ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

"IMPLEMENTACION DE TARJETA DE DESARROLLO UTILIZANDO EL MODULO TINI PLATFORM PARA EL LABORATORIO DE CONTROL EN LA EIE"

TESIS DE GRADO

PREVIA OBTENCIÓN DEL TÍTULO DE

INGENIERO EN ELECTRÓNICA Y COMPUTACIÓN

PRESENTADO POR:

CRISTINA JUDITH GUERRERO COELLO

DANIEL GUSTAVO MALDONADO HURTADO

RIOBAMBA - ECUADOR

2010

A Dios, por bendecirnos con la Vida.

A nuestros Padres por darnos la oportunidad de estudiar y brindarnos siempre su apoyo y amor incondicional que nos ha permitido llegar a alcanzar esta meta.

A nuestro tutor, profesores y amigos, por habernos guiado y ayudado de una u otra forma a culminar este trabajo.

Dedicado a mi Dios, por darme la oportunidad de vivir, por ser la fortaleza de mis días, por guiarme y darme decisión para seguir adelante a pesar de las adversidades.

A mi familia por su ayuda incondicional y por estar junto a mí en todo momento.

A Daniel Gustavo por ser un gran compañero de trabajo y amigo por mucho tiempo.

Y finalmente dedico este trabajo de todo corazón a mi Madre, la persona que más admiro en este mundo, por darme la vida, por ser mi compañera y amiga, por ser un ejemplo de vida, porque con su esfuerzo, sacrificio y amor me ha brindado su apoyo sincero durante toda mi carrera y mi vida entera. Los quiero y los llevare siempre en mi corazón.

Cristina

El presente trabajo de titulación se lo dedico a mi familia; a mi padre Guido, por entregarme la gran herencia del aprendizaje; a mi madre Teresita, por cobijarme con sus amorosos brazos, que siempre me dan fuerzas, tranquilidad y alegría para alcanzar mis metas; a mis hermanas Raquel y Nora, por apoyarme; a mis amigos Diego, Jorge y Juan, por brindarme su ayuda siempre que la necesitaba; y a mi compañera de Tesis Cristina, por permitirnos terminar este trabajo de graduación con su gran empeño y capacidad.

Daniel

NOMBRE	**FIRMA**	**FECHA**
Ing. Iván Menes C. **DECANO DE LA FACULTAD DE INFORMÁTICA Y ELECTRÓNICA**	**…….**	**……..**
Ing. José Guerra **DIRECTOR DE LA ESCUELA DE INGENIERÍA ELECTRÓNICA.**	**…….**	**……..**
Ing. Hugo Moreno **DIRECTOR DE TESIS**	**…….**	**……..**
Ing. Paúl Romero **MIEMBRO DEL TRIBUNAL**	**…….**	**……..**
Tlgo. Carlos Rodríguez **DIRECTOR DPTO. DOCUMENTACION**	**…….**	**……..**
NOTA DE LA TESIS	**…….**	

"Nosotros, **CRISTINA JUDITH GUERRERO COELLO** y **DANIEL GUSTAVO MALDONADO HURTADO**, somos los responsables de las ideas, doctrinas y resultados expuestos en esta Tesis de Grado, y el patrimonio intelectual de la misma pertenecen a la Escuela Superior Politécnica de Chimborazo"

Cristina Judith Guerrero Coello

Daniel Gustavo Maldonado Hurtado

ÍNDICE DE ABREVIATURAS

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CAN	Controller Area Network
CMOS	Complementary Metal-Oxide-Semiconductor
comm API	Java Communications Api
CCP	Captura Comparación PWM.
CPU	Unidad Central de Procesamiento
EEPROM	Electrically-Erasable Programmable ROM
FTP	File Transfer Protocol
HMI	**I**nterfaz Humano Máquina
HTTP	Hypertext Transfer Protocol
I/O	Input / Output
IDE	Integrated Development Environment
JDK	Java Development Kit
JRE	Java Runtime Environment
JVM	Java Virtual Machine
LAN	Red de Área Local
LCD	Liquid Crystal Display
LED	Diodo Emisor de Luz
LSI	Large-scale Integration
MS-DOS	MicroSoft Disk Operating System

PC	Computadora Personal
PWM	Pulse-Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTS/CTS	Ready To Send / Clear To Send
SRAM	Static RAM
SODIMM	Small Outline DIMM (Dual In-line Memory Module)
TCP/IP	Transfer Control Protocol / Internet Protocol
TINI	Tiny InterNet Interface
TINI OS	Sistema Operativo de TINI™
TINI SDK	TINI Software Development Kit
TTL	Transistor-Transistor Logic
TTY	Terminal Virtual Serial
UART	Universal Asynchronous Receiver Transmitter
URL	Uniform Resource Locator
XML	Extensible Markup Language

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Los Sistemas Embebidos son microcomputadores, pequeños, de bajo costo y que realizan una función específica. Desde hace varios años, los Sistemas Embebidos han sido ampliamente utilizados en las aéreas Industriales, Investigativas, Domésticas, etc; sin embargo no es ampliamente difundido su funcionamiento, uso o construcción, ya que generalmente su presencia es desapercibida en la vida cotidiana.

La Plataforma TINI, es un sistema embebido proveído por Dallas Semiconductor como una herramienta multidisciplinaria, programable en Java, que provee I/O integradas como serial, CAN, 1 Wire, TTL I/O, y Ethernet; implementa soporte para varios protocolos de la pila TCP/IP; y su principal función es interconectar redes TCP/IP con Sistema Embebidos Propietarios en su protocolo nativo.

El objetivo principal de esta Tesis fue la implementación de una tarjeta de desarrollo de aplicaciones utilizando el sistema embebido TINI™, dotando de un Equipo de Aprendizaje Didáctico que permite realizar prácticas de Sistema Embebidos, para el laboratorio de control en la Escuela de Ingeniería Electrónica.

La metodología utilizada en el desarrollo de esta Tesis inició con la adquisición de conocimientos necesarios a través de investigación y autoaprendizaje basados en documentos publicados referentes al tema. Logrando Diseñar e Implementar a través de la Experimentación un Sistema que cumpla con los requerimientos de esta Tesis.

Se obtuvo un equipo de aprendizaje compuesto por un Laboratorio TINI, que incluye la Tarjeta de Desarrollo implementada y la plataforma TINI; y una Guía de Aprendizaje escrita e interactiva; que en conjunto permiten conocer, explorar y explotar las capacidades del Sistema Embebido TINI.

CAPÍTULO I

MARCO REFERENCIAL

1.1 ANTECEDENTES

Los Sistemas Embebidos han inundado el mercado como alternativas eficientes, económicas y sencillas para realizar distintas tareas de monitorización y control, basadas en plataformas extensamente conocidas.

Los Sistemas Embebidos se encuentran en todo dispositivo electrónico, como teléfonos móviles, equipos de audio y video, electrodomésticos, etc. Siendo una de sus principales funciones la capacidad de trabajar en redes basadas en protocolos TCP/IP, tales como las Redes de Área Local LAN, y el Internet.

Dentro de la escuela de Ingeniería Electrónica, en el rediseño curricular del año 2008 se implemento la nueva carrera de Ingeniería en Electrónica, Control y Redes Industriales, convirtiéndose en algo indispensable conocer los Sistemas Embebidos como una herramienta de control.

En la actualidad, los estudiantes de la Escuela de Ingeniería Electrónica desconocen casi en totalidad la funcionalidad de los Sistemas Embebidos, especialmente para Sistemas de Monitoreo y Control.

La Escuela de Ingeniería Electrónica se encuentra implementando un Laboratorio de Control, el cual planea poseer variedad de equipos orientados a dicha rama.

1.2 JUSTIFICACIÓN

Esta Tesis da solución a un problema que ocurre actualmente en la Escuela de Ingeniería Electrónica, que es la falta de equipos y laboratorios para el área de Control y Monitorización o conjuntos de aprendizaje para la utilización de Sistemas Embebidos.

Con la realización e implementación de esta Tesis, se logró dotar de un Laboratorio de Sistemas Embebidos, que consta de los elementos descritos a continuación.

Para el desarrollo del proyecto, se utilizó como base el sistema embebido **TINI™,** su micro núcleo ***LINUX***, sus funcionalidades ***JAVA™*** y soporte para redes TCP/IP.

Una **TARJETA DE DESARROLLO** que posee funcionalidades para la simulación de monitorización y control de Sistemas Propietarios mediante LEDs, interruptores, transductores y motores que se consideran como actuadores y sensores.

Una **GUÍA DE APRENDIZAJE** de la utilización y programación del sistema embebido; estructurado con teoría, ejemplos y ejercicios prácticos que son implementados y probados mediante la tarjeta de desarrollo.

Es por esto que resultó ser un proyecto práctico y muy aplicable, además de brindar beneficios a los estudiantes de la Escuela de Ingeniería Electrónica; en el que se pudo aplicar todos los conocimientos adquiridos en las aulas de clase complementado con la investigación de los autores.

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Implementar una tarjeta de desarrollo de aplicaciones utilizando el sistema embebido TINI™ para el laboratorio de control en la Escuela de Ingeniería Electrónica.

1.3.2 OBJETIVOS ESPECÍFICOS

- Investigar y operar la tarjeta embebida TINI (Tiny InterNet Interface).

- Implementar una tarjeta de desarrollo capaz de comunicarse a un sistema embebido TINI™ y poseer funcionalidades para Simulación de monitorización y control.
- Desarrollar una guía de aprendizaje escrita e interactiva con soporte teórico y ejercicios prácticos, que aporte a la enseñanza del sistema embebido TINI™, cuyos resultados sean corroborados mediante la tarjeta de desarrollo.
- Desarrollar aplicaciones que realicen control y monitorización de forma remota, mediante el uso de Redes basadas en TCP/IP, tales como redes LAN o Internet.

CAPÍTULO II

MARCO TEÓRICO

2.1 SISTEMAS EMBEBIDOS

Un **sistema embebido** es un sistema de computación diseñado para realizar una o algunas pocas funciones dedicadas, frecuentemente en tiempo real. En un sistema embebido la mayoría de los componentes se encuentran incluidos en la placa base aunque muchas veces los dispositivos no lucen como computadoras. Por lo general los sistemas embebidos se pueden programar directamente en el lenguaje assembler del microcontrolador incorporado sobre el mismo o con algún lenguaje de programación como C, C++ o BASIC.

Sus ventajas principales son su bajo costo y consumo. Los sistemas embebidos suelen usar un procesador relativamente pequeño y una memoria pequeña.

2.1.1 COMPONENTES DE UN SISTEMA EMBEBIDO

En la parte central se encuentra el microprocesador. Es decir, la CPU o unidad que aporta capacidad de cómputo al sistema, pudiendo incluir memoria interna o externa.

La comunicación adquiere gran importancia en los sistemas embebidos. Lo normal es que el sistema pueda comunicarse mediante interfaces estándar de cable o inalámbricas.

El subsistema de presentación suele ser una pantalla gráfica, táctil, LCD, alfanumérico, etc.

Se denominan actuadores a los posibles elementos electrónicos que el sistema se encarga de controlar. Posee un módulo de E/S analógicas y digitales que suelen emplearse para digitalizar señales analógicas procedentes de sensores, activar diodos LED, reconocer el estado abierto o cerrado de un pulsador, etc.

El módulo de reloj es el encargado de generar las diferentes señales de reloj a partir de un único oscilador principal.

El módulo de energía se encarga de generar las diferentes tensiones y corrientes necesarias para alimentar los circuitos del Sistema Embebido. Siendo el consumo de energía determinante cuando el sistema embebido se alimenta con baterías.

En general, un Sistema Embebido consiste en un sistema con microprocesador cuyo hardware y software están específicamente diseñados y optimizados para resolver un problema concreto eficientemente. Normalmente un Sistema Embebido interactúa continuamente con el entorno para vigilar o controlar algún proceso mediante una serie de sensores. Un sistema embebido simple cuenta con un microprocesador, memoria, unos pocos periféricos de E/S y un programa dedicado a una aplicación concreta almacenado permanentemente en la memoria. El término embebido hace referencia al hecho de que el microcomputador está encerrado o instalado dentro de un sistema mayor y su existencia como microcomputador puede no ser aparente.

Un sistema embebido complejo puede utilizar un sistema operativo como apoyo para la ejecución de sus programas, sobre todo cuando se requiere la ejecución simultánea de los mismos.

2.1.2 APLICACIONES DE UN SISTEMA EMBEBIDO

Los lugares donde se pueden encontrar los sistemas embebidos son numerosos y de varias naturalezas:

- En electrodomésticos tales como: televisores, decodificadores para la recepción de televisión digital, videos, lavadoras, alarmas, teléfonos inalámbricos, etc. Incluso una PC tiene sistemas embebidos en el monitor, impresora, y periféricos en general, adicionales a la CPU de la PC.
- En una fábrica, para controlar un proceso de montaje o producción.
- Equipos de medicina en hospitales y ambulancias.
- Máquinas de revelado automático de fotos.
- Cajeros automáticos, etc.

2.2 PLATAFORMA TINI

Tiny InterNet Interface (TINI) es una plataforma desarrollada por Dallas Semiconductor que provee una vía simple, flexible y barata para diseñar una amplia gama de dispositivos de hardware que puedan conectarse directamente a las redes corporativas y domesticas. Esta plataforma es una combinación de poderosos conjuntos de chips que proveen procesamiento, control, comunicación a nivel de dispositivos, capacidades de trabajar en red y un Runtime Environment Programable en Java donde las características del hardware son expuestas al desarrollador del software a través de un conjunto de Interfaces de Programación de Aplicaciones de Java más conocido como API de *Java*.

La combinación de capacidades de entrada y salida en la tarjeta TINI, con la pila de protocolos TCP/IP y un ambiente de programación de Java, permite a los

programadores crear rápidamente aplicaciones que proveen no solo el control local sino también acceso global a los dispositivos basados en TINI. Las capacidades de trabajar en red de la TINI extiende la conectividad de cualquier dispositivo contiguo, permitiendo la interacción con sistemas y usuarios remotos a través de aplicaciones de red estándar, como los exploradores Web.

TINI está diseñada para cubrir los requerimientos funcionales para aplicaciones de red embebidas comerciales e industriales.

TINI puede ser usada para tareas embebidas tradicionales *stand-alone*, como monitorización y control de un sistema ó dispositivo local. Sin embargo, la capacidad de red que posee la Tarjeta TINI es utilizada por la mayoría de aplicaciones en la actualidad, como:

Controles Industriales.- El soporte para el Controller Area Network (CAN) integrado en la TINI es muy instrumental en la implementación del equipamiento de automatización de fábrica, los switches de Red y actuadores.

Equipo de monitorización y control basado en Web.- Permite la recolección de datos y diagnósticos remotos por web, para la monitorización de un dispositivo.

Conversión de Protocolos.- Los sistemas basados en TINI pueden ser usados para conectar dispositivos propietarios a Redes Ethernet. Dependiendo de las capacidades I/O del sistema propietario.

Monitores Ambientales.- TINI en red con 1-Wire, permite la consulta de sensores y reportar la información obtenida a un host remoto.

2.2.1 ARQUITECTURA HARDWARE

La plataforma TINI esta conformada por chips LSI, otros chips más pequeños y varios componentes discretos como resistencias, capacitores y cristales. El Hardware mínimo de TINI contiene los siguientes chips LSI:

- Microcontrolador
- Flash ROM
- Static RAM

El microcontrolador es el corazón de cualquier diseño de Hardware en la TINI, y ejecuta directamente la porción de código nativo del Runtime Environment. El microcontrolador usado en las implementaciones actuales de la TINI es el **DS80C400.** Es un microcontrolador pequeño con soporte para distintas formas de entrada y salida incluyendo Serial y CAN, también provee varios pines de propósito general que pueden ser usados para realizar tareas de control simple como manejar Relés y LEDs de estado.

La memoria Flash almacena el Runtime Environment de TINI y satisface lo siguiente:

- La memoria contenida es conservada inclusive en ausencia de alimentación del sistema.

- La memoria es reprogramable.

La Static RAM contiene el área de datos del sistema, y también el *garbage collector* en el cual todos los objetos de Java se ubican. También almacena todos los archivos del sistema de información. Para que el sistema de archivos persista en ausencia de energía se necesita que la Static RAM tenga un backup de batería (nonvolatized).

Los dispositivos periféricos, aparte de la memoria pueden ser conectados directamente a los buses de datos y direcciones del microcontrolador (etiquetados como “Parallel I/O expansion”). Dos de estos periféricos son usados comúnmente en los sistemas TINI: Controlador Ethernet y reloj de tiempo real. La configuración mostrada en la figura extiende el alcance de los sistemas embebidos a las redes Ethernet.

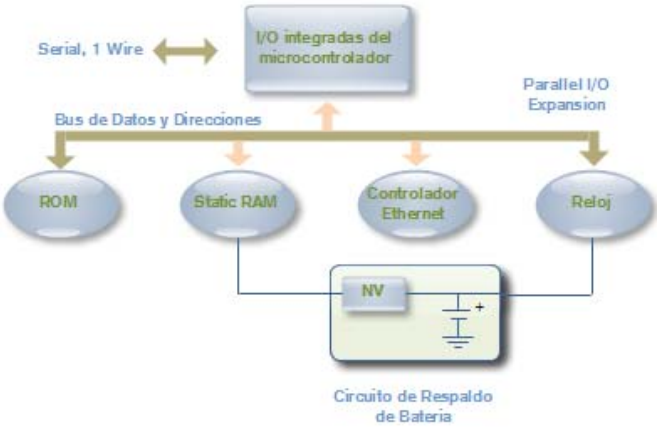


Figura.II.01. Funciones de aplicación de hardware TINI
Fuente: Libro The TINI™ Specification and Developer’s Guide

El circuito de respaldo de batería realiza dos funciones, primero mantiene el reloj corriendo en ausencia del poder principal Vcc, asegurando que un tiempo preciso pueda ser siempre leído desde el reloj, la celda de litio es la que realiza esta tarea. Además la celda en conjunto con un chip pequeño conocido como SRAM nonvolatizer

(NV) mantiene los contenidos de la Static RAM en ausencia del poder principal. La principal razón de no volatizar la SRAM es permitir que la información del sistema de archivos persista aun en ausencia de la energía.

Mapa de Memoria

Un mapa de memoria específica donde la memoria y otros dispositivos periféricos son decodificados en el espacio de direcciones del microcontrolador. El mapa de memoria usado en la TINI consiste en tres elementos distintos:

- Código
- Datos
- Periféricos

Los segmentos de Código y Datos son ocupados por los chips de memoria, y el segmento de periféricos es ocupado por otro tipo de componentes de Hardware como el controlador de Ethernet y el reloj de tiempo real. Otros dispositivos periféricos que soporten una interface de bus paralelo compatible con el bus del microcontrolador también pueden ser mapeados en el segmento de periféricos.

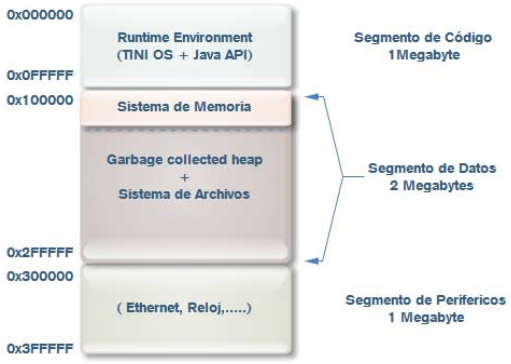


Figura.II.02. Mapa de Memoria
Fuente: Libro The TINI™Specification and Developer's Guide

I/O Integradas

Un gran rango de dispositivos que son interesantes para trabajar en red con la TINI no tienen soporte para conectarse a un bus paralelo. Frecuentemente estos dispositivos tienen alguna forma de interface serial, que resulta en un ancho de banda de comunicación menor. Pero las interfaces seriales también reducen el número de pines requeridos, simplificando la comunicación y a menudo bajando los costos comparado con dispositivos que tienen interfaces de bus paralelas. Los protocolos de comunicaciones seriales de bajo nivel soportados por la TINI son los siguientes:

- **Comunicación Serial.-** Protocolos Seriales Sincrónicos, usando interfaces de dos cables, y comunicación serial asincrónica basada en el estándar RS232-C. El controlador TINI provee dos circuitos UART integrados.
- **Controller Area Network (CAN).-** Provee un bus de comunicación serial confiable que es comúnmente usado en aplicaciones de control industrial y de automotores.
- **1-Wire Network.-** Desarrollada por Dallas Semiconductor, la red 1-Wire esta conformada por pequeños sensores, actuadores y elementos de memoria que comparten un solo conductor para comunicación y alimentación.
- **TTL I/O.-** Estos son pines bidireccionales y de propósito general del microcontrolador que pueden ser usados para varias tareas de control.

Al utilizar las capacidades de entrada y salida integradas en el microcontrolador envés de las I/O mapeadas en memoria, se disminuye el conteo de dispositivos y el costo de

comunicación con dispositivos externos porque la carga del microcontrolador es menor.

Por ejemplo, el núcleo del CPU puede correr a toda velocidad ejecutando el Runtime Environment mientras el UART está enviando y recibiendo caracteres seriales al mismo tiempo. Si la comunicación se la hiciera a través del bus paralelo, el CPU deja de realizar lo que esta haciendo en ese momento y ejecuta instrucciones para leer o escribir datos en el dispositivo.

Diseño de Referencia del Hardware

La TINI Board Model 400 o **DSTINIm400** a sido desarrollada para brindar a los diseñadores de sistemas, una implementación referencial. Permitiendo tanto a los diseñadores de hardware como de software iniciar sus prototipos y desarrollar el trabajo sin la necesidad de gran inversión de dinero y tiempo.

EL DSTINIm400 cubre los siguientes propósitos:

Implementación Referencial.- Todos los detalles de diseño son públicos. Los desarrolladores de hardware tienen la libertad de utilizar esta información cuando desarrollen sus propios sistemas basados en la TINI.

Herramienta de Desarrollo.- Provee fácil acceso a la mayoría de las capacidades I/O de la plataforma, permitiendo a los diseñadores la rápida conexión a hardware externo personalizado y desarrollar sus aplicaciones.

Componente de un Sistema.- El DSTINIm400 es un diseño *fully specified*. Ha sido fuertemente probado y funcionalmente caracterizado sobre voltajes y temperaturas y por lo tanto puede ser utilizado como el componente núcleo de aplicaciones de red embebidas comerciales e industriales.

El DSTINIm400 incluye las siguientes importantes características:

- 1 Megabyte de SRAM nonvolatile.
- 1 Megabyte de Flash memory para el código del sistema.
- Disponible en un módulo de 144 pines SODIMM
- 10 Base-T Ethernet controller
- Real-time clock
- DS80C400

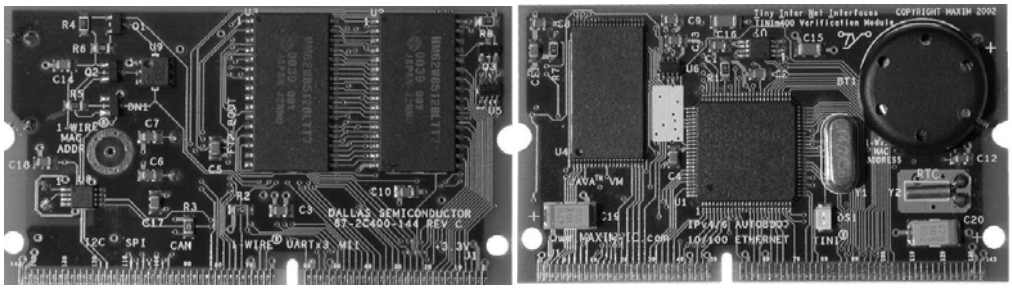


Figura.II.03. Vista Inferior y Superior de DSTINIm400
Fuente: Documento Electrónico "Getting Started with TINI"

Socket para DSTINIm400: DSTINIs400

Este socket sirve para aplicaciones de desarrollo y prototipos. Su principal función es brindar conectores físicos que interconecten la DSTINIm400 con otros equipos como

son redes Ethernet, dispositivos seriales o redes 1-Wire, además de proporcionarle energía. La DSTINIs400 posee los siguientes conectores (Ver Figura 4):

- **144-Pin SODIMM Connector**. Es una ranura especial donde se coloca el módulo DSTINIm400.
- **9-Pin Female DB9 Connector.** Se utiliza para tener una conexión con una PC estándar. Es utilizado comúnmente para cargar el Runtime Environment y la aplicación bootstrap.
- **9-Pin Male DB9 Connector.** Se utilizan en la mayoría de aplicaciones TINI que ocupan dispositivos seriales.
- **RJ45.** Se ocupa para dar conexión a una red Ethernet
- **RJ11.** Da acceso a una red 1-Wire usando un cable telefónico estándar.
- **Power Jack.** Necesita un poder de alimentación de +5V DC regulada.

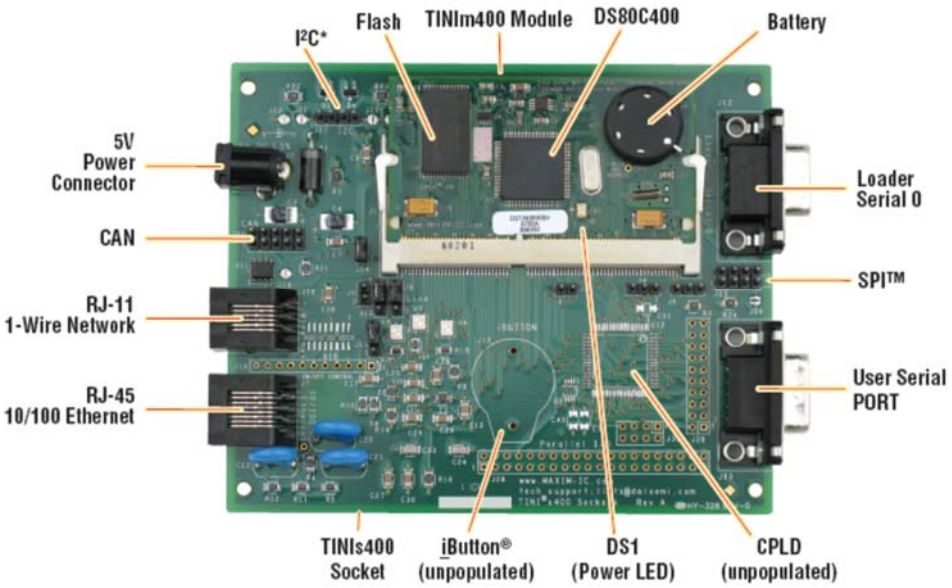


Figura.II.04. DS80C400-KIT# que incluye DSTINIs400 y DSTINIm400
Fuente: Data Sheet DS80C400-Kit

Características del socket DSTINIs400

Tabla.II.I. Características del socket DSTINIs400

CARACTERISTICAS	TAMAÑO (mm)	MODULOS COMPATIBLES	ALIMENTACION
conectores 1-Wire, CAN, serial, 10/100	120 x 100	DSTINIm400	5V DC * 10% >150 mA

Fuente: Documento Electrónico Getting Started with TINI

Características del modulo DSTINIm400

Tabla.II.II. Características del módulo DSTINIm400

PROCESADOR	CARACTERISTICAS	FORMATO	SOCKETS COMPATIBLES
DS80C400	1Mb RAM, 1Mb Memoria flash	144-Pin SODIMM	DSTINIs400

Fuente: Documento Electrónico Getting Started with TINI

2.2.2 ARQUITECTURA SOFTWARE

2.2.2.1 TINI RUNTIME ENVIRONMENT

Una gran cantidad de software es requerido para liberar a los desarrolladores de aplicaciones de preocupaciones sobre detalles de crear infraestructura de capas para soportar la ejecución de múltiples tareas, pilas de protocolos de red, y una API (Application Programming Interface). El Runtime Environment fue desarrollado desde el principio como una parte integral de la plataforma en general. El Software que comprende el Runtime de la TINI puede ser dividido en dos categorías: Código Nativo ejecutado directamente por el microcontrolador y una API interpretada como bytecodes por la Maquina Virtual de Java (JVM). El código de aplicación es escrito en

Java y utiliza la API para explotar las capacidades del Runtime Nativo y los recursos de hardware. Una representación gráfica del Runtime Environment es mostrada en la Figura 5.

En la TINI, las aplicaciones Java tienen privilegios completos y acceso a todos los recursos del sistema. Esta particularidad es importante para las aplicaciones embebidas porque están acopladas muy cerca de los dispositivos físicos. Además en la TINI, generalmente no existe un administrador del sistema que realice configuración y mantenimiento. Esto significa que la aplicación es responsable de configurar y controlar el sistema completo.

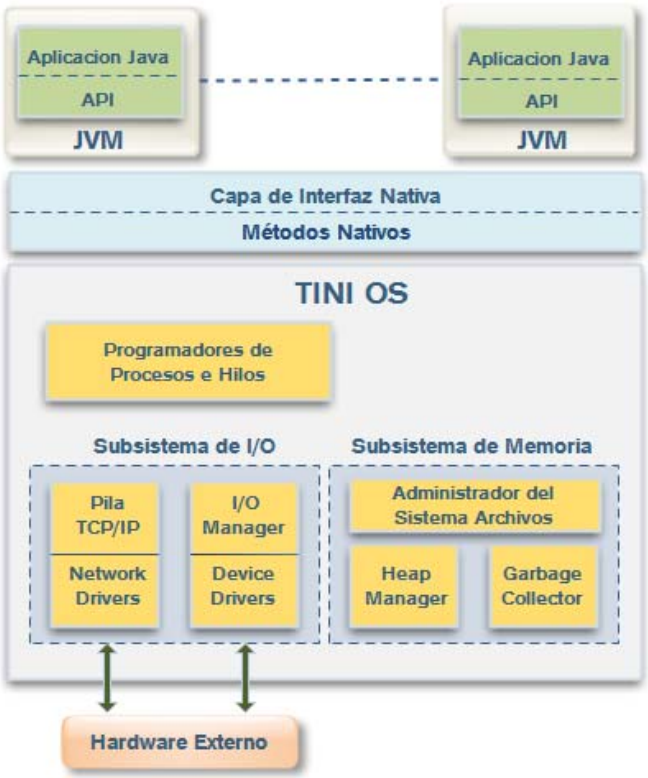


Figura.II.05. TINI Runtime Environment
Fuente: Libro The TINI™ Specification and Developer's Guide

DESCRIPCION GENERAL DEL TINI API

La porción API del Runtime Environment combina clases de diferentes paquetes definidos en el Java Development Kit (JDK) de Sun, con las clases específicas de TINI que exponen las capacidades del sistema y no tienen análogos en otras plataformas de Java mas extensas. Las clases específicas de TINI son definidas como subpaquetes dentro del paquete raíz com.dalsemi. Las clases que son incluidas en el Runtime Environment son conocidas como la porción del API incorporada.

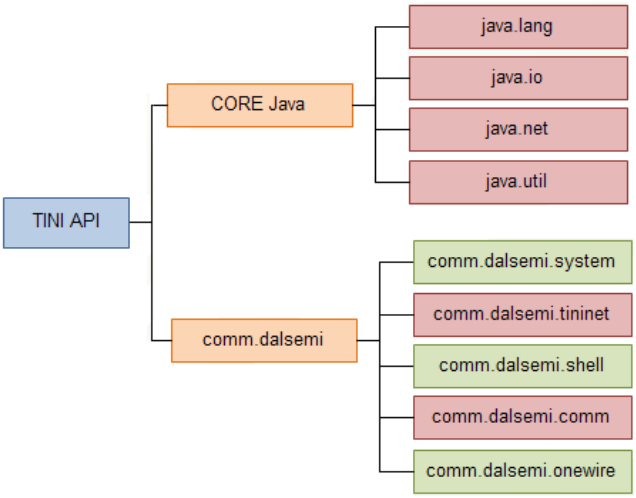


Figura.II.06. Estructura del TINI API
Fuente: Los Autores

PAQUETES Core Java.- La API incluye implementaciones para la mayoría de las clases en los siguientes paquetes Core Java.

- java.lang
- java.io
- java.net
- java.util

PAQUETES com.dalsemi

com.dalsemi.system. Las clases en este paquete proveen acceso a varias formas de I/O integradas incluyendo el puerto serial síncrono de dos cables (2-wire), el bus de datos del microcontrolador, y puertos de pines individuales. También contiene clases para la configuración de los recursos del sistema como el reloj, el temporizador Watch Dog y las interrupciones externas.

com.dalsemi.tininet. Este paquete contiene una clase llamada *TININet* que provee métodos estáticos para consultas y configuración de algunos parámetros de Red, como la dirección IP y la máscara de subred. Los subpaquetes del com.dalsemi.tininet, proveen soporte para los protocolos de red como DHCP (Dynamic Host Configuration Protocol), ICMP (Internet Control Message Protocol), y DNS (Domain Name System).

com.dalsemi.shell. Las clases en este paquete y sus subpaquetes implementan la infraestructura para aplicaciones interpretadoras de comandos (shell). Las clases en los subpaquetes de com.dalsemi.shell implementan los servidores Telnet y FTP. Estos servidores también pueden ser usados por aplicaciones distintas del interpretador de comandos para proveer acceso a aplicaciones cliente Telnet y FTP.

com.dalsemi.comm. Este paquete contiene clases cercanas a bajo nivel para acceder a los controladores CAN. También contiene algunas clases para configuración y comunicación con los puertos seriales del sistema. Sin embargo, estas clases también son usadas por aplicaciones. El acceso al puerto serial es proveído e implementado en la API de comunicaciones de Java.

com.dalsemi.onewire. Este es el paquete raíz de la jerarquía para la API de 1-Wire. A diferencia de los paquetes descritos anteriormente, la API de 1-Wire también es soportada por otras plataformas de Java distintas de TINI. El paquete com.dalsemi.onewire.container provee clases conocidas como contenedores, que comprenden el comportamiento de chips 1-wire específicos. Para evitar consumir un valioso espacio en la memoria flash, clases contenedoras específicas del dispositivo no son incluidas en la API incorporada. Las clases contenedoras deben incluirse como parte de la aplicación.

MAQUINA VIRTUAL DE JAVA DE TINI

El espacio ocupado por la JVM de TINI es menos de 40 kilobytes. Sin considerar su pequeño tamaño, soporta muchas de las funcionalidades provistas por las implementaciones completas de una JVM, incluyendo las siguientes[1]:

- Soporte total para threads (multitarea).
- Soporte de todos los tipos primitivos.
- Cadenas
- Soporta pila de protocolos TCP-IP

[1] *Para mas información sobre las limitaciones de la JVM revisar el archivo Limitations.txt del TINI SDK disponible en la Zona de Descarga de la Guía de Aprendizaje Interactiva TINI/TDDC*

MÉTODOS NATIVOS

La capa nativa representa la colección de métodos nativos que soporta la API dando acceso a la infraestructura provista por el TINI OS. Incluye métodos para el acceso a la pila de protocolos de Red así como también a controladores de dispositivos que no trabajan en la red. También incluye métodos para configurar el acceso a los recursos del sistema como el temporizador Watch Dog y el reloj de tiempo real.[2]

Las aplicaciones que requieren métodos nativos personalizados, pueden proveer librerías nativas que pueden ser cargadas en el sistema en tiempo de corrido usando el método loadLibrary.

TINI OS

El Sistema Operativo de TINI esta en la capa mas baja del Runtime Environment. Es responsable de manejar los recursos del sistema como el acceso a la memoria, la programación de múltiples procesos de ejecución de Threads e interactuar con los componentes de hardware tanto internos como externos. El TINI OS se define en 3 componentes [3] :

- Programador de procesos y tareas.
- Administración del Subsistema de manejo de memoria.
- Administración del Subsistema de manejo de I/O.

---

[2] Para conocer el proceso de escritura y construcción de librerías nativas, revisar "Native_Methods.txt" y "Native_API.txt" del TINI SDK.

[3] Revisar el libro The TINI™Specification and Developer's Guide pg 15-18

Funcionamiento del Sistema

El área de código esta conformada por 3 partes distintas:

Figura.II.07. Segmento de Código expandido.
Fuente: Libro The TINI™ Specification and Developer's Guide

- El cargador.
- Entorno de ejecución.
- Aplicación primaria de Java.

La secuencia de carga se describe 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 lanza la aplicación primaria de java.

Generalmente se carga primero el command Shell o interpretador de comandos, este es muy útil para la etapa de desarrollo y pruebas, y además permite configurar parámetros del sistema, como la configuración de red. Una vez que se haya probado

rigurosamente una aplicación para un ambiente de producción, esta puede reemplazar el interpretador de comandos como la aplicación primaria de java.

El firmware proporcionado incluye un interpretador de comandos para el TINI OS que se carga en la memoria flash ROM y se ejecuta cuando se alimenta a TINI.

2.2.2.2 TINI SDK

El TINI Software Development Kit, conocido como Firmware, es un conjunto de archivos, clases y aplicaciones necesarias para el desarrollo de sistemas TINI. El firmware de TINI 1.1X es el más usado en la actualidad, cuya versión utilizada en esta Tesis fue la TINI 1.18. La última versión puede ser descargada gratuitamente de la página Web de Dallas Semiconductor[4]. Microcontrollers Tool Kit es un archivo ejecutable que sirve para cargar archivos fácilmente en la tarjeta TINI que solo funciona en Plataformas Windows, y es utilizado para cargar el firmware.

Dentro del TINI SDK existen algunos archivos importantes como son:

README.txt.- Contiene instrucciones detalladas de cómo instalar el TINI Runtime Environment, el arranque del sistema TINI y la inicialización de las configuraciones de RED. También contiene referencias a otros documentos en el SDK que describen el procedimiento para crear un ambiente de desarrollo completo.

---

[4] Descarga del Software en http://www.maxim-ic.com/products/tini/software/downloads.cfm

tini.jar.- Es un archivo de java localizado en el directorio bin y tiene 3 programas utilitarios importantes que son:

- JavaKit: Administra el proceso de carga del firmware y realiza otras tareas de mantenimiento del sistema. También puede ser utilizado para correr sesiones de usuario de Slush sobre una conexión serial.
- TINIConvertor: Toma los archivos class de su aplicación como entrada y genera una imagen binaria utilizable para la ejecución en la TINI.
- BuildDependency: Es una aplicación adicional que sirve de soporte para el TINIConvertor. Ayuda a construir más aplicaciones donde la inclusión o agregación de una clase pueda significar la inclusión de muchas clases de soporte.

tiniclasses.jar.- Es un archivo que está ubicado en el directorio bin, que contiene todos los archivos de clases que existen en el API de TINI. Este archivo debe ser incluido siempre en la opción -bootclasspath de javac cuando se compilan aplicaciones para la TINI.

```
javac –bootclasspath c:\tini1.18\bin\tiniclasses.jar [archivo.java]
```

tini.db.- Es una base de datos ASCII que contiene información acerca de todos los archivos de clases que se encuentran en el API de TINI. El TINIConvertor utiliza este archivo junto con los archivos de clase de la aplicación, para generar una imagen binaria utilizable para la interpretación de la maquina virtual de java de la TINI (TINI JVM).

Archivos Tbin.- Es una forma de resumir TINI binary y es una extensión por defecto utilizada para las imágenes binarias que van a ser cargadas en la flash ROM. Los archivos Tbin distribuidos con el TINI SDK, contienen las implementaciones del TINI Java Runtime y la aplicación Slush Shell.

El archivo tini_400.tbin contiene la imagen binaria del TINI Runtime Environment y algunas veces puede ser referido como el "firmware". Es la combinación del Sistema operativo nativo y el Java API. En cambio el slush400.tbin es una aplicación que tiene la imagen binaria del Shell de usuario, conocido como Slush.

2.2.3 ACCESO AL SISTEMA TINI

Debido al control embebido y la naturaleza concéntrica de entradas y salidas de la mayoría de aplicaciones de red embebidas, no existe hardware o soporte de API para la interface humana. Sin embargo, la visualización local e ingreso de datos puede ser obtenida realizando una conexión a una terminal a través de un enlace serial o Ethernet. Si un sistema TINI requiere interface humana puede ser implementado usando LCDs y teclados.

Las sesiones de Terminal permiten la monitorización del Sistema mediante el interpretador de comandos Slush, dando acceso a la administración de la TINI, y a la configuración, ejecución y prueba de las aplicaciones desarrolladas por los usuarios.

En TINI, una vez que se ha cargado el Runtime Environment, el interpretador de comandos Slush, y se ha inicializado el TINI Firmware; se puede iniciar sesiones de terminal sin necesidad de recargar este software fundamental en cada ocasión.

Una de las principales configuraciones de la TINI realizadas en una terminal conectada a través de un enlace serial es la configuración de la Red Ethernet. Con esto se logra poner en marcha una de las funciones importantes de la TINI que es la comunicación mediante una red Ethernet TCP/IP.

2.2.4 SLUSH

Es un pequeño interpretador de comandos cuya intención es dar una interface parecida a Unix al TINI Runtime Environment, proporcionando servidores seriales TTY (Terminal Virtual Serial), servidores telnet y FTP.

Slush es una aplicación de java que es interpretada por la maquina virtual de java de TINI (TINI JVM). Slush es menos que un sistema operativo completo pero mucho más que un Shell simple, provee una manera de ver y manipular el sistema de archivos, correr otras aplicaciones de java y controlar funciones del sistema como el *watch dog timer* y la configuración de red.

Slush está diseñado para ser un sistema multitarea y multiusuario, permitiendo sesiones de usuario simultáneas, estas características son usadas típicamente en la fase de desarrollo.

Slush provee accesibilidad en red usando las aplicaciones de *cliente telnet* para la interacción del usuario y *ftp* para transferir aplicaciones y archivos de información hacia y desde el sistema de archivos[5].

Después que una aplicación ha sido desarrollada y depurada es instalada en la flash ROM, reemplazando a Slush en la mayoría de veces.

2.2.5 TCP/IP NETWORKING

La característica más importante de la plataforma TINI es su gran conectividad de redes. Una aplicación en TINI como por ejemplo Slush, es capaz de configurar y usar la red.

El API de red TINI provee soporte especial para los protocolos mostrados en la figura 8. Existen seis protocolos de la capa de aplicación que son soportados en el API de TINI:

- HTTP (Hypertext Transfer Protocol)
- DNS (Domain Name System)
- DHCP (Dynamic Host Configuration Protocol)
- Telnet
- FTP (File Transfer Protocol)
- Ping (ICMP echo request/reply)

---

[5] *Para un mayor detalle de los comandos del Shell Slush, referirse a la Guía de Aprendizaje TINI/TDDC - Práctica #3*

Estos protocolos excepto ping son implementados usando las clases socket en uno de los paquetes del API de TINI como el mecanismo de transporte de red. Ping no es en realidad un protocolo sino una aplicación desarrollada dentro del subconjunto de ICMP (Internet Control Message Protocol). La clase Ping directamente invoca métodos nativos que son expuestos en el módulo ICMP de la pila de red.

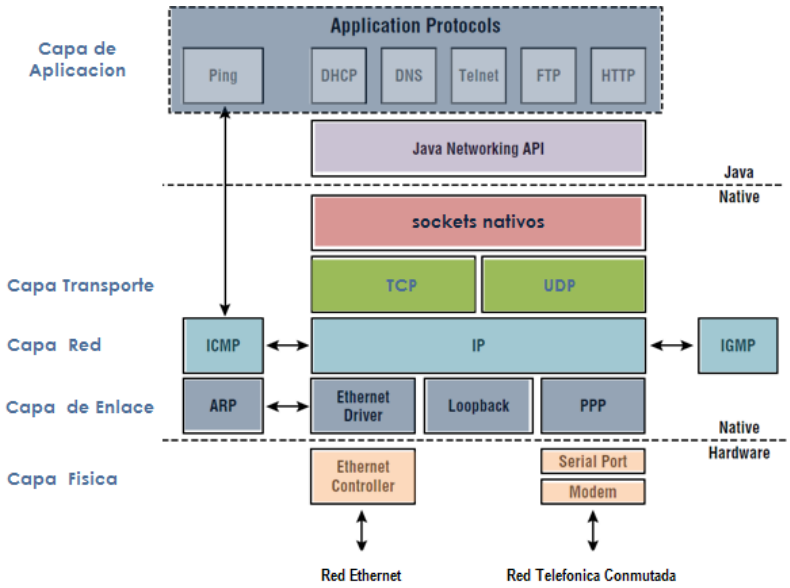


Figura.II.08. Protocolos soportados por TINI
Fuente: Libro The TINI™Specification and Developer's Guide

Configurando Parámetros de Red

En host no embebidos, las aplicaciones de red no necesitan preocuparse por configurar los parámetros de red básicos como la dirección IP y la mascara de subred ya que esta tarea es realizada por un administrador de red usando el programa utilitario de red proveído por el sistema operativo. Durante el desarrollo de aplicaciones esto también

se aplica en TINI. Las configuraciones de red son establecidas usando el comando de *slush ipconfig*[6], siendo los parámetros de red configurables los siguientes:

- Dirección IP (IP address)
- Mascara de Subred (Subnet mask)
- Dirección IP del Gateway (Gateway (router) IP address)
- Dirección DNS primaria (Primary DNS address)
- Dirección DNS secundaria (Secondary DNS address)
- Valor del time-out DNS (DNS time-out value)
- Nombre del Dominio (Domain name)
- Dirección IP del servidor DHCP
- Nombre del Host (Host name)
- Mailhost
- HTTP proxy server: La dirección IP de la máquina que reenvía solicitudes HTTP en su lugar. Por ejemplo: Si TINI está detrás de un firewall, el uso de un servidor proxy puede ser requerido para satisfacer las solicitudes HTTP de hosts fuera de la red local.
- HTTP proxy port: Es un integer de 16 bits que especifica el número de puerto en el cual el servidor HTTP proxy espera recibir las solicitudes HTTP.

---

[6] Para un mayor detalle de las opciones del comando ipconfig de Slush, referirse a la Guía de Aprendizaje TINI/TDDC - Práctica #4

2.3 JAVA

Java nace como parte de un proyecto de desarrollo de soporte de software para electrodomésticos diseñado por Ingenieros de Sun Microsystems en 1991. Este primer enfoque le da a Java una de sus más interesantes características: *la portabilidad*, dado que Java tenía que funcionar en numerosos tipos de CPU se desarrolló un código "neutro" que no dependía del tipo de electrodoméstico, el mismo que se ejecutaba sobre una máquina virtual denominada JVM, la cual interpretaba el código convirtiéndolo a código particular de la CPU utilizada. Logrando un lenguaje independiente de la plataforma sobre la que funcione.

Como lenguaje de programación para computadores, Java se introdujo a finales de 1995, con la incorporación de un intérprete Java en el programa Netscape Navigator versión 2.0, produciendo una verdadera revolución en Internet. En 1997 aparece la versión 1.1 con muchas mejoras y adaptaciones, luego Java 1.2 a finales de 1998 y finalmente Java 2.

AI programar en Java cualquier aplicación que se desarrolle se apoya en un gran número de clases preexistentes. Algunas de ellas las ha podido hacer el propio usuario, otras pueden ser comerciales, pero siempre hay un número muy importante de clases que forman parte del propio lenguaje, esto es la API o *Application Programming Interface* de Java.

El principal objetivo del lenguaje Java es llegar a ser el "nexo universal" que conecte a los usuarios con la información.

La compañía Sun describe el lenguaje Java como *"simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico"*. Todo ello describe bastante bien el lenguaje Java, aunque en algunas de esas características el lenguaje sea todavía bastante mejorable.

Plataforma de Desarrollo Java

Conocida como Java™ Development Kit o JDK, es un software distribuido gratuitamente por Sun Mycrosystem que provee un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java. Incorpora además la posibilidad de ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las variables (con el denominado Debugger).

El Java Runtime Environment (JRE) es una versión reducida del JDK destinada únicamente a ejecutar código Java (no permite compilar). El JRE esta compuesto por la JVM y un conjunto de librerías o APIs que contienen código Java compilado.

Los IDEs (Integrated Development Environment) son entornos de desarrollo integrados. En un mismo programa es posible escribir el código Java, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar debug gráficamente, que simplifica bastante el trabajo frente a la versión que incorpora el JDK basada en la utilización de una consola (denominada habitualmente ventana de comandos de MS-DOS, en Windows NT/95/98) bastante difícil y pesada de

utilizar. Estos entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con componentes ya desarrollados los cuales se incorporan al proyecto o programa. Sin embargo, al utilizar los IDEs, los ficheros resultantes son de mayor tamaño que los basados en clases estándar y existen algunos fallos de compatibilidad entre plataformas. Los entornos de desarrollo integrados comúnmente utilizados son NetBeans y Eclipse.

El compilador de Java

Se trata de una de las herramientas de desarrollo incluidas en el *JDK*. Realiza un análisis de sintaxis del código escrito en los ficheros fuente de Java *(con extensión .java).* Si no encuentra errores en el código, genera los ficheros compilados *(con extensión .class)* caso contrario muestra la línea o líneas erróneas. En el JDK de Sun dicho compilador se llama *javac.exe*. Tiene numerosas opciones, algunas de las cuales varían de una versión a otra[7].

Java Virtual Machine (JVM)

La JVM es el intérprete de Java. Interpreta y convierte código neutro a código particular de la CPU utilizada, evitando tener que realizar un programa diferente para cada CPU o plataforma y ejecuta los "bytecodes" (ficheros compilados con extensión *.class*) creados por el compilador de Java (Javac.exe).

---

[7] Consultar la documentación de la versión del JDK utilizado, para información detallada de las opciones del compilador.

Las variables PATH y CLASSPATH

El desarrollo y ejecución de aplicaciones en Java exige que las herramientas para compilar (Javac.exe) y ejecutar (Java.exe) se encuentren accesibles. El ordenador, desde una ventana de comandos de MS-DOS, sólo es capaz de ejecutar los programas que se encuentran en los directorios indicados en la variable PATH del ordenador (o en el directorio activo). Si se desea compilar o ejecutar código en Java, el directorio donde se encuentran estos programas (java.exe y javac.exe) deberá ser en el PATH. Tecleando PATH en una ventana de comandos de MS-DOS se muestran los nombres de directorios incluidos en dicha variable de entorno.

Java utiliza además una nueva variable de entorno denominada CLASSPATH, la cual determina donde buscar tanto las clases o librerías de Java (la API de Java) como otras clases de usuario.

2.3.1 PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA

Variables ó Primitivos en Java

Uno de los pilares principales de la programación es el uso de variables. En Java se encuentra muchos tipos de variables, capaces de almacenar todos los tipos de datos que se necesiten. Pero al estar hablando de programación Orientada a Objetos se tiene que diferenciar variables de objetos. Las variables, en Java, son llamados

primitivos. Simplemente almacenan ***un valor***. En cambio, un ***objeto*** es un conjunto de ***variables y métodos***.

Definición de Variables

Al igual que C, Java requiere que se declaren los tipos de todas las variables empleadas. La sintaxis de iniciación es la misma que C.

OPERADORES

Los operadores básicos de Java son (+, - , *, /). Además existen los operadores decremento e incremento: -- y ++ respectivamente y los operadores lógicos (!, ==, !=, <, >, <=, >=, &&, ||)

CADENAS DE CARACTERES

En Java no hay un tipo predefinido para cadenas de caracteres, en su lugar hay la clase ***String***, que es la que soporta las distintas operaciones con cadenas de caracteres.

ÁMBITO DE LAS VARIABLES

El ámbito de las variables en Java viene definida por los corchetes: *{ }*; una vez definida una variable en un código dejará de existir cuando se acabe el bloque de código en el que se definió. Los bloques de código empiezan con “{“ y acaban en “}”, por lo que la variable dejará de existir cuando se cierre el corchete que esté justo antes que ella en el código. Por otro lado es ilegal en Java definir una variable que ha sido definida anteriormente.

ARRAYS

En Java los arrays son un objeto. Como tales se crean mediante el comando new.

SENTENCIAS CONDICIONALES

Ejecutan un código u otro en función de que se cumpla o no una determinada condición. Siendo los principales:

- If then Else
- Switch

BUCLES

Son instrucciones que nos permiten repetir un bloque de código mientras se cumpla una determinada condición.

- Bucle while
- Bucle for

BREAK Y CONTINUE

Son sentencias íntimamente relacionadas con los bucles. La sentencia *break* detiene la ejecución del cuerpo del bucle y sale de este, continuándose ejecutando el código que hay tras el bucle. Es utilizado comúnmente para forzar la salida del bloque de ejecución de una instrucción condicional.

La sentencia *continue* también detiene la ejecución del cuerpo del bucle, pero en esta ocasión no se sale del bucle, sino que se pasa a la siguiente iteración de este.

RETURN

Su misión tiene que ver con el control de flujo: se deja de ejecutar código secuencialmente, pasa el valor especificado al código invocador y devuelve el control al código que invocó el método.

2.3.1.1 DEFINICIÓN DE CLASES, MÉTODOS Y OBJETOS

CLASE.- Es un conjunto de primitivos y métodos relacionados entre sí. Una clase es la "plantilla" que se utilizan para crear objetos. Todos los objetos pertenecen a una determinada clase. Las distintas clases tienen distintas relaciones de herencia entre sí: una clase que deriva de otra hereda todos los métodos y variables de la clase de la que se deriva o clase padre.

Definición de una clase.- La forma mas general de definir una clase es la siguiente:

```
[modificador] class nombre_clase [extends nombreClasePadre][implements interface ]{
  declaración de variables;
  declaración de Métodos;
}
```

Los campos que están entre corchetes son optativos; `nombre_clase` es el nombre de la clase que se va a crear, `nombreClasePadre` es el nombre de la clase de la cual se heredan variables y métodos (clase padre). En cuanto al contenido del último corchete ya se explicará más adelante su significado.

Modificadores

- *public*
- *ninguno*
- *abstract*
- *protected*
- *prívate*
- *static*
- *final*

CREACIÓN DE MÉTODOS

Un método es parecido a lo que en programación estructurada se llama función. Es un fragmento de código al que se llama, acompañado de datos de entrada y produce una determinada salida. Para esto se debe indicar el tipo de los primitivos de entrada, el tipo de los primitivos de salida, el nombre del método y el trabajo que realizará.

El esquema completo de un método estándar es:

```
 [modificadores] [tipoPrimitivoSalida] [nombreMétodo] ([tipoPrimitivoEntrada]
[nombrePrimitivoEntrada]) {
  //Código del método
  return primitivoSalida;
}
```

Sobrecarga de Métodos.- En Java es posible tener varios métodos con el mismo nombre, pero a los cuales se pasan distintos parámetros. Según los parámetros que se pasen se invoca a uno u otro método.

Constructores.- Son métodos cuyo nombre coincide con el nombre de la clase y que nunca devuelven ningún tipo de dato, no siendo necesario indicar que el tipo de dato devuelto es void. Son utilizados para inicializar los valores de los objetos y realizar las operaciones que sean necesarias para la generación de este objeto. Admite sobrecarga, permitiendo invocar al constructor que mas convenga al crear un objeto.

CREACIÓN Y REFERENCIA DE OBJETOS

Un objeto no es más que la instancia de una clase, es decir, el conjunto de primitivos y métodos definidos en la clase, que ahora definen un objeto determinado. Para crear un objeto, se utiliza el comando ***new***, su sintaxis es la siguiente:

```
nombreClase nombreObjeto;
nombreObjeto = new nombreClase();
```

Con el primer comando se crea un puntero que apunta a una variable tipo `nombreClase`, con el segundo inicializamos el objeto al que apunta el `nombreObjeto`, reservando espacio para sus variables.

***this*.-** Es una variable de solo lectura proporcionada por Java que contiene una referencia al objeto en el que se usa dicha variable permitiendo que el objeto pueda referenciarse a sí mismo. También es utilizado para diferenciar las variables locales de un método o constructor, de las variables del objeto.

EXCEPCIONES Y CONTROL DE ERRORES

try/catch Es una estructura para manejar excepciones y control de errores. La sentencia *try* inicia el bloque de código que será manejado en caso de errores y la

sentencia *catch* indica el tipo de excepción que se capturará. Si una excepción no es capturada por el listado de las clausulas catch, la JVM inicia el reporte e interrumpe todos los hilos de ejecución.

```
try{
    …
} catch(tipo1 exception){…}
  catch(tipo2 exception){…}
```

La herencia en java

Java permite el empleo de la herencia, característica muy potente que permite definir una clase tomando como base a otra clase ya existente. Esto es una de las bases de la reutilización de código, en lugar de copiar y pegar.

En java, la herencia se especifica agregando la palabra **EXTENDS** después del nombre de la clase. Al heredar de una clase base, heredaremos tanto los atributos como los métodos, mientras que los constructores son utilizados, pero no heredados.

2.3.1.2 INTERFACES

El concepto de ***interface*** fue creado con el objetivo de no privar a Java de la potencia de la herencia múltiple. Una interface es como una clase pero con dos diferencias: Sus métodos no realizan ninguna tarea porque están vacios y al definirla en vez de utilizar la palabra clave *"class"* se utiliza la palabra *"interface".* Sus variables son de tipo *public*

y requieren ser inicializadas dentro del cuerpo de la interface, una vez que toman un valor, este será constante para todos los objetos que implementen dicha interface.

Sin embargo, cuando una clase implementa una interface obligatoriamente se compromete a sobrescribir cada uno de los métodos de dicha interface. Caso contrario la clase en cuestión se convertiría en una clase abstracta, es decir, no se podría crear ningún objeto de ella.

Para que un método sobrescriba a otro ha de tener el mismo nombre, se le han de pasar los mismos datos, ha de devolver el mismo tipo de dato y ha de tener el mismo modificador que el método al que sobrescribe.

Por último decir que aunque una clase sólo puede heredar propiedades de otra clase puede implementar cuantas interfaces se desee, recuperándose así en buena parte la potencia de la herencia múltiple.

2.3.1.3 THREADS

Un sistema capaz de ejecutar varias aplicaciones simultáneamente es conocido como *"Sistema Multiproceso*". Cada aplicación que se arranca es un proceso dentro del ordenador. Los procesos están siempre "*protegidos*" unos a otros, es decir poseen su propia región de memoria del ordenador que es *"privada"* para los demás procesos. Multitarea describe un escenario muy similar al multiproceso.

Un Thread conocido como "*proceso ligero"* es una secuencia de código en ejecución dentro del contexto de un proceso que no puede ejecutarse por si solo. Dentro de cada proceso puede haber uno o varios hilos ejecutándose que requieren la supervisión de su proceso padre para correr. Cada proceso puede correr varios hilos los cuales están realizando diferentes tareas.

Por otro lado los distintos threads de un proceso comparten el mismo espacio de memoria, permitiendo a diferencia de los procesos, que un thread pueda acceder a las variables de los demás Threads. La Figura 9 muestra la relación entre hilos y procesos[8].

Figura.II.09. Relación entre hilos y procesos
Fuente: http://zarza.usal.es/~fgarcia/docencia/poo/01-02/trabajos/S3T3.pdf

VIDA DE UN THREAD

Recién Nacido: Se reserva un espacio de memoria para el Thread y se inicializan las variables, mediante la creación de un objeto de tipo Thread, pero aun no está en cola de ejecución.

[8] *Más información en http://zarza.usal.es/~fgarcia/docencia/poo/01-02/trabajos/S3T3.pdf*

Ejecutable: El Thread esta listo para ser ejecutado y se encuentra en cola de ejecución junto con otros Threads. Periódicamente la CPU cambia de Thread, es decir, coge el Thread que estaba ejecutando, lo pone en cola y escoge otro Thread de la cola de ejecución dependiendo siempre de la prioridad de los hilos, caso contrario los ejecuta secuencialmente.

Corriendo: El Thread está siendo ejecutado. Y libera al procesador cuando:

- Se acaba su tiempo de ejecución.
- Se bloquea con los métodos: sleep(), wait(), suspend(), o por alguna I/O.
- Termina su cuerpo de ejecución o se invoca al método stop(). Donde pasa a ser un thread muerto.

Bloqueado: El thread se bloquea, esperando que pase el tiempo de espera antes de acceder a cola de ejecución.

Muerto: El thread ya no existe. Sea porque se termino su cuerpo de ejecución o porque se lo mato con stop().

THREADS EN JAVA

Java es un sistema multiproceso, que soporta Threads de modo nativo (sin utilizar librerías), repartiendo cada uno de los hilos de una aplicación java entre las distintas CPU de un sistema multiprocesador, de modo transparente al programador.

Existen dos mecanismos para crear un thread en Java: implementando la interfaz Runnable que posee un único método *run* que se debe sobrescribir, o extendiendo la clase Thread.

Para ejecutar un thread en java se utiliza el método start() que llamara al método run. Y para detener el thread por un momento se utiliza el método sleep(int), siendo int el tiempo a esperar en milisegundos.

Clases Relacionadas con los Hilos

Java proporciona soporte para hilos a través de una simple interfaz y un conjunto de clases. La interfaz de Java y las clases que incluyen funcionalidades sobre hilos son las siguientes:

- Thread
- Runnable
- ThreadDeath
- ThreadGroup
- Object

Todas estas clases son parte del paquete Java.lang.

2.3.2 JAVA PARA TINI

2.3.2.1 DESARROLLO DE APLICACIONES PARA TINI

Típicamente el desarrollo y prueba de una aplicación para TINI requiere los siguientes pasos:

1. **Crear un archivo fuente.-** El archivo fuente contiene líneas de código con sintaxis de java que permiten declarar clases, métodos, objetos, variables, etc. que ayudan a crear aplicaciones que satisfagan el objetivo del programador.

   Este archivo generalmente se escribe en cualquier editor de texto ASCII como por ejemplo el blog de notas, siendo necesario guardarlo con la extensión .java.

2. **Compilar el archivo fuente.-** Un compilador es un programa que lee un código escrito en un lenguaje fuente y lo traduce a un lenguaje de máquina llamado lenguaje objeto. Como parte importante de este proceso de traducción, el compilador informa al usuario de la presencia de errores en el código fuente[9].

   El compilador javac se encuentra en el directorio *bin* por debajo del directorio *java*, donde se haya instalado el JDK. El compilador de Java cambia el código fuente Java a byte-codes, que son los componentes que entiende la *Máquina Virtual de Java* y que se guardan en un archivo ".class"

---

[9] *Existen varios compiladores para Java, en esta Tesis se utilizó la versión jdk1.6.0_13 pero se recomienda utilizar la ultima versión del Java Development Kit (JDK) de Sun disponible en www.sun.com*

3. **Convertir el archivo clase.-** TINIConvertor es una herramienta de línea de comandos provista por Dallas Semiconductor para convertir archivos ".class" de Java en aplicaciones de TINI. Esta herramienta de Java esta incluida en el TINI SDK. Puede ser encontrado en el archivo **tini.jar** en el directorio **bin** del **TINI SDK.**

   TINIConvertor es utilizado para obtener archivos ".tini", los cuales pueden ser cargados en el sistema de archivos TINI y ejecutados con el comando de Slush "java".

   El formato del archivo ".class" de Java es el núcleo de la fortaleza de Java como lenguaje de programación y contiene toda la información que una máquina virtual necesita para ejecutar los byte-codes de la clase en cualquier plataforma. Este formato usa ampliamente cadenas para hacer referencia a campos, métodos, y otras clases. Sin embargo, esto es muy exigente para el procesador de 8 bits que posee la TINI, por esta razón el TINIConvertor se utiliza para cambiar la referencia de cadenas a índices de tablas que contienen respectivamente campos, métodos y clases incluidas en el API de TINI, facilitando a la maquina virtual de TINI el manejo de las exigencias de Java.

   TINIConvertor no genera código nativo del microcontrolador de TINI, sino genera un archivo binario que contiene los byte-codes de java que pueden ser interpretados por la TINI JVM.

TINIConvertor tiene muchas opciones que pueden ser un poco confusas. Inclusive para las aplicaciones más simples, se deben utilizar tres argumentos obligatorios en TINIConvertor. El primero es el nombre del archivo ".class" que se desea convertir. El segundo es el nombre de la base de datos y el tercero es el nombre del archivo de salida. Estos argumentos pueden ser entregados en cualquier orden siempre y cuando se anteponga la etiqueta –f al nombre del archivo ".class", -d a la base de datos y –o al archivo de salida ".tini"; un esquema general del comando se muestra a continuación.

```
java -classpath  [Directorio bin del TINI SDK]\ tini.jar TINIConvertor
-f [PATH\Archivo.class]
-d [Directorio bin del TINI SDK]\ tini.db
-o [PATH\Archivo.tini]
```

4. **Cargar la imagen convertida.-** Mediante FTP se carga el archivo de salida ".tini" al sistema de archivos TINI.

5. **Ejecutar la imagen convertida.-** Se puede comprobar el funcionamiento de las aplicaciones de Java utilizando una sesión Telnet y el comando de Slush "java" seguido del nombre el archivo de salida ".tini"

2.3.2.2 CLASES Y MÉTODOS UTILIZADOS EN TINI

Para el desarrollo de esta Tesis, se ocupó un gran conjunto de métodos y clases incluidos en la Java API y en la TINI API. Las más importantes son:

1. BitPort.

2. comm API.
3. Sockets.
4. HTTPServer.

CLASE BitPort

El acceso a los pines individuales del microcontralador DS80C400 es logrado usando la clase BitPort contenida en el paquete *com.dalsemi.system*. El siguiente constructor de la clase BitPort es usado para conectar un objeto BitPort a un pin específico.

```
Public BitPort(byte bitname)
```

El pin es especificado por el parámetro bitname. Valores válidos para bitname son definidos como constantes públicas en la clase BitPort. Los nombres de constantes son formados concatenando la cadena “Port”, seguido del número de puerto y posición del Bit en el puerto. Por ejemplo, la siguiente declaración crea un objeto BitPort conectado al puerto del microcontrolador p5.3

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

Los métodos set y clear de la clase bitport son usados para controlar el estado del pin.

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

Cuando el método *clear* es llamado, el pin es manejado activamente a un low (a un 0 lógico). Si *set* es llamado siguiendo una invocación de clear, el pin será cambiado a un high (a un 1 lógico).

El método *read* devuelve el valor muestreado del pin. Si en el tiempo de muestreo el pin estaba en high (1 lógico), *read* devuelve 1. Caso contrario, *read* devuelve 0.

```
public int read()
```

JAVA COMMUNICATIONS API (comm API)

Definido por SUN Microsystems como una extensión de la plataforma de Java. Este API es definido y parcialmente implementado en el paquete javax.comm. La porción específica de la plataforma de las implementaciones del comm API para TINI, se encuentra en el paquete com.dalsemi.comm. Para la mayoría de las aplicaciones no existe una buena razón para usar las clases del puerto serial en el com.dalsemi.comm directamente.

En la TINI, los controladores para el puerto serial están siempre instalados y disponibles en el Runtime Environment y por lo tanto no se necesita de ningún archivo adicional.

A continuación se detalla las clases y métodos de Java indispensables para la comunicación a través del puerto serial en la TINI.

Adquiriendo y Configurando los Puertos Seriales

Se trabajó con objetos *SerialPort*. SerialPort es una subclase de la clase abstracta *CommPort*. CommPort provee una abstracción genérica del puerto de comunicaciones

además de proveer métodos para cambiar las configuraciones del puerto y realizar la adquisición de streams para la lectura y escritura de datos en la capa física del puerto.

Los objetos CommPort no pueden ser directamente creados usando el operador *new*, sino invocando el método *open* en el objeto CommPortIdentifier.

La clase CommPortIdentifier administra el acceso a los puertos definido por los controladores del puerto físico. También provee un mecanismo para notificar a la aplicación cuando el estado del dueño del puerto cambia, esto puede ser útil cuando múltiples aplicaciones desean compartir un único puerto. La habilidad de compartir puertos a través de múltiples procesos es soportada por la TINI.

Los objetos CommPortIdentifier pueden ser creados invocando uno de los siguientes métodos getPortIdentifier

```
public static CommPortIdentifier getPortIdentifier(String portName)
throws NoSuchPortException

public static CommPortIdentifier getPortIdentifier(CommPort port)
throws NoSuchPortException
```

Después de tener un objeto con CommPortIdentifier, se puede llamar a un método *open* para obtener el objeto CommPort.

```
public synchronized CommPort open(String appname, int timeout)
throws PortInUseException
```

El propietario de un puerto de comunicaciones es mutuamente exclusivo. En otras palabras, múltiples procesos no pueden acceder simultáneamente al puerto físico. El método *open* solo funciona cuando a obtenido acceso directo al puerto o cuando el valor del *input Time-Out* a terminado. Si el puerto está siendo utilizado por otro

proceso y culmina el Time-Out, mientras se espera que el proceso entregue la propiedad del puerto, se lanza una *PortInUseException.* El método *open* también necesita una representación del nombre de la aplicación. Esta cadena es usada para identificar el propietario del puerto. El propietario del puerto renuncia al mismo invocando el método *close* en el objeto CommPort.

```
public void close()
```

El objeto CommPort debe ser emitido a un objeto SerialPort antes de que se pueda alterar las configuraciones del puerto. La clase SerialPort provee métodos públicos "setter" para configurar parámetros individuales así también como los métodos públicos simétricos "getter" para consultar los parámetros en su valor actual. Existen algunos parámetros que son típicamente configurados antes de transmitir o recibir datos del puerto físico:

- Baud rate o velocidad de transmisión.
- Numero de bits de datos.
- Numero de bits de parada.
- Tipo de paridad.
- Control de flujo.

En la TINI, las configuraciones por defecto son 115200 bps, 8 bits de datos, 1 bit de parada, sin paridad y sin control de flujo. Los valores soportados para el número de bits de datos, bits de parada, control de flujo y tipos de paridad son definidos como constantes integer publicas en la clase SerialPort. Los parámetros más comunes para la

configuración del puerto serial pueden ser configurados con una invocación simple del método setSerialPortParams del objeto SerialPort.

```
public void setSerialPortParams(int baudrate, int dataBits, int stopBits, int parity)
throws UnsupportedCommOperationException
```

Todas las configuraciones son ingresadas como enteros. El número de bits de datos, bits de parada, y modo de paridad son suministrados usando las constantes del SerialPort. El baud rate es simplemente un valor entero igual a la velocidad deseada.

Si alguno de los valores de los parámetros es invalido, entonces setSerialPortParams lanzará una UnsupportedCommOperationException. Si esto ocurre no se completará ninguno de los cambios de los parámetros.

Control de Flujo

Otro escenario que puede ser configurado antes de iniciar la transferencia de datos seriales es el modo de control de flujo, el control de flujo es un mecanismo que le permite al receptor decirle al emisor que haga una pausa cuando el buffer receptor de datos interno esta cerca de llenarse, esto evita la perdida de datos por desbordamiento del buffer. Los siguientes módulos de control de flujo son soportados por el comm API

- Ninguno
- RTS/CTS (Control de flujo por Hardware)
- XON/XOFF (Control de flujo por Software)

Si no se especifica un control de flujo, ambos lados en la comunicación transmiten cuando necesitan, dejando sin protección contra el desborde del buffer de recepción. Esto no será un problema, dependiendo del protocolo serial empleado entre los extremos del canal de datos, sin embargo si un lado del canal transmite un stream continuo de datos, el receptor debería dedicarse a prestar servicios al buffer receptor o corre el riesgo de perder información. Este es un problema potencial en sistemas multitarea.

El modo de control de flujo es configurado mediante el método SetFlowControlMode del objeto SerialPort. El actual modo de control de flujo es utilizado por el controlador y puede ser consultado cualquier momento usando el método getFlowControlMode.

```
public void setFlowControlMode(int flowcontrol)
throws UnsupportedCommOperationException

public int getFlowControlMode()
```

El control de flujo puede ser especificado en un solo sentido de la comunicación, inclusive se puede utilizar un modo de control de flujo para las comunicaciones entrantes, y otro para las salientes.

En la TINI de los puertos seriales internos serial0, serial1, y serial 4, solo el último soporta todas las señales de control de flujo de Hardware (handshake).

Enviando y Recibiendo Datos Seriales

Una aplicación transmite y recibe datos seriales usando los métodos *read* y *write* en los streams de entrada y salida respectivamente. Los objetos java.io.InputStream y

java.io.OutputStream son adquiridos por los métodos getInputStream y getOutputStream en el objeto SerialPort.

```
public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException
```

Existen exactamente un InputStream y un OutputStream conectado a cada puerto serial. Sin embargo estos objetos bloquean los métodos de lectura o escritura hasta lograr completar la operación, es por esto que en la práctica se ocupan los eventos del puerto serial, que permiten realizar una comunicación serial eficiente.

Eventos del Puerto Serial

El comm API provee un mecanismo de notificación asincrónica de los eventos interesantes del puerto serial como un cambio de estado en las líneas de control o cuando la información esta disponible. En la TINI los eventos son propagados por un proceso demonio (daemon thread), que escucha los cambios de estado en los controladores del puerto serial. El daemon thread es creado cuando el primer evento listener es registrado usando el método addEventListener en la clase SerialPort, el argumento de este método requiere una instancia de una clase que implementa la interface SerialPortEventListener.

```
public void addEventListener(SerialPortEventListener listener)
throws TooManyListenersException

public void removeEventListener()
```

Los eventos listener son removidos automáticamente cuando se cierra el puerto.

Para cada evento del puerto serial existe un método con el prefijo notifyOn. Los listener escogen los eventos que requieren ser notificados. Invocando el método apropiado notifyOn* en el objeto serialPort. Por ejemplo para recibir información cuando hay datos disponibles en el puerto serial un listener invoca un método notifyOnDataAvailable.

```
public void notifyOnDataAvailable(boolean enable)
```

Un valor enable igual a TRUE habilita la notificación para el evento especificado, la notificación puede ser desactivada en cualquier momento con el mismo método cuando el valor enable es igual a FALSE.

La interface SerialPortEventListener define el método serialEvent que es invocado cuando ocurre la notificación de un evento que listener a solicitado.

```
public void serialEvent(SerialPortEvent ev)
```

Los listener invocan el método getEventType en el objeto SerialPortEvent y lo pasan al serialEvent para determinar el origen del evento.

```
public int getEventType()
```

getEventType devuelve el tipo del evento codificado en un integer. Existen algunos eventos del puerto serial definidos como constantes publicas en SerialPortEvent

Existen dos eventos importantes dentro del comm API de la TINI

- DATA_AVAILABLE
- OUTPUT_BUFFER_EMPTY

Estos proveen notificaciones del estado de los buffers de transmisión y recepción del driver serial. Cuando estos eventos son usados apropiadamente, le permiten a la aplicación maximizar el throughput (desempeño) serial con una mínima carga del CPU y sin tareas dedicadas.

Una aplicación puede sacar periódicamente los datos recibidos en el puerto serial (polling) mediante un método especial, sin embargo es muy ineficiente en el uso del CPU. Dependiendo del tipo de dispositivo serial y las actividades de la aplicación, es difícil determinar la frecuencia a la que se está revisando la entrada de datos, inclusive si no se soporta algún tipo de control de flujo, puede provocar perdida de datos o desbordamiento del buffer.

Una aplicación puede evitar el polling ocupando el evento DATA_AVAILABLE, este método es generado cuando los datos son recibidos por el puerto serial. Cuando el listener recibe la notificación de este evento generalmente lee todos los datos disponibles en el input stream y los entrega a otro proceso en la aplicación para ser procesados en el futuro. La ventaja del evento DATA_AVAILABLE es que no requiere bloquear ninguno de los métodos de lectura del input stream.

El manejar el flujo de datos de salida es menos crítico y más fácil, el evento OUTPUT_BUFFER_EMPTY es generado cuando el buffer de transmisión es vaciado. El listener puede usar este evento para mover información de un buffer arbitrariamente grande al puerto serial en bloques más pequeños y manejables. Este evento puede ser

usado como una alternativa al método write que se bloqueara si el buffer transmisor del serial esta lleno.

Cambios en el estado de las líneas de control definidos como entradas en los puertos seriales DTE pueden ser detectadas registrando cualquiera de los siguientes eventos.

- CD (Carrier Detect)
- CTS (Clear To Send)
- DSR (Data Set Ready)
- RI (Ring Indicate)

El método getNewValue de la clase SerialPortEvent puede ser usado para determinar el sentido de la transición.

```
public boolean getNewValue()
```

Devuelve un valor TRUE si la señal especificada esta activada caso contrario devuelve un valor FALSE. Esto se ocupa generalmente cuando se manejan las comunicaciones con un modem serial.

Puertos Seriales de Tini

Como se ha mencionado TINI Runtime Enviroment soporta hasta cinco puertos seriales. Los puertos seriales son designados desde serial 0 hasta serial 4. Los UARTs usados por el serial 0, serial 1, y serial 4 están integrados en el microcontrolador de TINI por esta razón se los llama puertos seriales internos. Los UARTs usados por el serial 2 y 3, requieren de un chip externo dedicado dual-UART, estos son referidos

como puertos seriales externos. El controlador del puerto serial interno no tiene que realizar mucho trabajo para cargar o descargar datos desde el UART.

Los puertos internos soportan el control de flujo XON/XOFF, un solo conjunto de líneas de control de Hardware son compartidas entre los puertos internos. Esto implica que solo un puerto a la vez puede ser utilizado con el control de flujo RTS/CTS, el puerto que posea las señales de control de flujo puede ser configurado usando el método setRTSCTSFlowControlEnable definido en la clase com.dalsemi.system.TINIOS.

```
public static boolean setRTSCTSFlowControlEnable(int portNumber,boolean enable)
throws UnsupportedCommOperationException
```

El número de puerto debe especificar uno de los puertos seriales internos. Si enable es TRUE, las señales de control de hardware serán dedicadas al puerto especificado. Si enable es FALSE, las señales son liberadas para ser usadas en la clase com.dalsemi.system.BitPort, como entradas y salidas TTL de propósito general.

Al desarrollar aplicaciones que controlen dispositivos seriales, se debe tomar en cuenta que al momento de arrancar la TINI transmite mensajes de progreso en el serial 0 a una velocidad de 115200 bps. Esto puede causar confusión para algunos dispositivos seriales embebidos porque es una información no solicitada y transmitida a una velocidad que puede ser diferente de la configurada para recibir datos.

SOCKETS

Los sockets son un sistema de comunicación entre procesos de diferentes máquinas de una red. Siendo los sockets el punto de comunicación por el cual un proceso puede emitir o recibir información. Los sockets utilizan una serie de primitivas para establecer el punto de comunicación, conectarse a una máquina remota en un determinado puerto que esté disponible, escuchar en él, leer o escribir y publicar información en él, y finalmente para desconectarse.

Java proporciona mecanismos potentes y a la vez sencillos para construir programas para redes a través de clases especializadas para establecer conexiones de red completas entre un servidor y un cliente, como son las clases Socket y ServerSocket incluidas en el paquete java.net.

La clase socket implementa una de las partes de la comunicación bidireccional entre un programa Java y otro programa en la red. Un objeto de este tipo proporciona métodos para la entrada/salida a través de streams que facilitan la lectura y escritura a través de sockets.

De forma adicional, la clase ServerSocket, implementa un socket el cual los servidores pueden utilizar para escuchar y aceptar peticiones de conexión de clientes. Este objeto no realiza el servicio, sino que crea un objeto Socket en función del cliente para realizar toda la comunicación a través de él.

Por otra parte, al intentar conectar a través de la Web, la clase URL y clases relacionadas (URLConnection, URLEncoder) son más apropiadas que las clases de sockets. Pero de hecho, las clases URL no son más que una conexión a un nivel más alto a la Web y utilizan como parte de su implementación interna los sockets.

Modelo de comunicaciones con Java

El modelo de sockets más simple es cuando el servidor establece un puerto y espera durante un cierto tiempo (timeout segundos), a que el cliente establezca la conexión. Cuando el cliente solicite una conexión, el servidor abrirá la conexión socket con el método accept(). Luego el cliente establece una conexión con la máquina host a través del puerto que se designe en puerto#. El cliente y el servidor se comunican con manejadores InputStream y OutputStream.

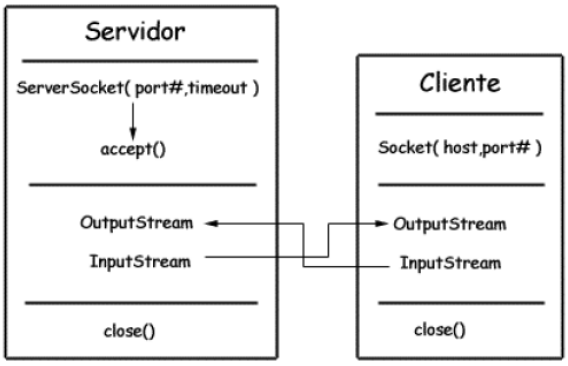


Figura.II.10. Funcionamiento de Sockets cliente – servidor
Fuente: http://www.docstoc.com/docs/23955078/Sockets-en-Java

Apertura de Sockets

Al programar el CLIENTE, la apertura del socket se obtiene a través de la creación de un objeto socket, mediante el siguiente constructor.

```
Socket Cliente;
Cliente = new Socket("nombre _server", numeroPuerto_server);
```

Donde *nombre_server* es un string que representa una dirección IP codificada en notación decimal punteada o un nombre DNS, y *numeroPuerto_server* es un número de puerto de 16 bits en el cual el servidor escuchara por peticiones del cliente.

La apertura del socket servidor se obtiene a través de la creación de un objeto ServerSocket, mediante el siguiente constructor.

```
ServerSocket Servicio;
Servicio = new ServerSocket(numeroPuerto_server );
```

A la hora de la implementación de un servidor también se requiere crear un objeto socket mediante el método *accept()* del objeto ServerSocket para que esté atento a las conexiones que le puedan realizar clientes potenciales y poder aceptar esas conexiones:

```
Socket socketServicio = null;
socketServicio = Servicio.accept();
```

Siendo importante en toda operación que se realice con sockets la captura de excepciones.

Creación de Streams

Se utiliza el DataInputStream para crear un stream de entrada que esté listo para recibir información.

```
DataInputStream entrada;
entrada = new DataInputStream( socket.getInputStream() );
```

El stream de salida es creado para enviar información a través de un socket utilizando las clases PrintStream o DataOutputStream:

```
PrintStream salida;
salida = new PrintStream( socket.getOutputStream() );
```

La clase PrintStream tiene métodos para la representación textual de todos los datos primitivos de Java. Sus métodos write y println() tienen una especial importancia en este aspecto. No obstante, para el envío de información también se puede utilizar DataOutputStream.

La clase DataOutputStream permite escribir cualquiera de los tipos primitivos de Java, muchos de sus métodos escriben un tipo de dato primitivo en el stream de salida. De todos esos métodos, el más útil quizás sea writeBytes().

Cierre de Sockets

Al final de la comunicación se cierran los canales de entrada y salida que se hayan abierto durante la ejecución de la aplicación mediante el método *close()* de cada objeto. Siendo importante cerrar primero los streams relacionados con un socket antes que el propio socket, ya que de esta forma se evitan posibles errores de escrituras o lecturas sobre descriptores ya cerrados.[10]

---

[10] *http://www.docstoc.com/docs/23955078/Sockets-en-Java*
http://www.webtaller.com/construccion/lenguajes/java/lecciones/sockets-java.php

HTTPServer

La clase HTTPServer en el paquete com.dalsemi.tininet.http implementa un muy simple servidor HTTP. Este soporta solo las solicitudes HTTP GET y responde información estática contenida en los archivos en algún directorio específico. Su intención no es ser una aplicación Web server dedicada sino proveer una capacidad de dar servicios HTTP que una aplicación puede ocupar. HTTPServer ofrece un desempeño razonable, y la sobrecarga que impone en una aplicación es relativamente pequeña. Logrando así que los requerimientos de procesamiento de las aplicaciones en foreground se vean poco afectados por el servidor.

Un objeto HTTPServer es creado usando uno de los siguientes constructores. Si el constructor no especifica el puerto que va a ser usado, el número de puerto TCP se pone por defecto igual a 80.

```
public HTTPServer() throws HTTPServerException
public HTTPServer(int httpPort) throws HTTPServerException
```

Si el puerto especificado se encuentra en uso por otro thread o proceso, el constructor no es capaz de crear un ServerSocket para escuchar las conexiones en el puerto específico devolviendo una HTTPServerException. El directorio raíz de HTTP y la página index por defecto son "webroot" e "index.html" respectivamente. Para cambiar estos valores una aplicación puede usar los siguientes métodos:

```
public void setHTTPRoot(String httpRoot)
public void setIndexPage(String indexPage)
```

Una vez creado el ServerSocket, se invoca un método serviceRequest. Este se bloquea indefinidamente esperando por una conexión entrante de un cliente.

```
public int serviceRequests() throws HTTPServerException
```

Este método provocará que el servidor cree un nuevo thread por cada solicitud. La aplicación típicamente dedica un solo thread que invoca el método serviceRequest en un bucle infinito. Ninguna otra acción es requerida de la aplicación por el HTTPServer para continuar procesando las solicitudes GET del cliente.

Ya que el HTTPServer es muy pequeño y simple por diseño, este no satisface todos los requerimientos de aplicaciones como lo haría un servidor web de propósito general. Sin embargo, existen servidores HTTP de grado comercial completamente caracterizados y poderosos escritos para la TINI que soportan el Java servlet API.

Aplicaciones que necesitan acceso a información provista por servidores web usan la clase familiar URL en el paquete java.net. Existe un parámetro de configuración que puede ser configurado por una aplicación usando las clases URL: Un servidor Proxy. Frecuentemente redes corporativas están protegidas detrás de un firewall y la única vía para que una solicitud HTTP pueda alcanzar el internet es a través de un servidor proxy. Un servidor Proxy es simplemente una máquina que recibe solicitudes de un cliente y las reenvía hacia otro servidor. El servidor proxy tiene privilegios especiales para comunicarse con los host fuera del firewall. La clase TININet provee los siguientes métodos para configurar el uso del servidor proxy.

```
public static boolean setProxyServer(String proxyServer)
public static boolean setProxyPort(int proxyPort)
```

El método setProxyServer toma el nombre del servidor como un string que representa una dirección IP codificada en notación decimal punteada o un nombre DNS. El

método setProxyPort toma un valor integer especificando el número de puerto de 16 bits en el cual el servidor proxy recibe las solicitudes HTTP. Tanto el servidor como el puerto son persistentes a través del inicio del sistema. Si setProxyServer es invocado con un string vacio, se deshabilitará el uso del servidor proxy. Por defecto, las clases de manejo de protocolos de URL no usan un proxy.[11]

2.3.3 NETBEANS

NetBeans es un entorno de desarrollo integrado (IDE por sus siglas en inglés). Esto quiere decir que integra todas las herramientas necesarias para poder desarrollar aplicaciones. Originalmente la programación en Java era algo complicada porque Java cuenta con una enorme cantidad de librerías y funciones que era preciso aprenderse de memoria, viendo esto muchas compañías construyeron diferentes entornos de programación para facilitar la tarea del programador. Entre los más populares surgió Eclipse que reinó como el único y más importante IDE de Java durante varios años. Posteriormente Sun Microsystems desarrollo su propio IDE, creado por las mismas personas que crearon Java años antes, este IDE fue NetBeans y después de varios años de desarrollo ha llegado a ser tan útil y poderoso como Eclipse o quizás un poco más. Dentro de la estructura del IDE de NetBeans se destaca lo siguiente:

---

[11] *Para mas información, Revisar el libro The TINI™Specification and Developer's Guide pg. 127-130*

Área de Proyectos: Mediante el manejo de pestañas, se puede acceder a clases, librerías, etc. que conforman el proyecto, los archivos que se encuentran almacenados en el proyecto y los servicios a los cuales accede el mismo.

Área de Programación: Es el lugar en donde se ingresan las líneas de código de java. Cuenta con un sistema de detección de errores y ayuda sensible al contexto.

Área de Compilación y Ejecución: Visualiza el proceso de compilación y ejecución de la aplicación java, informando todos los sucesos y errores, permitiendo depurar la aplicación y comprobar su funcionamiento.

Navegador: Permite explorar los contenidos del proyecto: clases, métodos, archivos, tareas, etc.

Otros: Como todo ambiente de desarrollo incluye gran variedad de barras de menús, herramientas, búsqueda, estado, etc.

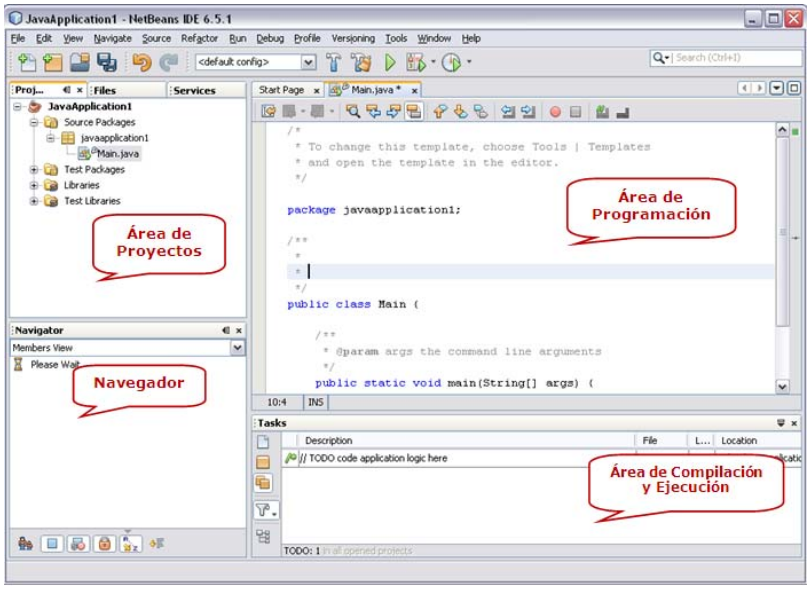


Figura.II.11. Entorno de Desarrollo de Netbeans en funcionamiento
Fuente: Los Autores

2.3.4 ANT en Java NetBeans

ANT es una herramienta de construcción que provee soporte para compilar, empaquetar, implementar y documentar aplicaciones de java. Las especificaciones para el comando *ant* son definidas en términos de *sentencias XML.* Un archivo de construcción define un proyecto que consiste de un conjunto de tareas. El proceso de construcción es ejecutado llamando al objetivo de las tareas, donde cada tarea es ejecutada por un objeto que implementa una interfaz particular Task. La principal ventaja de esta aproximación es la portabilidad, ya que los archivos de construcción son independientes del sistema operativo. Una desventaja no tan grande es que los comandos Shell no pueden ser usados para especificaciones.

La especificación *ant* es usualmente escrita en un archivo llamado *build.xml*. Este archivo sigue la sintaxis XML, y especifica un proyecto (identificado con la etiqueta *project*), que consiste en un conjunto de objetivos (identificados con la etiqueta *target*), el cual consiste en un conjunto de elementos task. Cada elemento corresponde a una tarea específica que será realizada por *ant.*

ant en TINI.- Ya que el proceso de compilación, conversión y cargado de aplicaciones en TINI es tedioso, es usualmente automatizado usando archivos make, archivos ant, o las interfaces de usuario graficas provistas por algunos ambientes de desarrollo.

Código de Conversión

Se utilizó una solución basada en ant ocupando el paquete llamado *tiniant*. Este paquete provee los elementos *task* de TINI que pueden ser usados para convertir un archivo *.class en un archivo *.tini. Los atributos mas importantes son *outputfile* para especificar el nombre del archivo de salida, *database* para especificar la base de datos del TINI API, *include* para los nombres de los archivos fuente y *classpath* para la localización del TINI SDK. También es necesario utilizar el archivo *convert* que especifica qué va ha ser incluido y qué va ha ser excluido. Todos estos parámetros deben ser incluidos en el archivo build.xml del proyecto mediante el siguiente código:

```
<!--La siguiente línea es usada para incluir la tarea tini haciéndola disponible para el
proyecto. -->
<taskdef name="tini" classname="net.geeba.ant.Tini"/>
<!-- Ubicación del SDK de TINI. -->
<property name="tini.dir" value="c:\tini1.18"/>
<!-- Base de Datos del TINI ROM API. -->
<property name="tini.db" value="${tini.dir}\bin\tini.db"/>
<!-- Ubicación del tini.jar -->
<property name="tini.jar" value="${tini.dir}\bin\tini.jar"/>
<!-- Creación del target convertir -->
<target name=" Tini_convertir ">
<!-- Directorio que contendrá los archivos.tini en la carpeta del proyecto -->
<mkdir dir=".\tini"/>
<!-- La tarea convertir convierte los archivos *.class de la carpeta “build” a
archivo.tini especificado por el atributo outputfile, usando la base de datos
especificada por la propiedad tini.db e incluyendo las definiciones de clases desde
tini.jar -->
<tini outputfile=".\tini\archivo.tini" database="${tini.db}" classpath="${tini.jar}">
<convert dir="build"/>
</tini>
</target>
```

Código de Transferencia

Otro proceso que puede ser automatizado mediante el uso de ant es la carga de archivos *.tini a través de FTP, en Netbeans las librerías de ant incluyen el soporte para

la utilización de este y otros protocolos, por lo tanto tan solo es necesario crear un nuevo target en el build.xml que permita la carga de archivos por FTP.

```
<!-- Creacion del target ftp-upload -->
<target name=" Tini_ftp-upload ">
<!-- Definir la dirección IP de la tarjeta TINI, el nombre de usuario, contraseña,y el
directorio donde se recibiran los archivos transferidos en la TINI -->
<ftp server="192.168.1.15"
userid="root"
password="tini"
depends="yes"
remotedir="/">
<!-- definir el directorio que contiene el archivo *.tini -->
<fileset dir="tini"/>
</ftp>
</target>
```

Para mayor detalle de la creación de aplicaciones TINI mediante Netbeans, referirse a la Guía de Aprendizaje Escrita, a partir de la Práctica #6

2.4 PIC16F887A

El microcontrolador PIC16F877 de Microchip pertenece a una gran familia de microcontroladores de 8 bits (bus de datos) que tienen las siguientes características generales que los distinguen de otras familias:

- Arquitectura Harvard
- Tecnología RISC
- Tecnología CMOS

Estas características se conjugan para lograr un dispositivo altamente eficiente en el uso de la memoria de datos y programa y por lo tanto en la velocidad de ejecución[12].

Oscilador

En la siguiente tabla se muestran los rangos de frecuencia así como los capacitores recomendados para un oscilador en base a cristal.

Tabla.II.III. Rangos de frecuencia de capacitores para oscilador.

Modo	Frecuencia típica	Capacitores recomendados	
		C1	C2
LP	32 khz 200 khz	68 a 100 pf 15 a 30 pf	68 a 100 pf 15 a 30 pf
XT	100 khz 2 Mhz 4 Mhz	68 a 150 pf 15 a 30 pf 15 a 30 pf	150 a 200 pf 15 a 30 pf 15 a 30 pf
HS	8 Mhz 10 Mhz 20 Mhz	15 a 30 pf 15 a 30 pf 15 a 30 pf	15 a 30 pf 15 a 30 pf 15 a 30 pf

Fuente: http://lc.fie.umich.mx/~jrincon/apuntes%20intro%20PIC.pdf

Cristal externo: En los tres modos mostrados en la tabla anterior se puede usar un cristal o resonador cerámico externo. En la siguiente figura se muestra la conexión de un cristal a los pines OSC1 y OS2 del PIC.

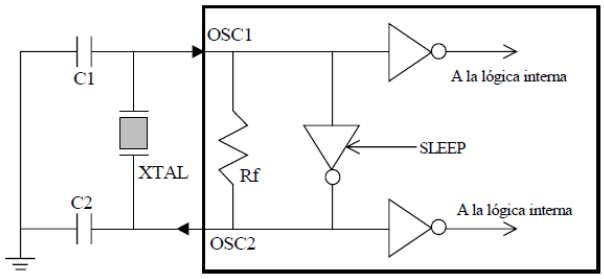


Figura.II.12. Conexión de un cristal externo al PIC
Fuente: http://lc.fie.umich.mx/~jrincon/apuntes%20intro%20PIC.pdf

[12] *http://lc.fie.umich.mx/~jrincon/apuntes%20intro%20PIC.pdf*

Características generales del PIC16F877

CPU:

- Tecnología RISC
- Sólo 35 instrucciones que aprender
- Todas las instrucciones se ejecutan en un ciclo de reloj, excepto los saltos que requieren dos
- Frecuencia de operación de 0 a 20 MHz (200 nseg de ciclo de instrucción)
- Opciones de selección del oscilador

Memoria:

- 8k x 14 bits de memoria Flash de programa
- 368 bytes de memoria de datos (RAM)
- 256 bytes de memoria de datos EEPROM
- Lectura/escritura de la CPU a la memoria flash de programa
- Protección programable de código
- Stack de hardware de 8 niveles

Reset e interrupciones:

- Hasta 14 fuentes de interrupción
- Reset de encendido (POR)
- Timer de encendido (PWRT)
- Timer de arranque del oscilador (OST)

- Sistema de vigilancia Watchdog timer.

Otros:

- Modo SLEEP de bajo consumo de energía
- Programación y depuración serie “In-Circuit” (ICSP) a través de dos pines
- Rango de voltaje de operación de 2.0 a 5.5 volts
- Alta disipación de corriente de la fuente: 25mA
- Rangos de temperatura: Comercial, Industrial y Extendido
- Bajo consumo de potencia:
- Menos de 0.6mA a 3V, 4 Mhz
- 20 µA a 3V, 32 Khz
- menos de 1µA corriente de standby (modo SLEEP).

Periféricos:

Tabla.II.IV. Modulos del PIC

Periférico	PIC16F874 PIC16F877	Características
3 a 5 Puertos paralelos	PortA, B,C,D,E	con líneas digitales programables individualmente
3 Timers	Timer0	Contador/Temporizador de 8 bits con pre-escalador de 8 bits
	Timer1	Contador/Temporizador de 16 bits con pre-escalador
	Timer2	Contador/Temporizador de 8 bits con pre-escalador y post-escalador de 8 bits y registro de periodo
2 módulos CCP	Captura	16 bits, 1.5 nseg de resolución máxima
	Comparación	16 bits, 200 nseg de resolución máxima
	PWM	10 bits
1 Convertidor A/D	AN0,...,AN7	de 10 bits, hasta 8 canales
Puertos Serie	SSP	Puerto Serie Síncrono
	USART/SCI	Puerto Serie Universal
	ICSP	Puerto serie para programación y depuración “in circuit”
Puerto Paralelo Esclavo	PSP	Puerto de 8 bits con líneas de protocolo

Fuente: http://lc.fie.umich.mx/~jrincon/apuntes%20intro%20PIC.pdf

Diagrama y descripción de pines del PIC16F877A

En la siguiente figura se muestra el diagrama de pines. La descripción de la función de cada unos de los pines se encuentra en el Anexo A.

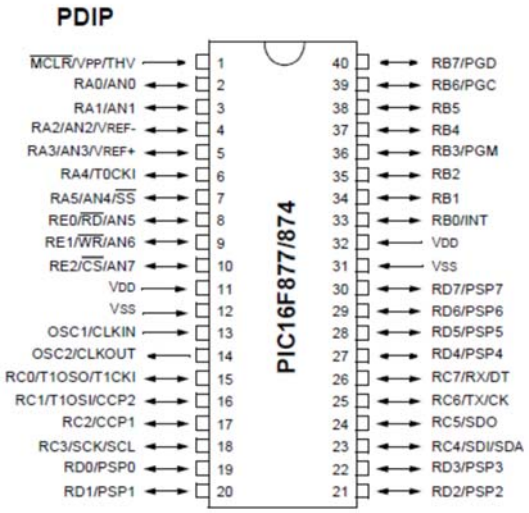


Figura.II.13. Empaquetado del PIC
Fuente: http://lc.fie.umich.mx/~jrincon/apuntes%20intro%20PIC.pdf

2.5 MICROCODE

Microcode studio es un ambiente de desarrollo integrado visual muy poderoso con capacidad de depuración en circuito ICD diseñado específicamente para el compilador PICBasic PRO de microEngineering Labs, es fácil configurar las opciones de su compilador, ensamblador y programador. Los errores de compilación y ensamblaje pueden ser fácilmente identificados y reconocidos usando la ventana de resultados de error. La sintaxis de programación de microcode es muy parecida a la programación en basic por lo que resulta muy sencillo programar. Adicionalmente brinda acceso a la

mayoría de los registros de los microcontroladores PIC y funciones adicionales para el manejo de comunicaciones seriales, LCD, PWM, etc.

En resumen, MicroCode Studio es una herramienta la cual permite escribir el código del programa, corregir errores de sintaxis, ordenar visualmente las subrutinas, etc. MicroCode queda enlazado con PICBASIC y IC-PROG, de manera que una vez que se termina el programa, se compila y se genera el archivo *.HEX, los programas son guardados en formato Picbasic *.BAS.

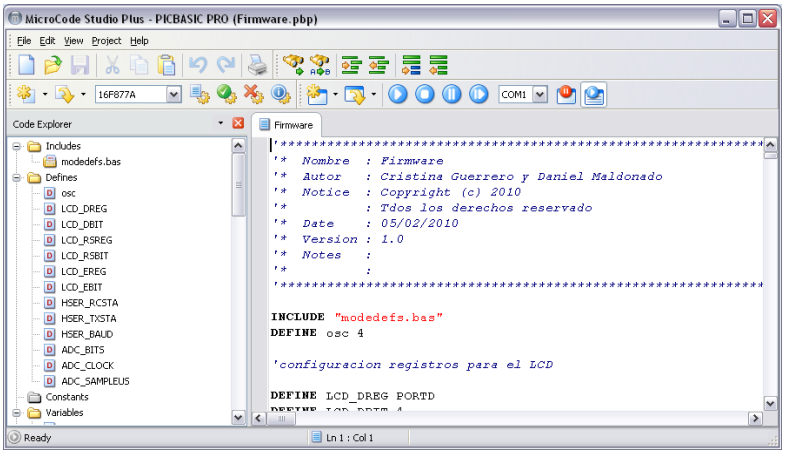


Figura.II.14. Ambiente de programación Microcode
Fuente: Los Autores.

CAPÍTULO III

TARJETA DE DESARROLLO TDDC

3.1 DISEÑO TDDC

3.1.1 REQUERIMIENTOS DEL SISTEMA

La siguiente figura muestra un modelo en el cual TINI es utilizado como un convertidor de protocolos o link entre un dispositivo embebido propietario y una red Ethernet. El dispositivo propietario puede comunicarse con el mundo exterior usando el puerto serial RS-232, el CAN, o tal vez algún tipo de interface paralela. La aplicación de Java ejecutándose en la TINI realiza la tarea de comunicarse con el dispositivo conectado en su lenguaje nativo (usando el protocolo de comunicaciones específico del dispositivo) y presenta el resultado al sistema remoto alcanzado a través de una Red TCP/IP. El link

proveído por TINI es bidireccional permitiendo al sistema remoto controlar y monitorear el sistema embebido propietario.

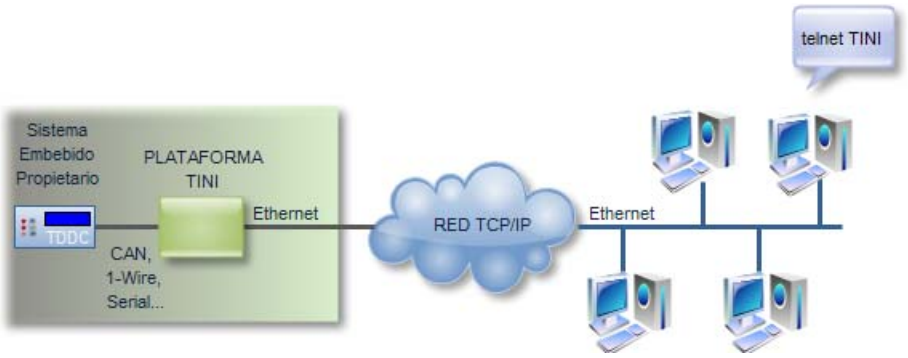


Figura.III.15. Conversión de Protocolos
Fuente: Libro The TINI™Specification and Developer's Guide

La Figura 15 se enfoca en un sistema embebido que controla y provee conectividad a un solo dispositivo a la Red. Sin embargo, TINI también puede servir para interconectar varios tipos de Redes reduciendo la brecha entre redes pequeñas de dispositivos mas baratos y livianos que una PC, y el mundo exterior de redes TCP/IP como es el internet.

En general, las aplicaciones de TINI son para interconectarse a otros equipos y redes más que con los humanos, es por esto que la Tarjeta de Desarrollo TDDC fue diseñada de manera que permita la simulación de algunos dispositivos embebidos propietarios que realicen tareas de control habituales y que permitan la exploración y explotación de la mayoría de las capacidades del sistema embebido TINI. Como son:

- Comunicación bidireccional serial RS232
- Capacidad de trabajar en Redes TCP/IP
- Monitorización de Sensores Digitales.
- Control y monitorización de Actuadores Digitales.

- Control y monitorización de velocidad de motores.
- Monitorización y Registro de datos adquiridos de Sensores Analógicos.
- Comunicación a través de Sockets.
- Explorar la capacidad de TINI para brindar monitorización y Control vía HTTP.

En base a los requerimientos descritos anteriormente, la Tarjeta de Desarrollo TDDC esta diseñada en dos etapas:

- Diseño de Hardware
- Diseño de Software

3.1.2 DISEÑO DE HARDWARE

El diseño de Hardware incluye los circuitos necesarios para poder satisfacer los requerimientos del Sistema.

El diagrama de bloques de la TDDC se muestra en la siguiente figura.

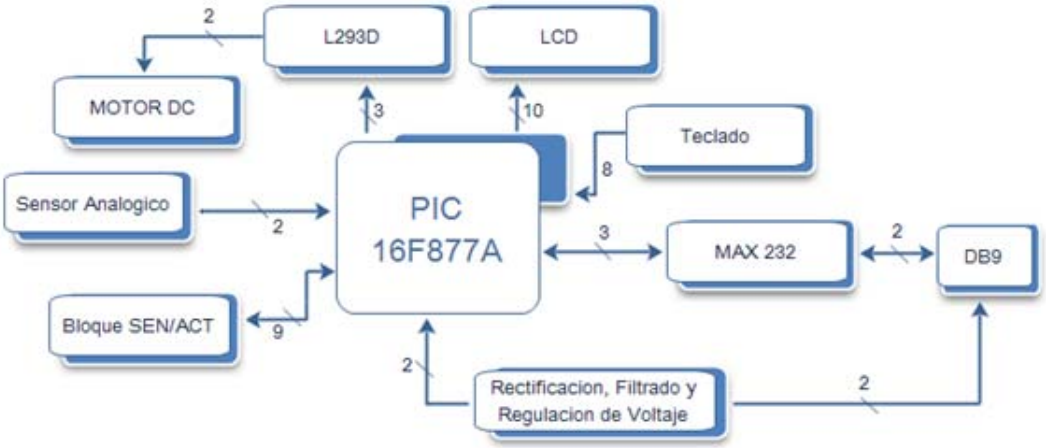


Figura.III.16. Diagrama de Bloques de la Tarjeta de Desarrollo TDDC
Fuente: Los Autores

Se utilizó un microcontrolador PIC16F877A por sus capacidades de I/O que incluyen 40 pines de conexión externa, modulo USART, 2 módulos CCP, manejo de interrupciones, convertidor Analógico Digital; además incluye 8k X 14 bits de memoria para la programación del PIC.

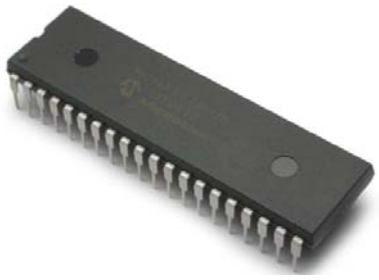


Figura.III.17. PIC16F877A
Fuente: http://aliatron.com/loja/catalog/images/encap_40p.gif

Para la interfaz gráfica de la TDDC se ocupó un LCD alfanumérico de 4 líneas por 20 columnas (4 X 20) compatible con Hitachi.

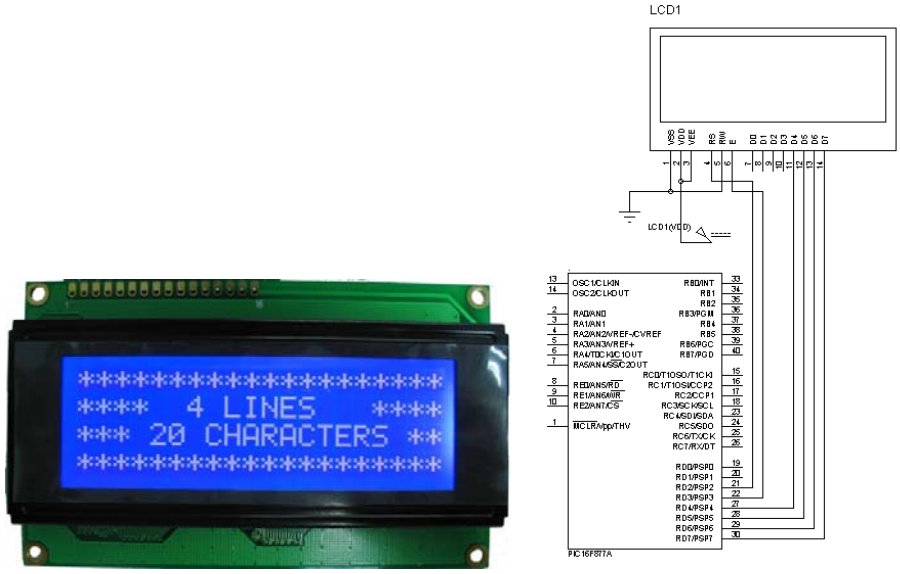


Figura.III.18. Esquema de conexión del LCD
Fuente: Los Autores

Como periférico de entrada, para el ingreso de caracteres alfanuméricos y selección de opciones, se ocupó un teclado matricial de 4 por 4.

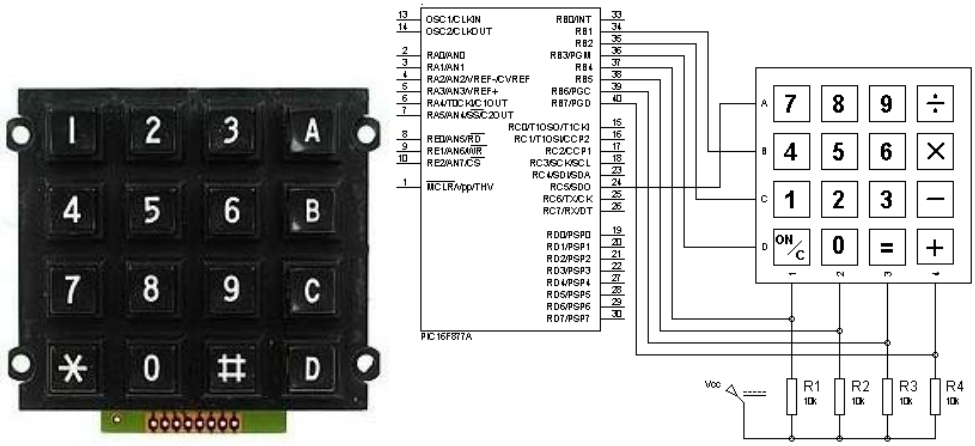


Figura.III.19. Esquema de conexión del teclado matricial
Fuente: Los Autores

Para establecer la comunicación serial con la TINI, se utilizó el modulo USART integrado en el PIC, que corresponde a los pines 25, 26 (Tx y Rx respectivamente).

Para poder manejar la recepción de datos de manera asíncrona, se ocupo la interrupción del puerto RB0, correspondiente al pin 33.

Fue necesario usar el integrado MAX232 para la conversión de niveles TTL – RS232, debido a que el conector DB9 del serial4 del socket DSTINIs400 trabaja en niveles RS232.

Para la conexión del MAX232, los pines de nivel TTL se interconectan al PIC, mientras que los pines de nivel RS232 se unen a un conector DB9 hembra, siguiendo el estándar de conexión de dispositivos DTE.

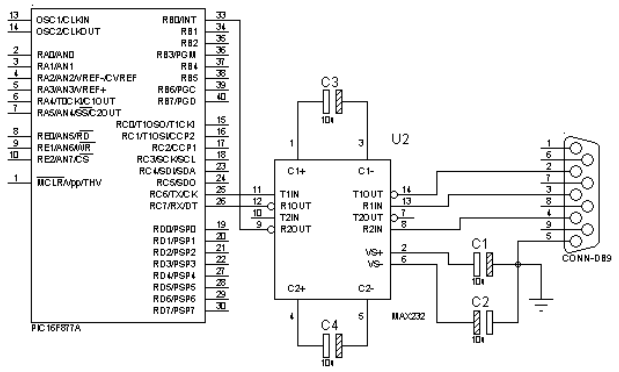


Figura.III.20. Esquema de conexión del integrado Max232 para la comunicación Serial.
Fuente: Los Autores

Para la simulación del comportamiento de los actuadores digitales, se utilizó LEDs activados mediante interruptores transistores. De igual manera, la simulación de los sensores digitales se logró con el uso de switches de dos posiciones.

Adicionalmente se permite la conexión en paralelo de actuadores reales a través de conectores, y de sensores reales a través de un interruptor de selección para la comprobación de los resultados.

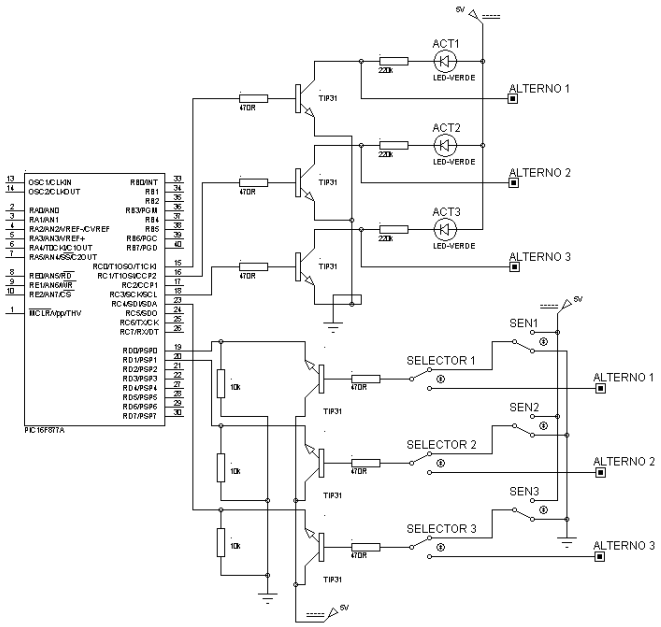


Figura.III.21. Esquema de conexión de los Sensores y Actuadores
Fuente: Los Autores

Para el control de la velocidad de un motor DC de 12 V se utilizó el modulo CCP1 del microcontrolador correspondiente al pin 17, generando la señal PWM necesaria para activar o desactivar el motor. Adicionalmente se utilizó el integrado L293D para controlar el sentido de giro del motor.

Se manejó uno de los dos puentes H provistos por el L293D para intercambiar la polaridad de alimentación DC del motor conectado a los pines 3 y 6 del chip. Este control se logró ingresando la señal digital proveniente de los pines 8 y 9 del PIC a los pines 2 y 7 del L293D respectivamente[13]. La señal del PWM ingresa al pin 1 correspondiente al enable del puente H, permitiendo activar o desactivar la alimentación DC en el motor logrando un control de velocidad.

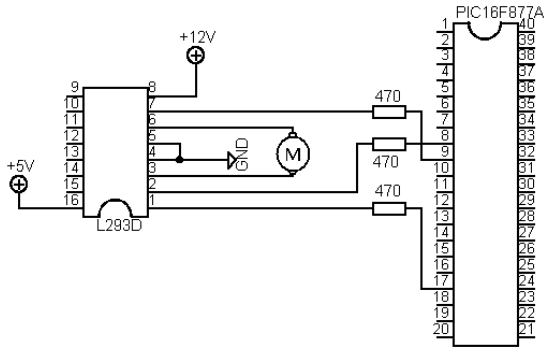


Figura.III.22. Esquema de conexión del Motor a través del puente H L293D
Fuente: Los Autores

Para conseguir una señal de un sensor analógico se utilizó un transductor que varía el valor de su resistencia de acuerdo a la intensidad luminosa del ambiente. A través de un divisor de voltaje se transforma esa variación en una señal eléctrica analógica de voltaje que es ingresada en el canal uno del convertidor analógico digital incorporado

---

*[13] Mas información en la Zona de Descarga, sección Datasheet / L293D.pdf disponible en la **GUIA DE APRENDIZAJE INTERACTIVA TINI/TDDC***

en el microcontrolador correspondiente al pin 2. El divisor de voltaje fue diseñado con la ayuda de un potenciómetro de 100k el cual permite la calibración del sensor.

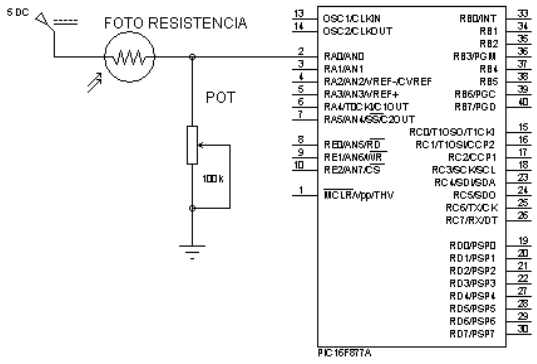


Figura.III.23. Esquema de conexión del Sensor Analógico
Fuente: Los Autores

Por último, se diseño una fuente de voltaje DC que suministre la energía necesaria para alimentar los circuitos y dispositivos electrónicos del Laboratorio TINI, definiendo los siguientes requisitos:

- Alimentación para la TDDC de 5V DC regulada, a un amperio max.
- Alimentación para la TINI de 5V DC regulada, a un amperio max.
- Alimentación para le motor DC, de 12V DC

Para reducir el voltaje se ocupo un trasformador de 110 a 6 Voltios de 2 Amps con derivación central; para la etapa de rectificación se utilizó dos Diodos Rectificadores de 3 Amps; un capacitor electrolítico de 4700µF a 16V para el filtrado obteniendo en esta etapa un Voltaje de 12V DC, suficiente para alimentar el motor. Este mismo voltaje ingresa a dos integrados 7805, los cuales regulan la señal a 5V DC que sirven para alimentar a la TDDC y la TINI independientemente.

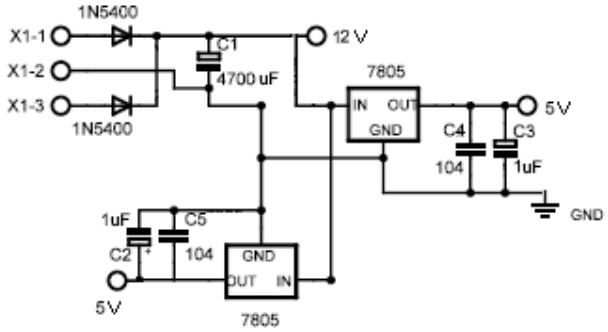


Figura.III.24. Esquema de la Fuente DC de la TINI y TDDC
Fuente: Los Autores

3.1.3 DISEÑO DE SOFTWARE

El software para la Tarjeta de Desarrollo TDDC, que de ahora en adelante será referido como el "Firmware TDDC" fue diseñado para cumplir con los requerimientos del sistema.

La Tarjeta de desarrollo TDDC permite simular la tarea especifica de cuatro sistemas embebidos propietarios cuyo único medio de comunicación con el exterior es utilizando el protocolo serial RS232. Los sistemas que simula la TDDC son los siguientes:

1. Comunicación bidireccional con terminales remotos a través de caracteres de texto ASCII.
2. Control y Monitorización local y remota de actuadores digitales. Monitorización local y remota de sensores digitales.
3. Control y Monitorización local y remota de velocidad PWM de un motor DC.
4. Monitorización local y remota de un Sensor Analógico.

El Firmware TDDC fue diseñado con los siguientes procedimientos para cumplir las tareas antes descritas.

- Configuración de los registros del PIC para manejar LCD, comunicación serial asincrónica, conversión Analógica Digital, I/O en cada uno de los puertos, y definición de variables necesarias para la programación.
- Diseño de una presentación inicial de la TDDC.
- Activación de la interrupción del pin RB0, modificando el registro INTCON con el valor 90h; definición del procedimiento RECIBE para manejar las interrupciones; definición de la variable OPERACIÓN que permite identificar el modo de operación de la TDDC cuando se origina la interrupción.
- Diseño de la presentación, inicialización de las variables y definición del procedimiento de los cuatro modos de operación.
- Generación del modelo del barrido del teclado.
- En el modo de operación Opción 1, se realiza el barrido del teclado, si se ingresa un número o carácter se visualiza en el LCD y se escribe en el puerto serial; si se recibe una interrupción, se descarga el dato serial del puerto y se lo imprime en el LCD; se diseñaron procedimientos para manejar una visualización correcta de los datos en el LCD. Si se escribe o se recibe un carácter asterisco se lo escribe en el puerto, se termina la operación y se regresa al menú principal.

- En el modo de operación Opción 2, se realiza el barrido del teclado y se comprueba el estado de los Sensores, si el estado de algún sensor a cambiado, se identifica el nuevo estado del sensor, se visualiza en el LCD y se informa el cambio en el puerto serial; a continuación se comprueba si se ha presionado alguna tecla, en caso de haberse seleccionado “1”, “2” o “3”, se cambia el estado del actuador seleccionado, se visualiza en el LCD y se informa el cambio en el puerto serial. En caso de recibir una interrupción, se descarga el dato serial, si se recibe un “0” se informa el estado de todos los sensores y actuadores en el puerto serial; si se recibe un “1”, “2” o “3”, se cambia el estado del actuador seleccionado, se visualiza en el LCD y se informa el cambio en el puerto serial. Si se escribe o se recibe un carácter asterisco se lo escribe en el puerto, se termina la operación y se regresa al menú principal.
- En el modo de operación Opción 3, se realiza el barrido del teclado y se presta atención a las interrupciones, en caso de recibir una interrupción, se descarga el dato serial y se lo procesa. Si se escribe o se recibe uno de los siguientes caracteres se realiza la acción indicada. Donde el primer carácter es aquel que se selecciona desde el teclado matricial, y el segundo carácter es el que se espera recibir a través del puerto serial.
    - 1, H: Configura la máxima velocidad.
    - 3, S: Detener por completo el motor.
    - 2, +: Incrementar la velocidad en 6.67%
    - 8, -: Decremento de la velocidad en 6.67%
    - 6, D: Configurar un sentido de giro horario.

- 4, I: Configurar un sentido de giro anti horario
- 5, V: Informa en el puerto serial la velocidad PWM actual.
- G: Informa en el puerto serial el sentido de giro actual.

Después se actualiza el ancho de pulso del PWM y el sentido de giro del motor, y en el LCD se visualiza el porcentaje y el sentido de giro actual. Todas las acciones que se realicen en el control de velocidad y sentido de giro son informadas a través del puerto serial. Si se escribe o se recibe un carácter asterisco se lo escribe en el puerto, se termina la operación y se regresa al menú principal.

- En el modo de operación Opción 4, se realiza el barrido del teclado y se presta atención a las interrupciones, cada segundo se realiza una conversión A/D de la señal del sensor y se la visualiza en el LCD como un porcentaje. En caso de recibir una interrupción se descarga el dato serial, si es el carácter “0”, se realiza una conversión A/D de la señal del sensor, se la visualiza en el LCD como un porcentaje y se escribe el porcentaje en el puerto serial. Si se escribe o se recibe un carácter asterisco se lo escribe en el puerto, se termina la operación y se regresa al menú principal.

3.1.4 IMPLEMETACIÓN TDDC

Para la implementación de la Tarjeta de Desarrollo TDDC se realizaron cada uno de los siguientes pasos que son ilustrados mediante las figuras.

Los circuitos descritos en el diseño de hardware fueron implementados usando una protoboard. Esto permitió realizar las pruebas de funcionamiento de cada parte de la TDDC. El diseño de software fue implementado mediante microcode y cargado al microcontrolador PIC con la ayuda de icprog.

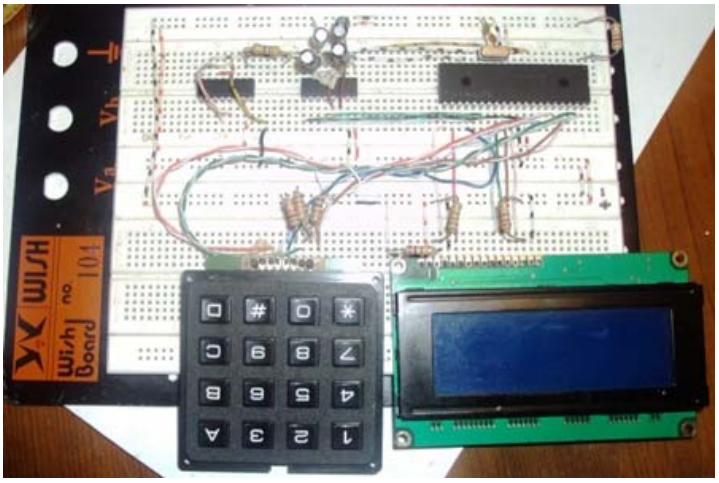


Figura.III.25. Implementación parcial TDDC mediante protoboard.
Fuente: Los Autores

Una vez comprobado el funcionamiento del diseño de Hardware y Software, se procedió a implementar el circuito impreso. Las pistas fueron diseñadas ocupando Eagle. El esquema de conexión de los Dispositivos y los diseños de las pistas de la placa pueden ser encontrados en el Anexo B y Anexo C respectivamente.

Las placas resultantes del circuito impreso son mostradas a continuación.

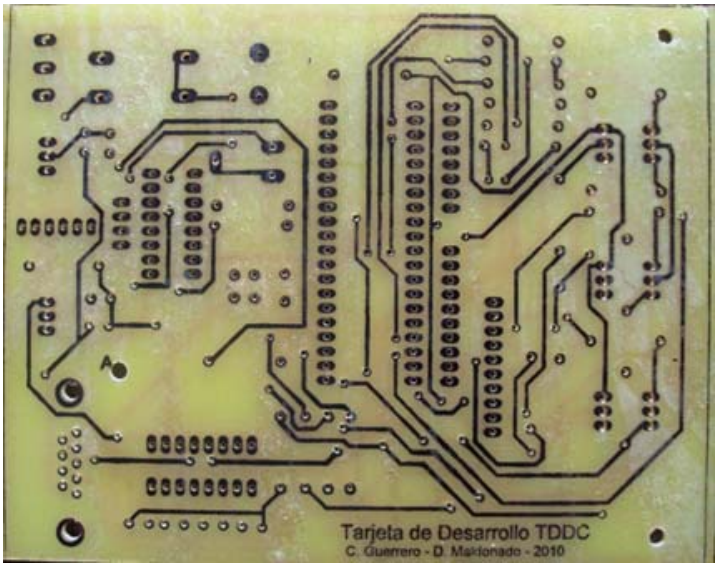


Figura.III.26. Circuito Impreso Tarjeta de Desarrollo TDDC, cara superior.
Fuente: Los Autores

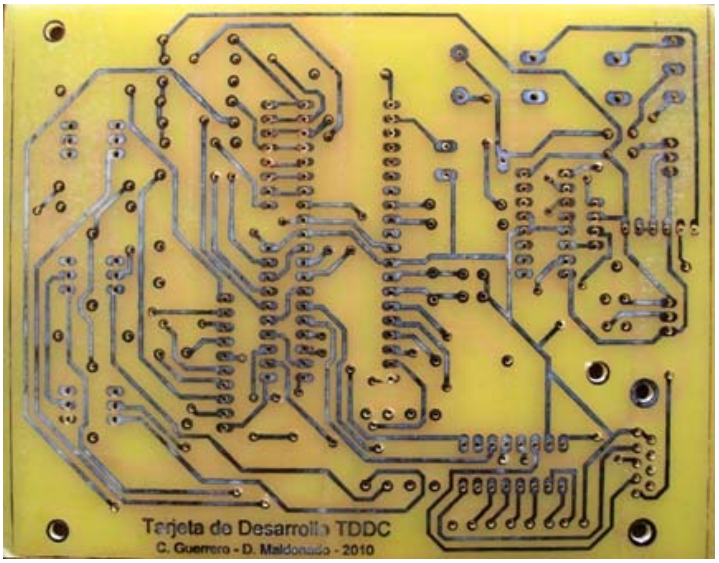


Figura.III.27. Circuito Impreso Tarjeta de Desarrollo TDDC, cara inferior.
Fuente: Los Autores

Después se procedió a soldar todos los componentes en sus respetivas posiciones, uniendo pistas entre cara y cara donde era necesario, gracias a esto se obtuvo una tarjeta como se ve en la siguiente figura.

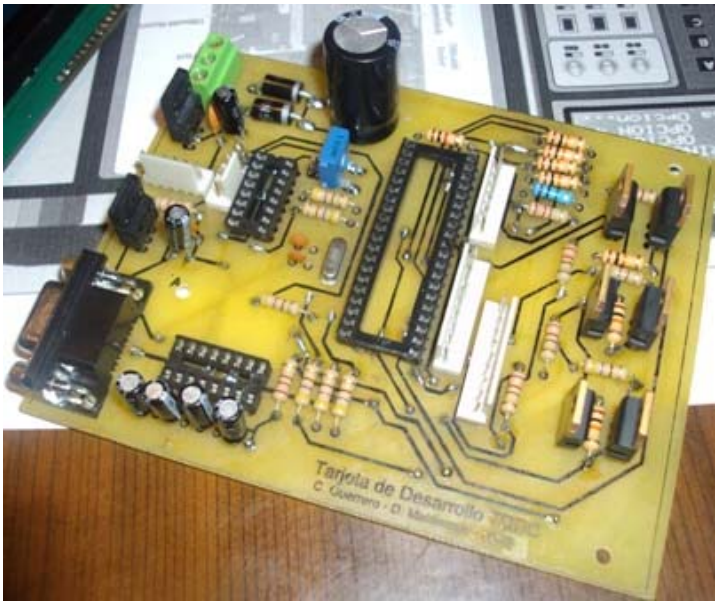


Figura.III.28. Tarjeta de desarrollo TDDC
Fuente: Los Autores

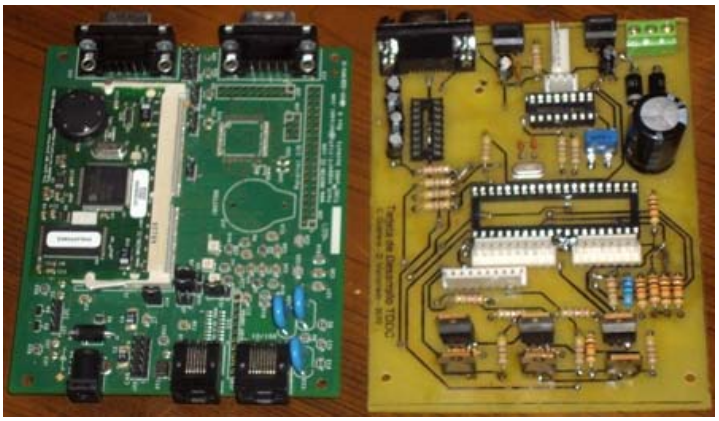


Figura.III.29. A la izquierda socket DSTINIs400, a la derecha TDDC.
Fuente: Los Autores

Inmediatamente se comprobó el funcionamiento de la tarjeta de Desarrollo como soporte para el aprendizaje del sistema embebido TINI, estas pruebas se las realizaron

aun si soldar definitivamente todos los componentes, ya que no tenían un soporte definitivo.

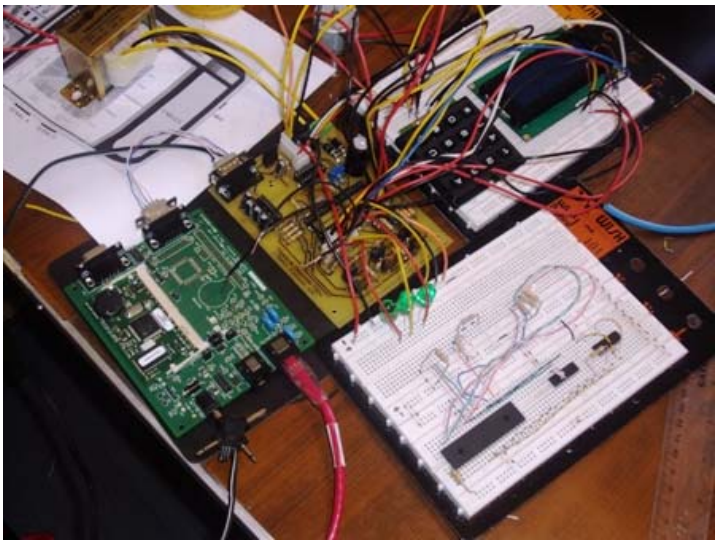


Figura.III.30. Comprobación del funcionamiento de la implementación de la TDDC.
Fuente: Los Autores

Con esto se completó la implementación de la Tarjeta de Desarrollo TDDC.

3.2 LABORATORIO TINI

Para cumplir con los objetivos de esta Tesis, fue necesario integrar la Tarjeta de Desarrollo TDDC con el sistema embebido TINI en un solo conjunto que permita el fácil acceso a las funciones de cada una.

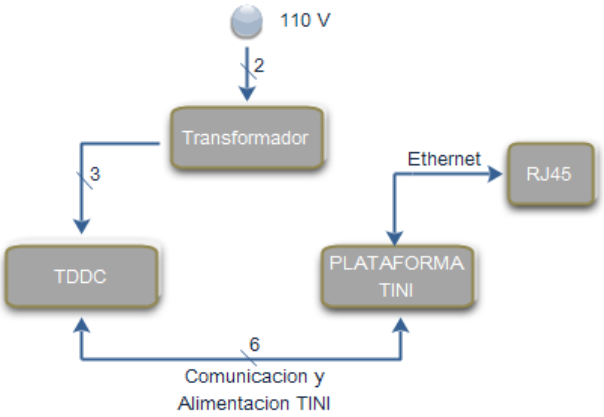


Figura.III.31. Diagrama de Bloques del Laboratorio TINI
Fuente: Los Autores

Es por esto, que se construyó el Laboratorio TINI, el cual permite la fácil realización de cada una de las prácticas de la Guía de aprendizaje TINI/TDDC.

Como componentes individuares que debieron ser ubicados e interconectados en el laboratorio TINI podemos mencionar:

- Sistema Embebido TINI, conformado por el DSTINIs400 y el DSTINIm400.
- Tarjeta de Desarrollo TDDC.
- Trasformador AC de 110V a 6V con derivación central.
- LCD Alfanumérico 4X20.
- Teclado Matricial.
- Motor DC de 12V.
- Fotoresistencia.
- Interruptor de Encendido.
- LEDs e interruptores.
- Conectores seriales, Ethernet y de alimentación AC.

Debido a la facilidad de manipulación y buena presentación se escogió como material de trabajo el acrílico, plastidecord, vinil y aluminio. Una vez identificado el espacio necesario para ubicar los componentes, se realizó las plantillas que permitieron cortar las partes de manera adecuada, como se muestra en las siguientes figuras.

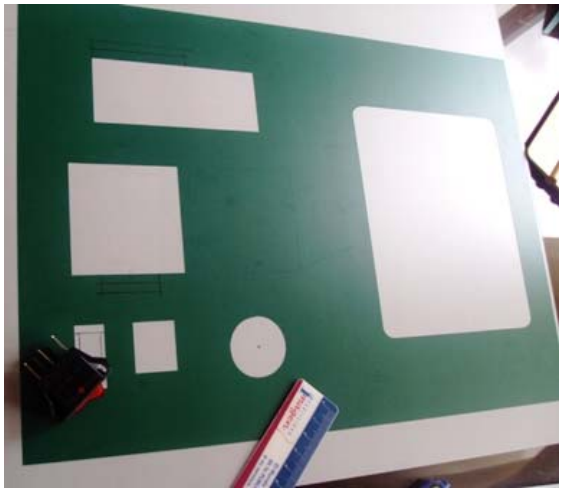


Figura.III.32. Plantilla para la ubicación de los componentes en el Laboratorio TINI
Fuente: Los Autores

Figura.III.33. Comprobación de los espacios distribuidos.
Fuente: Los Autores

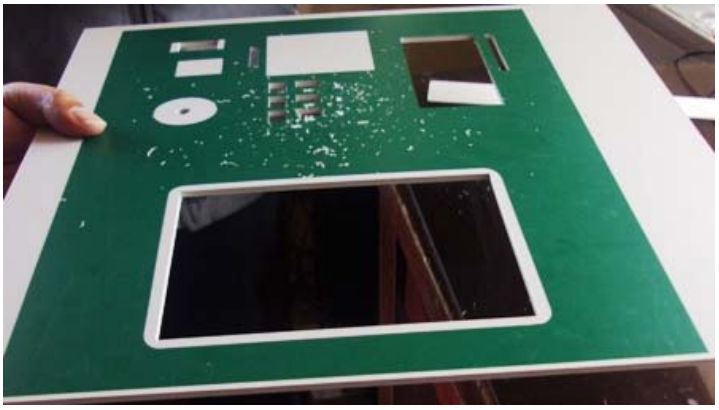


Figura.III.34. Realización de los cortes y perforaciones.
Fuente: Los Autores

Una vez terminada la distribución de los componentes de la parte superior del Laboratorio, se identificó el espacio para los conectores laterales y se formó la estructura que brindó el soporte a todo el equipo considerando los espacios necesarios en el interior de la caja.

Figura.III.35. Distribución del espacio interior del laboratorio TINI
Fuente: Los Autores

Figura.III.36. Comprobación de las medidas de armado de la caja.
Fuente: Los Autores

Una vez que se terminó y se comprobó el ensamblaje de la caja, se procedió a colocar los elementos de la tapa y de los laterales, soldando los dispositivos y asegurándolos mediante silicona. También se procedió a colocar los acabados del Laboratorio TINI.

Figura.III.37. Colocación del acabado de la parte superior del Laboratorio TINI.
Fuente: Los Autores

Figura.III.38. Ubicación de los componentes individuales en la parte superior del Laboratorio.
Fuente: Los Autores

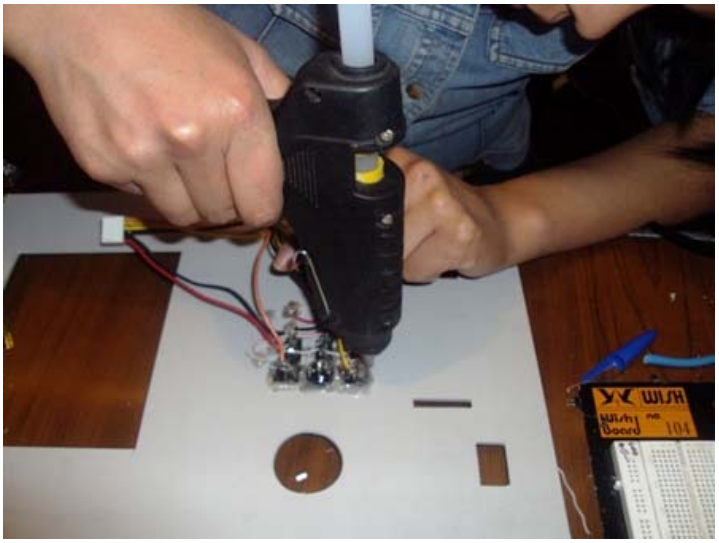


Figura.III.39. Soldadura y sujeción de los dispositivos ubicados en la tapa del Laboratorio
Fuente: Los Autores

Figura.III.40. Colocación Final de la tapa superior con los acabados.
Fuente: Los Autores

Una vez concluida la estructuración de la caja del Laboratorio TINI se ubicó e interconectó los dispositivos, para poder cerrar la tapa.

Figura.III.41. Interconexión de los elementos internos del Laboratorio TINI.
Fuente: Los Autores

Figura.III.42. Terminado final del Laboratorio TINI.
Fuente: Los Autores

CAPÍTULO IV

GUIA DE APRENDIZAJE TINI/TDDC

4.1 DISEÑO GUIA DE APRENDIZAJE TINI/TDDC

La Guía de Aprendizaje TINI/TDDC es una parte fundamental de la Tesis. Su propósito principal fue, junto con la Tarjeta de Desarrollo TDDC, implementar un equipo de aprendizaje didáctico que explore y explique las principales capacidades del sistema embebido TINI.

La Guía de Aprendizaje TINI/TDDC responde a los mismos requerimientos de la Tarjeta de Desarrollo TDDC. Además, permite familiarizarse con el método de desarrollo, depuración y ejecución de aplicaciones para el sistema embebido TINI, administración

del sistema mediante el Shell Slush; y entrega una gran cantidad de conceptos y práctica para el aprendizaje del Lenguaje de programación Java y del IDE NetBeans.

Para cubrir estos requerimientos, la guía de aprendizaje fue estructurada en tres partes principales.

Apartado Teórico del Sistema Embebido TINI y programación en Java.

Introducción a la administración y configuración básica del sistema de archivos y acceso a la red del Sistema embebido TINI a través del Shell Slush.

Interacción del sistema embebido TINI con la Tarjeta de Desarrollo TDDC mediante aplicaciones en Java.

Para la elaboración de cada una de las prácticas de la Guía de Aprendizaje se siguió un diseño secuencial. Es decir, cada uno de los conocimientos impartidos en una Práctica de la Guía es la base fundamental para las siguientes prácticas, logrando integrar la mayor parte de las capacidades de la TINI.

Los tópicos incluidos en la Guía de Aprendizaje se enlistan en la Tabla V.

Tabla.IV.V. Estructura de la Guía de Aprendizaje

PRESENTACIÓN
RECOMENDACIONES PARA EL ESTUDIANTE
TEMA I. INTRODUCCIÓN TEÓRICA
TINI
Java
TEMA II. ADMINISTRACIÓN Y CONFIGURACIÓN BASICA DE TINI
Practica # 1
Carga del Runtime Environment y Slush
Practica # 2
Sesiones de Terminal seriales TINI
TEMA III. ADMINISTRACIÓN Y CONFIGURACIÓN DE TINI A TRAVÉS DE ETHERNET
Practica # 3
Sesiones de Terminal Telnet, examen detallado del Shell Slush
Practica # 4
Pila de protocolos TCP/IP soportados por TINI
TEMA IV. LABORATORIO TINI CON JAVA
Practica # 5
Creando una aplicación básica TINI
Práctica # 6
Creando una aplicación TINI Con NetBeans
Práctica # 7
Manejo de I/O de TINI - LED Intermitente
Practica # 8
Comunicación Serial Bidireccional TINI - TDDC
Practica # 9
Monitorización y Control de sensores y actuadores Digitales
Practica # 10
Control y Monitorización de velocidad PWM
Practica # 11
Monitorización y Registro de información de un Sensor Analógico
Practica # 12
Conversión de Protocolos – Manejo de Sockets en TINI
Practica # 13
Servidor HTTP
Practica # 14
Control de un Actuador Digital a través de un Web Browser.

Fuente: Los Autores

4.2 DESCRIPCION DE LOS TOPICOS DE LA GUIA DE APRENDIZAJE TINI/TDDC.

En los apartados ***Presentación*** y ***Recomendaciones para el Estudiante,*** se da una pequeña introducción al objetivo y contenidos de la Guía de Aprendizaje,

enmarcándola como una parte de esta Tesis y haciendo referencia a la utilización de la Tarjeta de Desarrollo TDDC y al sistema embebido TINI.

La Guía de Aprendizaje TINI/TDDC inicia con un capitulo teórico amplio sobre el sistema embebido TINI y una introducción a la programación Java, brindando los conocimientos básicos necesarios para poder cumplir los objetivos de cada una de las prácticas.

Practica # 1.- Carga del Runtime Environment y Slush

Se procedió a cargar el "Runtime Environment" de TINI y el interpretador de comandos "Slush" necesarios para la operación del Sistema embebido TINI, a través de un Software cargador serial "MTK" proveído por Dallas Semiconductor. Se describió la instalación y configuración del software MTK, la carga del Runtime Environment y Slush, y la inicialización del TINI Firmware.

Practica # 2.- Sesiones de Terminal seriales TINI

Se comprobó el acceso al Sistema TINI mediante sesiones de terminal conectadas a través del puerto serial, utilizando el Software "MTK" provisto por Dallas Semiconductor. Se describió la inicialización de la comunicación serial entre una PC y el sistema embebido TINI, la autenticación de la Sesión, algunos comandos básicos de administración del Shell Slush, y por último se indicó como configurar los parámetros básicos de Red Ethernet en el sistema embebido TINI, habilitando y comprobando su capacidad de trabajar en Red.

Practica # 3.- Sesiones de Terminal Telnet, examen detallado del Shell Slush

Se comprobó el acceso al Sistema TINI mediante sesiones de terminal Telnet conectadas a través de una red Ethernet, utilizando un cliente telnet. Se describió los pasos necesarios para establecer una comunicación Ethernet entre una PC y el sistema embebido TINI, y se hizo un estudio detallado de los comandos de Slush.

Practica # 4.- Pila de protocolos TCP/IP soportados por TINI

En esta práctica se especificaron los protocolos TCP/IP soportados por la TINI, mediante la configuración de los parámetros de Red a través del comando ipconfig y la comprobación de funcionamiento de dichos protocolos. Se estudió detalladamente las opciones y usos del comando ipconfig. También se comprobó el uso del servidor FTP para la carga de archivos en la TINI.

Practica # 5.- Creando una aplicación básica TINI

Mediante esta práctica, se explicó y se experimento el procedimiento para el desarrollo y prueba de aplicaciones para la TINI mediante el JDK y el TINI SDK. Se detallo el código y comandos utilizados para la creación del archivo fuente, compilación, conversión del archivo compilado en un formato admisible por TINI, cargado de la imagen convertida, y ejecución de la Aplicación en TINI.

Práctica # 6.- Creando una aplicación TINI Con NetBeans

Se presentó el Entorno de Desarrollo Integrado de Java NetBeans, en el cual se demostró la facilidad para crear una Aplicación para la TINI en comparación con el procedimiento de la práctica anterior. Adicionalmente se describió el uso de la herramienta *Ant*, que permite automatizar los procedimientos de conversión y cargado de las aplicaciones en la TINI.

Práctica # 7.- Manejo de I/O de TINI - LED Intermitente

Se estudio la posibilidad de ocupar los pines que provee el microcontrolador DS80C400 como Entradas o Salidas digitales de propósito general, y como pueden ser manipuladas directamente por una aplicación de Java. Se indica también la forma de incluir librerías de TINI en el IDE NetBeans y acceder a las clases específicas de la plataforma. Mediante un LED incorporado en el modulo DSTINIm400 se demostró la manipulación directa de un Pin del microcontrolador.

Practica # 8.- Comunicación Serial Bidireccional TINI - TDDC

Los sistemas basados en TINI pueden ser usados para conectar dispositivos propietarios a Redes Ethernet, dependiendo de las capacidades de entrada y salida del sistema propietario. Normalmente este es un trabajo que puede ser realizado por una PC o estación de trabajo. Sin embargo, TINI puede hacer el mismo trabajo por un costo y tamaño menor. En esta práctica se introdujo la funcionalidad de la Tarjeta de Desarrollo para simular el funcionamiento de algunos Sistemas Propietarios, en este caso se utilizó el modo de operación “opción 1”.

El medio de comunicación utilizado entre la TINI y la TDDC es un enlace serial basado en el protocolo RS232. Se demostró que es posible implementar una aplicación en Java que pueda comunicarse con un sistema propietario en su protocolo nativo, conociendo previamente el funcionamiento del protocolo de software de este sistema, es decir, la forma en la que se comunica la TDDC en el modo de operación “opción 1”. Para el manejo del puerto serial en la TINI se usaron las clases SerialPort, y se implementaron las interfaces Runnable y SerialPortEventListener, introduciendo el uso de Threads; y adicionalmente se ocupo uno de los de pines I/O del microcontrador para poder satisfacer los requerimientos de comunicación de la TDDC.

El modelo de comunicación serial implementado en esta práctica, brinda la plantilla de soporte a cada una de las siguientes prácticas de la Guía de Aprendizaje.

Practica # 9.- Monitorización y Control de sensores y actuadores Digitales.

Una tarea tradicional de control de los sistemas propietarios, implementada en la Tarjeta de Desarrollo TDDC opción 2, es la monitorización y control de sensores y actuadores Digitales. La TDDC brinda el mecanismo para realizar el control y monitorización a través de una comunicación por el puerto serial. En esta práctica, se enseñó como desarrollar una aplicación en Java que implemente el protocolo adecuado para poder comunicarse con la Opción 2 de la TDDC a través de un puerto serial RS232. Además, debido a que el acceso a la aplicación que se ejecuta en la TINI es a través de una conexión Telnet, se logra generalizar la manera en la que TINI brinda

el soporte necesario para realizar una Monitorización y Control Remoto de un sistema embebido propietario.

Practica # 10.- Control y Monitorización de velocidad PWM

Mediante la Opción 3 de la TDDC, se realiza el Control y Monitorización local y remota de la velocidad y sentido de giro de un motor DC mediante PWM y puentes H. En esta práctica se evidencia la necesidad de estructurar un modelo de comunicación, que identifique la función de cada byte recibido o transmitido serialmente, permitiendo reafirmar la versatilidad de Java para adaptar mediante programación la forma en que la TINI se comunica con el sistema embebido propietario.

Practica # 11.- Monitorización y Registro de información de un Sensor Analógico

Para muchas aplicaciones reales se requiere llevar un registro del estado de un sensor para poder hacer el estudio de un entorno. Este registro debe estar acompañado de una etiqueta de fecha y hora que permita su correcta identificación. La Opción 4 de la TDDC informa a través de una comunicación serial el estado de un sensor analógico cuando recibe una solicitud.

En esta práctica se aprendió como crear y escribir en un archivo almacenado en la memoria de la TINI la información recibida de un dispositivo propietario, mediante una aplicación en Java. Además se ve directamente como manejar dos Threads en la misma aplicación, donde uno de ellos realiza el registro automático cada cierto periodo de tiempo, y el otro un registro manual el momento en que es solicitado por el Usuario.

Practica # 12.- Conversión de Protocolos – Manejo de Sockets en TINI

Una de las principales aplicaciones de TINI es actuar como convertidor de protocolos, es decir que puede recibir datos a través de una conexión Ethernet mediante paquetes TCP/IP, y trasmitir estos datos en el puerto serial ocupando el protocolo nativo del dispositivo propietario; y viceversa.

En esta práctica se ocupo las clases Sockets, y mediante la generación de una aplicación servidor que se ejecuta en la TINI, se pudo comunicar la Opción 1 de la TDDC con una aplicación de Java ejecutándose en un computador cliente, comprobando así la comunicación bidireccional de dispositivos que ocupan distintos protocolos físicos y lógicos ocupando a la TINI como un intermediario.

Practica # 13.- Servidor HTTP

Se utilizó el modulo HTTPServer incluido en el TINI SDK, que implementa un pequeño y liviano servidor Web mediante sockets. De esta manera se pudo ver la capacidad de TINI para brindar acceso remoto al sistema a través de un cliente tan simple y popular como es un explorador Web.

Practica # 14.- Control de un Actuador Digital a través de un Web Browser.

Mediante la implementación de un Applet en Java, y el diseño de una página web que ejecute el Applet, fue posible demostrar el acceso al control de uno de los actuadores de la TDDC en la Opción 2, mediante un simple explorador Web.

En esta práctica se conjugan casi en totalidad todos los conocimientos adquiridos a través de la Guía de Aprendizaje TINI/TDDC, ya que se conjuga el uso de las clases HTTPServer, Sockets, SerialPort, BitPort, y Threads.

De esta manera se concluye la Guía de Aprendizaje, dejando las bases y la Propuesta de la creación de una aplicación basada en Web que permita el acceso a cada una de las Opciones de la Tarjeta de Desarrollo TDDC, para acentuar y confirmar los conocimientos adquiridos a través de este equipo de aprendizaje didáctico.

4.3 GUÍA DE APRENDIZAJE INTERACTIVA TINI/TDDC

La Guía de Aprendizaje Interactiva TINI/TDDC fue implementada como un complemento a la Guía de Aprendizaje TINI/TDDC, que facilita el acceso y la comprensión de cada uno de los contenidos de las prácticas de la Guía para el aprendizaje del sistema embebido TINI. Mediante un interface de reproducción flash, se acceden y se ubican rápidamente los contenidos individuales de la Guía, además integra un video Tutorial de los procedimientos más importantes de cada una de las prácticas, y el código Fuente.

La Guía de Aprendizaje Interactiva TINI/TDDC incluye una zona de descargas, en la cual se puede acceder a todo el Software necesario para la realización de las prácticas, Libros Electrónicos, y documentos técnicos a los cuales se hizo referencia a lo largo de toda la tesis. Además se puede acceder a las aplicaciones implementadas de cada una

de las prácticas. Mediante las siguientes figuras se describe brevemente la conformación de la Guía de Aprendizaje Interactiva TINI/TDDC.

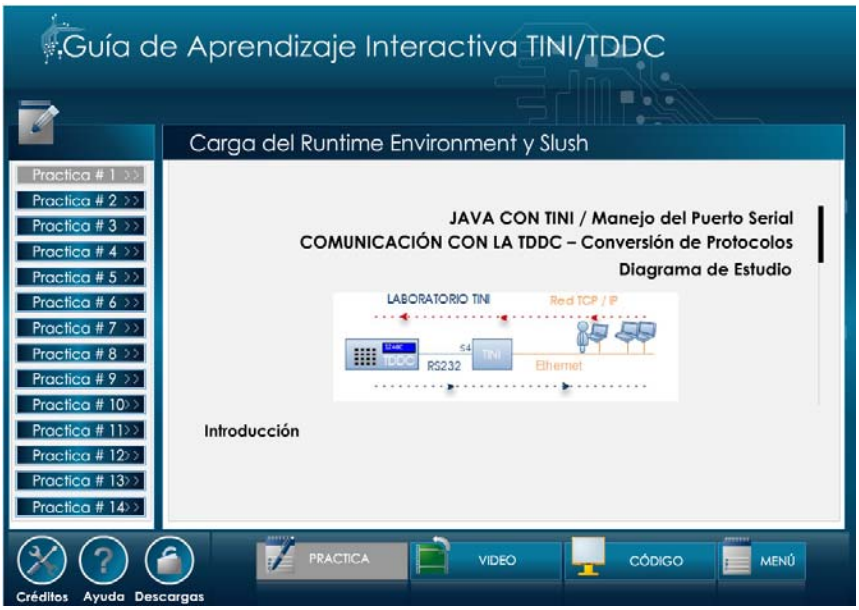


Figura.IV.43. Visualización del contenido de la práctica en la Guía de Aprendizaje.
Fuente: Los Autores

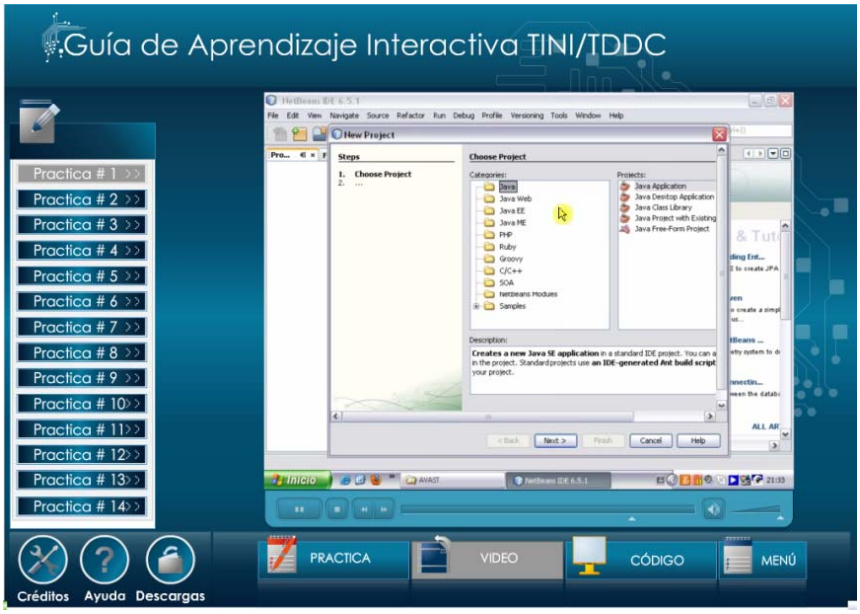


Figura.IV.44. Reproducción del Video Tutorial de la práctica.
Fuente: Los Autores

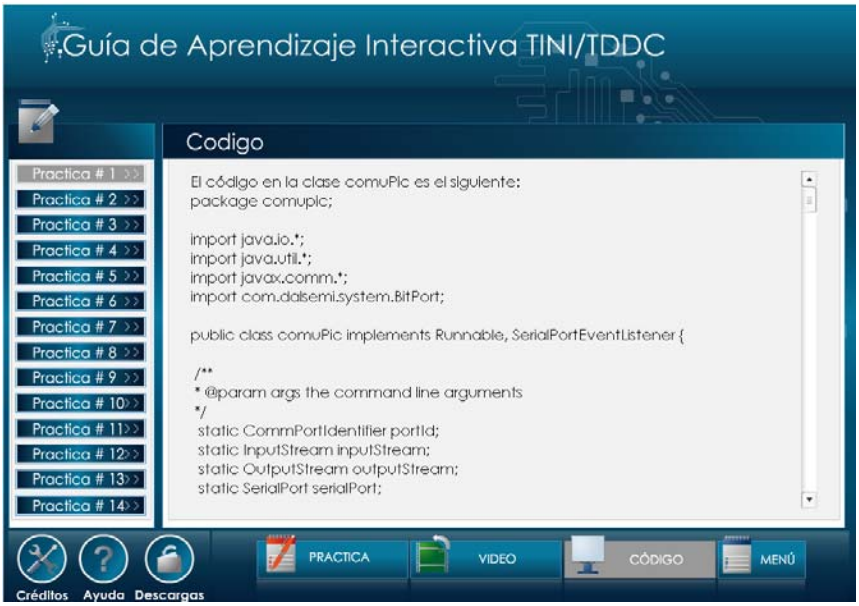


Figura.IV.45. Accediendo al código fuente de la Práctica.
Fuente: Los Autores

Figura.IV.46. Zona de Descarga de la Guia de Aprendizaje Interactiva TINI/TDDC
Fuente: Los Autores

CAPÍTULO V

PRUEBAS Y RESULTADOS

5.1 FUNCIONAMIENTO DEL LABORATORIO TINI

Una vez finalizada la construcción e implementación del Laboratorio TINI y la Guía de Aprendizaje TINI/TDDC se procedió a comprobar los resultados obtenidos de los cuatro modos de operación de la Tarjeta de Desarrollo TDDC.

La TDDC visualiza en su pantalla un Menú Principal de los Modos de Operación.

Figura.V.47. Visualización del Menú Principal en el LCD de la TDDC
Fuente: Los Autores

Figura.V.48. Ejecución de las pruebas del Laboratorio TINI
Fuente: Los Autores

En el modo de operación Opción 1, se realizó el envió bidireccional de caracteres desde la consola de la TINI y desde el teclado matricial del Laboratorio TINI.

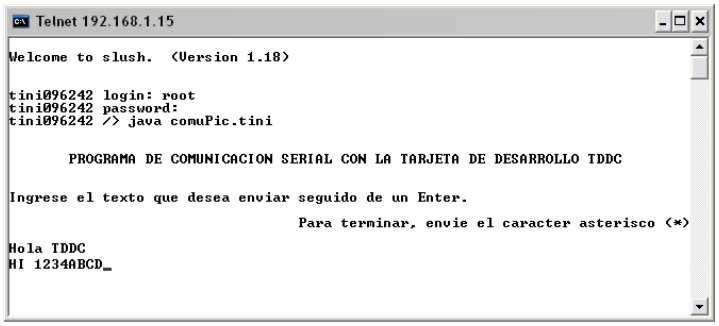


Figura.V.49. Acceso desde la consola TINI al modo de operación Opción 1
Fuente: Los Autores

Figura.V.50. Comprobación del modo de operación Opción 1 en el Laboratorio TINI.
Fuente: Los Autores

Figura.V.51. Visualización en el LCD de la TDDC del modo de operación Opción 1.
Fuente: Los Autores

En el modo de operación Opción 2, se comprobó el control y la monitorización de los sensores y actuadores digitales de la TDDC, a través de la consola de la TINI y desde el teclado matricial del Laboratorio TINI.

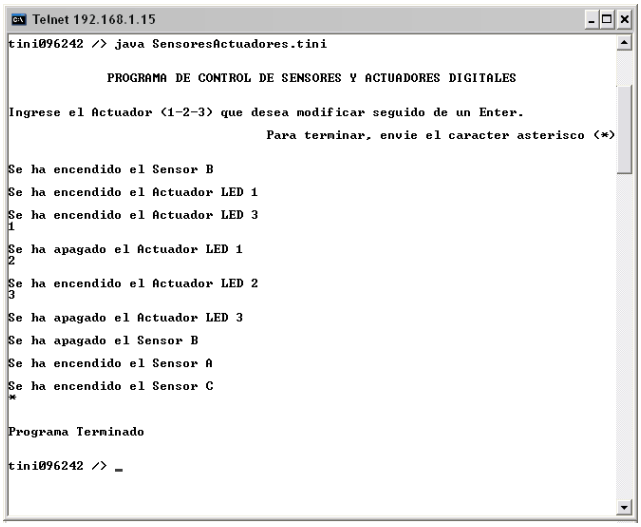


Figura.V.52. Monitorización y Control desde la consola TINI al modo de operación Opción 2
Fuente: Los Autores

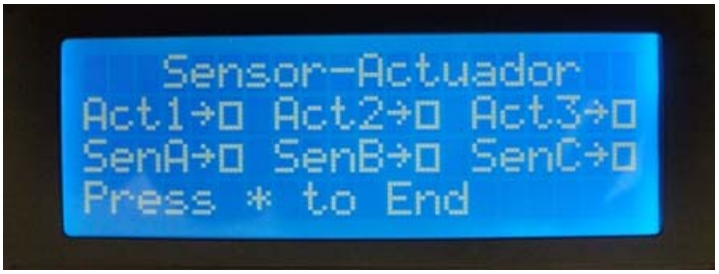


Figura.V.53. Inicio del modo de operación Opción 2
Fuente: Los Autores

Figura.V.54. Activación de dos Actuadores y un Sensor.
Fuente: Los Autores

Figura.V.55. Comprobación en el LCD de la TDDC del estado de los Sensores y Actuadores
Fuente: Los Autores

En el modo de operación Opción 3, se comprobó el control y la monitorización de la velocidad de un motor DC por PWM, a través de la consola de la TINI y desde el teclado matricial del Laboratorio TINI.

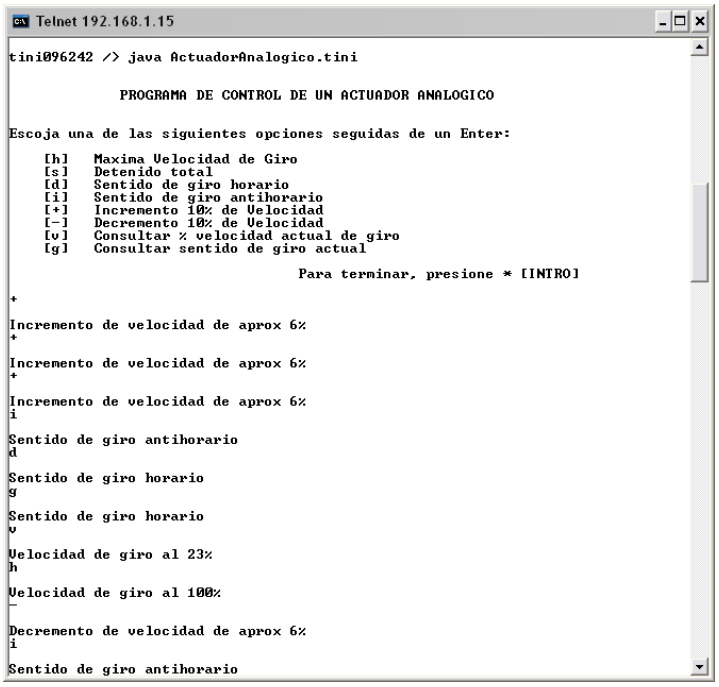


Figura.V.56. Control desde la consola TINI de la Velocidad PWM de un motor
Fuente: Los Autores

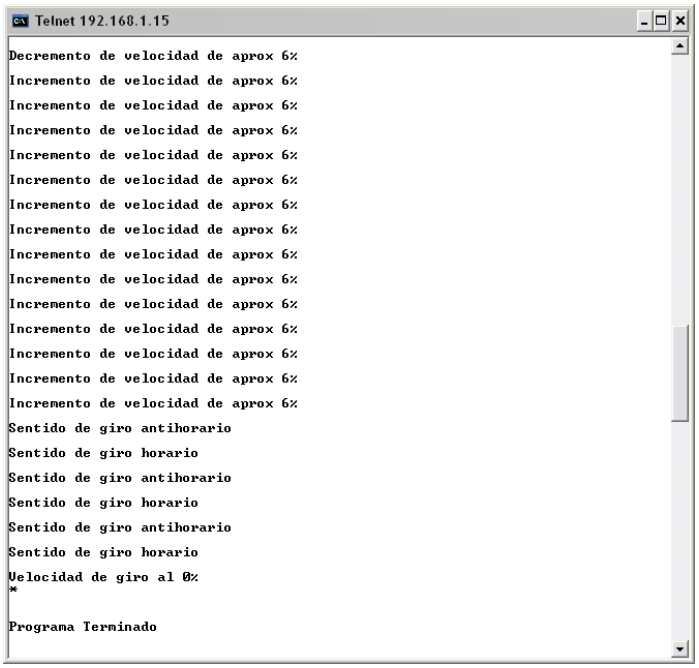


Figura.V.57. Monitorización desde la consola TINI de la Velocidad PWM de un motor
Fuente: Los Autores

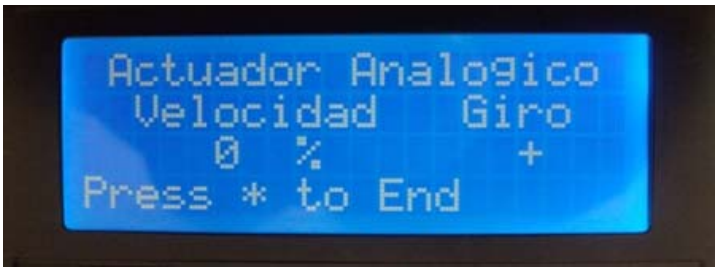


Figura.V.58. Inicio del modo de operación Opción 3.
Fuente: Los Autores

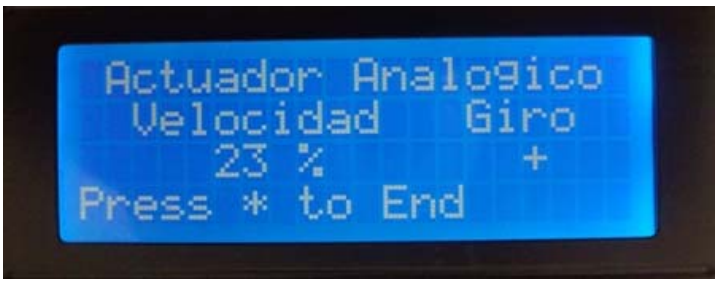


Figura.V.59. Verificación mediante el LCD del estado del Motor
Fuente: Los Autores

Figura.V.60. Control desde el teclado de la TDDC de la Velocidad PWM de un motor
Fuente: Los Autores

En el modo de operación Opción 4, se comprobó la monitorización y registro del estado de un sensor analógico, a través de la consola de la TINI y desde el teclado matricial del Laboratorio TINI.

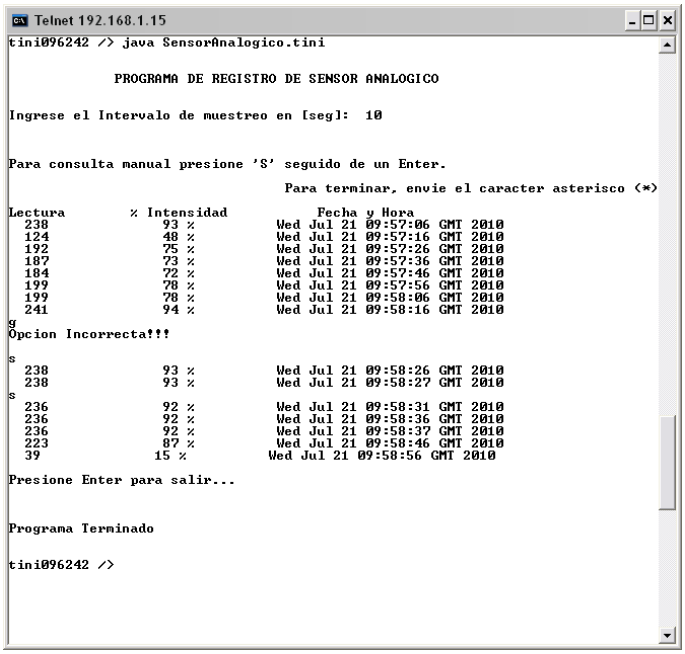


Figura.V.61. Monitorización desde la consola TINI del registro del Sensor Analógico
Fuente: Los Autores

Figura.V.62. Manipulación del Sensor analógico mediante la variación del ambiente.
Fuente: Los Autores

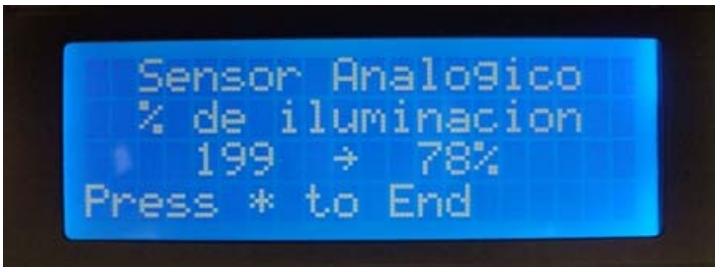


Figura.V.63. Verificación mediante el LCD del estado del Sensor Analógico.
Fuente: Los Autores

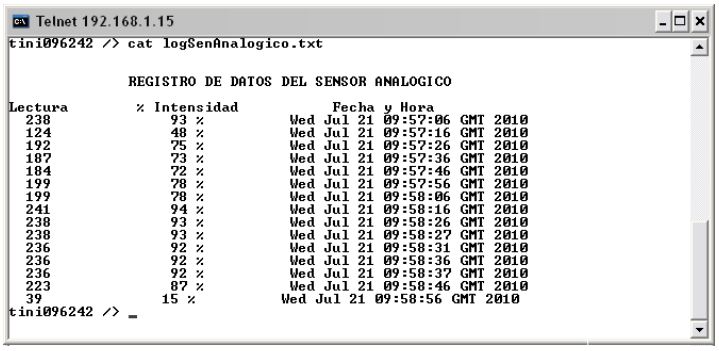


Figura.V.64. Verificación del registro del sensor analógico en la consola de la TINI.
Fuente: Los Autores

CONCLUSIONES

1. Se realizó una extensa investigación sobre el funcionamiento y operación de la plataforma TINI que permitió explorar y experimentar el sistema embebido.
2. Se investigó, aprendió y utilizó el Lenguaje de programación Java, que permite crear aplicaciones orientadas a objetos y multiplataforma.
3. Se diseñó e implementó la Tarjeta de Desarrollo TDDC, compuesta por un microcontrolador PIC16F877A y dispositivos digitales y analógicos; capaz de comunicarse con el Sistema embebido TINI para simulación de control y monitorización remoto de tareas realizadas por Sistemas Propietarios.
4. Se implementó un equipo de aprendizaje didáctico llamado Laboratorio TINI, que interconecta la Plataforma TINI con la Tarjeta de Desarrollo TDDC, y brinda fácil acceso a las funciones de dicho conjunto; que incluye conectores de comunicación, y acceso a los dispositivos digitales y analógicos de la TDDC.
5. Se diseñó y desarrollo la Guía de Aprendizaje TINI/TDDC, que provee soporte Teórico y ejercicios prácticos a través de creación de Aplicaciones de Java, aportando a la enseñanza del sistema embebido TINI, y cuyos resultados son corroborados a través de la Tarjeta de Desarrollo TDDC.
6. En base a esta Tesis se pudo demostrar la importancia de los Sistemas Embebidos para realizar distintas tareas de Control en campos Tecnológicos, Académicos, Investigativos e Industriales.

7. Se concluye que Java es un lenguaje de programación de alto nivel orientado a objetos, que brinda muchas ventajas con respecto a otros lenguajes de programación, ya que permite la generación de aplicaciones multiplataforma.

RECOMENDACIONES

1. Los elementos que conforman el Laboratorio TINI son sensibles a daños en caso de manipulaciones incorrectas, es por esto que se recomienda que este equipo sea utilizado por Estudiantes que posean conocimientos básicos de Electrónica, y que ejecuten las instrucciones de la Guía de Aprendizaje de forma correcta.
2. Para las distintas conexiones utilizar los cables provistos en el Laboratorio TINI o hacer referencia a los requerimientos y conexión de cada práctica de la Guía de aprendizaje para evitar daños al equipo.
3. Se recomienda la difusión de Java en la Escuela de Ingeniería Electrónica, ya que es un lenguaje que puede ser ocupado para realizar una gran gama de aplicaciones relacionadas con la Electrónica, logrando facilitar el entendimiento de los códigos fuente de la Guía de Aprendizaje, lo cual permitirá que esta Tesis sirva como una herramienta de aprendizaje de implementación de aplicaciones de Java para el control.
4. En caso de existir cualquier duda sobre los temas expuestos en la implementación de esta Tesis se recomienda realizar un estudio profundo de los Libros y documentos electrónicos provistos en la Zona de Descarga de la Guía de Aprendizaje Interactiva TINI/TDDC.

RESUMEN

La Tarjeta de Desarrollo de Control (TDDC) se diseño e implementó para simular sistemas propietarios, que utilicen el sistema embebido Tiny InterNet Interface (TINI) para permitir el control y monitorización de forma remota a través de Redes TCP/IP, dotando de un equipo de aprendizaje didáctico de Sistemas Embebidos al Laboratorio de Control de la Escuela de Ingeniería Electrónica de la Escuela Superior Politécnica de Chimborazo.

Para la implementación de la TDDC se utilizó el microcontrolador PIC16F877A como la CPU, un LCD y teclado matricial para la HMI; LEDs, interruptores de dos posiciones, un motor DC y una fotoresistencia, para simular actuadores y sensores digitales y analógicos respectivamente. Siendo el medio de comunicación con el exterior un puerto serial RS232. La TDDC funciona en cuatro modos de Operación que simulan diferentes sistemas embebidos propietarios.

Se utilizó la plataforma TINI modelo DS80C400-KIT#, para el diseño y desarrollo de la Guía de Aprendizaje TINI/TDDC, que incluye una serie de aplicaciones Java para explotar las capacidades del Sistema Embebido TINI corroboradas con la TDDC. La unión de la Tarjeta de Desarrollo TDDC y la plataforma TINI son llamadas Laboratorio TINI.

Obteniendo un equipo de aprendizaje didáctico de sistemas embebidos conformados por el Laboratorio TINI, y la Guía de Aprendizaje TINI/TDDC, que permitirá a los estudiantes de la Escuela de Ingeniería Electrónica de la ESPOCH conocer, explorar y explotar las capacidades del Sistema Embebido TINI.

SUMMARY

Control Development Card (TDDC) was designed and implemented to simulate proprietary systems, using the Tiny InterNet Interface embedded system (TINI) to allow remote control and monitoring via TCP/IP networks, giving a didactic learning equipment of Embedded Systems to the Control Laboratory of the School of Electronic Engineering at Escuela Superior Politécnica de Chimborazo.

To implement the TDDC, a microcontroller PIC16F877A was used as the CPU, an LCD and matrix keypad for HMI, LEDs, toggle switches, a DC motor and a photoresist, to simulate digital and analog actuators and sensors respectively. As the communication system with the outside was used a RS232 serial port. The TDDC operates in four modes of operation that simulate different proprietary embedded systems.

TINI platform DS80C400-KIT # model was used for the design and development of the TINI/TDDC Learning Guide, which includes a number of Java applications to exploit the capabilities of TINI Embedded System with TDDC to corroborate them. The union of the TDDC Development Card and the TINI platform are called TINI Laboratory.

Was obtained an Embedded Systems didactic learning equipment comprised of TINI Laboratory and TINI/TDDC Learning Guide, allowing to know, to explore and to exploit the capabilities of TINI Embedded System, to the students from the School of Electronic Engineering, ESPOCH.

ANEXOS

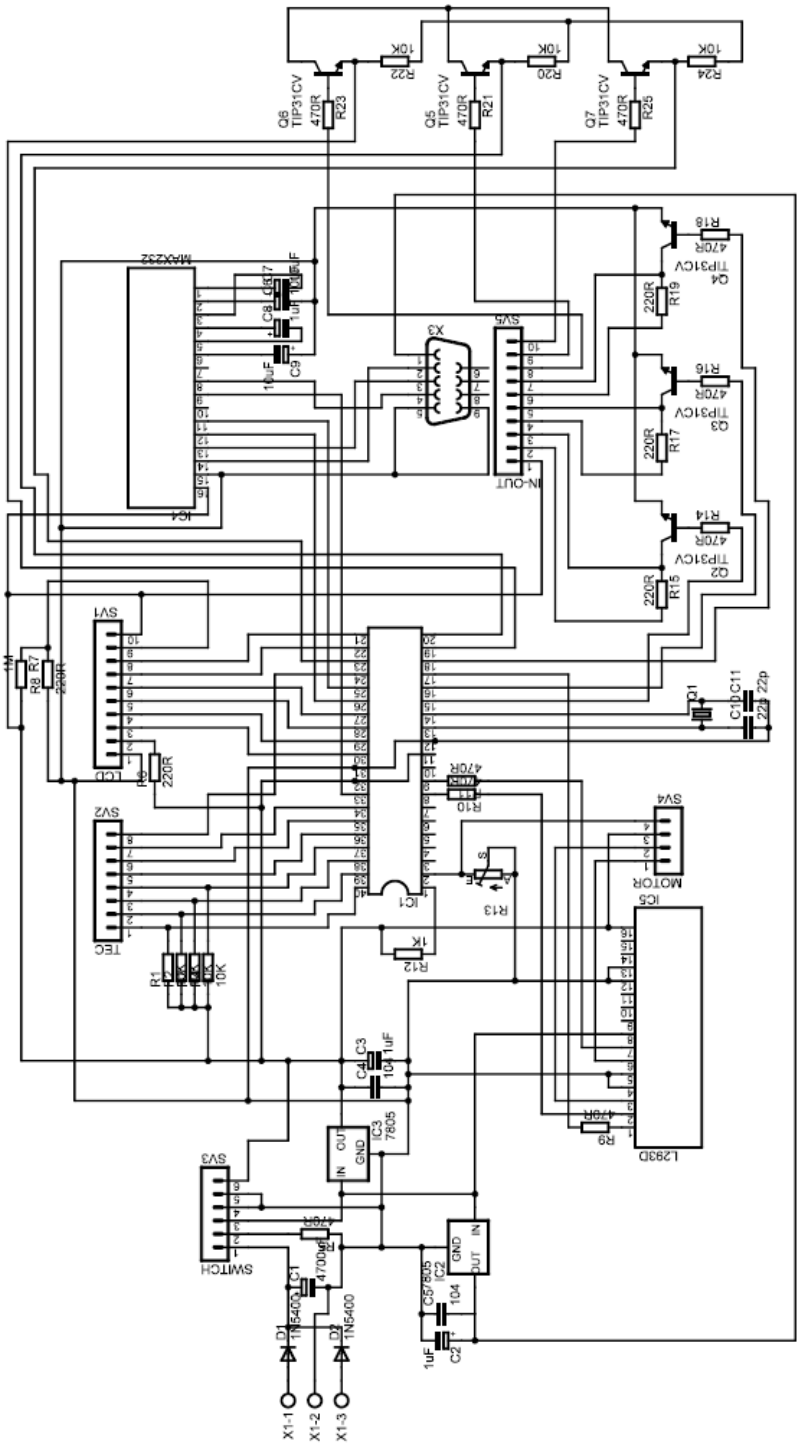
ANEXO A.- Tabla de Descripción de Pines del PIC16F877A

NOMBRE DEL PIN	PIN	TIPO	TIPO DE BUFFER	DESCRIPCIÓN
OSC1/CLKIN	13	I	ST/MOS	Entrada del oscilador de cristal / Entrada de señal de reloj externa
OSC2/CLKOUT	14	O	-	Salida del oscilador de cristal
MCLR/Vpp/THV	1	I/P	ST	Entrada del Master clear (Reset) o entrada de voltaje de programación o modo de control high voltaje test
				PORTA es un puerto I/O bidireccional
RA0/AN0	2	I/O	TTL	RAO: puede ser salida analógica 0
RA1/AN1	3	I/O	TTL	RA1: puede ser salida analógica 1
RA2/AN2/ Vref-	4	I/O	TTL	RA2: puede ser salida analógica 2 o referencia negativa de voltaje
RA3/AN3/Vref+	5	I/O	TTL	RA3: puede ser salida analógica 3 o referencia positiva de voltaje
RA4/T0CKI	6	I/O	ST	RA4: puede ser entrada de reloj el timer0.
RA5/SS/AN4	7	I/O	TTL	RA5: puede ser salida analógica 4 o el esclavo seleccionado por el puerto serial síncrono.
				PORTB es un puerto I/O bidireccional. Puede ser programado todo como entradas
RBO/INT	33	I/O	TTL/ST	RB0 pude ser pin de interrupción externo.
RB1	34	I/O	TTL	
RB2	35	I/O	TTL	
RB3/PGM	36	I/O	TTL	RB3: puede ser la entada de programación de bajo voltaje
				Pin de interrupción
RB4	37	I/O	TTL	Pin de interrupción
RB5	38	I/O	TTL	Pin de interrupción. Reloj de programación serial
RB6/PGC	39	I/O	TTL/ST	
RB7/PGD	40	I/O	TTL/ST	
				PORTC es un puerto I/O bidireccional
RCO/T1OSO/T1CKI	15	I/O	ST	RCO puede ser la salida del oscilador timer1 o la entrada de reloj del timer1
RC1/T1OS1/CCP2	16	I/O	ST	RC1 puede ser la entrada del oscilador timer1 o salida PMW 2
RC2/CCP1	17	I/O	ST	RC2 puede ser una entrada de captura y comparación o salida PWN
RC3/SCK/SCL	18	I/O	ST	RC3 puede ser la entrada o salida serial de reloj síncrono para modos SPI e I2C
				RC4 puede ser la entrada de datos SPI y modo I2C
RC4/SD1/SDA	23	I/O	ST	RC5 puede ser la salida de datos SPI
RC5/SD0	24	I/O	ST	RC6 puede ser el transmisor asíncrono USART o el reloj
RC6/Tx/CK	25	I/O	ST	síncrono.
				RC7 puede ser el receptor asíncrono USART o datos síncronos
RC7/RX/DT	26	I/O	ST	
				PORTD es un puerto bidireccional paralelo
RD0/PSP0	19	I/O	ST/TTL	
RD1/PSP1	20	I/O	ST/TTL	
RD2/PSP2	21	I/O	ST/TTL	
RD3/PSP3	22	I/O	ST/TTL	
RD4/PSP4	27	I/O	ST/TTL	
RD5/PSP5	28	I/O	ST/TTL	
RD6/PSP6	29	I/O	ST/TTL	
RD7/PSP7	30	I/O	ST/TTL	

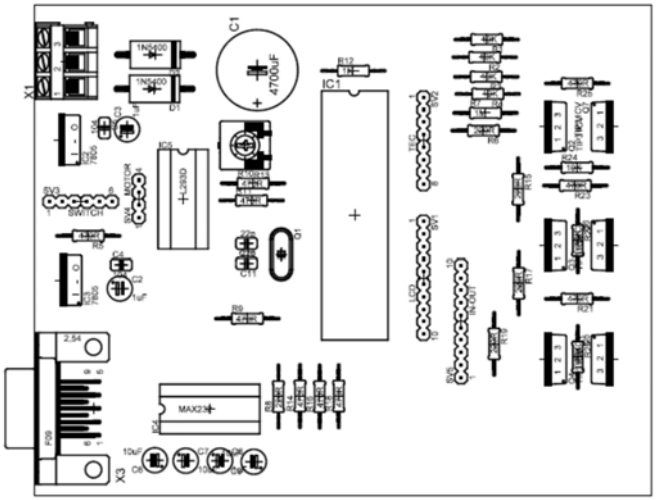
REO/RD/AN5 **RE1/WR/AN** **RE2/CS/AN7**	8 9 10	I/O I/O I/O	ST/TTL ST/TTL ST/TTL	PORTE es un puerto I/O bidireccional REO: puede ser control de lectura para el puerto esclavo paralelo o entrada analógica 5 RE1: puede ser escritura de control para el puerto paralelo esclavo o entrada analógica 6 RE2: puede ser el selector de control para el puerto paralelo esclavo o la entrada analógica 7.
Vss	12.31	P	-	Referencia de tierra para los pines lógicos y de I/O
Vdd	11.32	P	-	Fuente positiva para los pines lógicos y de I/O
NC	-	-	-	No está conectado internamente

Fuente: http://www.monografias.com/trabajos18/descripcion-pic/descripcion-pic.shtml

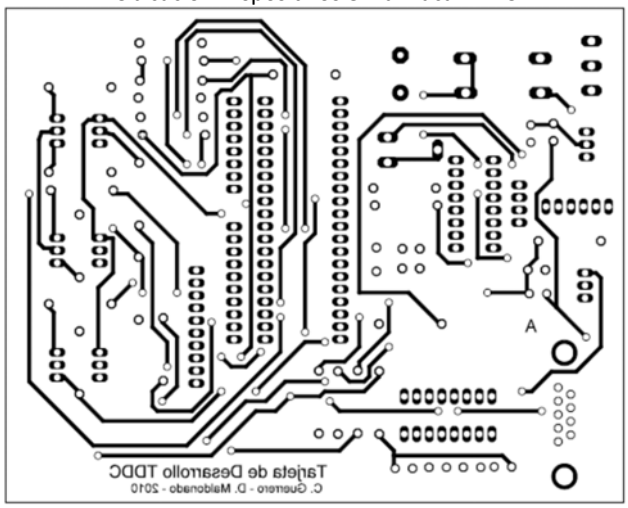
ANEXO B.- Esquema de conexiones del circuito impreso de la Tarjeta de Desarrollo TDDC.



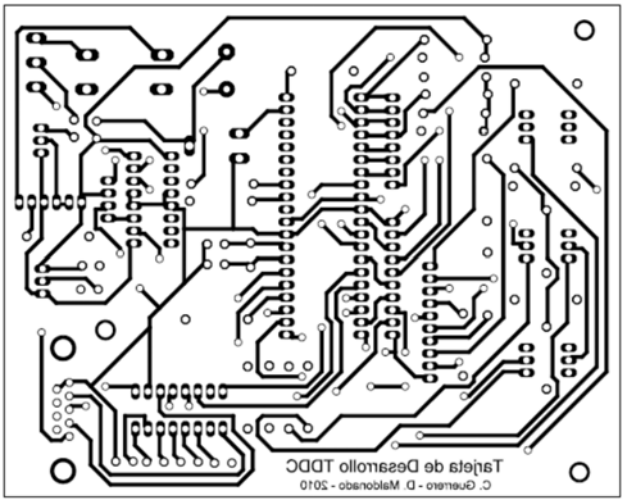
ANEXO C.- Diseño de pistas del Circuito Impreso TDDC.



Ubicación Dispositivos en la Placa TDDC



Pistas cara superior placa TDDC



Pistas cara inferior placa TDDC

BIBLIOGRAFÍA

LIBROS

DEAN, J.S. y DEAN R.H. Introducción a la programación con java. Mexico D.F. – Mexico: McGraw-Hill, 2009. pp 52-561.

EISENREICH, D. y DeMUTH, B. Designing Embedded Internet Devices. Burlington - Estados Unidos: Elsevier Science, 2003. 577p.

LOOMIS, D. The TINI™ Specification and Developer's Guide, New Jersey - Estados Unidos: ADDISON-WESLEY, 2001. 560 p.

REYES,C.A. Aprenda rápidamente a programar Microcontroladores PIC. Quito - Ecuador: Gráficas Ayerve C.A, 2004. 191p.

DOCUMENTOS ELECTRONICOS

MAXIM. Datasheet DS80C400-Kit. Maxim Integrated Products. 2009. 97p.

http://datasheets.maxim-ic.com/en/ds/DS80C400-KIT.pdf

(2009-06-23)

MAXIM. Getting Started with TINI. Maxim Integrated Products. 2004. 66p

http://www.maxim-ic.com/products/tini/pdfs/TINI_GUIDE.pdf

(2009-05-07)

MICRO ELECTRONICS. Datasheet Tip31. 1p.

http://www.datasheetcatalog.org/datasheet/MicroElectronics/mXuwuts.pdf

(2010-01-12)

Microchip. Datasheet 16F877A. Microchip Technology Inc. 1999. 201p.

http://ww1.microchip.com/downloads/en/devicedoc/39582b.pdf

(2009-08-18)

OTERO, A. Tutorial Básico de Java. Versión 3.0. javaHispano. 2007. 148p.

http://www.sergioandresgarcia.com/Informatica/Tutoriales/Tut_Bas_Java.pdf

(2009-11-14)

SGS-THOMSON. Datasheet L293D. SGS-Thomson Microelectronics. 1996. 7p.

http://www.datasheetcatalog.org/datasheet/SGSThomsonMicroelectronics/mXyzuxsr.pdf

(2009-12-08)

STMicroelectronics. Datasheet L7800 Series. STMicroelectronics GROUP OF COMPANIES. 2003. 29p.

http://focuslab.lfp.uba.ar/Extension/BDS2005/7800.pdf

(2010-05-10)

TEXAS. Datasheet Max232. Texas Instruments Incorporated 2002. 8p.

http://focus.ti.com/lit/ds/symlink/max232.pdf

(2009-10-15)

BIBLIOGRAFIA DE INTERNET

JAVA

http://java.sun.com

(2009-08-14)

http://www.codexion.com/tutorialesjava/getStarted/cupojava/win32.html#win32-2b

(2009-10-16)

http://www.docstoc.com/docs/23955078/Sockets-en-Java

(2010-06-20)

http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte3/cap3-1.html

(2009-08-07)

http://www.webtaller.com/construccion/lenguajes/java/lecciones/sockets-java.php

(2010-06-20)

http://zarza.usal.es/~fgarcia/docencia/poo/01-02/trabajos/S3T3.pdf

(2010-01-06)

MICROCONTROLADORES

http://aliatron.com/loja/catalog/images/encap_40p.gif

(2010-06-24)

http://lc.fie.umich.mx/~jrincon/apuntes%20intro%20PIC.pdf

(2009-09-12)

TINI

http://www.maxim-ic.com/

(2009-05-11)

http://www.maxim-ic.com/products/tini/software/downloads.cfm

(2009-05-25)

VARIOS

http://es.wikipedia.org/wiki/File_Transfer_Protocol

(2009-11-15)

http://es.wikipedia.org/wiki/Telnet

(2009-11-13)