	java.util.Hashtable **getCurrentEnvironment**()
// Regresa la identificacion del usuario actual.	byte **getCurrentUID**()
// Regresa el nombre del usuario enlazado.	lang.String **getCurrentUserName**()
// Regresa el nombre del shell.	lang.String **getName**()
// Regresa la versión del Shell.	String **getVersion**()
// Enlaza un usuario al sistema y le asigna privilegios del nivel apropiado.	int **login(**java.lang.String userName, java.lang.String password)

A continuación se muestran más clases que están integradas en el microcontrolador y que se utilizan implícitamente para la comunicación con Internet, estas clases son parte de la familia de protocolos TCP/IP.

DHCPClient soporta la obtención dinámica de una nueva dirección IP o recupera una dirección IP perdida de un reinicio de servidor DHCP.

DHCPClient — com.dalsemi.tininet.dhcp

Object
↳ Thread — Runnable
↳ **DHCPClient**

	void	**addDHCPListener**(DHCPListener newListener)
❄	public	**DHCPClient**(DHCPListener DL) throws IllegalStateException
❄	public	**DHCPClient**(DHCPListener DL,byte[] serverIP, byte[] localIP) throws IllegalStateException
	void	**removeDHCPListener**(DHCPListener thisListener)
	void	**run( )**
	void	**stopDHCPThread( )**

Esta clase acepta los mensajes de estado del servidor DHCP.

DHCPListener — com.dalsemi.tininet.dhcp

Object
↳ **DHCPListener**

	void	**ipError**(String error)
	void	**ipLeased**(DHCPListener DL)
	void	**ipLost( )**
	void	**ipRenewed( )**

La clase DNSClient permite obtener un DNS.

DNSClient — com.dalsemi.tininet.dns

Object
↳ **DNSClient**

❄		**DNSClient( )**
	String[]	**getByIP**(byte[] ip)
	String[]	**getByIP**(String ip)
	String[]	**getByName**(String name)
	void	**setDNSTimeout**(int timeout)
	void	**setPrimaryDNS**(String dns1)
	void	**setSecondaryDNS**(String dns2)

Un servidor simple que usa FTP. Este servidor usa un ServerSocket para escuchar sobre un puerto específico (por defecto el puerto 21) las peticiones de conexión FTP. Para cada conexión se crea una sesión FTP.

FTPServer — com.dalsemi.shell.server.ftp

Object
↳ Thread — Runnable
↳ com.dalsemi.shell.server.Server
↳ **FTPServer**

	void	**broadcast**(String sendThis)
❄		**FTPServer()** throws java.io.IOException
		FTPServer(int port) throws IOException
	String	**getConnectionMsgFile**( )
	String	**getWelcomeMsgFile()**
	boolean	**isAnonymousAllowed()**
	boolean	**isRootAllowed()**
	String	**logAnon()**

Esta clase encapsula toda la funcionalidad de una sesión serial. El servidor serial escucha las peticiones de conexión al puerto serial. La sesión permite entrar al sistema, entonces maneja la comunicación entre el usuario y el sistema TINI®.

SerialSession — com.dalsemi.shell.server.serial

Object
↳ Thread — Runnable
↳ com.dalsemi.shell.server.Session
↳ **SerialSession**

	String	**getNextCommand**() throws java.io.IOException
	void	**login()** throws java.io.IOException
	void	**updatePrompt**(String withThis)

Un servidor sencillo que usa el protocolo TELNET. Usa un ServerSocket para escuchar las peticiones de conexión TELNET. Todos los comandos de proceso los maneja la sesión TELNET creada, no el servidor.

TelnetServer — com.dalsemi.shell.server.telnet

Object
↳ Thread — Runnable
↳ com.dalsemi.shell.server.Server
↳ **TelnetServer**

❑	String	**getWelcomeFile()**
❑	boolean	**isRootAllowed()**
❄		**TelnetServer()** throws java.io.IOException
❄		**TelnetServer**(int port) throws java.io.IOException

Esta clase condensada encapsula toda la funcionalidad de un intérprete de comandos (*shell*) para TINI®. Interactúa entre un proceso y el sistema operativo, y proporciona el ambiente actual y la identificación de usuario del proceso actual.

TINIShell com.dalsemi.shell

Object
↳ **TINIShell**

	byte	**adminUID()**
	void	**execute**(java.lang.Object[] commandLine, server.SystemInputStream in, server.SystemPrintStream out, server.SystemPrintStream err, java.util.Hashtable env) throws.Exception
	java.util.Hashtable	**getCurrentEnvironment()**
	byte	**getCurrentUID()**
	String	**getCurrentUserName()**
	String	**getFromCurrentEnvironment**(String key)
	String	**getName()**
	java.util.Hashtable	**getSystemEnvironment()**
	int	**getUIDByUserName**(String username)
	String	**getUserNameByUID**(byte uid)
	String	**getVersion()**
	boolean	**isAdmin**(byte uid)
	boolean	**isCurrentUserAdmin()**
	int	**login**(String userName, String password)
	void	**logout**(java.lang.Object info)
		TINIShell( )

ANEXO D. Entrada/Salida Paralelo

La Tabla D.1 muestra las señales de control del bus de datos y direcciones del microcontrolador, con las que se pueden conectar directamente un dispositivo externo. Se agrupan en señales de Datos, Dirección y Control.

Tabla D.1. Señales de control de bus.

Identificación de Señal	Nombre de Señal	Descripción
D0 – D7	Bus de Datos	Bus de datos bidireccional ancho de 8 bits
A0 – A19	Bus de Direcciones	Bus de Direcciones de 20 bits de ancho
CE0 – CE3	Habilitación de Chip	Líneas de habilitación de chip usadas para seleccionar memoria o añadir periféricos.
PCE0 – PCE3	Habilitación de Chip Periférico	Líneas de habilitación de chips periféricos comúnmente para habilitar memorias para almacén de datos.
PSEN	Habilitación de almacén de programa	Línea de control de programa (lectura) de dispositivos externos de memoria habilitados por las líneas CE.
RD	Lectura	Línea de lectura para datos de dispositivos de memoria u otro periférico habilitado por las líneas PCE.
WR	Escritura	Usada para escribir datos de memoria a otros dispositivos periféricos.
DRST	Reset de Dispositivo	Terminal 3.4 del microcontrolador. No es una señal formal definida en el bus paralelo. Se usa para el reinicio de dispositivos externos.

La Tabla D.2 lista todas las terminales de puerto que pueden ser controladas directamente por una aplicación Java con su uso típico.

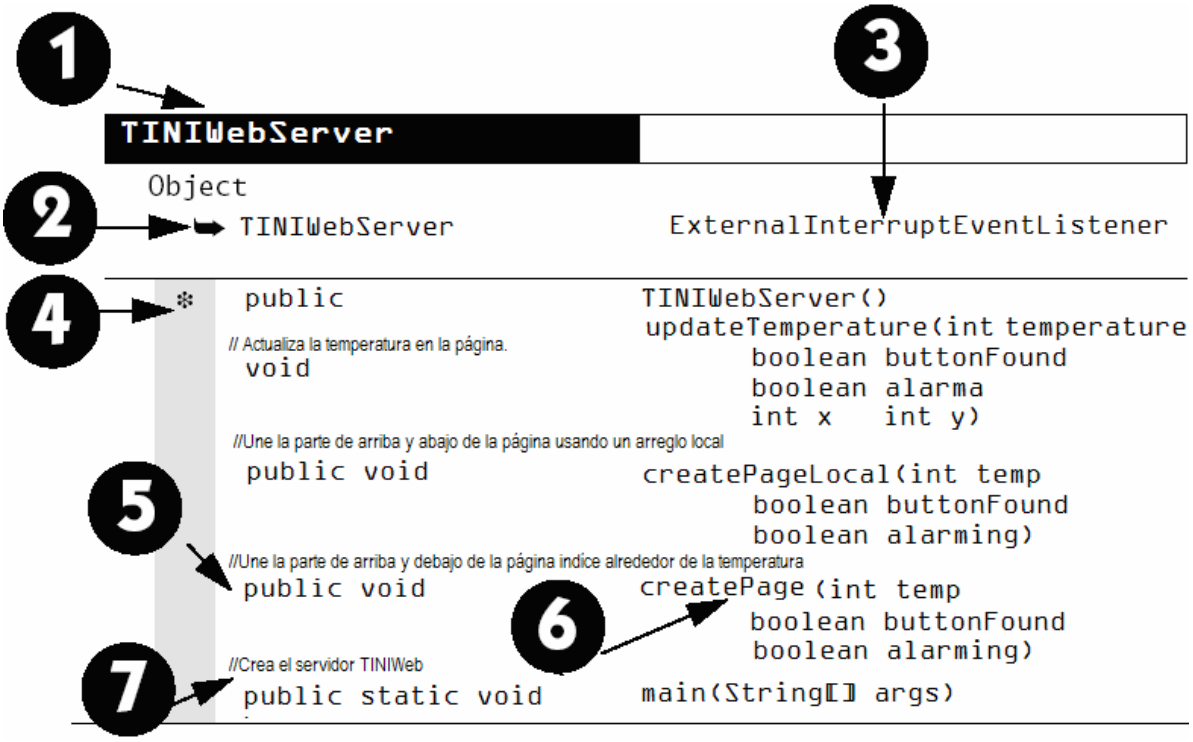
Tabla D. 2. Terminales de puerto accesibles por JAVA.

Número/Nombre del pin de Microcontrolador	Uso por defecto.
4 (P3.0)	Recepción de datos por serial0
5 (P3.1)	Transmisión de datos por serial0
6 (P3.2)	Interrupción de Controlador Ethernet
7 (P3.3)	Interrupción externa de propósito general
10 (P3.4)	Reset de dispositivo externo
11 (P3.5)	Red Interna 1 Wire
17 (P5.4)	Habilitación de Chip 0
16 (P5.5)	Habilitación de Chip 1
15 (P5.6)	Habilitación de Chip 2
14 (P5.7)	Habilitación de Chip 3

Considerando las cuatro terminales altas del puerto 5 (p5.4-5.7). Su uso normal es de decodificación de hardware interconectado al bus del controlador. Si un sistema no utiliza las señales PCE para propósitos de decodificación lógica, entonces estas terminales se pueden utilizar como terminales de puerto de propósito general. Si el sistema usa una de las señales para habilitar un dispositivo entonces ninguna de las restantes se puede usar como terminal de propósito general.

ANEXO E. Clases creadas para la aplicación

Estas son las clases realizadas de la aplicación realizada para la prueba del sistema.



1.- El nombre de la clase
2.- La cadena de superclases. Cada clase es una subclase de la que le precede.
3.- Los nombres de las interfaces implementadas por cada clase.
4.- Un constructor: Otros iconos que pueden aparecer en esta columna.

- ○ abstracto
- ● final
- □ estático
- ■ estático final
- © protegido
- ✍ campo

5.- El tipo de variable que regresa un método o el tipo de declaración de una variable.
6.- El nombre de la clase miembro. Si es un método, la lista de parámetros y la cláusula opcional.
7.- Descripción de la función que realiza la clase miembro.

TINIWebServer implementa un servidor web usando la clase HTTPServer. Escucha las peticiones para iniciar servidores. Revisa y actualiza el valor de la temperatura medida en la página.

TINIWebServer		
Object		
↳ **TINIWebServer**		ExternalInterruptEventListener
❄ public	TINIWebServer( )	
// Actualiza la temperatura en la página.		
void	updateTemperature(int temperature boolean buttonFound boolean alarma int x int y)	
//Une la parte de arriba y abajo de la página usando un arreglo local		
public void	createPageLocal(int temp boolean buttonFound boolean alarming)	
//Une la parte de arriba y debajo de la página indíce alrededor de la temperatura		
public void	createPage(int temp boolean buttonFound boolean alarming)	
//Inicio de varios servidores.Checa la temperatura y actualiza la página		
public void	drive( )	
//Crea el servidor TINIWeb		
public static void	main(String[] args)	

Esta clase ejecuta un servidor web.

WebWorker		
Object		
↳ **WebWorker**		Runnable
❄ public	WebWorker(Object lock)	
//Obtiene el directorio raíz del servidor web		
String	getWebRoot( )	
//Obtiene la página por defecto del servidor web		
String	getWebPage( )	
// Ejecuta el hilo del servidor		
void	run( )	

La clase TemperaturaWorker se ocupa de la comunicación con el dispositivo 1-Wire.

TemperatureWorker	
Object	
↳ TemperatureWorker	Runnable
❄ public	TemperatureWorker( )
//Regresa la temperatura actual	
public int	getCurrentTemperature( )
//Regresa el estado de alarma	
public boolean	alarming( )
//Pone el estado de alarma alta	
public void	setalarmhigh(int alarmHigh)
//Pone el estado de alarma baja	
public void	setalarmlow(int alarmLow)
//Regresa la alarma alta	
public int	getalarmhigh( )
//Regresa la alarma baja	
public int	getalarmlow( )
//Reporta presencia de diapositivo	
public boolean	buttonFound( )
//Ejecuta el hilo del servidor de temperatura	
public void	run( )

LCD es una clase con la función de inicializar al dispositivo LCD y permitir la escritura de texto conectada al sistema.

LCD	
Object	
↳ LCD	
❄ public	LCD( )
//Parametros de LCD lineas y caracteres por línea	
public	LCD(int lines int width)
//Parametros de LCD lineas,caracteres por línea,dirección espacio	
public	LCD(int lines int width int adress)
//Inicializa el LCD	
private void	LCDinit(int adress)
//Escribe una cadena en el LCD	
public void	write(string txt)
//Coloca el cursor en la posicion dada	
public void	positionCursor(int x int y)
//Mueve el cursor a la posición inicial	
public void	home( )
//Limpia la pantalla LCD	
public void	clear( )

ANEXO F. Inicialización del Sistema

INICIANDO CON EL SISTEMA

Este anexo es de guía para el proceso de configuración inicial del sistema mínimo desarrollado. Siguiendo las instrucciones siguientes, el usuario deberá establecer comunicación con el sistema y cargar el firmware. A continuación se incluye una lista de los materiales y software requeridos.

Requerimientos de Hardware

- El sistema mínimo basado en el microcontrolador DS80C400.
- Computadora Personal con Sistema Operativo que reúna los requerimientos del JDK de Sun Microsystems y con puerto RS232 o USB.
- Cable serial o USB.
- Cable Ethernet cruzado.
- Fuente de alimentación de 5V DC.

Requerimientos de Software

- JDK (Java Development Kit) de Sun Microsystems versión 1.4.2.
  Disponible en: http://www.java.sun.com/javase/index.jsp o ftp.dalsemi.com/pub/tini/index.html.
- Java Communications API de Sun Microsystems.
  Disponible en: http://java.sun.com/products/javacomm/index.jsp.
- TINI Software Development Kit Version 1.1 o posterior.
  Disponible en: http://www.ibutton.com/TINI/software/index.html.

Proceso de Inicialización

1. Instalando el JDK

El JDK es un programa ejecutable, se puede instalar en cualquier ubicación, la cual se referirá como *JAVA_HOME*. Una vez que la instalación se ha completado se puede ver la siguiente estructura de archivos.

```
03/03/2007  10:29 PM    <DIR>          bin
02/20/2007  07:00 AM             4,519 COPYRIGHT
03/03/2007  10:29 PM    <DIR>          demo
03/03/2007  10:29 PM    <DIR>          include
03/03/2007  10:29 PM    <DIR>          jre
03/03/2007  10:29 PM    <DIR>          lib
03/03/2007  10:29 PM            17,932 LICENSE
03/03/2007  10:29 PM            28,905 LICENSE.rtf
```

```
03/03/2007  10:29 PM            16,703 README.html
03/03/2007  10:29 PM             9,215 README.txt
02/20/2007  07:00 AM        11,849,754 src.zip
03/03/2007  10:29 PM    10,367 THIRDPARTYLICENSEREADME.txt
```

NOTA: Las fechas mostradas en el listado son las correspondientes a la fecha de instalación del JDK. Los nombres de archivo y tamaño mostrados son para la versión JDK 1.4.2.

2. Instalando las Comunicaciones API de Java

1. Descomprimir el archivo en cualquier directorio, el cual será consecuentemente referido como *COMMAPI*. En este directorio se encontrarán los siguientes archivos:

```
03/03/2007  03:59 PM             3,335 apichanges.html
03/03/2007  04:00 PM            28,043 comm.jar
03/03/2007  03:59 PM             8,141 COMM2.0_license.txt
03/03/2007  03:59 PM             5,374 CommAPI_FAQ.txt
02/20/2007  11:44 AM    <DIR>          javadocs
03/03/2007  03:59 PM               467 javax.comm.properties
03/03/2007  03:59 PM             2,182 jdk1.2.html
03/03/2007  03:59 PM             3,715 PlatformSpecific.html
03/03/2007  03:59 PM             3,913 Readme.html
03/03/2007  03:59 PM             1,821 ReceiveBehavior.html
02/20/2007  11:44 AM    <DIR>          samples
03/03/2007  04:00 PM            27,648 win32com.dll
```

2. Copiar win32com.dll de COMMAPI al directorio % JRE_HOME%\jre\bin.
3. Copiar comm.jar de COMMAPI al directorio % JRE_HOME %\jre\lib\ext.
4. Copiar javax.comm.properties de COMMAPI a %JRE_HOME%\jre\lib.
5. Ir a TINI_HOME y crear un archivo .BAT con la siguiente línea (descripción sensible a mayúsculas y minúsculas):

```
%JAVA_HOME%\bin\java -classpath %TINI_HOME%\bin\tini.jar JavaKit
```

6. Dar doble clic al archivo batch creado.

Esto pone a funcionar el JDK, el programa que se usa para interactuar con el sistema, como se muestra en la figura F.1. Se usará JDK frecuentemente, por eso el archivo batch ahorra tiempo. Si todo se instaló apropiadamente, se podrán ver los puertos COM que se tienen en la computadora.

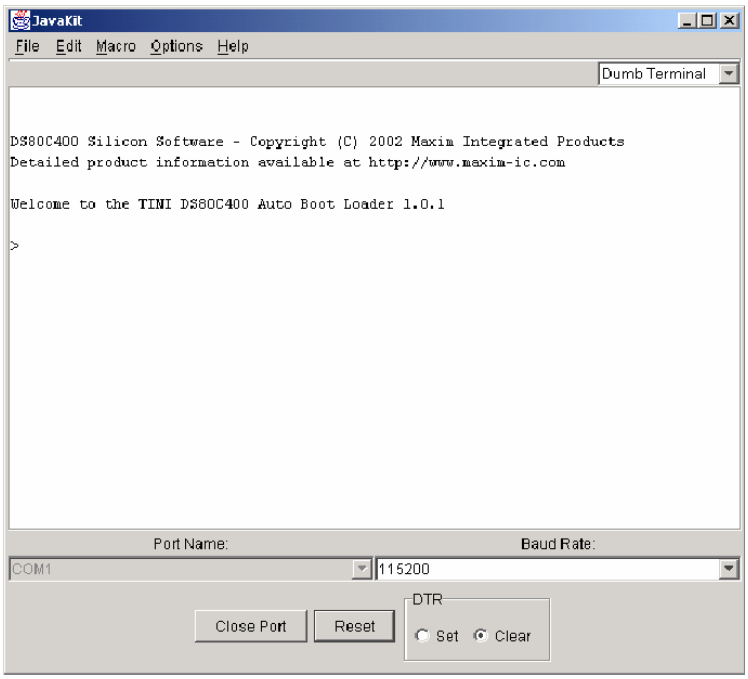


Figura F.1. Prompt del cargador JavaKit.

3. Conectando el sistema

1. Verificar que el sistema no esté alimentado.
2. Colocar el *jumper* DTR-Reset.
3. Conectar el cable USB a la tarjeta.
4. Conectar el cable cruzado Ethernet al puerto Ethernet de la computadora, para enlazar la conexión entre la computadora y la tarjeta del sistema.
5. Conectar el adaptador de alimentación a la tarjeta del sistema.
6. Conectar el adaptador de alimentación a la toma de alimentación.

El resultado de estos pasos se muestra en la figura F.2.

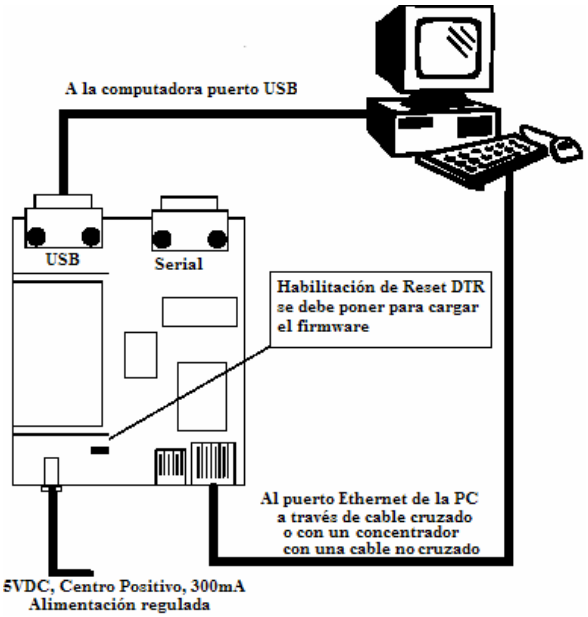


Figura F.2. Instalación típica del hardware.

4. Instalando el software TINI

Descomprimir el paquete en cualquier directorio, éste se referirá en adelante como TINI_HOME. En este directorio se deben encontrar estos archivos:

```
03/03/2007  04:35 PM    <DIR>          bin
03/03/2007  04:35 PM    <DIR>          docs
03/03/2007  04:35 PM    <DIR>          examples
05/15/2003  09:15 AM             3,413 index.html
05/15/2003  09:15 AM            13,178 License.txt
03/03/2007  04:35 PM    <DIR>          native
05/15/2003  09:15 AM            18,050 README.txt
03/03/2007  04:35 PM    <DIR>          src
               3 File(s)         34,641 bytes
```

5. Cargando el Firmware

Hay tres cosas que se necesitan para descargar y ejecutar al sistema. Lo primero es el firmware TINI, que está en el archivo `%TINI-HOME%\bin\tini.hex`. Después la biblioteca de las clases Java, que se encuentra en `%TINI-HOME%\bin\tiniapi.hex`. Finalmente, se necesita la línea de comandos interactiva que se encuentra en `%TINI-HOME%\examples\slush\Slush.hex`.

1. En JDK, seleccionar el puerto COM a usar para la entrada/salida serial. Este es cualquier puerto COM para conectar el sistema a la computadora.
2. Asegurarse que la velocidad sea de 115200.
3. Presionar el botón OPEN PORT.
4. Cuando el botón de RESET se habilite, presionarlo. Esto envía una señal de arranque al sistema. Este procedimiento inicia el "cargador" que permite comunicarse con el sistema y empotrar el firmware.
5. En el área de texto del prompt deben aparecer las siguientes líneas:

```
TINI loader 05-15-00 17:45
Copyright (C) 2000 Dallas Semiconductor. All rights reserved.
>
```

6. Seleccionar en el menú FILE la opción LOAD. Cargar el siguiente archivo: `%TINI_HOME%\bin\tini.tbin`. El archivo incluye una descripción de los bancos de información, por lo que no es necesario especificar la dirección de descarga. Debe reportar si el (los) banco (os) se cargó (aron) correctamente, lo que tomará algunos segundos.
7. Limpiar el área con lo siguiente:

```
b18      //changes to bank 18
f0       //fills bank 18 with 0's, effectively erasing it
```

8. Seleccionar FILE/LOAD una vez más y cargar el siguiente archivo: `%TINI-HOME%\bin\slush.tbin`

9. Si todo se ha cargado, escribir 'e' para ejecutar. Se debe tener una salida similar a la que muestra la figura F.3.

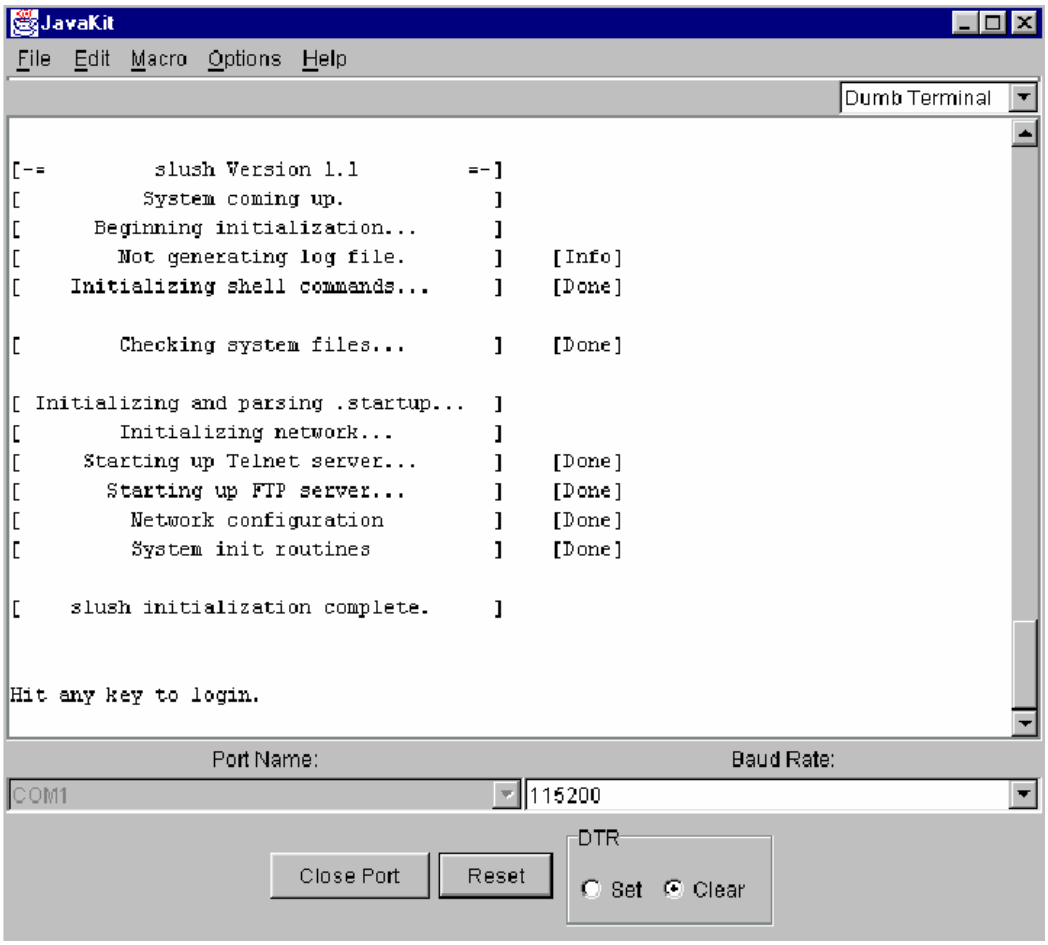


Figura F.3. Salida al cargar el firmware.

Después de presionar cualquier tecla se debe escribir el nombre de usuario y posteriormente la contraseña de entrada:

```
Welcome to slush.  (Version 1.01)

TINI login: root
TINI password:

TINI />
```

NOTA: Hay dos cuentas de usuario por defecto, '*guest*' con el password '*guest*' y '*root*' con password '*tini*'

10. Escribir "*help*" para obtener ayuda. También se puede realizar un comando específico. Por ejemplo: "*help ipconfig*".

11. Ahora se puede introducir la dirección IP usando el comando *ipconfig*. Se puede usar el DHCP setting para tener la IP dinámicamente o tener una IP fija. Por ejemplo:

```
ipconfig -a tu.IP.escrita.aquí \
         -m máscara.de red.escrita.aquí \
         -g la.gateway.escrita.aquí
```

    Nota: La opción -a se debe acompañar con la opción -m.

12. Si se pone la información de red correctamente usando ipconfig, se puede usar TELNET y FTP en el sistema. Ahora se puede probar con un ping al sistema de una PC de subred para confirmar la configuración.

NOTA: Una vez que se ha ingresado al sistema, todos los comandos son sensibles a mayúsculas y minúsculas.

6. ¡Listo!

¡Se ha completado! El sistema está listo para funcionar.

ANEXO G. Creación de aplicaciones TINI®

Este anexo es de ayuda para la realización de aplicaciones, como se carga y se ejecuta en el sistema. Al igual que los lenguajes de programación para aprender se usa al aceptado programa “Hola Mundo”.

Se usa la línea de comando, a través de una sesión TELNET, para correr la aplicación, desplegar la salida, interactuar con los archivos de sistema y los procesos de control.

G.1 Hola Mundo

Aunque este ejemplo no resalta exactamente las características de Java, es adecuado para describir el proceso paso a paso del desarrollo de una aplicación. Típicamente, para el desarrollo y prueba de aplicaciones se requieren 5 pasos:

1. Crear el archivo fuente.
2. Compilar el archivo fuente.
3. Convertir a archivo de clase.
4. Cargar la imagen convertida.
5. Ejecutar la imagen convertida.

Para la creación del archivo fuente se puede usar cualquier editor de texto o algún bloc de notas para programadores, por supuesto cuidando guardar el archivo con extensión *.java*, los siguientes tres pasos se realizan a través de la línea de comandos de MS-DOS. Y para ejecutar la aplicación se debe ingresar al sistema y usar su línea de comandos.

Paso 1: Crear el archivo fuente. Crear y guardar un archivo nombrado HelloWorld.java conteniendo el código de la figura 3.5.

```
Class HelloWorld {
      Public static void main (String[] args)
      {
            System.out.println(“Hola Mundo”);
       }
}
```

Figura G.1. Archivo fuente.

Paso 2: Compilando el archivo fuente: Se compila HelloWorld.java a un archivo de clase usando el compilador de Java, cambiando el prompt al directorio que contiene el archivo recién creado y se ejecuta el siguiente comando:

```
javac HelloWorld.java
```

Si el compilado produce error, generalmente se debe a que las clases de TINI® no están en el mismo directorio, entonces se debe especificar a través del parámetro “-bootclasspath”

y así el compilador sabrá donde se ubican las librerías que se usan, como el siguiente comando:

```
javac – bootclasspath c:\tini1.12\bin\tiniclasses.jar HelloWorld.java
```

Si es satisfactorio, se tiene un nuevo archivo llamado HelloWorld.class en el directorio de trabajo actual.

Paso 3: Convirtiendo el archivo clase. Para este paso se usa la utilidad TINIConvertor para generar una salida binaria adecuada para la ejecución en TINI®. Esta aplicación está implícita en el TINI SDK y está en el archivo tini.jar y se ejecuta en la línea de comandos.

TINIConvertor requiere de los parámetros: archivo fuente o directorio (-f), base de datos API (-d) y archivo de salida (-o).

```
java -classpath c:\tini1.02\bin\tini.jar TINIConvertor -f HelloWorld.class -d
                c:\tini\tini1.02\bin\tini.db -o  HelloWorld.tini
```

En el ejemplo, la aplicación consiste solo de una clase, mientras que la mayoría de las aplicaciones consisten de varias clases en uno o más paquetes y para ejecutarlas hay que cambiar el nombre por el del directorio de la raíz de los paquetes.

TINIConvertor genera un archivo con el nombre dado con la opción –o en este caso: HelloWorld.tini.

Paso 4: Cargar la imagen convertida. Se usa el cliente FTP, provisto con el sistema operativo, para conectar a TINI® y transferir la imagen binaria al sistema de archivos de TINI®.

```
C:\tini1.02\HelloWorld>ftp 192.168.6.159
Connected to 192.168.6.159.
220 Welcome to slush. (Version 1.02) Ready for user login.
User (192.168.6.159:(none)): root      nombre del usuario
331 root login allowed. Password required.
Password:
230 User root logged in.
ftp>
```

Después de establecer la conexión y registrarse, se puede transferir HelloWorld.tini. Primero se escribe “bin” en el prompt de FTP para asegurar que la imagen binaria no se altere en la transferencia.

```
ftp> bin
200 Type set to Binary
```

Se transfiere usando el comando put.

```
ftp> put HelloWorld.tini
200 PORT Command successful.
150 BINARY connection open, putting HelloWorld.tini
226 Closing data connection.
ftp: 171 bytes sent in 0.00Seconds 171000.00Kbytes/sec.
```

Finalmente se cierra la sesión FTP escribiendo bye o quit en el prompt.

```
ftp> bye
221 Goodbye.
```

Es posible revisar que el archivo se transfirió satisfactoriamente usando el comando ls en el prompt de la sesión TELNET.

```
TINI /> ls –l
total 3
drwxr-x 1    root admin     2 Jan 28 14:45.
-rwxr-- 1    root admin    17 Jan 28 15:46   HelloWorld.tini
drwxr-x 1    root admin     3 Jan 28 14:45   etc
```

El archivo HelloWorld.tini ahora aparece en el directorio raíz del sistema de archivos.

Otra forma de hacer la transferencia sin usar línea de comandos, y que muchos desarrolladores prefieren, es con archivos creados con las instrucciones. Por ejemplo, con el Windows FTP, se crea un archivo con el contenido de la figura 3.6.

```
root
tini
bin
put HelloWorld.tini
```

Figura G.2. Load.cmd.

Si se le llama load.cmd se usa lo siguiente para transferir el archivo sin ninguna interacción con el prompt de comando de FTP.

```
C:\TINI\tinil.02\myapps\HelloWorld>ftp –s:load.cmd 192.168.6.159
```

Usando la opción -s el cliente FTP lee el archivo especificado y ejecuta cada línea como si se escribiera manualmente en el prompt.

Paso 5: Ejecutando la imagen convertida. Ahora sólo hay que ejecutar la aplicación usando el comando java en la línea de comandos dentro del sistema. Para ingresar al sistema se debe abrir una sesión TELNET con la dirección del sistema.

```
C:/>telnet 192.168.6.159
```

Después de ingresar el nombre de usuario y contraseña se tiene la línea de comandos del sistema y se puede ejecutar la aplicación.

```
TINI/>java HelloWorld.tini
Hola Mundo
TINI/>
```

HelloWorld.tini realizó y produjo la salida esperada. Al terminar el programa, el control de la sesión del usuario regresa a la línea de comandos.