UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERIA ELECTRICA Y ELECTRONICA

DISEÑO DE UN SISTEMA SCADA MEDIANTE EL LENGUAJE DE PROGRAMACIÓN JAVA

INFORME DE SUFICIENCIA

PARA OPTAR EL TITULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

ELIO SULPICIO LEGUIA LOAYZA

PROMOCION
2003 – II

LIMA –PERU
2008

DISEÑO DE UN SISTEMA SCADA MEDIANTE EL LENGUAJE DE PROGRAMACIÓN JAVA

A mis padres, por inculcarme el amor al estudio y a mis hermanos por contribuir en ello

SUMARIO

El presente informe corresponde al diseño de un sistema SCADA para controlar el encendido o apagado de un motor. El sistema tendrá un control de retardo antes del encendido, registra también en una base de datos el día y la hora en que se prendió o apagó el motor así como el nombre del usuario que manipuló, también un control de 8 salidas por el puerto paralelo. El Software SCADA esta desarrollada utilizando el lenguaje de programación Java y el protocolo JDBC para la comunicación con la base de datos, en este caso particular se ha probado con el motor de base de datos Access debido a que no se necesita instalar lo que permite realizar las pruebas sólo copiando en cualquier computadora el software completo. Para escribir el programa usamos el entorno de desarrollo integrado Eclipse lo que permite escribir código Java con suma facilidad.

INDICE

INTRODUCCIÓN

Las computadoras ya son parte de nuestras vidas lo tomamos como una herramienta que nos permite hacemos la vida más fácil. Una de las aplicaciones en Ingeniería electrónica son el control, supervisión y adquisición de datos conocido como SCADA. Se trata de un software especialmente diseñado para funcionar en computadoras de control de producción proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, etc.) y controlando el proceso de forma automática desde la pantalla del computador. Además provee toda la información que se genera en el proceso productivo. Todo esto, software y hardware se denominan generalmente sistema SCADA. Vengo trabajando ya varios años con un lenguaje considerada en la actualidad como uno de los mas interesantes, no solo por ser un lenguaje que no necesita licencia sino que es multiplataforma es decir que puede correr sin problemas con distintos sistema operativos incluyendo celulares y/o dispositivos móviles, estoy hablando nada menos que de JAVA un lenguaje que se escribe una vez y se ejecuta literalmente donde sea, eso es lo que me motivo para investigar realizando algunas experiencias con el puerto paralelo utilizando el API **javax.comm.** Para el entorno gráfico usamos la librería Swing que es parte del JFC (Java Fundation CLasses) los cuales se usan como cualquier componente de Windows, para la conexión a la base de datos usamos el protocolo JDBC. El entorno que escogí para escribir el programa, es el IDE (Entorno de desarrollo Integrado) conocido como Eclipse versión 3.1 que me permite escribir con suma facilidad código Java. En el transcurso de la investigación utilicé algunos libros tanto de Java como los manuales de los dispositivos de comunicación, no encontrando mucha información al respecto los Libros de Java sólo muestran el lenguaje propiamente dicho como las estructuras de los algoritmos priorizando la comunicación a bases de datos, afortunadamente existen foros en Internet donde se discute temas relacionados con el lenguaje , inclusive existen comunidades de desarrollo en java en distintos países como en el Perú allí fue que encontré las librerías de comunicación tanto por el puerto paralelo como por el puerto serial.

CAPITULO I

PLANTEAMIENTO DE INGENIERÍA DEL PROBLEMA

1.1 Descripción del Problema.

Se tiene la necesidad de controlar el encendido o apagado de un motor utilizando un computador, así como registrar toda la información concerniente a dicho proceso en una base de datos, es decir se necesita saber en cualquier momento la estadística de encendido y apagado de un motor, así como la identificación del personal que manipula el equipo. El sistema deberá proporcionar en cualquier instante:

- Fecha, hora, minuto y segundo en que se prendió o apagó el motor
- Retardo en segundos para el encendido y apago del motor
- El mayor o menor rango de tiempo que estuvo prendido o apagado el motor en una fecha determinada.
- Cuánto tiempo estuvo parado el motor.
- El tiempo promedio en que el motor esta prendido que puede ser diario, semanal mensual o anual.
- Quien fue el personal que prendió o apago el motor en un momento determinado.
- Si en un instante determinado estuvo apagado o prendido el motor.
- Una señal de audio indicando el momento en que se prende o apaga el motor.

Para registrar toda la información concerniente a dicho proceso se deberá almacenar en un motor de base de datos como Access o MySQL toda la información necesaria como las fechas, horas, segundos y la identificación del personal. Cada vez que el personal utiliza el computador para prender o apagar el motor, el sistema automáticamente debe registrar el momento mismo en que se realizo en la base de datos y proceder con enviar la señal respectiva al puerto paralelo del computador para el encendido físico del motor con un retardo controlado. Una vez registrado en la base de datos usamos el lenguaje SQL para consultar absolutamente toda la información guardada.

El sistema debe desarrollarse en Java un lenguaje multiplataforma amigable para el usuario, es decir un personal sin necesidad de tener amplios conocimientos de cómputo pueda manejar sin problema con sólo saber el nombre de usuario y clave. Para esto el

Interfaz debe ser un entorno gráfico el cual mostrará en pantalla en forma gráfica todo el proceso tal como se vería en forma física. El personal a cargo de manipular el sistema debe estar plenamente identificado con su clave respectiva registrado en la base de datos, los cuales ingresarán al sistema con su usuario y clave para un mejor control. El equipo necesario para el sistema SCADA que da solución a lo planteado: Computadora, Tarjeta de Interfase y seguridad para la comunicación entre el puerto y los equipos de potencia como el motor, cables, Motor de 220 V, fuente de 12V continua.

Figura 1.1: Equipo necesario para el scada

Para el análisis de requerimientos usaremos el lenguaje de modelo unificado UML. En este caso usaremos el diagrama de casos de uso donde muestra la relación entre los actores y los casos de uso del sistema lo cual representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa

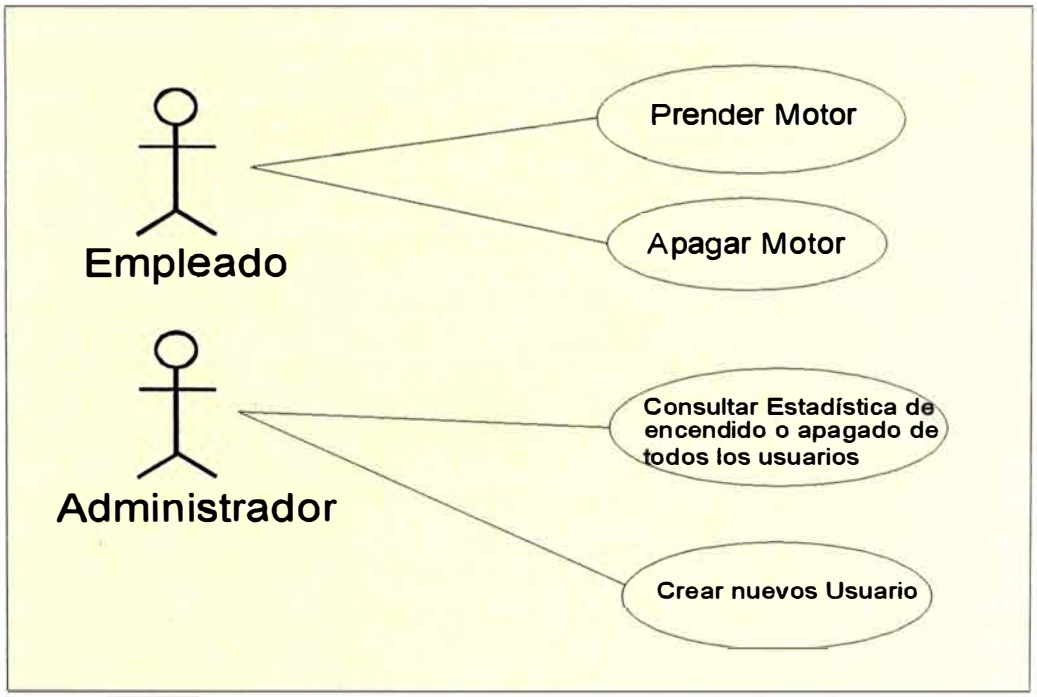


Figura 1.2: Diagrama de casos de uso

1.2 Objetivos del trabajo.

Conocer y demostrar la factibilidad de crear un control SCADA de una forma simple, sencilla y a bajo costo, demostrando que el lenguaje Java también puede ser usado para el control de equipos externos tal como se usa con C o C++, con la gran ventaja que puede ejecutarse en cualquier equipo sin preocuparse por el sistema operativo, es decir se escribe una sola vez y se ejecuta donde sea.

Utilizando un computador vemos lo sencillo que es guardar toda la información de los equipos en el proceso productivo, no solamente eso, aprovechamos la gran capacidad de cálculo de las computadoras personales para no sobrecargar al equipo de campo, justamente aquí radica la gran ventaja, es decir todo el cálculo tanto de control como ingreso a la base de datos se realiza desde el computador sin afectar a los dispositivos que se están controlando. Se debe construir también la tarjeta que nos sirve de interfase entre el computador y el motor a controlar es decir la tarjeta que permite conectar desde el puerto paralelo de la PC al dispositivo de potencia que en este caso particular es el motor.

1.3 Evaluación del problema

Según lo planteado la mejor solución sería usar un sistema SCADA (control, supervisión y adquisición de datos). Es decir utilizando un computador personal podemos controlar cualquier equipo externo que en este caso particular es un motor y una señal de alarma constituido por un parlante. Como el hardware ya esta diseñado, nuestra preocupación sería utilizar un algoritmo adecuado para tal fin. Escogemos el lenguaje de programación Java para escribir el algoritmo por que es un lenguaje libre minimizando los costos, podemos usar cualquier motor de base de datos como MySQL o PostGreSQL, en este caso particular usamos el Access por que es simple y puede ejecutarse directamente sin necesidad de instalar.

1.4 Limitaciones del trabajo

El sistema SCADA plateado como solución será con conexión directa a los dispositivos de campo por el puerto serial o paralelo, la solución más simple sería por el puerto paralelo por que la conexión es casi directa, la única preocupación es el aislamiento de las fuentes de alta tensión ya que el puerto paralelo es más delicado donde cualquier cruce puede malograr la placa. Para el caso del puerto serial necesitaríamos un microcontrolador para transformar la salida serial a paralela lo que en costos seria mayor. El sistema scada se construirá con conexión directa utilizando un cable DB25 para puerto

paralelo, sin embargo se puede generalizar para el control por medio de Internet utilizando applets de Java.

1.5 Síntesis del trabajo

Como el hardware ya esta instalado todo se limita al control a través del puerto paralelo por intermedio de una interfaz, guardar toda la información del control en una base de datos que en este caso particular es Access por que además de trabajar en Windows nos permite que el sistema puede ser manejado sin necesidad de instalar, se podría correr por ejemplo desde un USB. Para la comunicación de los puertos usamos el API javax.comm

CAPITULO II

MARCO TEÓRICO CONCEPTUAL

2.1 Sistema SCADA

En la actualidad ya no existe prácticamente nada que no se pueda controlar por medio de un computador, recoger o almacenar dato concerniente a los dispositivos de campo en un proceso productivo, es decir podemos guardar toda la información en una base de datos de los equipos que están en operación además de controlar dichos equipos. Un ejemplo simple es controlar el encendido y apagado de un motor guardando toda la información en la base de datos como cuando se prendió, cuando se apagó o cuanto tiempo estuvo prendido, etc. El conjunto software y hardware se conoce como sistema SCADA .

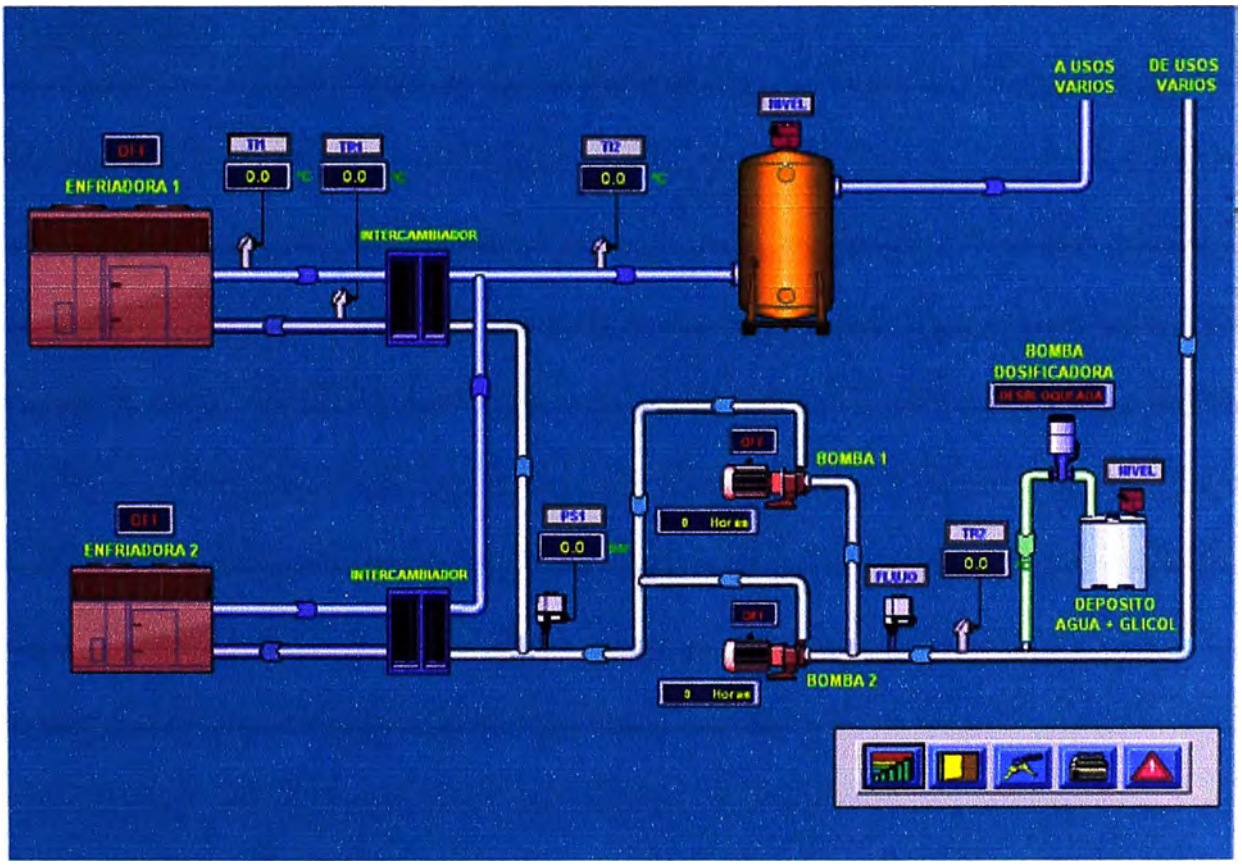


Figura 2.1: Ejemplo de un Scada para refrigeración y aire acondicionado

2.2 Usos de un sistema SCADA

- Posibilidad de crear paneles de alarma, que exigen la presencia del operador para reconocer una parada o situación de alarma, con registro de incidencias.
- Generación de históricos de señal de planta, que pueden ser volcados para su proceso sobre una base de datos. Ejecución de programas, que modifican la ley de control, o incluso el programa total sobre el autómata, bajo ciertas condiciones.
- Posibilidad de programación numérica, que permite realizar cálculos aritméticos de elevada precisión sobre la CPU del ordenador, y no sobre la del autómata.
- Con ellas, se pueden desarrollar aplicaciones basadas en el PC, con captura de datos, análisis de señales, presentaciones en pantalla, envío de resultados a disco e impresora, etc.

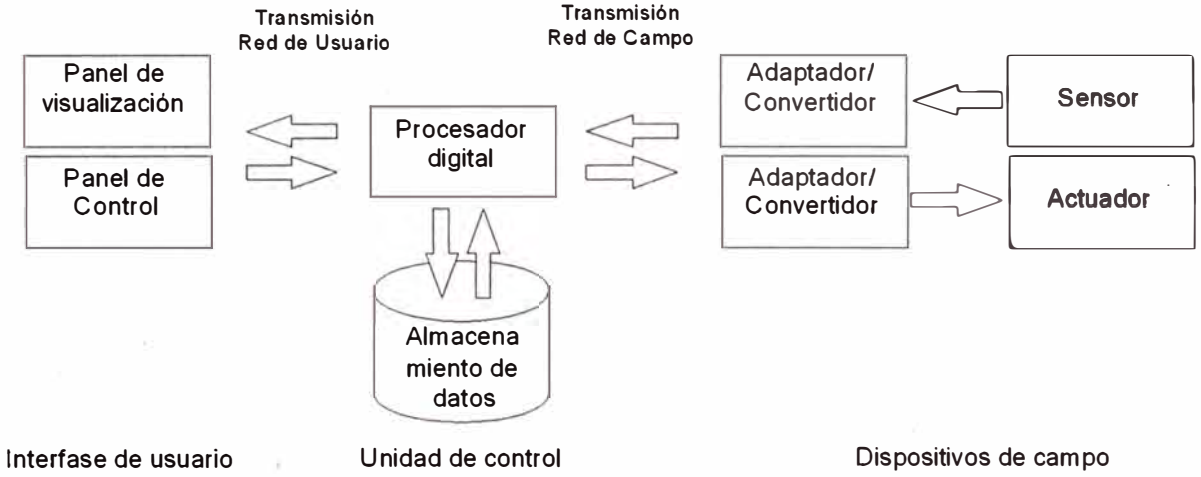


Figura 2.2: Esquema básico de un sistema SCADA

2.3 Objetivos de un sistema SCADA

- Deben ser sistemas de arquitectura abierta, capaces de crecer o adaptarse según las necesidades cambiantes de la empresa.
- Deben comunicarse con total facilidad y de forma transparente al usuario con el equipo de planta y con el resto de la empresa (redes locales y de gestión).
- Deben ser programas sencillos de instalar, sin excesivas exigencias de hardware, y fáciles de utilizar, con interfaces amigables con el usuario.

2.4 Módulos de un SCADA

Los módulos o bloques software que permiten las actividades de adquisición, supervisión y control son los siguientes:

Configuración: permite al usuario definir el entorno de trabajo de su SCADA, adaptándolo a la aplicación particular que se desea desarrollar.

Interfaz gráfico del operador: proporciona al operador las funciones de control y supervisión de la planta. El proceso se representa mediante sinópticos gráficos almacenados en el ordenador de proceso y generados desde el editor incorporado en el SCADA o importados desde otra aplicación durante la configuración del paquete.

Módulo de proceso: ejecuta las acciones de mando preprogramadas a partir de los valores actuales de variables leídas. La programación se realiza por medio de bloques de programa en lenguaje de alto nivel (como C, Basic, Java, etc.).

Gestión y archivo de datos: se encarga del almacenamiento y procesado ordenado de los datos, de forma que otra aplicación o dispositivo pueda tener acceso a ellos.

Comunicaciones: se encarga de la transferencia de información entre la planta y la arquitectura hardware que soporta el SCADA, y entre ésta y el resto de elementos informáticos de gestión.

2.5 Redes industriales

En la empresa coexisten una serie de equipos y dispositivos dedicados al control de una máquina o una parte cerrada de un proceso. Entre estos dispositivos están los autómatas programables, ordenadores de diseño y gestión, sensores, actuadores, etc.

El desarrollo de las redes industriales ha establecido una forma de unir todos estos dispositivos, aumentando el rendimiento y proporcionando nuevas posibilidades. Las ventajas que se aportan con una red industrial son, entre otras, las siguientes:

- Visualización y supervisión de todo el proceso productivo.
- Toma de datos del proceso más rápida o instantánea.
- Mejora del rendimiento general de todo el proceso.
- Posibilidad de intercambio de datos entre sectores del proceso y entre departamentos.
- Programación a distancia, sin necesidad de estar a pie de fábrica.

2.6 Niveles en una red industrial

En una red industrial coexistirán equipos y dispositivos de todo tipo, los cuales suelen agruparse jerárquicamente para establecer conexiones lo más adecuadas a cada área. De esta forma se definen cuatro niveles dentro de una red industrial:

Nivel de gestión: es el nivel más elevado y se encarga de integrar los niveles siguientes en una estructura de fábrica, e incluso de múltiples factorías. Las máquinas aquí

conectadas suelen ser estaciones de trabajo que hacen de puente entre el proceso productivo y el área de gestión. Se emplea una red de tipo LAN (Local Área Network) o WAN (Wide Área Network).

Nivel de control: se encarga de enlazar y dirigir las distintas zonas de trabajo. A este nivel se sitúan los autómatas de gama alta y los ordenadores dedicados a diseño, control de calidad, programación, etc. Se suele emplear una red de tipo LAN.

Nivel de campo y proceso: se encarga de la integración de pequeños automatismos (autómatas compactos, multiplexores de E/S, controladores PID, etc.) dentro de las subredes. En el nivel más alto de estas redes se suelen encontrar uno o varios autómatas modulares, actuando como maestros de la red o maestros flotantes. En este nivel se emplean los buses de campo.

Nivel de E/S: es el nivel más próximo al proceso. Aquí están los sensores y actuadores, encargados de manejar el proceso productivo y tomar las medidas necesarias para la correcta automatización y supervisión.

2.7 Redes LAN industriales

Son las redes más elevadas jerárquicamente. Los estándares más conocidos y extendidos son dos:

MAP (Manufacturing Automation Protocol): nació como un producto especialmente diseñado para el entorno industrial, lo que hace que sea de mayor éxito en LAN industriales. Fué impulsado por General Motors y normalizado por el IEEE. No actúa a nivel de bus de campo, pero establece pasarelas hacia estos buses mediante terminales. También permite integración en redes WAN.

ETHERNET: diseñada por Xerox Corporation y registrada posteriormente junto con Digital e Intel. Es compatible con el modelo OSI en los niveles 1, 2 y 3 (el último a través de puentes). Permite topología en Bus o arbol con comunicación semidúplex. Las velocidades van desde los 10 Mbits/s a los 100 Mbits/s de Fast-Ethernet. Es uno de los estándar de red que más rápidamente evolucionan, debido a su uso masivo en redes ofimáticas.

2.8 Bus de campo

El bus de campo constituye el nivel más simple y próximo al proceso dentro de la estructura de comunicaciones industriales. Está basada en procesadores simples y utiliza un protocolo mínimo para gestionar el enlace entre ellos. Los buses de campo más recientes permiten la comunicación con buses jerárquicamente superiores y más potentes.

2.9 Paneles de operación (OP)

Los OP (Operator Panel) son paneles que facilitan el acceso visual del operario por intermedio de la pantalla del computador de acuerdo al sistema de automatización, lo cual se podría decir esta dentro de lo que se conoce como "Human Machine Interface" (Interfaz Humano con la Maquinaria). Gracias a los paneles de operación por que tiene que ser amigable y sencillo resulta cómodo que cualquier persona pueda manejar sin ningún problema un sistema scada.

Algunas posibilidades de los OP son:

- Acceso rápido y sencillo a los datos del sistema.
- Supervisión y control del proceso.
- Visualización del proceso (sólo en OP gráficos).
- Modificación de parámetros y órdenes (sólo en algunos casos).

Todo OP posee los siguientes componentes:

- CPU
- Pantalla.
- Teclado.
- Puerto(s) de comunicación.
- Ranuras de expansión (según modelo y fabricante).

2.10 Puertos de comunicación con la PC

Los puertos de comunicación de la PC nos permiten la comunicación del computador con cualquier dispositivo externo como por ejemplo una impresora o un ratón, en nuestro caso la comunicación con el motor y la alarma constituido por el parlante, son muy importantes en la Electrónica ya que permiten utilizar una computadora personal para controlar todo tipo circuitos electrónicos por ejemplo, en actividades de automatización de procesos y adquisición de datos conocidos como SCADA y otras actividades que demandan precisión.

Para transmitir datos en forma simple en las computadoras se pueden hacer de dos formas básicas En serie y en paralelo. En un esquema de transmisión de datos en serie un dispositivo envía datos a otro a razón de un bit a la vez a través de un cable. Por otro lado, en un esquema de transmisión de datos en paralelo un dispositivo envía datos a otro a una tasa de n número de bits a través de n número de cables a un tiempo. La manera más común de enviar datos es utilizando ocho líneas para transmitir un byte a la vez. Actualmente se usa el puerto de comunicaciones Universal conocido popularmente como puerto USB, actualmente se usa con frecuencia para comunicarse con cámaras

digitales, celulares, memorias de almacenamiento conocidos como memory key o inclusive la comunicación con discos duros, obviamente también existe conversores de USB a puerto paralelo o serial.

2.10.1 El puerto paralelo de la PC

Manejar el puerto paralelo de la PC es bastante simple, sin embargo debemos tener cuidado al conectar dispositivos a dicho puerto ya que implica el riesgo de daños permanentes a la tarjeta madre de la PC. El puerto paralelo de las computadoras, de acuerdo a la norma IEEE 1284, está compuesto por un bus de comunicación bidireccional de 8 bits de datos, además de un conjunto de líneas de protocolo. Las líneas de comunicación cuentan con un retenedor que mantiene el último valor que les fue escrito hasta que se escribe un nuevo dato,

Vamos a analizar en primer lugar el de tipo unidireccional. En este caso se distinguen dos elementos: Cuando se coloca datos en la parte transmisora se informa a la parte receptora que la información están disponibles; entonces esta lee la información en las líneas de datos e informa a la parte transmisora que ha tomado la información sincronizando así su respectivo acceso a las líneas de datos, luego la parte transmisora no colocará nueva información en las líneas de datos hasta que la parte receptora remueva la información y le indique a esta que ya ha tomado los datos, todo esto se le conoce como handshaking, que significa que ambas partes están de acuerdo.

El hardware del puerto paralelo de una típica PC utiliza un conector hembra de tipo D de 25 patitas (DB-25) tal como se muestra en la figura.

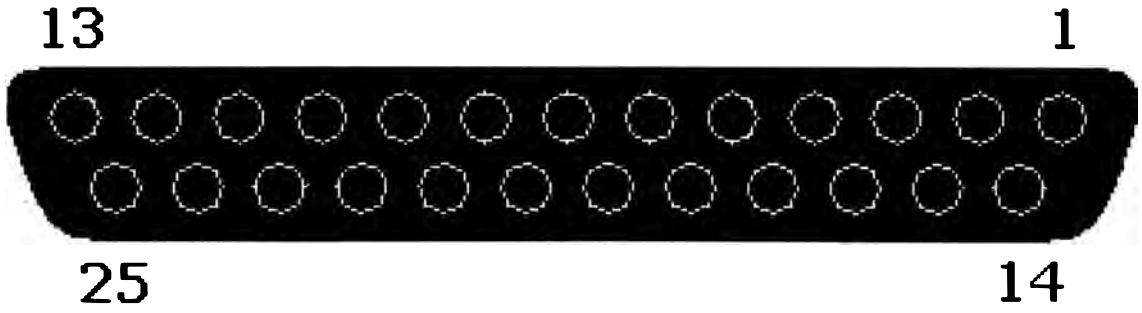


Figura 2.3: Vista del conector hembra DB-25

Tabla 2.1: Descripción de los pines del puerto DB-25

Patita	E/S	Polaridad activa	Descripción
1	Salida	0	Strobe
2 ~ 9	Salida	-	Líneas de datos (bit 0/ 2, bit 7/ 9)
10	Entrada	0	Línea acknowledge (activa cuando el sistema remoto toma datos)
11	Entrada	0	Línea busy (si está activa, el sistema remoto no acepta datos)
12	Entrada	1	Línea Falta de papel (si está activa, falta papel en la impresora)
13	Entrada	1	Línea Select (si está activa, la impresora se ha seleccionado
14	Salida	0	Línea Autofeed (si está activa, la impresora inserta una nueva línea por cada retorno de carro)
15	Entrada	0	Línea Error (si está activa, hay un error en la impresora
16	Salida	0	Línea Init (Si se mantiene activa por al menos 50 micro-segundos, ésta señal autoinicializa la impresora)
17	Salida	0	Línea Select input (Cuando está inactiva, obliga a la impresora a salir de línea)
18 ~ 25	-	-	Tierra eléctrica

Características eléctricas:

- Tensión de nivel alto: 3.3 o 5 V.
- Tensión de nivel bajo: 0 V.
- Intensidad de salida máxima: 2.6 mA.
- Intensidad de entrada máxima: 24 mA.

2.10.2 Puerto Serie RS-232.

Tal vez una de las mas populares conexiones que se hacen a una PC sea el conocido popularmente como puerto serial que permite al computador comunicarse con todo tipo de dispositivos periféricos: módems, impresoras, scanner, lectores de código de barras, etc. El RS232 es un estándar de comunicaciones propuesto por la Asociación de Industrias Electrónicas (EIA) y es la última de varias versiones anteriores. Antiguamente se utilizaba para conectar terminales a un ordenador Host. Se envían datos de 7, 8 o 9 bits. La velocidad se mide en baudios (bits/segundo) y sólo son necesarios dos cables, uno de transmisión y otro de recepción. Lo mas importante del estándar de comunicaciones es la funciones especifica de cada pin de entrada y salida de datos porque nos encontramos básicamente con dos tipos de conectores los de 25 pines y los de 9 pines, es probable que se encuentre mas la versión de 9 pines aunque la versión de 25 permite muchas mas información en la transferencia de datos. Las señales con la que actúa el puerto son digitales (0 - 1) y la tensión a la que trabaja es de 12 Voltios, resumiendo: 12Vlts. = Logica "0" , -12Vlts = Logica "1" .

Por lo visto no es compatible directamente con TTL que usa entre 0 y 5 Voltios Para compatibilizar usaremos el integrado MAX232.

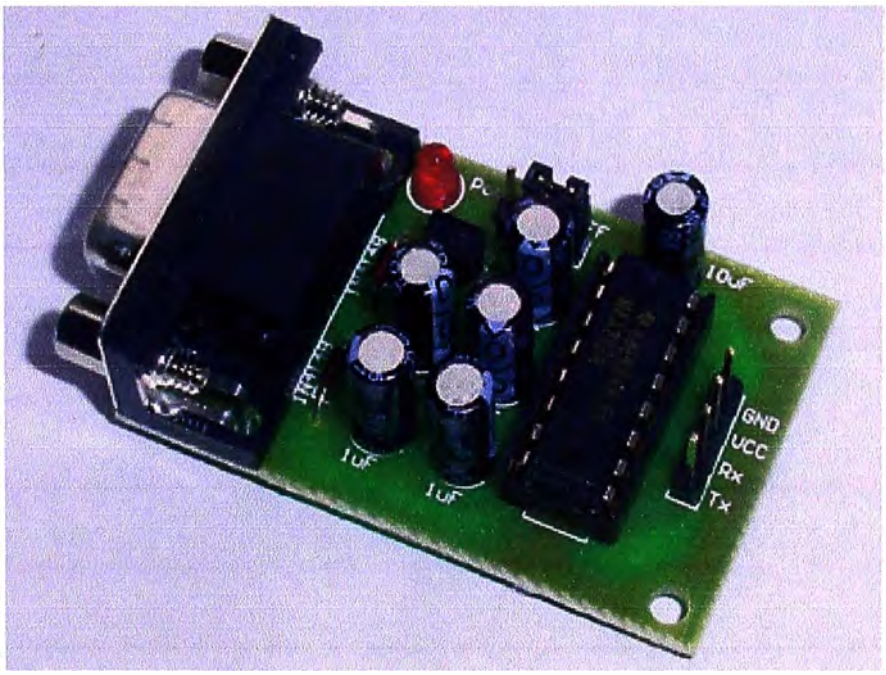


Figura 2.4: MAX232 convertidor de RS232 a TTL

Tabla 2.2: Tipo de señales en los pines del puerto RS-232

Pin	Función	E/S
TXD	Transmitir Datos	Señal de salida
RXD	Recibir Datos	Señal de entrada
RTS	Solicitud de envió	Señal de salida
DTR	Terminal de datos listo	Señal de salida
CTS	Libre para envió	Señal de entrada
DSR	Equipo de datos listo	Señal de entrada
DCD	Detección de portadora	Señal de entrada
SG	Tierra	Referencia para señales
RI	Indicador de llamada	Señal de entrada

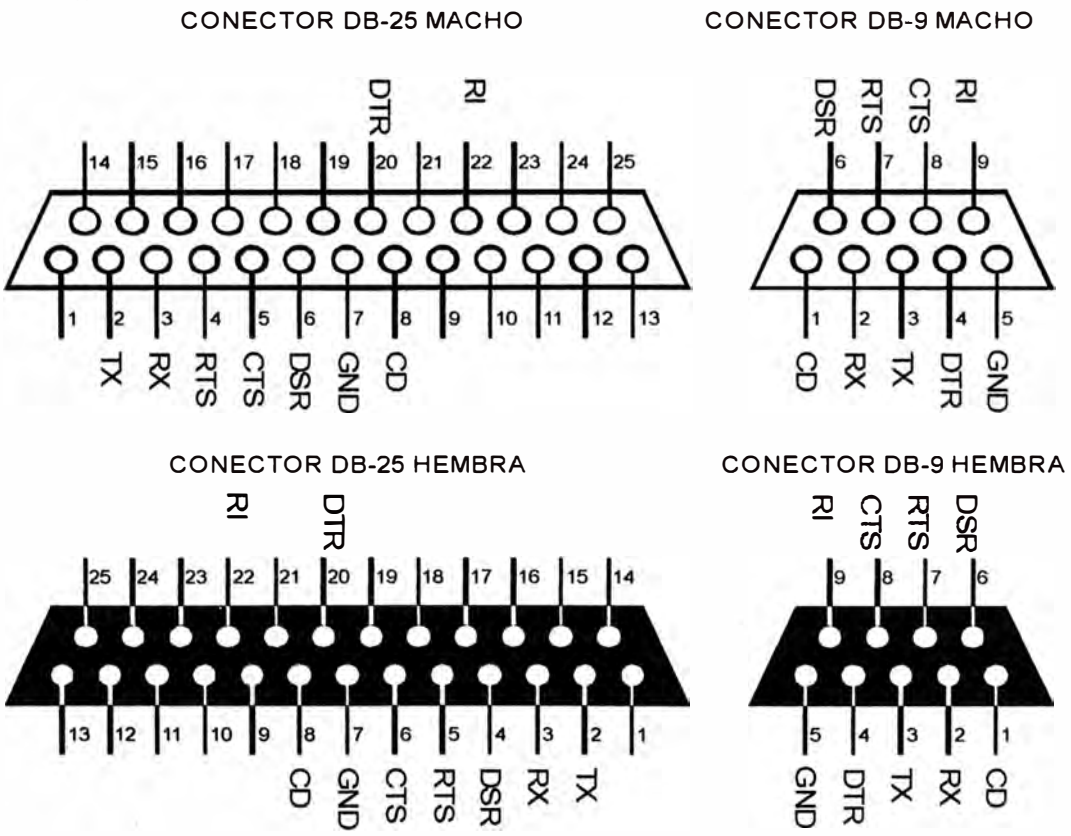


Figura 2.5: Vista de los conectores RS-232

Tabla 2.3: Descripción y ubicación de los pines del puerto RS-232

Conector 25 pines	Conector 9 pines	Nombre	Descripción
1	1	-	Masa chasis
2	3	TxD	Transmit Data
3	2	RxD	Receive Data
4	7	RTS	Request to send
5	8	CTS	Clear to send
6	6	DSR	Data Set Ready
7	5	SG	Signal Ground
8	1	DCD	Data Carrier Detect
15	-	TxC	Transmit Clock
17	-	RxC	Receive Clock
20	4	DTR	Data Terminal Ready
22	9	RI	Ring Indicator
24	-	RTxC	Transmin/Receive Clock

Existen hasta prácticamente 25 señales más pero no son muy usadas y para usos con el microcontrolador generalmente no son necesarias.

Los pines que portan los datos son RxD y TxD los demás se encargan de otros trabajos, el DTR indica que el computador esta encendido, DSR que el dispositivo conectado al puerto esta encendido, RTS que el computador al no estar ocupado puede recibir datos, al revés de CTS que lo que informa es que es el dispositivo el que puede recibir datos, DCD detecta que existen presencia de datos, etc.

Para controlar al puerto serie, la CPU emplea direcciones de puertos de E/S y líneas de interrupción (IRQ). En el AT-286 se eligieron las direcciones 3F8h (o 0x3f8) e IRQ 4 para el COM1, y 2F8h e IRQ 3 para el COM2. El estándar del PC llega hasta aquí, por lo que

al añadir posteriormente otros puertos serie, se eligieron las direcciones 3E8 y 2E8 para COM3-COM4, pero las IRQ no están especificadas. Cada usuario debe elegirlas de acuerdo a las que tenga libres o el uso que vaya a hacer de los puertos serie (por ejemplo, no importa compartir una misma IRQ en dos puertos siempre que no se usen conjuntamente, ya que en caso contrario puede haber problemas). Es por ello que últimamente, con el auge de las comunicaciones, los fabricantes de PCs incluyan un puerto especial PS/2 para el ratón, dejando así libre un puerto serie.

Antes de iniciar cualquier comunicación con el puerto RS232 se debe de determinar el protocolo a seguir dado que el estándar del protocolo no permite indicar en que modo se esta trabajando, es la persona que utiliza el protocolo el que debe decidir y configurar ambas partes antes de iniciar la transmisión de datos.

Siendo los parámetros a configurar los siguientes:

- Protocolo serie (numero bits-paridad-bits stop)
- Velocidad de puerto
- Protocolo de control de flujo (RTS/CTS o XON/XOFF).

El API de Comunicaciones Java que vamos a usar es el constituido por el paquete javax.comm, que proporciona JavaSoft, no forma parte del JDK, pero añade soporte a Java para dispositivos serie y paralelo.

2.11 Característica básica del lenguaje Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems en 1995. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++. A finales del 2006 e inicios del 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es libre).

Una de las APIs de JFC(Java Fundation Classes) para crear aplicaciones GUI es el llamado Swing escrito totalmente en Java el cual nos proporciona todos los componentes gráficos necesarios para que el entorno de la aplicación sea mas amigable. Justamente para nuestro proyecto usamos esta API.

Para crear nuestra aplicación en Java usamos el Eclipse que es un IDE (entorno de desarrollo integrado que nos permite escribir con suma facilidad código Java) bastante popular en el mercado. Se puede bajar de http://www.eclipse.org/downloads/

2.12 Breve descripción del IDE Eclipse

Una vez que se ha bajado sólo tenemos que copiar en cualquier carpeta y ejecutarlo

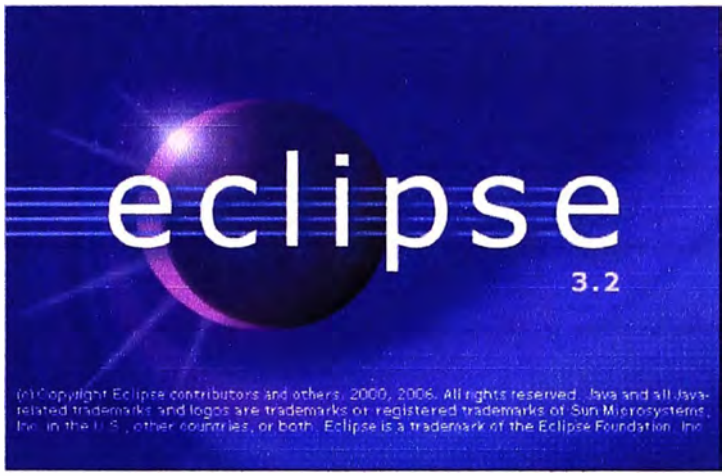


Figura 2.6: Vista inicial del IDE Elipse

Después de indicar el lugar de trabajo nos muestra la siguiente ventana

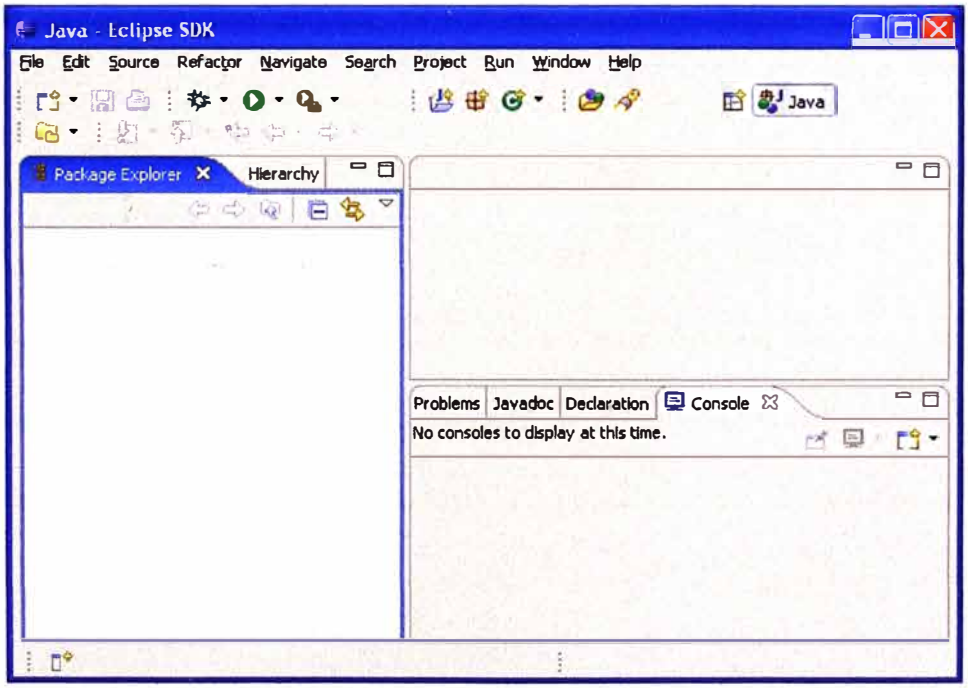


Figura 2.7: Ventana Principal de la aplicación

Existen varias ventanas internas que pueden activarse o no de acuerdo a la necesidad. Para empezar a trabajar debemos crear un proyecto nuevo tal como muestra la figura, en este caso lo denominaremos **Tesis**

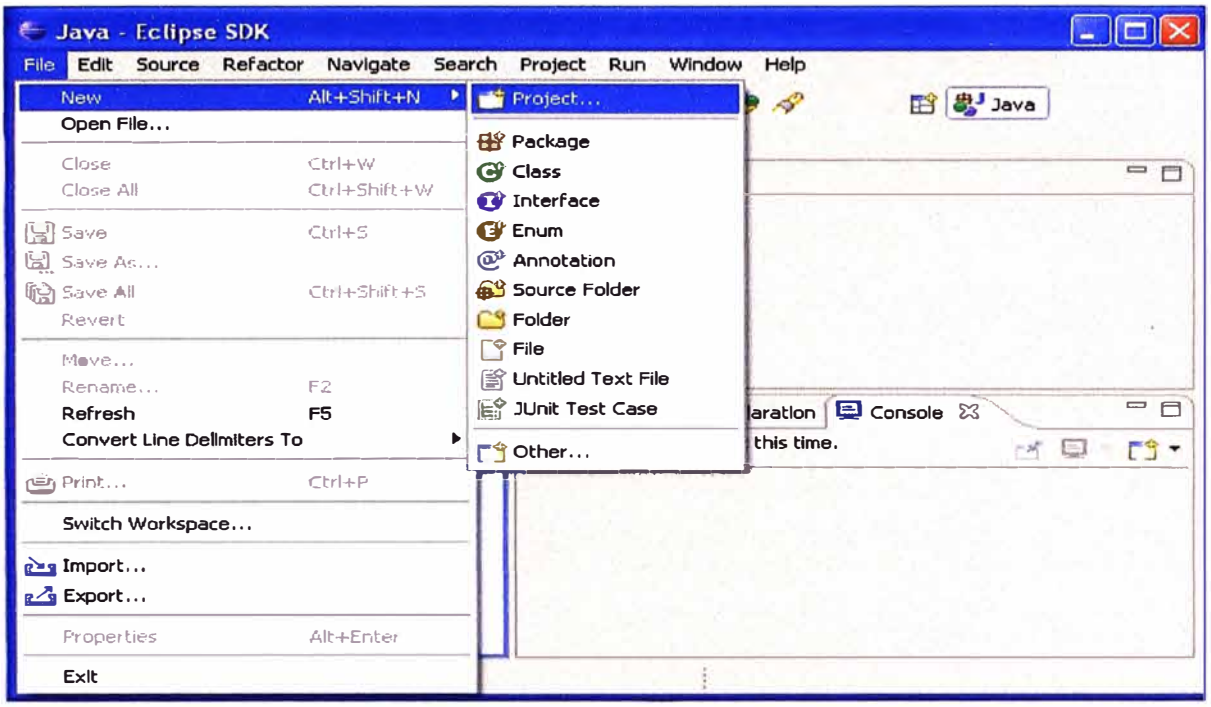


Figura 2.8: Muestra cómo se crea un proyecto Java

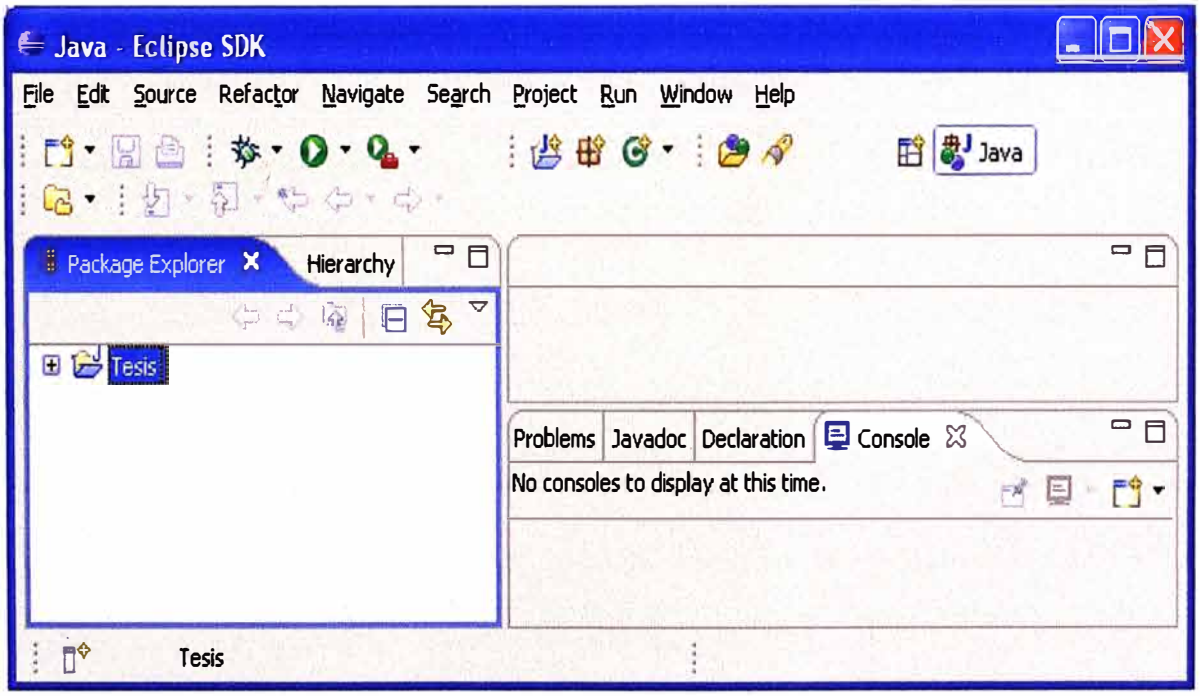


Figura 2.9: Muestra con el proyecto Tesis creado

Ahora ya podemos iniciar nuestra aplicación En java para esto creamos una nueva clase que le denominamos Inicio

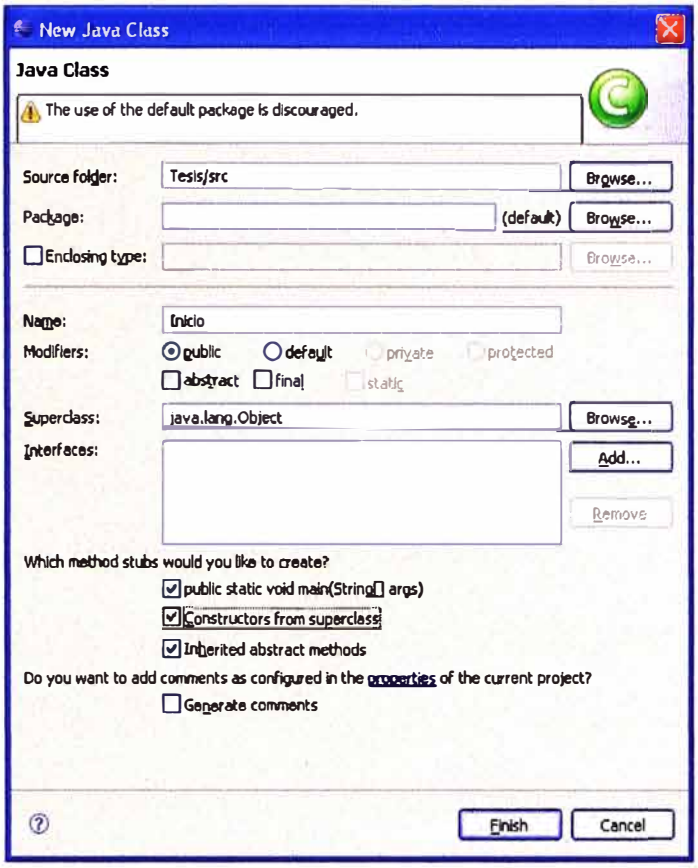


Figura 2.10: Ventana de creación de de la clase Java

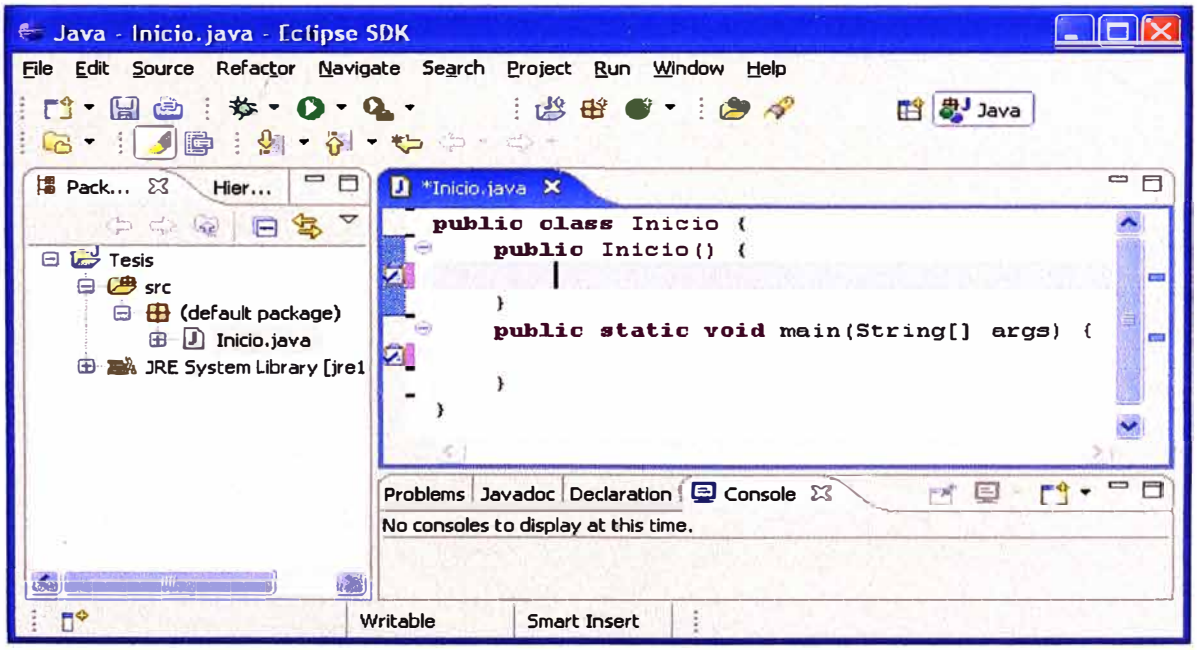


Figura 2.11: Ventana lista para escribir el programa Java

Lo primero que haremos es escribir una clase para mostrar una ventana tipo Windows lo cual debe heredar de todos los componentes Swing que esta dentro de un Librería conocida como JFC (Java Fundation classes) tal como se muestra:

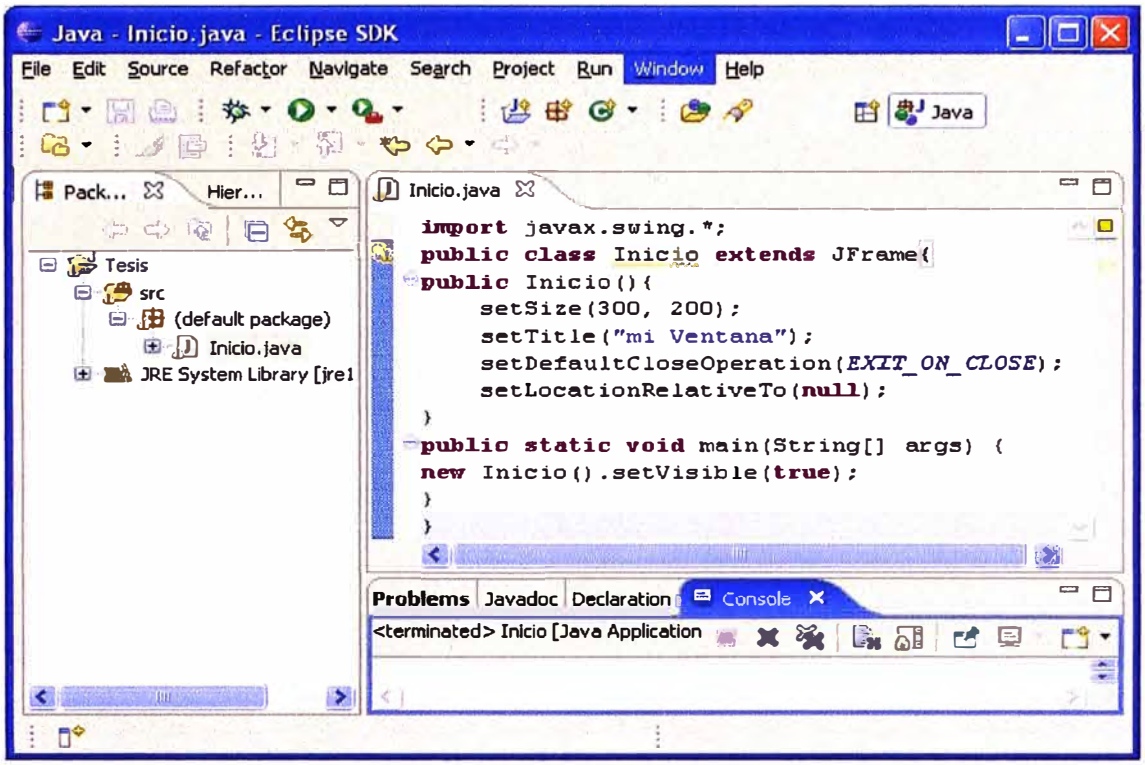


Figura 2.12: Muestra como se escribe el código de una ventana en Java

Como se ve, Eclipse nos facilita la creación de código Java, Verifica los errores de sintaxis u otros que se puedan presentar al Escribir nuestro programas. Por ejemplo el programa escrito arriba corresponde a una ventana como las clásicas de Windows. Creamos nuestra clase denominada Inicio que hereda de la clase JFrame donde colocamos utilizando set las propiedades a nuestra ventana.

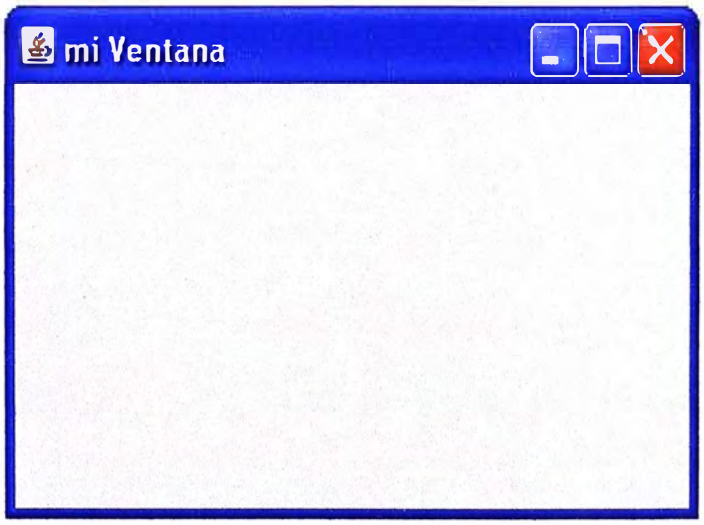


Figura 2.13: Ventana del código mostrado en la figura anterior

2.13 Estructura Básica de un programa en Java

```
public class ejemplo1 {
public static void main(String[] args) {
//Aquí se escribe el Algoritmo en Java
```

- Declaración de variables
- Entrada de datos
- Proceso de cálculo
- Salida de resultados

```
        }
}
```

Declaración de las variables

```
Tipo_de_dato nombre_de_variable;
Tipo_de_dato nombre1, nombre2, nombre3;
```

Tipo de datos que se usan en Java

```
short     entero corto: -127 a 128
byte      entero corto: 0 a 255
int       entero : -32767 a 32768
long      entero largo : 11 digitos positivos y negativos
float     real de baja precisión
double    real de alta precisión
char      caracter
String  Cadena de caracteres
boolean   Tipo lógico (V o F)
```

Ejm :

```
int a, b, c ;
double area;
```

Tabla 2.4: Operadores Aritméticos

Signo	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
%	Residuo

Tabla 2.5: Operadores adicionales

Operación	Matemáticas	Java
Potencia	a^b	Math.pow(a,b)
Raiz cuadrada	$\sqrt{a}$	Math.sqrt(a,b)
Raiz n	$\sqrt[n]{a}$	Math.pow(a,1/n)
Nº Aleatorio	[0,1>	Math.random()

Tabla 2.6: Operadores de Cadena

Signo	Significado
+	Concatenar

Mostrar datos en Pantalla. En Java utilizamos dos formas para mostrar datos en pantalla. Uno de ellos es:

System.out.println(**"texto"**+variable)

Ejm:
System.out.println(**"El resultado es: "**+variable);

Leer datos ingresados por teclado

Ejemplo: Variable=JOptionPane.showInputDialog(**"Mensaje"**);

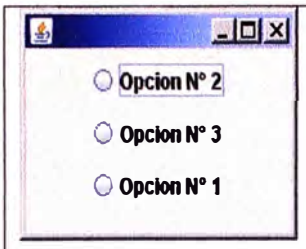
Ojo: Variable es de tipo String, así que debemos convertir a entero o decimal de acuerdo a los requerimientos

Tabla 2.7: Componentes más relevantes de swing

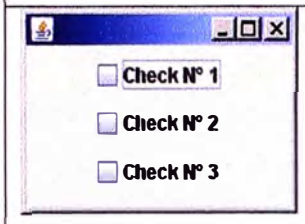
Componentes	Código Java

	`//Boton Aceptar`
	
	`JButton aceptar=new JButton("Aceptar");`
	`getContentPane().add(aceptar);`

	`//Caja de texto`
	
	`JTextField texto=new JTextField(8);`
	`getContentPane().add(texto);`

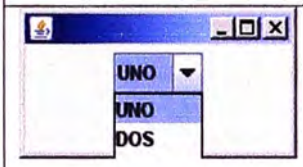



```
//Opciones
JRadioButton rb1=new JRadioButton("Opcion
N°1");
getContentPane().add(rb1);
JRadioButton rb2=new JRadioButton("Opcion
N° 2");
getContentPane().add(rb2);
JRadioButton rb3=new JRadioButton("Opcion
N° 3");
getContentPane().add(rb3);
ButtonGroup bg=new ButtonGroup();
getContentPane().add(rb1);
bg.add(rb1);bg.add(rb2);bg.add(rb3);
```




```
//Elemento Check

JCheckBox cb1=new JCheckBox("Check N° 1");
JCheckBox cb2=new JCheckBox("Check N° 2");
JCheckBox cb3=new JCheckBox("Check N° 3");
getContentPane().add(cb1);
getContentPane().add(cb2);
getContentPane().add(cb3);
```




```
//Combo
JComboBox cb=new JComboBox();
cb.addItem("UNO");
cb.addItem("DOS");
cb.addItem("TRES");
getContentPane().add(cb);
```

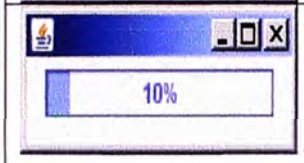

```
//Lista
DefaultListModel dlm=new
DefaultListModel();
dlm.addElement("LUNES");
dlm.addElement("MARTES");
dlm.addElement("MIERCOLES");
JList li=new JList(dlm);
JScrollPane sp=new JScrollPane(li);
li.setVisibleRowCount(3);
getContentPane().add(sp);
```

```
//Slider
JSlider s=new JSlider();
getContentPane().add(s);
```


```
//Spinner
JSpinner spn=new JSpinner();
spn.setValue(100);
getContentPane().add(spn);
```

```
//Imagen
JLabel imagen=new JLabel(new ImageIcon("uni.jpg"));
getContentPane().add(imagen);
```


```
//Area de texto
JTextArea ta=new JTextArea(6,12);
JScrollPane sp1=new JScrollPane(ta);
getContentPane().add(sp1);
```




```
//Barra de progreso
JProgressBar pb=new JProgressBar();
pb.setValue(10);
pb.setStringPainted(true);
getContentPane().add(pb);
```

2.14 API de comunicaciones En JAVA

El API de Comunicaciones Java, constituido por el paquete **javax.comm**, nos permite realizar comunicaciones con los puertos serie RS-232 y el paralelo IEEE-1284, en otras palabras podemos realizar aplicaciones de comunicaciones que utilizan los puertos de comunicaciones (tarjetas inteligentes, fax, etc) independientes de la plataforma.

Como El API de comunicaciones no se encuentra incluido en el JDK y es una extensión de este, deberemos instalar primero este nuevo API en las maquinas que vayamos a realizar el desarrollo.

2.14.1 Instalación del API de comunicaciones

Lo primero que haremos es obtener el API de comunicaciones, este se puede bajar fácilmente de Internet. Una vez que desempaquetemos el fichero procederemos a realizar los siguientes pasos:

- Copiamos el archivo Win32com.dll en el directorio \bin de JDK
  Ejemplo copiar a: C:\jdk1.1.6\bin
- Copiamos el archivo comm.jar al directorio \lib de JDK
  Ejemplo copiar a: c:\jdk1.1.6\lib
- Copiamos el archivo javax.comm.properties al directorio \lib de JDK
  Este archivo debe estar instalado caso contrario el sistema no podrá encontrar ningún puerto. Ejemplo copiar a c:\jdk1.1.6\lib.
- Añadir comm.jar a su classpath:
  - Si no tiene un classpath definido
    C:\>set CLASSPATH=c:\jdk1.1.6\lib\comm.jar
  - Si ya tiene un classpath definido:
    C:\>set CLASSPATH=c:\jdk1.1.6\lib\comm.jar;%classpath%

2.14.2 Características del API de comunicaciones

En el paquete de comunicaciones javax.comm tenemos una serié de clases que nos permiten tratar varios niveles de programación, estos niveles son los siguientes:

Nivel alto: En este nivel tenemos las clases CommPortIdentifier y CommPort que nos permiten el acceso a los puertos de comunicación.
Nivel medio: Con las clases SerialPort y ParallelPort que cubren el interfaces físico RS-232 para el puerto serie y IEEE-1284 para el puerto paralelo.
Nivel bajo: Este nivel ya toca el sistema operativo y en el se encuentra el desarrollo de drivers.

2.14.3 Servicios que nos proporciona este paquete

Poder obtener los puestos disponibles así como sus características.
Abrir y mantener una comunicación en los puertos.
Resolver colisiones entre aplicaciones. Gracias a este servicio podremos tener varias aplicaciones Java funcionando y
utilizando los mismos puertos y no solo sabremos que el puerto esta ocupado sino que podremos saber que aplicación lo esta utilizando.
Disponemos de métodos para el control de los puertos de entrada/salida a bajo nivel, de esta forma no solo nos limitamos a enviar y recibir datos sino que podemos saber en que estado esta el puerto. Así en un puerto serie podremos no solo cambiar los estados sino que podemos programar un evento que nos notifique el cambio de cualquier estado.
Programación a alto nivel
Para esta programación contamos con la clase CommPortIdentifier. Nos encontramos ante la clase principal del paquete ya que lo primero que debemos hacer antes de empezar a utilizar un puerto es descubrir si esta libre para su utilización. El primer programa que nos servirá como ejemplo lo encontramos en el Listado 1, este programa nos enseña la disponibilidad de los puertos. En la Figura A podemos ver el resultado del mismo, el puerto COM2 se encuentra ocupado por el programa "Una aplicación ocupa".

2.15 Algunos dispositivos para un sistema SCADA

2.15.1 Microcontroladores

Es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. En fin estas son básicamente algunas de sus partes...

- **Memoria ROM** (Memoria de sólo lectura)
- **Memoria RAM** (Memoria de acceso aleatorio)
- **Líneas de entrada/salida (I/O)** También llamados puertos
- **Lógica de control** Coordina la interacción entre los demás bloques.

Uno de los más populares microcontroladores son el famoso PIC son de tipo RISC y fabricado por Microchip Technology Inc. Uno de ello es el PIC16F84. El uso más difundido para los microcontroladores es en robótica como cerebro principal del robot.

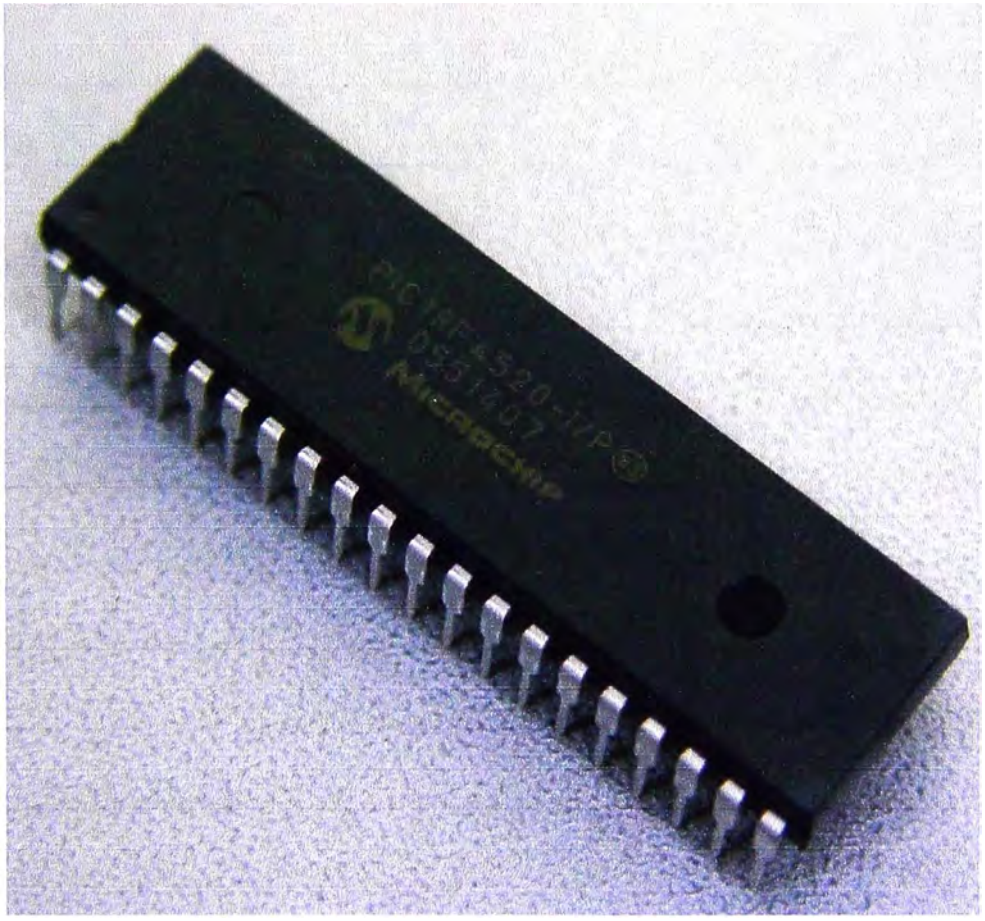


Figura 2.14 Microcontrolador PIC16F84

Por lo visto un microcontrolador es prácticamente un computador en un chip debido a que tiene todas las partes de un computador como Ingreso de datos, proceso, unidad lógica aritmética y salida.

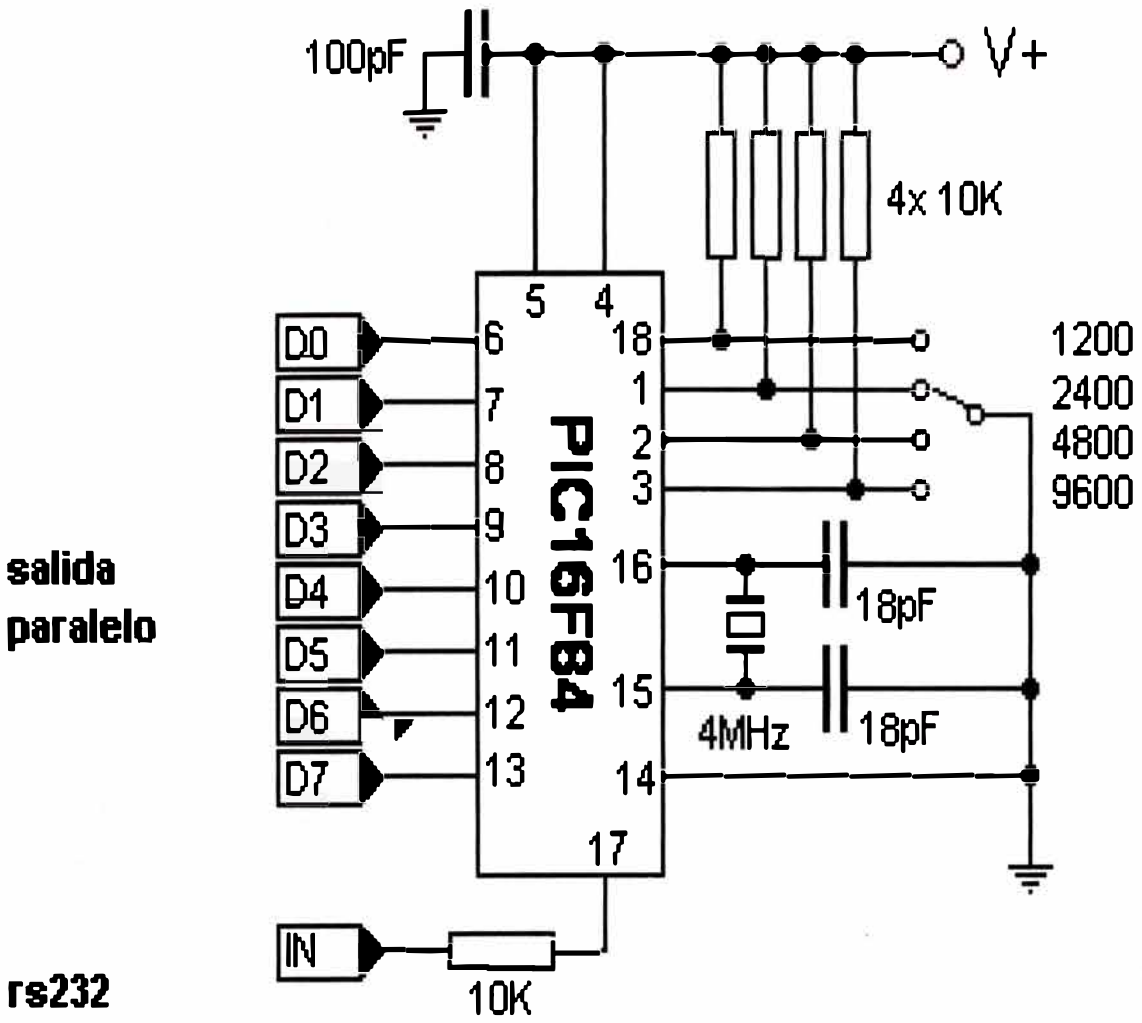


Figura 2.15: Circuito que convierte una salida serial RS232 a paralelo

2.15.2 Actuadores

Un actuador es un elemento que puede provocar un efecto sobre un proceso automatizado. Los actuadores son dispositivos capaces de generar una fuerza a partir de líquidos, de energía eléctrica y gaseosa. El actuador recibe la orden de un regulador o controlador y da una salida necesaria para activar a un elemento final de control como lo son las válvulas.Tenemos tres tipos de actuadores: *Hidráulicos* , *Neumáticos* y *Eléctricos* Estos actuadotes se pueden usar por ejemplo para manejar aparatos mecatrónicos. Si se necesita potencia los actuadores hidráulicos serían recomendables, para el caso de posicionamiento simple se usarían los neumáticos. Los actuadores eléctricos también son muy utilizados en los aparatos mecatrónicos, como por ejemplo, en los robots.

Algunos ejemplos de actuadores:

- **Cilindros neumáticos e hidráulicos**. Realizan movimientos lineales.
- **Motores (actuadores de giro) neumáticos e hidráulicos**. Realizan
- movimientos de giro por medio de energía hidráulica o neumática.
- **Válvulas**. Las hay de mando directo, motorizadas, electroneumáticas, etc. Se emplean para regular el caudal de gases y líquidos.
- **Resistencias calefactoras**. Se emplean para calentar.
- **Motores eléctricos**. Los más usados son de inducción, de continua, sin escobillas y paso a paso.
- **Bombas, compresores y ventiladores**. Movidos generalmente por motores eléctricos de inducción.

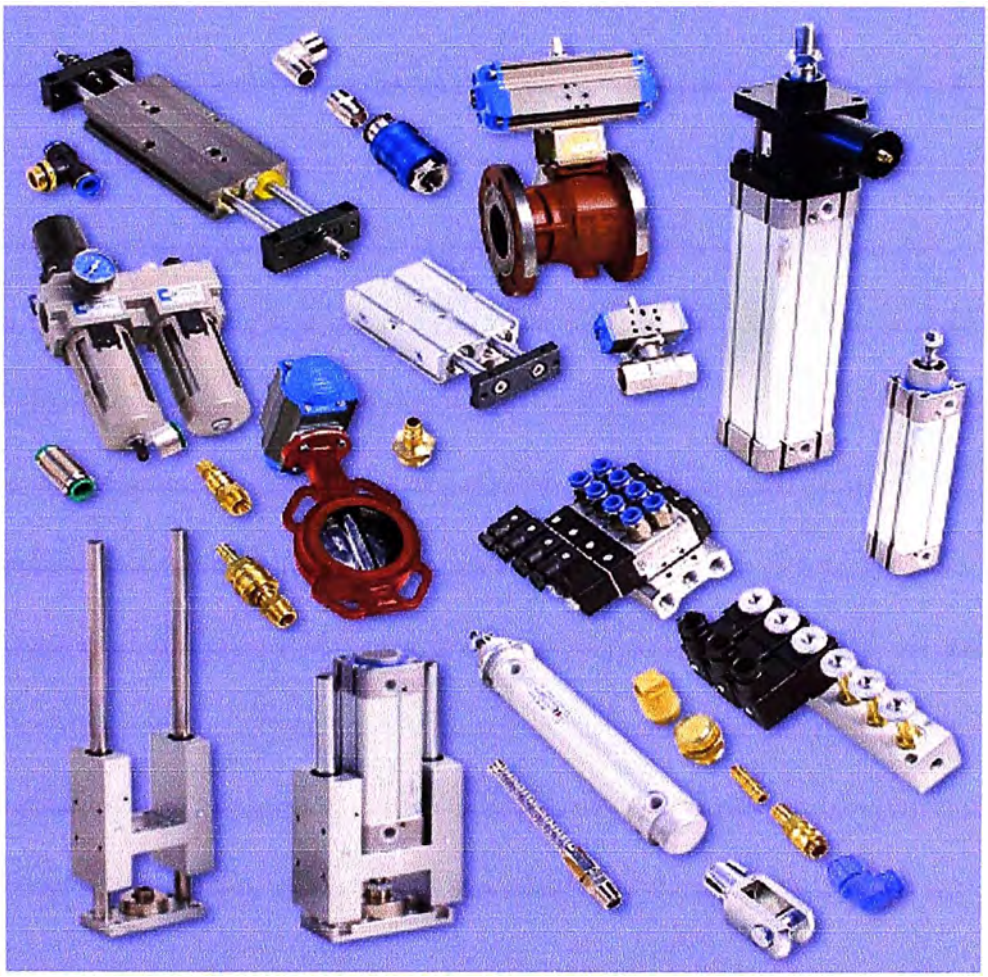


Figura 2.16: actuadores neumáticos y eléctricos

2.15.3 Sensor

Un sensor es un dispositivo que puede transformar magnitudes físicas o químicas, en magnitudes eléctricas. Esta transformación dependen del tipo de sensor y pueden ser por ejemplo temperatura, intensidad luminosa, distancia, aceleración, inclinación, desplazamiento, presión, fuerza, torsión, humedad, pH, etc. Una magnitud eléctrica obtenida puede ser una resistencia eléctrica (como en una RTD), una capacidad eléctrica (como en un sensor de humedad), una tensión eléctrica (como en un termopar), una corriente eléctrica (como un fototransistor), etc. Como se observa un sensor puede medir prácticamente todas las magnitudes físicas. Muchas veces podemos confundir entre un sensor y un transductor, sin embargo existen diferencias como que el sensor está siempre en contacto con la variable a medir o a controlar.

Recordando que la señal que nos entrega el sensor no solo sirve para medir la variable, si no también para convertirla mediante circuitos electrónicos en una señal estándar para fines de control de dicha variable en un proceso. Un sensor también puede decirse que es un dispositivo que convierte una forma de energía en otra.

Algunas aplicaciones de los sensores: Industria automotriz, Industria aeroespacial, Medicina, Industria de manufactura, Robótica, etc.

Figura 2.17: sensores de proximidad y fotoeléctricos

CAPITULO III

METODOLOGÍA PARA LA SOLUCIÓN DEL PROBLEMA

3.1 Creación de la base de datos.

Previo un análisis de requerimientos podemos crear la base de datos que nos servirá de base para almacenar toda la información requerida en el proceso, en este caso constituido por dos tablas por ejemplo la tabla Usuario donde guardaremos los datos de los operadores y otra tabla denominada historial donde se guardarán los datos históricos es decir la fecha y hora en que se manipulo los dispositivos de campo.

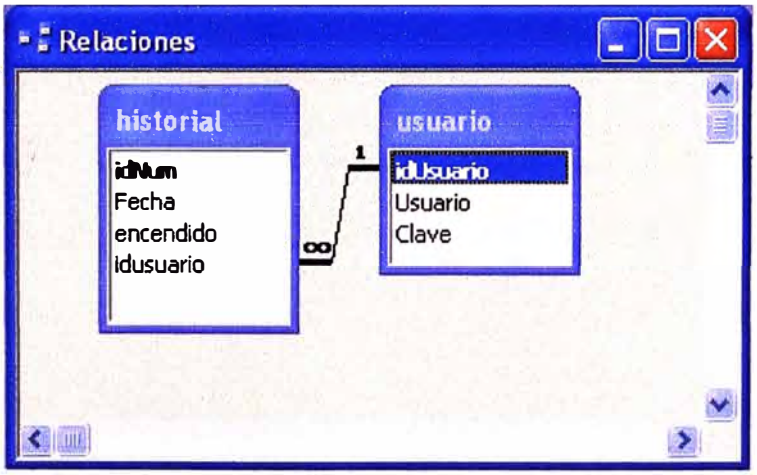


Figura 3.1: Modelo de la base de datos del SCADA

3.2 Creando las Interfaces

La ventana de validación es muy importante para verificar quién es el que esta usando el sistema para esto debemos hacer una conexión a la base de datos, en este caso usamos el puente JDBC-ODBC para conectar el lenguaje Java con la Base de Datos.

Figura 3.2: Ventana de validación de usuarios

3.2.1 Código Java para la ventana de validación

```
import java.awt.event.*;
import java.sql.*;
import javax.swing.*;
public class Inicio extends JFrame implements ActionListener{
private static final long serialVersionUID = 1L;
JButton cmdAceptar,cmdCancelar;
JTextField txtUsuario;
JPasswordField txtClave;
public static Connection cn=null;
public Inicio(){
// para que el frame pueda cerrarse
setDefaultCloseOperation(EXIT_ON_CLOSE);
setSize(320,200);
setLocationRelativeTo(null); //para centrar en pantalla
setTitle("Inicio");
setIconImage(new ImageIcon("img/key.gif").getImage());//Colocando el Icono
setResizable(false); // para que el formulario no cambie de tamaño
getContentPane().setLayout(null); //distribucion manual de componentes
txtUsuario=new JTextField();// Instanciando la caja de texto
txtUsuario.setBounds(150, 36, 100, 20); //Poniendo el tamaño y la ubicacion
getContentPane().add(txtUsuario); // Colocando la caja de texto
txtClave=new JPasswordField();//Instanciando la caja de texto para pasword
txtClave.setBounds(150, 76, 100, 20); //poniendo el tamaño y la ubicacion
getContentPane().add(txtClave);
```

```
cmdAceptar=new JButton("Aceptar",new ImageIcon("img/ok.gif"));
cmdAceptar.setMnemonic('A'); //Atajo para usar con el ALT
cmdAceptar.setBounds(40, 120, 110, 30); //colocando el boton aceptar
cmdAceptar.addActionListener(this);
getContentPane().add(cmdAceptar);
//colocando el boton cancelar
cmdCancelar=new JButton("Cancelar",new ImageIcon("img/cancelar.gif"));
cmdCancelar.setBounds(180, 120, 110, 30);
cmdCancelar.setMnemonic('C');
getContentPane().add(cmdCancelar);
cmdCancelar.addActionListener(this); // Evento escuchador para el boton Cancelar
JLabel fondo=new JLabel(new ImageIcon("img/inicio.gif"));// colocando el fondo
fondo.setBounds(0,0,320,180);
getContentPane().add(fondo);}
public static void main(final String[] args){
Inicio in=new Inicio();
in.setVisible(true);}
public void actionPerformed(ActionEvent e) {
if(e.getSource()==cmdCancelar){
System.exit(0);}
if(e.getSource()==cmdAceptar){
// Codigo para verificar si la clave es correcta
try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
String usuario=txtUsuario.getText();
String clave=String.valueOf(txtClave.getPassword());
cn=DriverManager.getConnection("jdbc:odbc:Driver={Microsoft Access Driver
(*.mdb)};DBQ=data/tienda.mdb");
ResultSet rs=cn.createStatement().executeQuery("select * from usuario where
idusuario='"+usuario+"' and clave='"+clave+"'");
if (rs.next()){
Principal p=new Principal();
this.setVisible(false);
p.setVisible(true);}
else{
System.out.println("Noooo");
```

```
}
}
catch(ClassNotFoundException cnfe){
System.out.println("...Error Clase no definida");
}
catch(SQLException sqle){
System.out.println("...Error de coneccion");
}
}
}
}
```

3.2.2 Código Java para Analizar los puertos

Este programa nos permite analizar la disponibilidad de todos los puertos soportados en nuestra maquina. También nos informara del tipo del puerto así como el propietario de este en caso de que se encuentre ocupado.

```
import java.io.*;
import java.util.*;
import javax.comm.*;
public class P1
{
   static Enumeration listaPort;
   static CommPortIdentifier idPort;
   public static void main(String[] args)
   {
      listaPort = CommPortIdentifier.getPortIdentifiers();
      ListaPuertos();
   }
   private static void ListaPuertos()
   {
      System.out.println("Los puertos disponibles son:");
      while (listaPort.hasMoreElements())
      {
         idPort = (CommPortIdentifier) listaPort.nextElement();
         System.out.print("PUERTO: " + idPort.getName() + " ");
         if (idPort.getPortType() == CommPortIdentifier.PORT_SERIAL)
         {
```

```
            System.out.println("RS-232 (" + idPort.getPortType() + ")");
        } else if (idPort.getPortType() == CommPortIdentifier.PORT_PARALLEL)
          {
            System.out.println("IEEE 1284 (" + idPort.getPortType() + ")");
          } else System.out.println("Tipo de puerto desconocido");
        if (idPort.isCurrentlyOwned())          // Describimos si esta disponible.
          System.out.println("OCUPADO por: " +
                             idPort.getCurrentOwner());
        else
          System.out.println("DISPONIBLE");
        System.out.println("-----------------------------------------");
      } }}
```

Lo primero que debemos hacer antes de intentar abrir un puerto será ver si ya tiene o no un propietario y para obtener la información de esto debemos obtener el objeto de **CommPortIdentifier** correspondiente al puerto que se realizara con alguno de los siguientes métodos:

getPortIdentifiers(): Es el método utilizado que nos entregará un enumerado con tantos objetos CommPortIdentifier como puertos dispongamos.

getPortIdentifier(String) Nos dará el objeto correspondiente al puerto que se le pase como parámetro, este será el método que normalmente usaremos ya que lo normal es que siempre nos conectemos por el mismo puerto. Este método deberá saber tratar la excepción NoSuchPortException que saltara en el caso de que solicitemos un puerto inexistente.

Una vez que tenemos el objeto del puerto tenemos una serie de métodos que nos permitirán obtener información del puerto y abrirlo para poder iniciar una comunicación. Estos métodos son los siguientes: getName() dará el nombre del puerto. En el caso de haber utilizado el método getPortIdentifier(String) será el mismo valor que pasamos como parámetro.

getPortType() Devuelve un entero que nos informa del tipo de puerto (serie o paralelo), para no tener que acordarnos del valor de cada tipo de puerto disponemos de las constantes PORT_PARALLEL y PORT_SERIAL.

isCurrentlyOwned() nos informa si esta libre o no el puerto. En caso de que este ocupado podremos saber quien lo esta utilizando mediante el método getCurrentOwner().

open(String, int) Abre y por lo tanto reserva un puerto, en el caso de que intentemos abrir un puerto que ya se encuentre en uso saltara la excepción PortInUseException. Los

parámetros que le debemos pasar son un String con el nombre de la aplicación que reserva el puerto y un int que indica el tiempo de espera para abrir el puerto.

Un ejemplo de este método se puede ver en el Listado 2. Este método nos da un objeto CommPort, realmente esto es una clase abstracta de la que heredan las clases SerialPort y ParallelPort. Una vez que se tiene este objeto se deberán encaminar sus salidas a OutputStream y las entradas a InputStream una vez realizado esto la escritura y lectura de los puertos serán tan fácil como utilizar los métodos de estas clases que pertenecen al standard del JDK.

addPortOwnershipListener(CommPortOwnershipListener) Permite añadir una clase que escuche los cambios de propietarios en los puertos.

removePortOwnershipListener(CommPortOwnershipListerner). Para eliminar un oyente de eventos si tenemos registrado uno

3.2.3 Código Java para Escribir en el puerto

Este programa escribe una frase por el puerto que le pasemos como parámetro.

```
 import java.io.*;
import java.util.*;
import javax.comm.*;
public class Escritor
{  static CommPortIdentifier idPort;
   static SerialPort sPort;
   static ParallelPort pPort;
   static OutputStream salida;
   static String datos = new String("HOLA! esto es una prueba");
   static evProp ev = new evProp();
   public static void main(String[] args)
{
      try     // Lo primero que hacemos es abrir el puerto
     {
       idPort = CommPortIdentifier.getPortIdentifier(args[0]);
       idPort.addPortOwnershipListener(ev);
     } catch (NoSuchPortException e)
      {System.err.println("ERROR al identificar puerto");
}
     try      // Abre el puerto necesario.
     {
```

```
if (idPort.getPortType() == CommPortIdentifier.PORT_SERIAL)
{
    sPort = (SerialPort) idPort.open("Escritor en serie", 30000);
          try {
            salida = sPort.getOutputStream();
          } catch (IOException e) {}
        }
        else {
    pPort = (ParallelPort) idPort.open("Escritor en paralelo", 30000);
          try
          {
            salida = pPort.getOutputStream();
          }
          catch (IOException e) { }
        }
}
catch (PortInUseException e)
        { System.err.println("ERROR al abrir puerto");
}
       try
       {
         salida.write(datos.getBytes());
       } catch (IOException e) { System.err.println("Error escritura");
}
}
}
class evProp implements CommPortOwnershipListener
{
   public void ownershipChange(int tipo)
   {
     System.out.print("Valor: " + tipo);
     if (tipo == CommPortOwnershipListener.PORT_OWNED)
       System.out.println(" Se abre el puerto");
     else if (tipo == CommPortOwnershipListener.PORT_UNOWNED)
       System.out.println(" Se cierra el puerto");
     else if (tipo == CommPortOwnershipListener.PORT_OWNERSHIP_REQUESTED)
```

```
        System.out.println(" Requerido puerto usado");
    }
}
```

Explicación de la Clase CommPort

Esta es una clase abstracta que describe los métodos comunes de comunicación y serán las clases que hereden de ellas (SerialPort y ParallelPort) la que añadan métodos y variables propias del tipo del puerto. Entre los métodos necesarios tenemos:

close() nos permitirá liberar el puerto que se reservo con open, este notificara el cambio de dueño a las clases que se hubiesen registrado con el método addPortOwnershipListener que se explico anteriormente.

getInputStream() nos permitirá enlazar la entrada del puerto al InputStream que nos devuelve para leer del puerto.

getOutputStream() nos permitirá enlazar la salida del puerto al OutputStream que nos devuelve para poder escribir en el puerto de la misma forma que si se escribiera en un fichero.

getInputBufferSize() nos informa del tamaño que tiene el buffer de entrada del puerto. Este tamaño se puede modificar con el método setInputBufferSize(int). Estos métodos no siempre dan el resultado esperado ya que sea la memoria disponible y el sistema operativo quien al final decida si realizar o no correctamente la operación. **getOutputBufferSize()** nos informa del tamaño que tiene el buffer de salida, como en el caso anterior contamos con un método para modificar el tamaño que es **setOutputBufferSize(int).** Al igual que pasaba con los métodos anteriores su correcto funcionamiento dependen del sistema operativo en si y del desarrollo del driver.

Una vez que hemos visto los principales métodos de la clase abstracta de la que heredan (ParallelPort y SerialPort) pasamos a ver con detenimiento las características de estas.

Explicación de la Clase SerialPort

En esta clase encontramos el interfaces de bajo nivel del puerto paralelo que cumple el standard RS-232. En la emisión de un carácter se realizarán las siguientes comprobaciones:

Ponemos el DTR (Indicando que estamos preparados)

Ponemos RTS (Solicito permiso para emitir)

Comprueba que esta DSR (Mira si el destinatario esta listo)

Esperamos a CTS (Se espera a que me autoricen a enviar datos)

Enviamos datos.

La clase SerialPort hereda de la clase abstracta CommPort y por lo tanto cuenta con sus métodos pero además de estos dispones de otros métodos y variables específicas para el tratamiento de los puertos serie.

CAPITULO IV

ANÁLISIS Y PRESENTACIÓN DE RESULTADOS

4.1 Análisis descriptivo de hardware creado.

Se desarrollo la aplicación java utilizando Eclipse que es un aplicativo que me permite escribir programas en Java con suma facilidad. Para la conexión física del puerto paralelo a los dispositivos externos creamos una placa con transistores para elevar la corriente y activar el Rele que me permite conectar dispositivos de potencia como motores o lámparas de prueba. Además me permite aislar de la etapa de potencia.

Sabemos que los niveles de tensión y corriente presentes en el puerto paralelo del PC responden a los estándares de la familia lógica TTL, es decir un estado alto que corresponde a 5V continua y un estado bajo idealmente 0V. Para el caso de corriente, cada pin del puerto paralelo puede proporcionar hasta 10mA llegando por periodos breves de tiempo hasta 20mA. Directamente se puede conectar desde la salida del puerto paralelo a circuitos integrados lógicos TTL ó CMOS sin necesidad de amplificar la señal, incluso es posible encender un diodo LED. Sin embargo para nuestro caso no podemos conectar directamente debido a que debemos controlar un motor de potencia, para esto utilizamos Reles como interruptor electromecánico, Cuando se alimenta la bobina del Relé, este genera un campo magnético que acciona el interruptor mecánico. Ese interruptor es el encargado de manejar la potencia en sí, quedando al circuito electrónico la labor de "mover" la bobina. Permitiendo así el aislar mecánicamente la sección de potencia de la de control. Una de las ventajas que tiene un Relé es la completa separación eléctrica entre la corriente de accionamiento (la que circula por la bobina del electroimán) y los circuitos controlados por los contactos, lo que hace que se puedan manejar altos voltajes o elevadas potencias con pequeñas tensiones de control. Todavía existe un problema, tampoco se puede conectar directamente un Relé ya que necesita una corriente que el puerto paralelo no lo puede proporcionar, entonces recurrimos al uso del transistor que lo usaremos en configuración corte / saturación. Esto implica que cuando está inactivo presenta una alta resistencia entre el emisor y el colector lo que impide que la corriente circule entre la batería y la bobina del Relé. Pero

cuando está activo (saturando) presenta una muy baja resistencia actuando como un interruptor cerrado que permite el paso de la corriente. Obviamente el transistor es mucho más rápido que el Relé debido a que su funcionamiento no esta ligada a partes mecánicas móviles como sucede con el Relé. Para que el transistor presente un estado de corte basta con no colocar señal en su base. En cambio, para que el transistor sature hay que dar a la base una señal cuya tensión y corriente sean suficientes para colocarlo en ese estado. Por seguridad la base no se conecta directamente al pin del puerto paralelo, sino que se hace en serie con una resistencia de 10K ohms la cual hace las veces de limitadora de corriente. También hay que tener en cuenta que la bobina del relé, cuando deja de recibir corriente, produce lo que se suele denominar como potencial inverso. Básicamente es una corriente que circula en sentido inverso al que estaba circulando cuando el transistor estaba saturando. Esta corriente inversa es muy peligrosa para el transistor y por ende la placa del computador, por lo que se debe colocar un diodo en paralelo con la bobina que deje circular la corriente en un sentido y no en otro. De esta forma el transistor queda protegido contra la corriente inversa producida por el Relé al dejar de funcionar. Los transistores que usamos dependen de la capacidad de tensión y corriente que puede soportar.

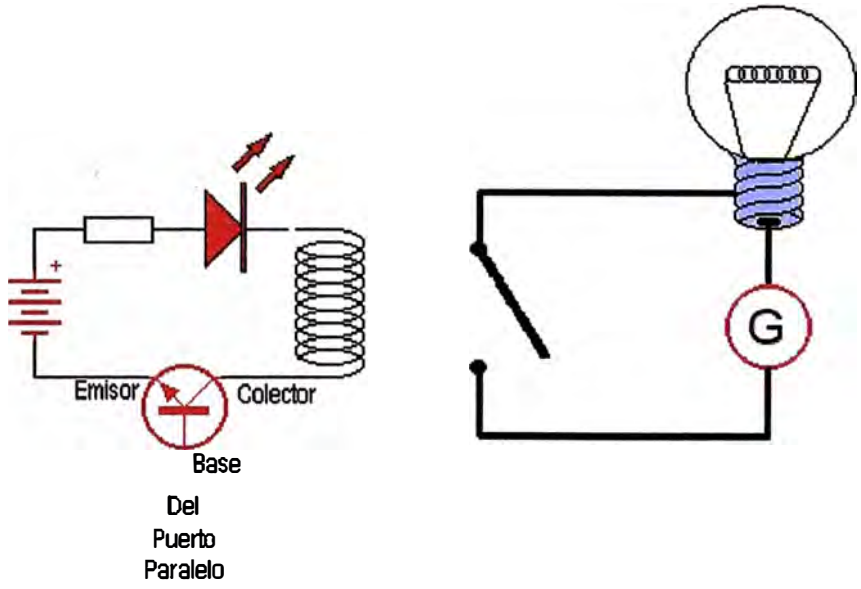


Figura 4.1: Circuito que activa un foco con Relé desde el puerto paralelo de la pc

En base al análisis anterior se construye el siguiente circuito que va a servir de base para la conexión de hasta 8 equipos al puerto paralelo de la pc. Tenga en cuenta que se tiene que alimentar con una fuente de corriente continua de 12 voltios para activar el Relé, además la alimentación de los equipos de potencia (220 voltios)

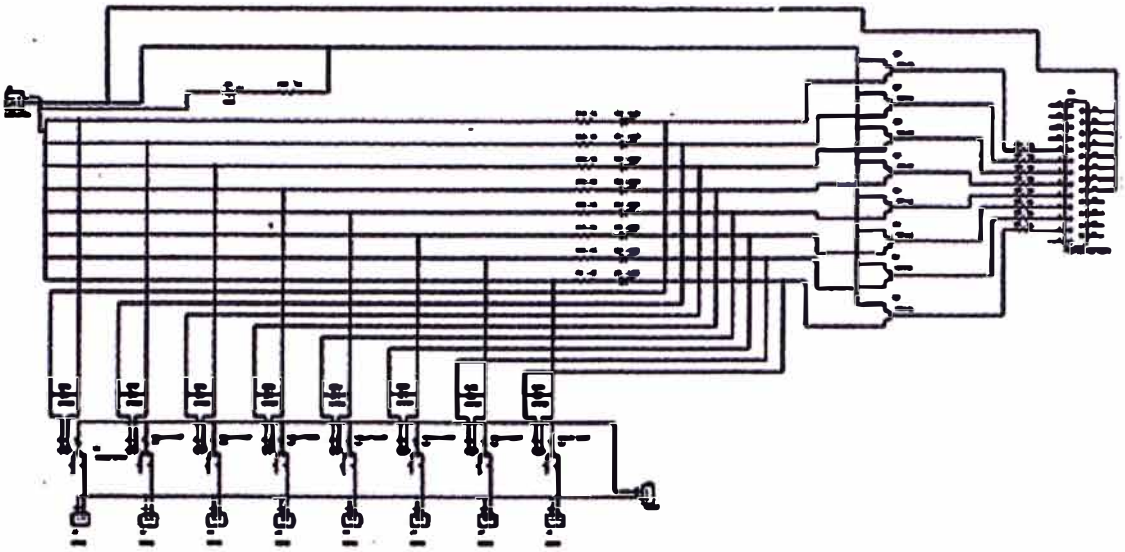


Figura 4.2: Circuito de control de 8 salidas para puerto paralelo

Construimos el circuito impreso correspondiente al circuito mostrado en la figura 4.2 para luego implementar el circuito físico con los componentes soldados.

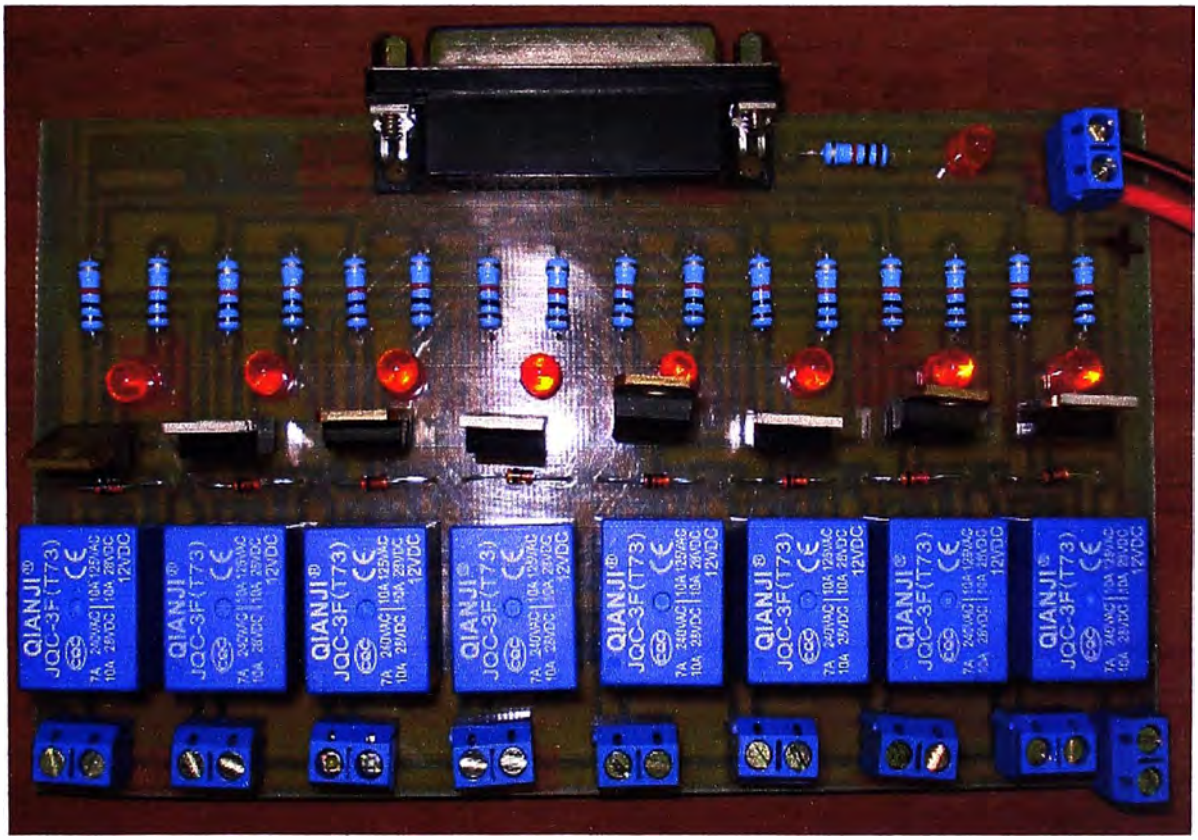


Figura 4.3: Placa para el control de Potencia por puerto paralelo vista 1

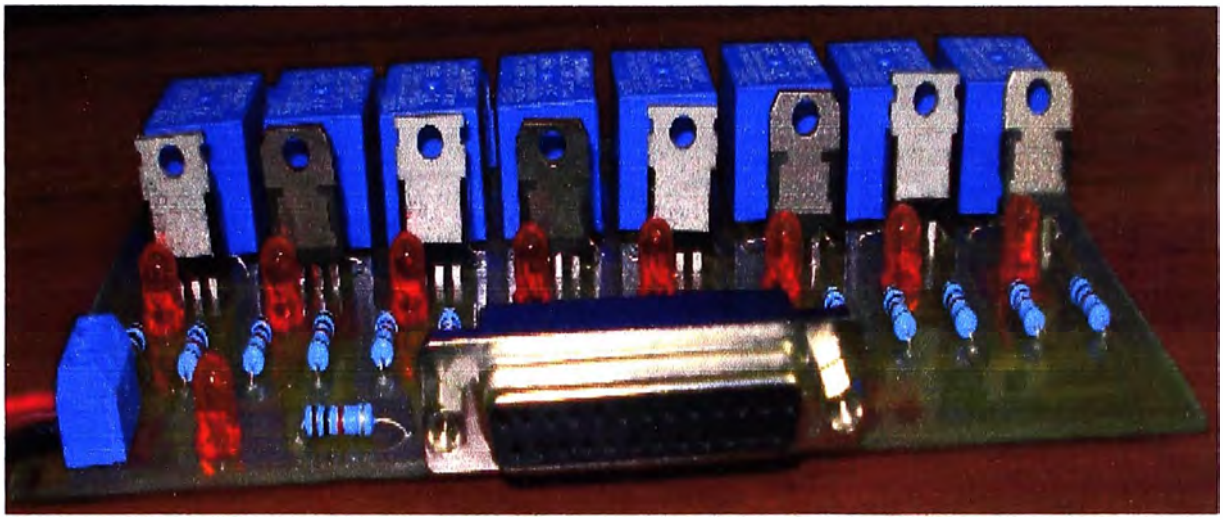


Figura 4.4: Placa para el control de Potencia por puerto paralelo vista 2

4.2 Análisis descriptivo del software creado. Mostraremos el manual del sistema creado totalmente en Java, con el fin de demostrar su funcionalidad probando y registrando datos de prueba.

4.2.1 Ingreso al sistema.

Para ingresar al sistema se necesita un usuario y clave que me permitirá validar si es o no el usuario registrado previamente. Esto se realiza conectándose a la base de datos y consultando a la tabla usuario. Previamente ya se registro los usuarios en la base de datos.

Figura 4.5: Ventana de ingreso al sistema

4.2.2 Dentro del sistema.

Una vez dentro, el sistema nos muestra la pantalla principal con un menú para el control. El sistema principal tiene varias opciones como por ejemplo para reconocer los puertos disponibles

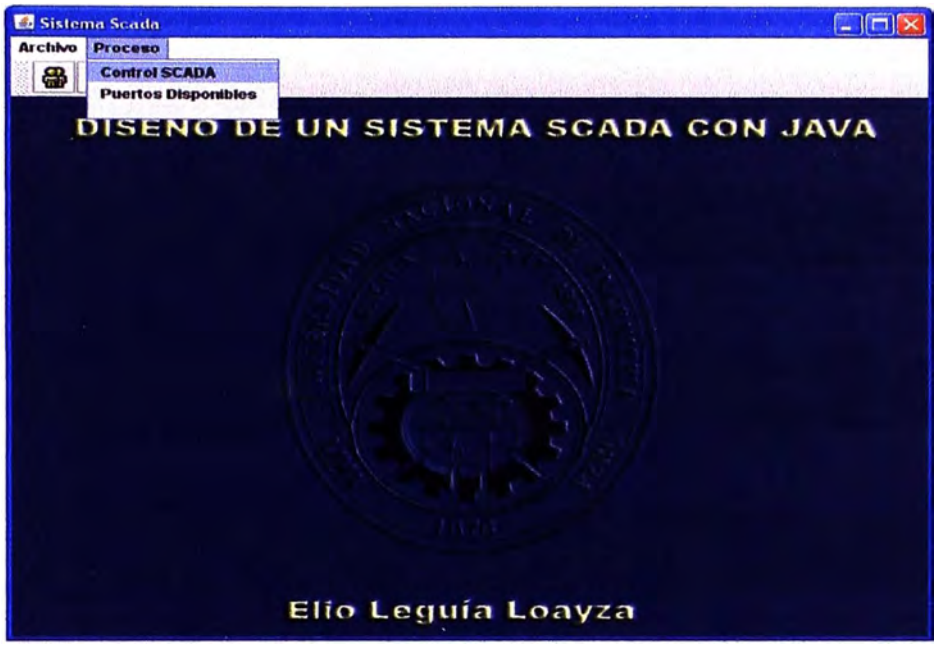


Figura 4.6: Ventana principal del sistema

4.2.3 Inicio de la aplicación.

Ejecutando el sistema de control del motor. El usuario puede colocar un retardo y hacer clic en switch para iniciar el conteo decrementando el valor que indica el retardo hasta que sea cero

Figura 4.7: Ventana de aplicación antes de prender

4.2.4 Ejecución de la aplicación.

Una vez transcurridos los 100 pulsos el altavoz indica con un sonido que el motor ya esta en funcionamiento Registrándose así la fecha y hora en que se encendió.

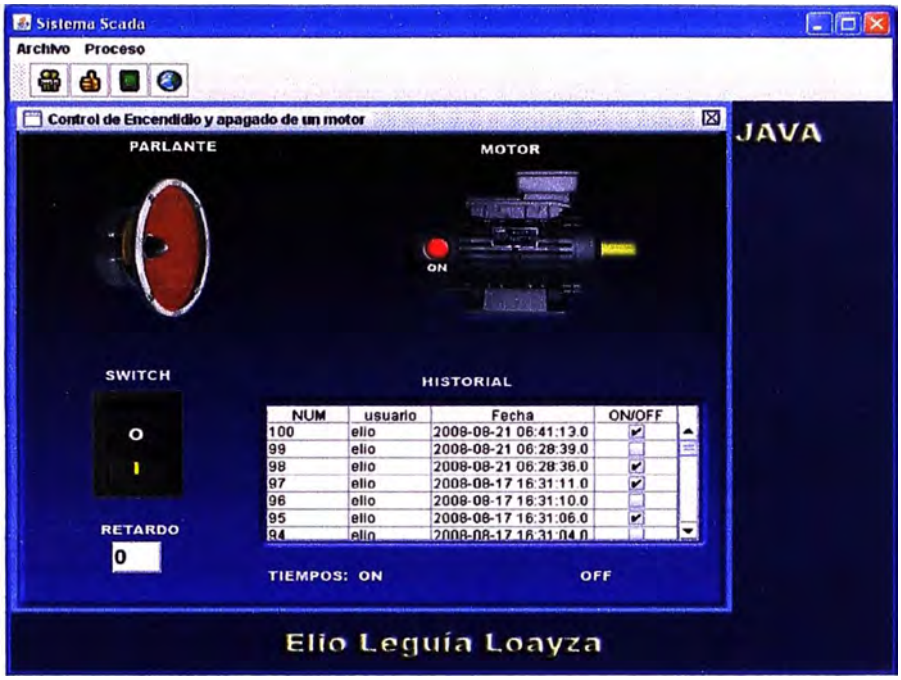


Figura 4.8: Ventana de aplicación en funcionamiento

4.3 Análisis teórico del sistema desarrollado.

El sistema scada desarrollado puede tener algunas dificultades si se quiere usar como un sistema de precisión donde el tiempo de respuesta sea imprescindible, aunque en este caso se guarda con precisión el momento mismo en que se dio la orden del prendido o apagado del dispositivo a controlar ya que se usa el motor de base de datos. El mismo sistema puede analizar toda la información guardada en la tabla historial en el momento o en cualquier fecha posterior. Se puede agregar formularios para realizar reportes de los análisis de datos. Necesitamos saber por ejemplo la estadística de prendido o apagado del motor. Por ejemplo si la base de datos contiene la siguiente información del prendido o apagado de los motores realizados por los usuarios los cuales han sido registrados. Tenemos el siguiente dato de muestra

Tabla 4.1: Usuarios que manipulan el motor

idUsuario	Usuario	Clave
C0001	Elio	123456
C0002	Juber	654321
C0003	Cesar	569854
C0004	Luis	879562

Tabla 4.2: Registro del encendido o apagado del motor

idNum	Fecha	encendido	idUsuario
1	12/05/08:08:45	true	C0001
2	12/05/08:14:46	false	C0001
3	13/05/08:08:50	true	C0002
4	14/05/08:16:30	false	C0003
5	15/05/08:09:00	true	C0004
6	16/05/08:14:00	false	C0003
7	17/05/08:20:00	true	C0002
8	18/05/08:10:00	false	C0001
9	20/05/08:14:00	true	C0003
10	21/05/08:19:00	false	C0004

Algunas consultas que se introdujeron en el programa desarrollado en SQL (lenguaje estructurado de consultas)

- **Listado de Usuario del sistema**

  SELECT * FROM usuario
- **Listado del último usuario que prendió el motor**

  SELECT * FROM usuario u, historial h WHERE u.idusuario=h.idusuario and fecha IN (SELECT MAX(fecha) FROM historial WHERE encendido=true)
- **Listado de todo el historial del sistema**

  SELECT * FROM historial
- **Listado del historial en una fecha determinada (fechaDet)**

  SELECT * FROM historial WHERE fecha=fechaDet
- **¿Cuántas veces se ha apagado el motor?**

  SELECT count(*) FROM WHERE encendido=true;
- **¿Quién y que día apagó el motor por ultima vez?**

  SELECT * FROM usuario u, historial h WHERE u.idusuario=h.idusuario and fecha IN (SELECT MAX(fecha) FROM historial WHERE encendido=false)

Como hemos visto El lenguaje de consultas SQL nos proporciona prácticamente todo lo que necesitamos saber de la base de datos. Lo único que debemos preocuparnos es que los datos se ingresen al momento de manejar el sistema para controlar el motor. En el caso que se necesite ampliar la información donde deben registrarse otros datos,

tendríamos que reconstruir y modelar la base de datos para nuevos requerimientos usando ya sea UML o un modelador sencillo como el Erwin

4.4 Presupuesto y tiempo de ejecución.

Costo del proyecto desarrollado:

- Una computadora que cuente con puerto Serial o paralelo para la comunicación cuyo costo en el mercado es de 600 dólares.
- Un Dispositivo para controlar equipos de potencia por puerto paralelo. Costo promedio de 100 dólares con mano de obra incluido
- Una fuente de alimentación de corriente continua 12 V. Costo 50 dólares
- Creación del software considerando el análisis e investigación. Presupuesto 500 dólares.
- Otros dispositivos como cables y carga de potencia para las pruebas
- Un motor para las pruebas.
-

Tiempo de ejecución:

- Investigación teórica de los puertos. 1 semana
- Prueba de comunicación de puertos con el API. 1 semana
- Creación de la placa para la comunicación del puerto. 1 semana
- Desarrollo del software Interfase gráfica 1 mes
- Recopilación, digitación del documento. 15 días

CONCLUSIONES Y RECOMENDACIONES

Con un sistema SCADA a medida se puede controlar casi cualquier cosa así como recoger todo tipo de información en el proceso productivo. En la actualidad la metodología del levantamiento de información y requerimientos se realizan casi de la misma forma que cuando se desarrollan sistema de información con bases de datos, algo así como cuando se realizan sistemas para un tienda comercial por ejemplo, sin embargo un Ingeniero electrónico además de desarrollar el sistema de control tiene que investigar y analizar los dispositivos que se van a controlar. Gracias al Internet podemos compartir nuestra investigación con muchos profesionales que intervienen en foros lo que facilita solución de cualquier dificultad casi inmediata. En la actualidad esta en prueba los APIs Java para controlar por medio del puerto USB cualquier dispositivo externo así como recoger la información de campo. Con Java también me facilita el uso con dispositivos móviles o a través de Internet por medio de los conocidos como Aplets lo que no es muy difícil de migración de aplicaciones swing desarrollada en el proyecto.

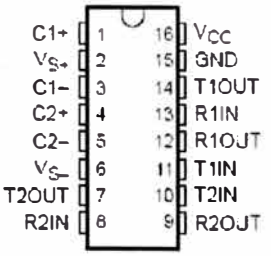
Una de la recomendaciones importante que debemos tener en cuenta es que el puerto paralelo es bastante delicado si no se toma en cuenta las recomendaciones puede malograr su placa definitivamente quedando inutilizada inclusive por manos experimentadas para esto debemos estar seguros de que la etapa de potencia debe estar aislada con respecto a la etapa de baja potencia.

Finalmente concluimos que Java no tiene nada que envidiar a otros lenguajes como C o C++ o Pascal debido a que pude inclusive controlar puertos que antes era casi de exclusividad del lenguaje C, ahora Java tiene incluso un valor agregado que es la propiedad de la multiplataforma lo que significa que se escribe una vez y se ejecuta donde sea.

ANEXO A
CIRCUITO INTEGRADO MAX232

- Meet or Exceed TIA/EIA-232-F and ITU Recommendation V.28
- Operate With Single 5-V Power Supply
- Operate Up to 120 kbit/s
- Two Drivers and Two Receivers
- ±30-V Input Levels
- Low Supply Current . . . 8 mA Typical
- Designed to be Interchangeable With Maxim MAX232
- ESD Protection Exceeds JESD 22 – 2000-V Human-Body Model (A114-A)
- Applications
  - TIA/EIA-232-F
  - Battery-Powered Systems
  - Terminals
  - Modems
  - Computers

MAX232 . . . D, DW, N, OR NS PACKAGE
MAX232I . . . D, DW, OR N PACKAGE
(TOP VIEW)



description/ordering information

The MAX232 is a dual driver/receiver that includes a capacitive voltage generator to supply EIA-232 voltage levels from a single 5-V supply. Each receiver converts EIA-232 inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V and a typical hysteresis of 0.5 V, and can accept ±30-V inputs. Each driver converts TTL/CMOS input levels into EIA-232 levels. The driver, receiver, and voltage-generator functions are available as cells in the Texas Instruments LinASIC™ library.

ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
0°C to 70°C	PDIP (N)	Tube	MAX232N	MAX232N
	SOIC (D)	Tube	MAX232D	MAX232
		Tape and reel	MAX232DR	
	SOIC (DW)	Tube	MAX232DW	MAX232
		Tape and reel	MAX232DWR	
	SOP (NS)	Tape and reel	MAX232NSR	MAX232
–40°C to 85°C	PDIP (N)	Tube	MAX232IN	MAX232IN
	SOIC (D)	Tube	MAX232ID	MAX232I
		Tape and reel	MAX232IDR	
	SOIC (DW)	Tube	MAX232IDW	MAX232I
		Tape and reel	MAX232IDWR	

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

Function Tables

EACH DRIVER

INPUT TIN	OUTPUT TOUT
L	H
H	L

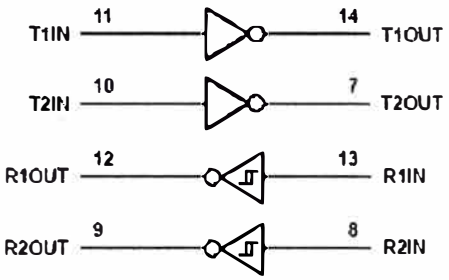
H = high level, L = low level

EACH RECEIVER

INPUT RIN	OUTPUT ROUT
L	H
H	L

H = high level, L = low level

logic diagram (positive logic)



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Input supply voltage range, V_{CC} (see Note 1) ..... –0.3 V to 6 V
Positive output supply voltage range, V_{S+} ..... V_{CC} – 0.3 V to 15 V
Negative output supply voltage range, V_{S-} ..... –0.3 V to –15 V
Input voltage range, V_I: Driver ..... –0.3 V to V_{CC} + 0.3 V
Receiver ..... ±30 V
Output voltage range, V_O: T1OUT, T2OUT ..... V_{S-} – 0.3 V to V_{S+} + 0.3 V
R1OUT, R2OUT ..... –0.3 V to V_{CC} + 0.3 V
Short-circuit duration: T1OUT, T2OUT ..... Unlimited
Package thermal impedance, θ_{JA} (see Note 2): D package ..... 73°C/W
DW package ..... 57°C/W
N package ..... 67°C/W
NS package ..... 64°C/W
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds ..... 260°C
Storage temperature range, T_{stg} ..... –65°C to 150°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: All voltage values are with respect to network ground terminal.
2. The package thermal impedance is calculated in accordance with JESD 51-7.

recommended operating conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	V
V_{IH}	High-level input voltage (T1IN, T2IN)		2			V
V_{IL}	Low-level input voltage (T1IN, T2IN)				0.8	V
R1IN, R2IN	Receiver input voltage				±30	V
T_A	Operating free-air temperature	MAX232	0		70	°C
		MAX232I	–40		85	

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted) (see Note 3 and Figure 4)

PARAMETER		TEST CONDITIONS	MIN	TYP‡	MAX	UNIT
I_{CC}	Supply current	V_{CC} = 5.5 V, All outputs open, T_A = 25°C		8	10	mA

‡ All typical values are at V_{CC} = 5 V and T_A = 25°C.
NOTE 3: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

DRIVER SECTION

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature range (see Note 3)

PARAMETER			TEST CONDITIONS		MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage	T1OUT, T2OUT	R_L = 3 kΩ to GND		5	7		V
V_{OL}	Low-level output voltage‡	T1OUT, T2OUT	R_L = 3 kΩ to GND			–7	–5	V
r_o	Output resistance	T1OUT, T2OUT	V_{S+} = V_{S-} = 0,	V_O = ±2 V	300			Ω
I_{OS}§	Short-circuit output current	T1OUT, T2OUT	V_{CC} = 5.5 V,	V_O = 0		±10		mA
I_{IS}	Short-circuit input current	T1IN, T2IN	V_I = 0				200	µA

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ The algebraic convention, in which the least positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels only.
§ Not more than one output should be shorted at a time.
NOTE 3: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

switching characteristics, V_{CC} = 5 V, T_A = 25°C (see Note 3)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
SR	Driver slew rate	R_L = 3 kΩ to 7 kΩ See Figure 2			30	V/µs
SR(t)	Driver transition region slew rate	See Figure 3		3		V/µs
	Data rate	One TOUT switching		120		kbit/s

NOTE 3: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

RECEIVER SECTION

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature range (see Note 3)

PARAMETER			TEST CONDITIONS		MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage	R1OUT, R2OUT	I_{OH} = –1 mA		3.5			V
V_{OL}	Low-level output voltage‡	R1OUT, R2OUT	I_{OL} = 3.2 mA				0.4	V
V_{IT+}	Receiver positive-going input threshold voltage	R1IN, R2IN	V_{CC} = 5 V,	T_A = 25°C		1.7	2.4	V
V_{IT-}	Receiver negative-going input threshold voltage	R1IN, R2IN	V_{CC} = 5 V,	T_A = 25°C	0.8	1.2		V
V_{hys}	Input hysteresis voltage	R1IN, R2IN	V_{CC} = 5 V		0.2	0.5	1	V
r_i	Receiver input resistance	R1IN, R2IN	V_{CC} = 5,	T_A = 25°C	3	5	7	kΩ

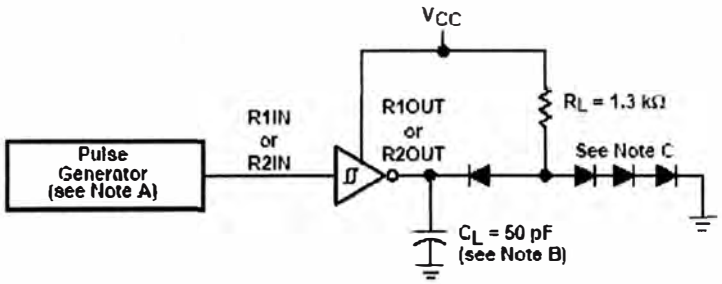
† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ The algebraic convention, in which the least positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels only.
NOTE 3: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

switching characteristics, V_{CC} = 5 V, T_A = 25°C (see Note 3 and Figure 1)

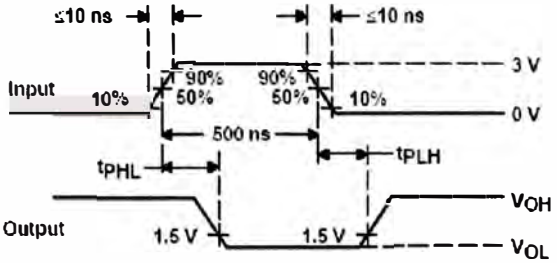
PARAMETER		TYP	UNIT
$t_{PLH(R)}$	Receiver propagation delay time, low- to high-level output	500	ns
$t_{PHL(R)}$	Receiver propagation delay time, high- to low-level output	500	ns

NOTE 3: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

PARAMETER MEASUREMENT INFORMATION



TEST CIRCUIT

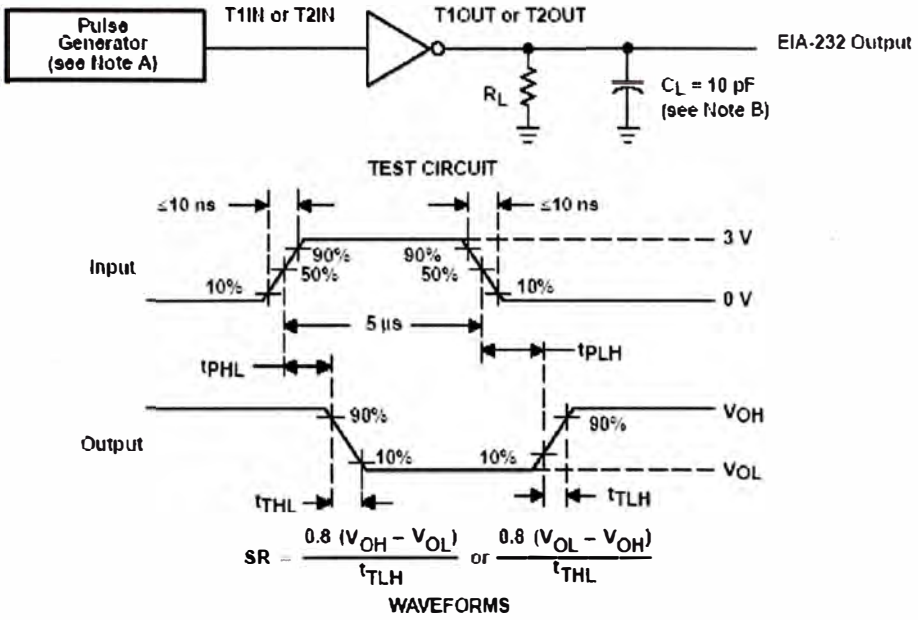


WAVEFORMS

NOTES: A. The pulse generator has the following characteristics: Z_O = 50 Ω, duty cycle ≤ 50%.
B. C_L includes probe and jig capacitance.
C. All diodes are 1N3064 or equivalent.

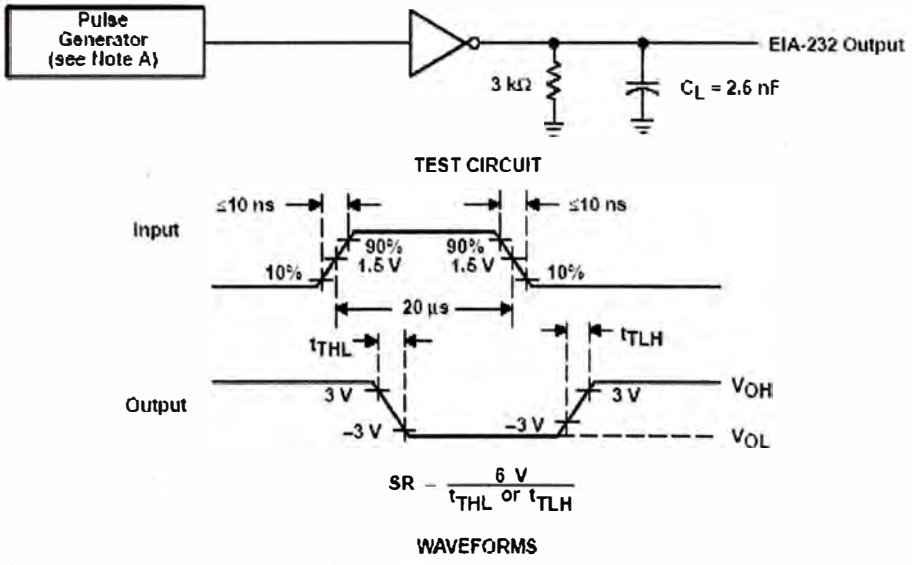
Figure 1. Receiver Test Circuit and Waveforms for t_{PHL} and t_{PLH} Measurements

PARAMETER MEASUREMENT INFORMATION



NOTES: A. The pulse generator has the following characteristics: Z_O = 50 Ω, duty cycle ≤ 50%.
B. C_L includes probe and jig capacitance.

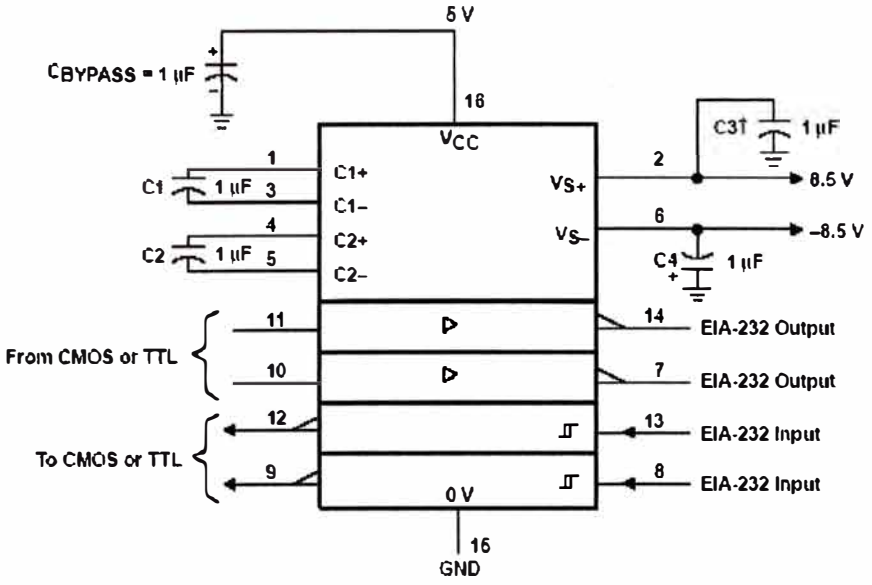
Figure 2. Driver Test Circuit and Waveforms for t_{PHL} and t_{PLH} Measurements (5-µs Input)



NOTE A: The pulse generator has the following characteristics: Z_O = 50 Ω, duty cycle ≤ 50%.

Figure 3. Test Circuit and Waveforms for t_{THL} and t_{TLH} Measurements (20-µs Input)

APPLICATION INFORMATION



† C3 can be connected to V_{CC} or GND.

Figure 4. Typical Operating Circuit

ANEXO B
MICROCONTROLADOR PIC16F84

MICROCHIP

PIC16F84A

18-pin *Enhanced* FLASH/EEPROM 8-Bit Microcontroller

High Performance RISC CPU Features:

- Only 35 single word instructions to learn
- All instructions single-cycle except for program branches which are two-cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- 1024 words of program memory
- 68 bytes of Data RAM
- 64 bytes of Data EEPROM
- 14-bit wide instruction words
- 8-bit wide data bytes
- 15 Special Function Hardware registers
- Eight-level deep hardware stack
- Direct, indirect and relative addressing modes
- Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt-on-change
  - Data EEPROM write complete

Peripheral Features:

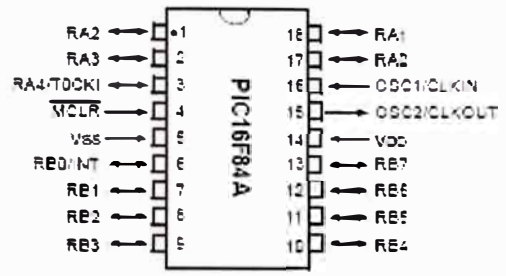
- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
  - 25 mA sink max. per pin
  - 25 mA source max. per pin
- TMR0: 8-bit timer/counter with 8-bit programmable prescaler

Special Microcontroller Features:

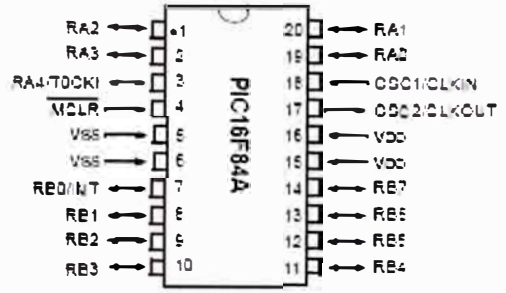
- 10,000 erase/write cycles *Enhanced* FLASH Program memory typical
- 10,000,000 typical erase/write cycles EEPROM Data memory typical
- EEPROM Data Retention > 40 years
- In-Circuit Serial Programming™ (ICSP™) - via two pins
- Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own On-Chip RC Oscillator for reliable operation
- Code protection
- Power saving SLEEP mode
- Selectable oscillator options

Pin Diagrams

PDIP, SOIC



SSOP



CMOS Enhanced FLASH/EEPROM Technology:

- Low power, high speed technology
- Fully static design
- Wide operating voltage range:
  - Commercial: 2.0V to 5.5V
  - Industrial: 2.0V to 5.5V
- Low power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 15 µA typical @ 2V, 32 kHz
  - < 0.5 µA typical standby current @ 2V

1.0 DEVICE OVERVIEW

This document contains device specific information for the operation of the PIC16F84A device. Additional information may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023), which may be downloaded from the Microchip website. The Reference Manual should be considered a complementary document to this data sheet, and is highly recommended reading for a better understanding of the device architecture and operation of the peripheral modules.

The PIC16F84A belongs to the mid-range family of the PICmicro® microcontroller devices. A block diagram of the device is shown in Figure 1-1.

The program memory contains 1K words, which translates to 1024 instructions, since each 14-bit program memory word is the same width as each device instruction. The data memory (RAM) contains 68 bytes. Data EEPROM is 64 bytes.

There are also 13 I/O pins that are user-configured on a pin-to-pin basis. Some pins are multiplexed with other device functions. These functions include:

- External interrupt
- Change on PORTB interrupt
- Timer0 clock input

Table 1-1 details the pinout of the device with descriptions and details for each pin.

FIGURE 1-1: PIC16F84A BLOCK DIAGRAM

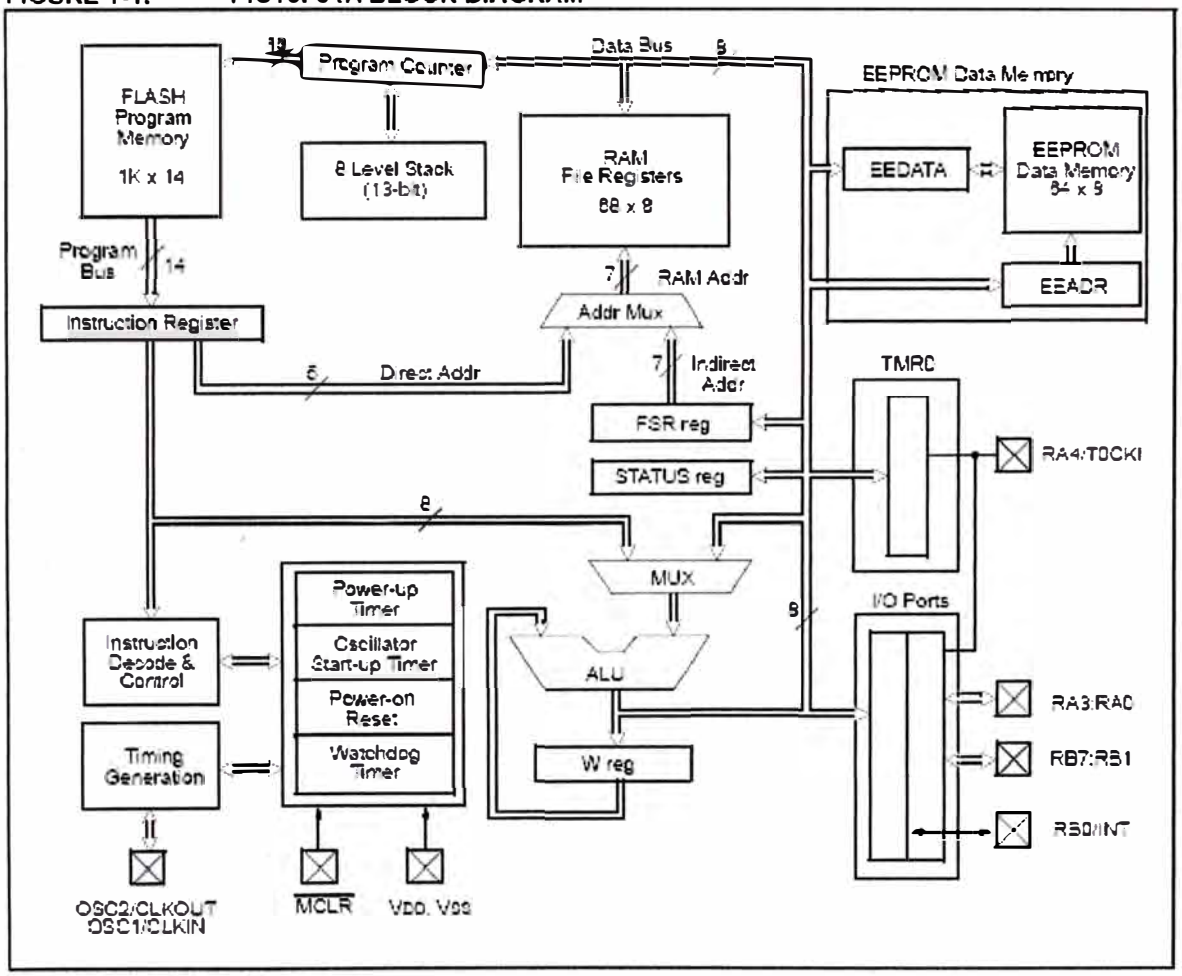


TABLE 1-1: PIC16F84A PINOUT DESCRIPTION

Pin Name	PDIP No.	SOIC No.	SSOP No.	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	16	16	18	I	ST/CMOS(3)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	15	15	19	O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. In RC mode, OSC2 pin outputs CLKOUT, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
$\overline{MCLR}$	4	4	4	I/P	ST	Master Clear (Reset) input/programming voltage input. This pin is an active low RESET to the device.
						PORTA is a bi-directional I/O port.
RA0	17	17	19	I/O	TTL	
RA1	18	18	20	I/O	TTL	
RA2	1	1		I/O	TTL	
RA3	2	2	2	I	TTL	
RA4/T0CKI	3	3	3	I	ST	Can also be selected to be the clock input to the TMR0 timer/counter. Output is open drain type.
						PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	6	6	7	I/O	TTL/ST(1)	RB0/INT can also be selected as an external interrupt pin.
RB1	7	7	8	I/O	TTL	
RB2	8	8	9	I/O	TTL	
RB3	9	9	10	I/O	TTL	
RB4	10	10	11	I/O	TTL	Interrupt-on-change pin.
RB5	11	11	12	I/O	TTL	Interrupt-on-change pin.
RB6	12	12	13	I/O	TTL/ST(2)	Interrupt-on-change pin. Serial programming clock.
RB7	13	13	14	I/O	TTL/ST(2)	Interrupt-on-change pin. Serial programming data.
VSS	5	5	5,6	P	—	Ground reference for logic and I/O pins.
VDD	14	14	15,16	P	—	Positive supply for logic and I/O pins.

Legend: I= input O = Output I/O = Input/Output P = Power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

2.0 MEMORY ORGANIZATION

There are two memory blocks in the PIC16F84A. These are the program memory and the data memory. Each block has its own bus, so that access to each block can occur during the same oscillator cycle.

The data memory can further be broken down into the general purpose RAM and the Special Function Registers (SFRs). The operation of the SFRs that control the "core" are described here. The SFRs used to control the peripheral modules are described in the section discussing each individual peripheral module.

The data memory area also contains the data EEPROM memory. This memory is not directly mapped into the data memory, but is indirectly mapped. That is, an indirect address pointer specifies the address of the data EEPROM memory to read/write. The 64 bytes of data EEPROM memory have the address range 0h-3Fh. More details on the EEPROM memory can be found in Section 3.0.

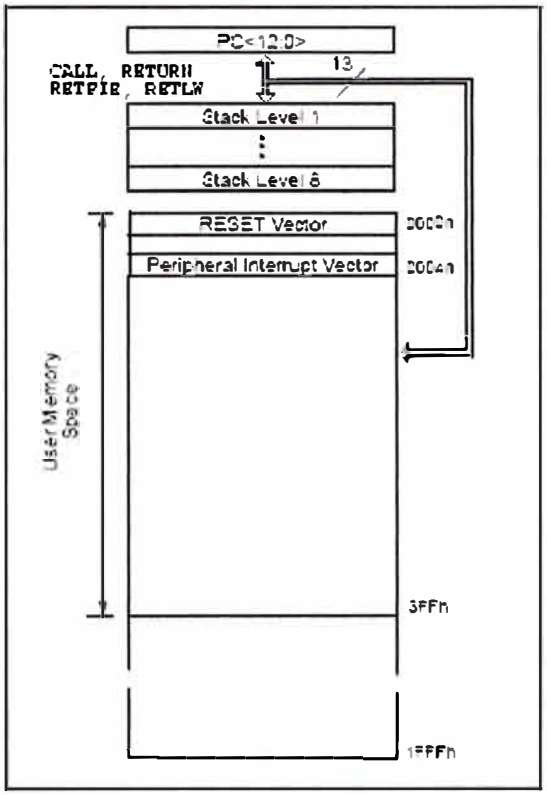
Additional information on device memory may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023).

2.1 Program Memory Organization

The PIC16FXX has a 13-bit program counter capable of addressing an 8K x 14 program memory space. For the PIC16F84A, the first 1K x 14 (0000h-03FFh) are physically implemented (Figure 2-1). Accessing a location above the physically implemented address will cause a wraparound. For example, for locations 20h, 420h, 820h, C20h, 1020h, 1420h, 1820h, and 1C20h, the instruction will be the same.

The RESET vector is at 0000h and the interrupt vector is at 0004h.

FIGURE 2-1: PROGRAM MEMORY MAP AND STACK - PIC16F84A



2.2 Data Memory Organization

The data memory is partitioned into two areas. The first is the Special Function Registers (SFR) area, while the second is the General Purpose Registers (GPR) area. The SFRs control the operation of the device.

Portions of data memory are banked. This is for both the SFR area and the GPR area. The GPR area is banked to allow greater than 116 bytes of general purpose RAM. The banked areas of the SFR are for the registers that control the peripheral functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register. Figure 2-2 shows the data memory map organization.

Instructions MOVWF and MOVF can move values from the W register to any location in the register file ("F"), and vice-versa.

The entire data memory can be accessed either directly using the absolute address of each register file or indirectly through the File Select Register (FSR) (Section 2.5). Indirect addressing uses the present value of the RP0 bit for access into the banked areas of data memory.

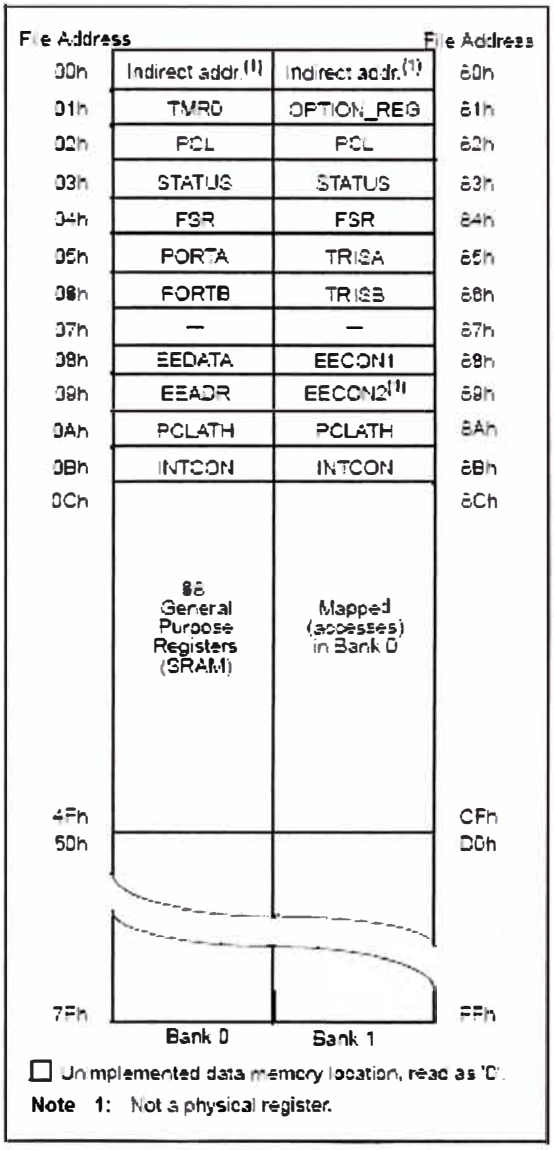
Data memory is partitioned into two banks which contain the general purpose registers and the special function registers. Bank 0 is selected by clearing the RP0 bit (STATUS<5>). Setting the RP0 bit selects Bank 1. Each Bank extends up to 7Fh (128 bytes). The first twelve locations of each Bank are reserved for the Special Function Registers. The remainder are General Purpose Registers, implemented as static RAM.

2.2.1 GENERAL PURPOSE REGISTER FILE

Each General Purpose Register (GPR) is 8-bits wide and is accessed either directly or indirectly through the FSR (Section 2.5).

The GPR addresses in Bank 1 are mapped to addresses in Bank 0. As an example, addressing location 0Ch or 8Ch will access the same GPR.

FIGURE 2-2: REGISTER FILE MAP - PIC16F84A



☐ Unimplemented data memory location, read as '0'.

Note 1: Not a physical register.

2.3 Special Function Registers

The Special Function Registers (Figure 2-2 and Table 2-1) are used by the CPU and Peripheral functions to control the device operation. These registers are static RAM.

The special function registers can be classified into two sets, core and peripheral. Those associated with the core functions are described in this section. Those related to the operation of the peripheral features are described in the section for that specific feature.

TABLE 2-1: SPECIAL FUNCTION REGISTER FILE SUMMARY

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on RESET	Details on page
Bank 0											
00h	INDF	Uses contents of FSR to address Data Memory (not a physical register)								---- ----	11
01h	TMR0	8-bit Real-Time Clock/Counter								xxxx xxxx	20
02h	PCL	Low Order 8 bits of the Program Counter (PC)								0000 0000	11
03h	STATUS(2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
04h	FSR	Indirect Data Memory Address Pointer 0								xxxx xxxx	11
05h	PORTA(4)	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	18
06h	PORTB(5)	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	19
07h	—	Unimplemented location, read as '0'								—	—
08h	EEDATA	EEPROM Data Register								xxxx xxxx	13,14
09h	EEADR	EEPROM Address Register								xxxx xxxx	13,14
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10
Bank 1											
80h	INDF	Uses Contents of FSR to address Data Memory (not a physical register)								---- ----	11
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	9
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	11
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	11
85h	TRISA	—	—	—	PORTA Data Direction Register					---1 1111	18
86h	TRISB	PORTB Data Direction Register								1111 1111	19
87h	—	Unimplemented location, read as '0'								—	—
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	13
89h	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	14
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0', q = value depends on condition

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> are never transferred to PCLATH.

2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ Reset.

3: Other (non power-up) RESETS include: external RESET through $\overline{MCLR}$ and the Watchdog Timer Reset.

4: On any device RESET, these pins are configured as inputs.

5: This is the value that will be in the port output latch.

ANEXO C

PUERTOS RS232 - LPT

1. Idea of serial and parallel port control cables

The idea of port control cables is very simple. Some cameras have shutter control output to which you can plug a pilot and control the shutter remotely. The idea is to connect a camera with a computer and control shutter remotely. To do this we need a special cable.

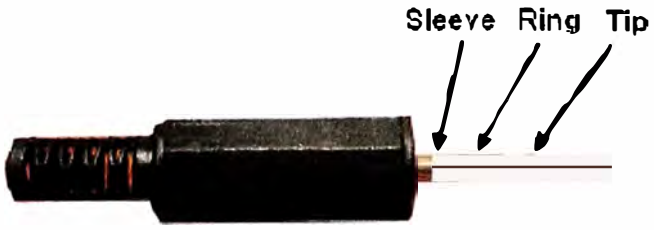
2.1 Needed parts (Serial Port Cable – RS232)

To make a simple serial port control cable you need:

- 1 x Resistor 2,2k Ohm
- 1 x Transistor (npn) (e.g. 2N2222)
- 1 x Diode (e.g. 1N4001)
- 1 x Female RS-232 plug (DB-9) 9 pins
- 1 x Stereo small jack 2,5mm
- Up to 10 meters of 2 wire cable

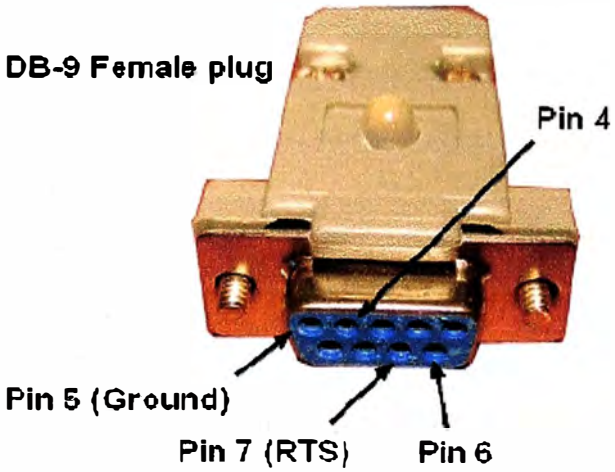
2.2 Circuit scheme and description

Camera output is a 2,5mm stereo jack (one size smaller than that used for walkmans headphones).



2,5mm stereo mini jack

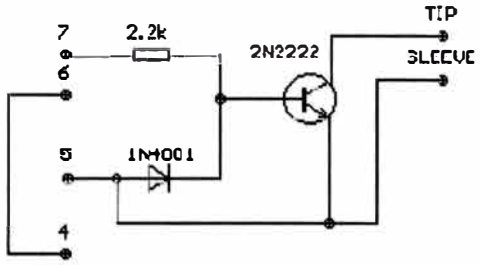
As a computer interface the RS-232 is used. Female plug (DB-9) for RS-232 port is shown below.



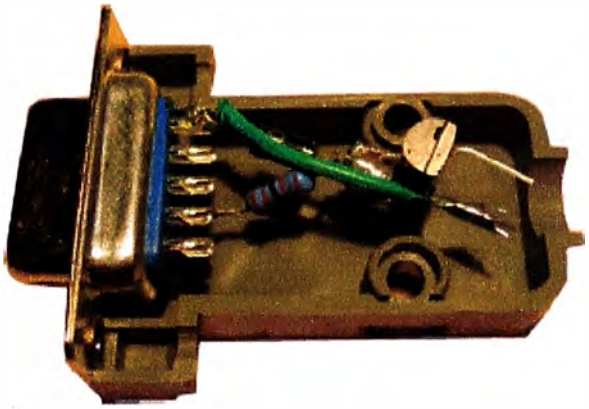
RS-232 Pins

Pin	Name	Description
1	CD	Carrier Detect
2	RX	Recieve Data
3	TX	Transmit Data
4	DTR	Data Terminal Ready
5	GND	Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send

Circuit is very simple. Scheme is shown below.



Numbers on the scheme are numbers of RS-232 pins. The idea of this circuit is to make a connection between TIP and SLEEVE when pin7 (RTS) is high (positive voltage), and to disconnect them when RTS is low (negative voltage). Voltage on the transistor base should be approximately 0.7V. Diode in this circuit only protects the transistor and does not play any other role. Not bad idea is to place the whole circuit in the DB-9 plug.



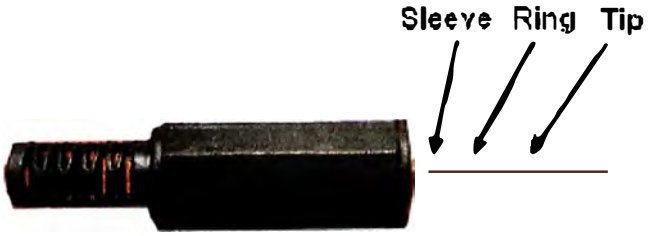
3.1 Needed parts (Parallel Port - LPT)

To make a simple parallel port control cable you need:

- 1 x Resistor 47k Ohm
- 1 x Transistor (npn) (e.g. 2N2222)
- 1 x Diode (e.g. 1N4001)
- 1 x male LPT plug (DB-25) 25 pin
- 1 x Stereo small jack 2,5mm
- Up to 10 meters of 2 wire cable

3.2 Circuit scheme and description

Camera output is a 2,5mm stereo jack (one size smaller than that used for walkman headphones).



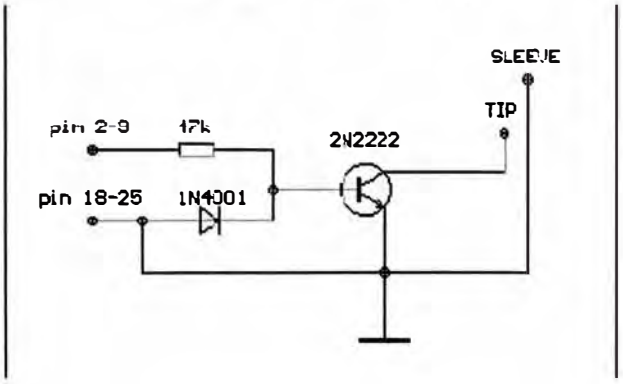
2,5mm stereo mini jack

As a computer interface the LPT (parallel) is used. Male plug (DB-25) for LPT port is shown below.

LPT Pins

Pin	Name	Description
1	STROBE	
2	D0	Data LSB
3	D1	Data
4	D2	Data
5	D3	Data
6	D4	Data
7	D5	Data
8	D6	Data
9	D7	Data MSB
10	ACK	
11	BUSY	
12	PAPER END	
13	SELECT STATUS	
14	AUTO LINE FEED	
15	ERROR	
16	INITIATE PRINTER	
17	SELECT PRINTER	
18-25	GND	Ground

Circuit is very simple and similar to this shown for RS-232 port. This time resistor can be attached to any of the pins from 2 to 9 (Delphinus will support it but some other programs only supports pin 2 so for better compatibility with other programs it's recommended to use pin 2). Ground can be attached to any pin from 18 to 25. Diode here is only to protect the transistor and can be omitted in most cases but we recommend to use it in this circuit.



4. Using Cables

Before connecting your cable to the camera check the output voltage and current in your circuit. Simply connect plug into RS-232/ LPT port. Run program Delphinus. Set long Time value (e.g. 360 sec.), which give you time to check the voltage. Select appropriate port address (typically COM1 or LPT1). Press "Camera" icon and check the voltage between **TIP** and **SLEEVE**. A few mV (typically up to 300mV) are acceptable, but if the voltage is higher than something is wrong **and you absolutely should NOT plug your cable into the camera.**

Remember: Whenever you want to connect or disconnect plug make sure that your computer is not running. Connecting or disconnecting cables on running computer can affect in port damage!

BIBLIOGRAFÌA

[1] Hortsmann , "Java 2 fundamentos " , edit.:p.hall, 2003

[2] Herbert Schildt. "Manual de Referencia. Java 2". Cuarta edición.

[3] Aquilino Rodríguez Penin "Sistemas Scada". 2ª Edición Marcombo 2007

[4] Jaworski, J. JAVA – "Guía de Desarrollo", Ed. Prentice-Hall, 1997.

[5] Web: http://java.sun.com

[6] Web: http://www.javahispano.com

[7] Web: http://www.lawebdelprogramador.com/foros/new.php?id=44&texto=Java

[8] Tutorial Eclipse: http://eclipsetutorial.forge.os4os.org/in1.htm

[9] IDE: http://www.eclipse.org/downloads/

[10] Programa Java: http://java.sun.com/