ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

CONSTRUCCIÓN DE UNA MAQUETA CON SISTEMA DIGITAL PARA EL CONTROL DE LA ILUMINACIÓN Y MONITOREO DE ACCESO POR PUERTAS Y VENTANAS MEDIANTE EL PUERTO SERIAL DE UN COMPUTADOR UTILIZANDO EL MICROCONTROLADOR ATMEGA.

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO EN ELECTRÓNICA Y TELECOMUNICACIONES

AUTOR:

CIFUENTES SARCHI JORGE ALFREDO

jor-g12_85@hotmail.com

DIRECTOR:

ING. PABLO WIGBERTO LÓPEZ MERINO

pablo.lopez@epn.edu.ec

Quito, Febrero, 2014

DECLARACIÓN

Yo CIFUENTES SARCHI JORGE ALFREDO, declaro bajo juramento que el trabajo aquí descrito es de mi autoría, que no ha sido previamente presentada para ningún grado o calificación profesional y que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

CIFUENTES SARCHI JORGE ALFREDO

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por CIFUENTES SARCHI JORGE ALFREDO, bajo mi supervisión.

Ing. PABLO WIGBERTO LÓPEZ MERINO

DIRECTOR DE PROYECTO

AGRADECIMIENTOS

Quiero agradecer a todas las personas que hicieron posible la culminación de este proyecto quienes fueron el pilar fundamental de mis éxitos.

A mi madre y mi padre por brindarme todo su apoyo y comprensión en todo momento de mi vida.

A mis hermanos quienes siempre me alentaron a seguir adelante y a toda mi familia que supieron apoyarme durante mi vida estudiantil.

A mi primo Pedro Martínez quien fue un gran apoyo en la realización de este proyecto y a mi novia Nathalia Puga quien siempre me apoyó para la culminación de mi meta.

Jorge Cifuentes

DEDICATORIA

Este trabajo va dedicado:

A mis padres por darme la fuerza y el coraje para seguir adelante y alcanzar mis objetivos trazados, siempre los tendré en mi corazón.

A todas las personas que han pasado por mi vida dejando una huella imborrable y me han ayudado a un constante mejoramiento personal.

Jorge Cifuentes

ÍNDICE

RESUMEN

Este proyecto se realizó durante el período comprendido entre el año 2010 y 2013, el propósito de la presente investigación consistió en controlar el encendido y apagado de las luces de una vivienda y el monitorear el acceso por puertas y ventanas a la misma. Para ello se utilizó el microcontrolador Atmega conectado a un PC por medio del puerto serial.

El presente proyecto se justifica debido a la importancia de la automatización en las actividades de la vida diaria como lo es el encendido y apagado de las luces y el monitoreo de acceso a la vivienda para la detección de intrusos permitiendo de esta manera reducir el esfuerzo que realizan las personas para realizar estas actividades manualmente y mejorar la seguridad del inmueble.

PRESENTACIÓN

La tecnología es una herramienta indispensable para el desarrollo y simplificación de las actividades en la vida diaria de las personas. Múltiples aparatos eléctricos y electrónicos que actualmente utilizamos fueron creados para facilitarnos las tareas ya sea en nuestro trabajo, estudio y especialmente en el hogar.

La mayoría de las viviendas modernas están provistas de un gran número de costosos bienes los cuales son codiciados por delincuentes quienes ingresan a las mismas como si fuesen los propios dueños sin que tengamos una manera de cómo poder detectar su intrusión a la vivienda. Estos hogares son amplias construcciones las cuales tienen muchas luminarias en todas las áreas como sala, cocina, baños, dormitorios etc. y para encender o apagar estas luces es necesario estar físicamente en el lugar donde se encuentra ubicado el interruptor lo cual resulta una tarea tediosa y a veces molesta.

La necesidad de automatizar el manejo de la iluminación y el monitoreo del acceso por puertas y ventanas dentro de una casa es de vital importancia para la comodidad y seguridad de la familia, el sistema permite que desde un mismo lugar tengamos la posibilidad de prender o apagar las luminarias de la vivienda a la vez que monitorea si una puerta o ventana se encuentra abierta o cerrada accionando una alarma si se vulnera la seguridad en el horario que el usuario lo disponga.

1 CAPÍTULO I: FUNDAMENTOS TEÓRICOS

1.1 MICROCONTROLADOR

1.1.1 INTRODUCCIÓN[1]

Controlador es un dispositivo que se emplea para el gobierno de uno o varios procesos. Aunque el concepto ha permanecido invariable a través de los tiempos, su implementación física ha variado frecuentemente. Hace tres décadas, los controladores se construían exclusivamente con componentes de lógica discreta; posteriormente se emplearon los microprocesadores, que se rodean con chips de memoria RAM, ROM y E/S sobre una tarjeta de circuito impreso.

En la década de los 70' los fabricantes de circuitos integrados iniciaron la difusión de un nuevo circuito de control, medición he instrumentación al que llamaron microcomputador en un solo chip, o de manera más exacta **microcontrolador**.

Un microcontrolador es un circuito integrado programable que contiene todos los componentes de un computador, aunque de limitadas prestaciones. Se emplea para controlar el funcionamiento de una tarea determinada y, debido a su reducido tamaño, suele ir incorporado en el propio dispositivo al que gobierna. Esta última característica es la que le confiere la denominación de "controlador incrustado".

El microcontrolador es un dispositivo electrónico dedicado a una sola tarea principal ya que en su memoria sólo decide un programa destinado a controlar una aplicación determinada, sus líneas de entrada/salida soportan el conexionado de sensores y actuadores del dispositivo a gobernar y todos los recursos complementarios disponibles tienen como única finalidad atender sus requerimientos. Una vez programado y configurado el microcontrolador solamente sirve para ejecutar la tarea asignada.

[1] www.rodrigog.com/pic/curso/Microcontroladores%20PIC.doc

1.1.2 HISTORIA

El microcontrolador es uno de los logros más sobresalientes del siglo XX, Hace un cuarto de siglo tal afirmación habría parecido absurda. Pero cada año, el microcontrolador se acerca más al centro de nuestras vidas, forjándose un sitio en el núcleo de una máquina tras otra, su presencia ha comenzado a cambiar la forma en que percibimos el mundo e incluso a nosotros mismos. Cada vez se hace más difícil pasar por alto al microcontrolador como otro simple producto en una larga línea de innovaciones tecnológicas.

Ninguna otra invención en la historia se ha diseminado tan aprisa por todo el mundo o ha tocado tan profundamente tantos aspectos de la vida humana. Hoy existen 15.000 millones de microchips de alguna clase en uso. De cara a esa realidad, ¿Quién puede dudar que el microcontrolador no sólo está transformando los productos que usamos, sino también nuestra forma de vivir y, por último, la forma en que percibimos de realidad?

El microcontrolador ha eclipsado hasta la revolución industrial. Evolucionando a mayor velocidad que ningún otro invento en la historia, la capacidad del microcontrolador ha aumentado 10.000 veces en los últimos 25 años.

El mayor atributo del microcontrolador es que puede integrar inteligencia a casi cualquier artefacto. Se le puede programar para adaptarse a su entorno, responder a condiciones cambiantes y volverse más eficiente y que responda a las necesidades únicas de sus usuarios.

El desarrollo del microcontrolador es un ejemplo palpable del “mejoramiento continuo”.

Un microcontrolador contiene toda la estructura de un sencillo pero completo computador contenidos en el corazón de un circuito integrado. Los resultados prácticos que pueden lograrse a partir de estos elementos son sorprendentes.

De bajo costo, bajo consumo de energía, fácil implementación, es una pieza clave para el desarrollo de la robótica.

La única limitación que tienen las aplicaciones de los microcontroladores actuales está en la imaginación de los desarrolladores, los campos más destacados de los microcontroladores son:

- Automatización industrial
- Medidas y control de procesos
- Enseñanza e investigación
- Periféricos para computadores
- Electrodomésticos
- Aparatos portátiles o de bolsillo
- Juguetes
- Instrumentación
- Electromedicina
- Robótica
- Domótica
- Sistemas de seguridad y alarmas

1.1.3 ARQUITECTURA BÁSICA[2]

Aunque inicialmente todos los microcontroladores adoptaron la arquitectura clásica de Von Neumann, en el momento presente se impone la arquitectura Harvard. La arquitectura de Von Neumann se caracteriza por disponer de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de buses único (direcciones, datos y control).

Como se muestra en la Figura 1.1, la arquitectura Harvard dispone de dos memorias independientes una que contiene sólo instrucciones y otra sólo datos. Ambas disponen de sus respectivos sistemas de buses de acceso y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en ambas memorias.

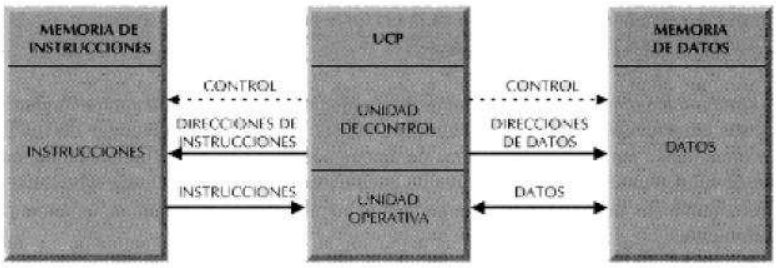


Figura 1.1. Arquitectura Harvard

La arquitectura Harvard dispone de dos memorias independientes para datos y para instrucciones, permitiendo accesos simultáneos.

Los microcontroladores PIC responden a la arquitectura Harvard.

[2] http://www.monografias.com/trabajos12/microco/microco.shtml

1.1.4 EL PROCESADOR O CPU

Es el elemento más importante del microcontrolador y determina sus principales características, tanto a nivel hardware como software.

Se encarga de direccionar la memoria de instrucciones, recibir el código OP de la instrucción en curso, su decodificación y la ejecución de la operación que implica la instrucción, así como la búsqueda de los operandos y el almacenamiento del resultado.

Existen tres orientaciones en cuanto a la arquitectura y funcionalidad de los procesadores actuales.

CISC: Un gran número de procesadores usados en los microcontroladores están basados en la filosofía CISC (Computadores de Juego de Instrucciones Complejo). Disponen de más de 80 instrucciones máquina en su repertorio, algunas de las cuales son muy sofisticadas y potentes, requiriendo muchos ciclos para su ejecución.

Una ventaja de los procesadores CISC es que ofrecen al programador instrucciones complejas que actúan como macros.

RISC: Tanto la industria de los computadores comerciales como la de los microcontroladores están decantándose hacia la filosofía RISC (Computadores de Juego de Instrucciones Reducido). En estos procesadores el repertorio de instrucciones máquina es muy reducido y las instrucciones son simples y, generalmente, se ejecutan en un ciclo.

La sencillez y rapidez de las instrucciones permiten optimizar el hardware y el software del procesador.

SISC: En los microcontroladores destinados a aplicaciones muy concretas, el juego de instrucciones, además de ser reducido, es "específico", o sea, las instrucciones se adaptan a las necesidades de la aplicación prevista. Esta filosofía se ha bautizado con el nombre de SISC (Computadores de Juego de Instrucciones Específico).

1.1.5 MEMORIA

En los microcontroladores la memoria de instrucciones y datos está integrada en el propio chip. Una parte debe ser no volátil, tipo ROM, y se destina a contener el programa de instrucciones que gobierna la aplicación. Otra parte de memoria será tipo RAM, volátil, y se destina a guardar las variables y los datos.

Hay dos peculiaridades que diferencian a los microcontroladores de los computadores personales:

No existen sistemas de almacenamiento masivo como disco duro o disquetes.

Como el microcontrolador sólo se destina a una tarea en la memoria ROM, sólo hay que almacenar un único programa de trabajo.

La RAM en estos dispositivos es de poca capacidad pues sólo debe contener las variables y los cambios de información que se produzcan en el transcurso del programa. Por otra parte, como sólo existe un programa activo, no se requiere guardar una copia del mismo en la RAM pues se ejecuta directamente desde la ROM.

Los usuarios de computadores personales están habituados a manejar Megabytes de memoria, pero, los diseñadores con microcontroladores trabajan con capacidades de ROM comprendidas entre 512 bytes y 8 k bytes y de RAM comprendidas entre 20 y 512 bytes.

Según el tipo de memoria ROM que dispongan los microcontroladores, la aplicación y utilización de los mismos es diferente. Se describen las cinco versiones de memoria no volátil que se pueden encontrar en los microcontroladores del mercado.

1.1.5.1 Rom con máscara

Es una memoria no volátil de sólo lectura cuyo contenido se graba durante la fabricación del chip. El elevado coste del diseño de la máscara sólo hace

aconsejable el empleo de los microcontroladores con este tipo de memoria cuando se precisan cantidades superiores a varios miles de unidades.

1.1.5.2 Otp (One Time Progammable)

El microcontrolador contiene una memoria no volátil de sólo lectura "programable una sola vez" por el usuario. OTP (One Time Programmable). Es el usuario quien puede escribir el programa en el chip mediante un sencillo grabador controlado por un programa desde un PC.

La versión OTP es recomendable cuando es muy corto el ciclo de diseño del producto, o bien, en la construcción de prototipos y series muy pequeñas.

Tanto en este tipo de memoria como en la EPROM, se suele usar la encriptación mediante fusibles para proteger el código contenido.

1.1.5.3 Eprom

Los microcontroladores que disponen de memoria EPROM (Erasable Programmable Read Only Memory) pueden borrarse y grabarse muchas veces. La grabación se realiza, como en el caso de los OTP, con un grabador gobernado desde un PC. Si, posteriormente, se desea borrar el contenido, disponen de una ventana de cristal en su superficie por la que se somete a la EPROM a rayos ultravioleta durante varios minutos. Las cápsulas son de material cerámico y son más caros que los microcontroladores con memoria OTP que están hechos con material plástico.

1.1.5.4 Eeprom

Se trata de memorias de sólo lectura, programables y borrables eléctricamente EEPROM (Electrical Erasable Programmable Read Only Memory).

Tanto la programación como el borrado, se realizan eléctricamente desde el propio grabador y bajo el control programado de un PC.

Es muy cómoda y rápida la operación de grabado y la de borrado.

No disponen de ventana de cristal en la superficie.

Los microcontroladores dotados de memoria EEPROM una vez instalados en el circuito, pueden grabarse y borrarse cuantas veces se quiera sin ser retirados de dicho circuito. Para ello se usan "grabadores en circuito" que confieren una gran flexibilidad y rapidez a la hora de realizar modificaciones en el programa de trabajo.

El número de veces que puede grabarse y borrarse una memoria EEPROM es finito, por lo que no es recomendable una reprogramación continua.

Son muy idóneos para la enseñanza y la Ingeniería de diseño debido a su gran facilidad para ser reprogramadas desde el computador sin la ayuda de luz ultravioleta como sucedía con las anteriores tecnologías.

Se va extendiendo en los fabricantes la tendencia de incluir una pequeña zona de memoria EEPROM en los circuitos programables para guardar y modificar cómodamente una serie de parámetros que adecuan el dispositivo a las condiciones del entorno.

Este tipo de memoria es relativamente lenta.

1.1.5.5 Flash

Se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar. Funciona como una ROM y una RAM pero consume menos y es más pequeña.

A diferencia de la ROM, la memoria FLASH es programable en el circuito. Es más rápida y de mayor densidad que la EEPROM.

La alternativa FLASH está recomendada frente a la EEPROM cuando se precisa gran cantidad de memoria de programa no volátil. Es más veloz y tolera más ciclos de escritura/borrado.

Las memorias EEPROM y FLASH son muy útiles al permitir que los microcontroladores que las incorporan puedan ser reprogramados "en circuito", es decir, sin tener que sacar el circuito integrado de la tarjeta. Así, un dispositivo con este tipo de memoria incorporado al control del motor de un automóvil permite que pueda modificarse el programa durante la rutina de mantenimiento periódico, compensando los desgastes y otros factores tales como la compresión, la instalación de nuevas piezas, etc. La reprogramación del microcontrolador puede convertirse en una labor rutinaria dentro de la puesta a punto.

1.1.6 PUERTAS DE ENTRADA Y SALIDA

La principal utilidad de las patitas que posee la cápsula que contiene un microcontrolador es soportar las líneas de E/S que comunican al computador interno con los periféricos exteriores.

Según los controladores de periféricos que posea cada modelo de microcontrolador, las líneas de E/S se destinan a proporcionar el soporte a las señales de entrada, salida y control.

1.1.7 RELOJ PRINCIPAL

Todos los microcontroladores disponen de un circuito oscilador que genera una onda cuadrada de alta frecuencia, que configura los impulsos de reloj usados en la sincronización de todas las operaciones del sistema.

Generalmente, el circuito de reloj está incorporado en el microcontrolador y sólo se necesitan unos pocos componentes exteriores para seleccionar y estabilizar la frecuencia de trabajo. Dichos componentes suelen consistir en un cristal de cuarzo junto a elementos pasivos o bien un resonador cerámico o una red R-C.

Aumentar la frecuencia de reloj supone disminuir el tiempo en que se ejecutan las instrucciones pero lleva aparejado un incremento del consumo de energía.

1.1.8 RECURSOS ESPECIALES

Cada fabricante oferta numerosas versiones de una arquitectura básica de microcontrolador. En algunas amplía las capacidades de las memorias, en otras incorpora nuevos recursos, en otras reduce las prestaciones al mínimo para aplicaciones muy simples, etc. La labor del diseñador es encontrar el modelo mínimo que satisfaga todos los requerimientos de su aplicación. De esta forma, minimizará el coste, el hardware y el software.

Los principales recursos específicos que incorporan los microcontroladores son:

• Temporizadores o "Timers".

• Perro guardián o "Watchdog".

• Protección ante fallo de alimentación o "Brownout".

• Estado de reposo o de bajo consumo.

• Conversor A/D.

• Conversor D/A.

• Comparador analógico.

• Modulador de anchura de impulsos o PWM.

• Puertas de E/S digitales.

• Puertas de comunicación.

1.1.9 TEMPORIZADORES O “TIMERS”

Se emplean para controlar periodos de tiempo (temporizadores) y para llevar la cuenta de acontecimientos que suceden en el exterior (contadores).

Para la medida de tiempos se carga un registro con el valor adecuado y a continuación dicho valor se va incrementando o decrementando al ritmo de los impulsos de reloj o algún múltiplo hasta que se desborde y llegue a 0, momento en el que se produce un aviso.

Cuando se desean contar acontecimientos que se materializan por cambios de nivel o flancos en alguna de las patitas del microcontrolador, el mencionado registro se va incrementando o decrementando al ritmo de dichos impulsos. La velocidad de trabajo del microcontrolador depende de la frecuencia máxima del cristal que se utilice.

1.1.10 PERRO GUARDIÁN O “WATCHDOG”

Cuando el computador personal se bloquea por un fallo del software u otra causa, se pulsa el botón del reset y se reinicializa el sistema. Pero un microcontrolador funciona sin el control de un supervisor y de forma continuada las 24 horas del día. El Perro guardián consiste en un temporizador que, cuando se desborda y pasa por 0, provoca un reset automáticamente en el sistema.

Se debe diseñar el programa de trabajo que controla la tarea de forma que refresque o inicialice al Perro guardián antes de que provoque el reset. Si falla el programa o se bloquea, no se refrescará al Perro guardián y, al completar su temporización, "ladrará y ladrará" hasta provocar el reset.

1.1.11 PROTECCIÓN ANTE FALLO DE ALIMENTACIÓN O “BROWNOUT”

Se trata de un circuito que resetea al microcontrolador cuando el voltaje de alimentación (VDD) es inferior a un voltaje mínimo ("brownout"). Mientras el voltaje

de alimentación sea inferior al de brownout el dispositivo se mantiene reseteado, comenzando a funcionar normalmente cuando sobrepasa dicho valor.

1.1.12 ESTADO DE REPOSO Ó DE BAJO CONSUMO

Son abundantes las situaciones reales de trabajo en que el microcontrolador debe esperar, sin hacer nada, a que se produzca algún acontecimiento externo que le ponga de nuevo en funcionamiento.

Para ahorrar energía, (factor clave en los aparatos portátiles), los microcontroladores disponen de una instrucción especial (SLEEP en los PIC), que les pasa al estado de reposo o de bajo consumo, en el cual los requerimientos de potencia son mínimos. En dicho estado se detiene el reloj principal y se "congelan" sus circuitos asociados, quedando sumido en un profundo "sueño" el microcontrolador. Al activarse una interrupción ocasionada por el acontecimiento esperado, el microcontrolador se despierta y reanuda su trabajo.

1.1.13 CONVERSOR A/D (CAD)

Los microcontroladores que incorporan un Conversor A/D (Analógico/Digital) pueden procesar señales analógicas, tan abundantes en las aplicaciones.

Suelen disponer de un multiplexor que permite aplicar a la entrada del CAD diversas señales analógicas desde las patitas del circuito integrado.

1.1.14 CONVERSOR D/A (CDA)

Transforma los datos digitales obtenidos del procesamiento del computador en su correspondiente señal analógica que saca al exterior por una de las patitas de la cápsula. Existen muchos efectores que trabajan con señales analógicas.

1.1.15 COMPARADOR ANALÓGICO

Algunos modelos de microcontroladores disponen internamente de un Amplificador Operacional que actúa como comparador entre una señal fija de referencia y otra variable que se aplica por una de las patitas de la cápsula. La salida del comparador proporciona un nivel lógico 1 ó 0 según una señal sea mayor o menor que la otra.

También hay modelos de microcontroladores con un módulo de tensión de referencia que proporciona diversas tensiones de referencia que se pueden aplicar en los comparadores.

1.1.16 MODULADOR DE ANCHURA DE IMPULSOS O PWM

Son circuitos que proporcionan en su salida impulsos de anchura variable, que se ofrecen al exterior a través de las patitas del encapsulado.

1.1.17 PUERTOS DE E/S DIGITALES

Todos los microcontroladores destinan algunas de sus patitas a soportar líneas de E/S digitales. Por lo general, estas líneas se agrupan de ocho en ocho formando Puertos.

Las líneas digitales de los Puertos pueden configurarse como Entrada o como Salida cargando un 1 ó un 0 en el bit correspondiente de un registro destinado a su configuración.

1.1.18 PUERTOS DE COMUNICACIÓN

Con objeto de dotar al microcontrolador de la posibilidad de comunicarse con otros dispositivos externos, otros buses de microprocesadores, buses de sistemas, buses de redes y poder adaptarlos con otros elementos bajo otras normas y protocolos.

Algunos modelos disponen de recursos que permiten directamente esta tarea, entre los que destacan:

UART: adaptador de comunicación serie asíncrona.

USART: adaptador de comunicación serie síncrona y asíncrona

Puerta paralela esclava para poder conectarse con los buses de otros microprocesadores.

USB (Universal Serial Bus), es un moderno bus serie para los PC.

Bus I2C, que es un interfaz serie de dos hilos desarrollado por Philips.

Uno de los factores que más importancia tiene a la hora de seleccionar un microcontrolador entre todos los demás es el soporte tanto software como hardware de que dispone. Un buen conjunto de herramientas de desarrollo puede ser decisivo en la elección, ya que pueden suponer una ayuda inestimable en el desarrollo del proyecto.

1.1.19 DIFERENCIA ENTRE UN MICROPROCESADOR Y UN MICROCONTROLADOR

El microprocesador es un circuito integrado que contiene la Unidad Central de Proceso (UCP), también llamada procesador, de un computador. La UCP está formada por la Unidad de Control, que interpreta las instrucciones, y el Camino de Datos, que las ejecuta (Figura 1.2).

Las patitas de un microprocesador sacan al exterior las líneas de sus buses de direcciones, datos y control, para permitir conectarle con la Memoria y los Módulos de E/S y configurar un computador implementado por varios circuitos integrados. Se dice que un microprocesador es un sistema abierto porque su configuración es variable de acuerdo con la aplicación a la que se destine.

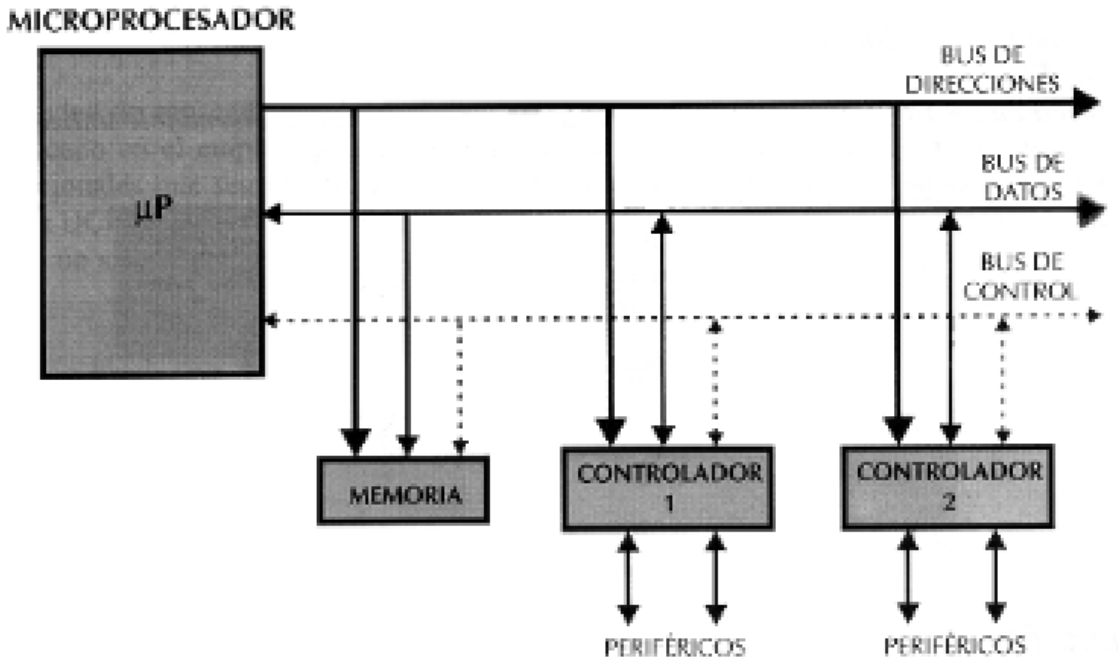


Figura 1.2. Estructura de un sistema abierto basado en un microprocesador.

La disponibilidad de los buses en el exterior permite que se configure a la medida de la aplicación, en la Figura 1.3 se indica que el microcontrolador es un sistema cerrado que permite la interconexión con distintos periféricos.

Si sólo se dispusiese de un modelo de microcontrolador, éste debería tener muy potenciados todos sus recursos para poderse adaptar a las exigencias de las diferentes aplicaciones. Esta potenciación supondría en muchos casos un despilfarro.

En la práctica cada fabricante de microcontroladores oferta un elevado número de modelos diferentes, desde los más sencillos hasta los más poderosos. Es posible seleccionar la capacidad de las memorias, el número de líneas de E/S, la cantidad y potencia de los elementos auxiliares, la velocidad de funcionamiento, etc. Por todo ello, un aspecto muy destacado del diseño es la selección del microcontrolador a utilizar.

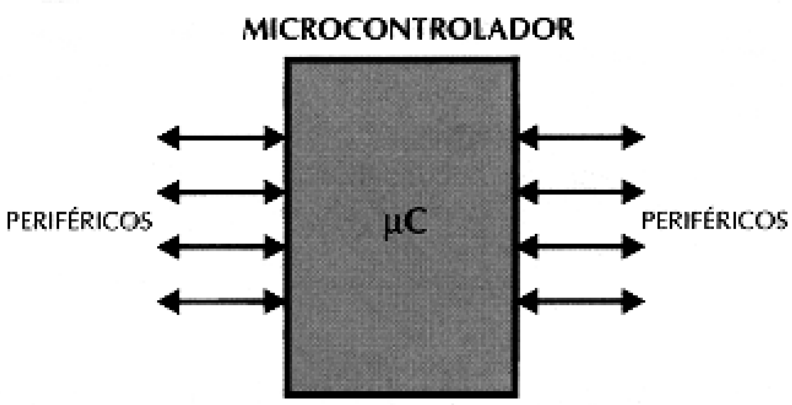


Figura 1.3. El microcontrolador es un sistema cerrado.

Todas las partes del computador están contenidas en su interior y sólo salen al exterior las líneas que gobiernan los periféricos.

1.2 DISPOSITIVOS ELECTRÓNICOS

1.2.1 REGULADORES DE VOLTAJE[3]

Dentro de los reguladores de voltaje con salida fija, se encuentran los pertenecientes a la familia LM78xx, donde “xx” es el voltaje de la salida. Estos son 5, 6, 8, 9, 10, 12, 15, 18 y 24V, entregando una corriente máxima de 1A y soporta consumos pico de hasta 2.2 A.

Poseen protección contra sobrecargas térmicas y contra cortocircuitos, que desconectan el regulador en caso de que su temperatura de juntura supere los 125°C.

Como se muestra en la Figura 1.4 los LM78xx son reguladores de salida positiva, mientras que la familia LM79xx es para voltajes equivalentes pero con salida

[3] http://www.ucontrol.com.ar/wiki/index.php?title=LM78xx

negativa. Así, un LM7805 es capaz de entregar 5V positivos, y un LM7912 entregara 9V negativos.

La capsula que los contiene es una TO-220, igual a la de muchos transistores de mediana potencia. Para alcanzar la corriente máxima de 1A es necesario dotarlo de un disipador de calor adecuado, sin él solo obtendremos una fracción de esta corriente antes de que el regulador alcance su temperatura máxima y se desconecte.

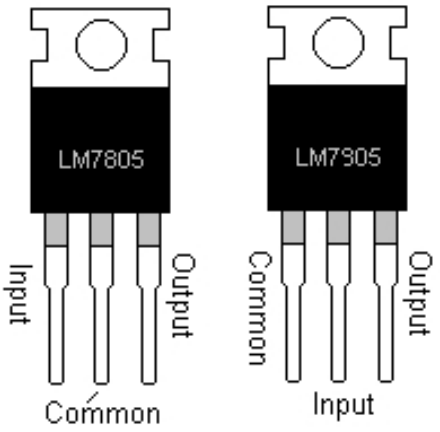


Figura 1.4. Reguladores de Voltaje

La potencia además depende de la tensión de entrada, por ejemplo, si tenemos un LM7812, cuya tensión de salida es de 12V, con una tensión de entrada de digamos 20V, y una carga en su salida de 0,5A, multiplicando la diferencia entre la tensión de entrada y la tensión de salida por la corriente que circulará por la carga nos da los vatios que va a tener que soportar el integrado:

(Vint - Vout) x Iout = (20 - 12) x 0.5 = 4W

La tensión de entrada es un factor muy importante, ya que debe ser superior en unos 3 voltios a la tensión de salida (es el mínimo recomendado por el fabricante), pero

todo el exceso debe ser eliminado en forma de calor. Si en el ejemplo anterior en lugar de entrar con 20 volts solo usamos 15V (los 12V de la salida más el margen de 3V sugerido) la potencia disipada es mucho menor:

(Vint - Vout) x Iout = (15 - 12) x 0.5 = 1.5W

De hecho, con 15V la carga del integrado es de 1,5W, menos de la mitad que con 20V, por lo que el calor generado será mucho menor y en consecuencia el disipador necesario también menor.

1.2.2 SENSORES[4]

Un sensor es un dispositivo capaz de medir magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas. Las variables de instrumentación pueden ser por ejemplo: temperatura, intensidad lumínica, distancia, aceleración, inclinación, desplazamiento, presión, fuerza, torsión, humedad, pH, etc. Una magnitud eléctrica 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 en un fototransistor), etc.

Un sensor se diferencia de un transductor en que el sensor está siempre en contacto con la variable de instrumentación con lo que Puede decirse también que es un dispositivo que aprovecha una de sus propiedades con el fin de adaptar la señal que mide para que la pueda interpretar otro dispositivo. Como por ejemplo el termómetro de mercurio que aprovecha la propiedad que posee el mercurio de dilatarse o contraerse por la acción de la temperatura. Un sensor también puede decirse que es un dispositivo que convierte una forma de energía en otra. Áreas de aplicación de los sensores: Industria automotriz, Industria aeroespacial, Medicina, Industria de manufactura, Robótica, etc.

[4] http://es.wikipedia.org/wiki/Sensor

Los sensores pueden estar conectados a un computador para obtener ventajas como son el acceso a una base de datos, la toma de valores desde el sensor, etc.

1.2.2.1 Características de un sensor

Entre las características técnicas de un sensor destacan las siguientes:

- Rango de medida: dominio en la magnitud medida en el que puede aplicarse el sensor.
- Precisión: es el error de medida máximo esperado.
- *Offset* o desviación de cero: valor de la variable de salida cuando la variable de entrada es nula. Si el rango de medida no llega a valores nulos de la variable de entrada, habitualmente se establece otro punto de referencia para definir el *offset*.
- Linealidad o correlación lineal.
- Sensibilidad de un sensor: relación entre la variación de la magnitud de salida y la variación de la magnitud de entrada.
- Resolución: mínima variación de la magnitud de entrada que puede apreciarse a la salida.
- Rapidez de respuesta: puede ser un tiempo fijo o depender de cuánto varíe la magnitud a medir. Depende de la capacidad del sistema para seguir las variaciones de la magnitud de entrada.
- Derivas: son otras magnitudes, aparte de la medida como magnitud de entrada, que influyen en la variable de salida. Por ejemplo, pueden ser condiciones ambientales, como la humedad, la temperatura u otras como el envejecimiento (oxidación, desgaste, etc.) del sensor.
- Repetitividad: error esperado al repetir varias veces la misma medida.

Un sensor es un tipo de transductor que transforma la magnitud que se quiere medir o controlar, en otra, que facilita su medida.

Pueden ser de indicación directa (ej. un termómetro de mercurio) o pueden estar conectados a un indicador (posiblemente a través de un convertidor analógico a

digital, un computador y un display) de modo que los valores detectados puedan ser leídos por un humano.

Por lo general, la señal de salida de estos sensores no es apta para su lectura directa y a veces tampoco para su procesado, por lo que se usa un circuito de acondicionamiento, como por ejemplo un puente de Wheatstone, amplificadores y filtros electrónicos que adaptan la señal a los niveles apropiados para el resto de la circuitería esto se lo hace ya que los niveles de la señal de salida son muy bajos a nivel de los micro voltios y se necesitan un circuito especial que amplifique esta señal para que pueda ser interpretada de mejor manera, luego esta señal analógica es digitaliza por medio de conversores análogo a digital para poder ser mostrada mediante un Display u otra fuente de visualización.

1.2.2.2 Resolución y precisión

La resolución de un sensor es el menor cambio en la magnitud de entrada que se aprecia en la magnitud de salida. Sin embargo, la precisión es el máximo error esperado en la medida.

La resolución puede ser de menor valor que la precisión. Por ejemplo, si al medir una distancia la resolución es de 0,01 mm, pero la precisión es de 1 mm, entonces pueden apreciarse variaciones en la distancia medida de 0,01 mm, pero no puede asegurarse que haya un error de medición menor a 1 mm. En la mayoría de los casos este exceso de resolución conlleva a un exceso innecesario en el coste del sistema. No obstante, en estos sistemas, si el error en la medida sigue una distribución normal o similar, lo cual es frecuente en errores accidentales, es decir, no sistemáticos, la repetitividad podría ser de un valor inferior a la precisión. Sin embargo, la precisión no puede ser de un valor inferior a la resolución, pues no puede asegurarse que el error en la medida sea menor a la mínima variación en la magnitud de entrada que puede observarse en la magnitud de salida.

1.2.2.3 Tipos de sensores[5]

A continuación se indican algunos tipos y ejemplos de sensores electrónicos:

• **Sensores de temperatura:** Termopar, Termistor

• **Sensores de deformación:** Galga extensiométrica

• **Sensores de acidez:** IsFET

• **Sensores de luz:** fotodiodo, fotorresistencia, fototransistor

• **Sensores de sonido:** micrófono

• **Sensores de contacto:** final de carrera

• **Sensores de imagen digital (fotografía):** CCD o CMOS

• **Sensores de proximidad:** sensor de proximidad

1.2.3 EL RELÉ[6]

El **relé** o **relevador** es un dispositivo electromecánico. Como se puede observar en la Figura 1.5 funciona como un interruptor controlado por un circuito eléctrico en el que, por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes.

Fue inventado por Joseph Henry en 1835.

---

[5] http://www.mitecnologico.com/Main/TiposSensores

[6] http://es.wikipedia.org/wiki/Rel%C3%A9

Dado que el relé es capaz de controlar un circuito de salida de mayor potencia que el de entrada, puede considerarse, en un amplio sentido, como un amplificador eléctrico. Como tal se emplearon en telegrafía, haciendo la función de repetidores que generaban una nueva señal con corriente procedente de pilas locales a partir de la señal débil recibida por la línea. Se les llamaba "relevadores". De ahí "relé".

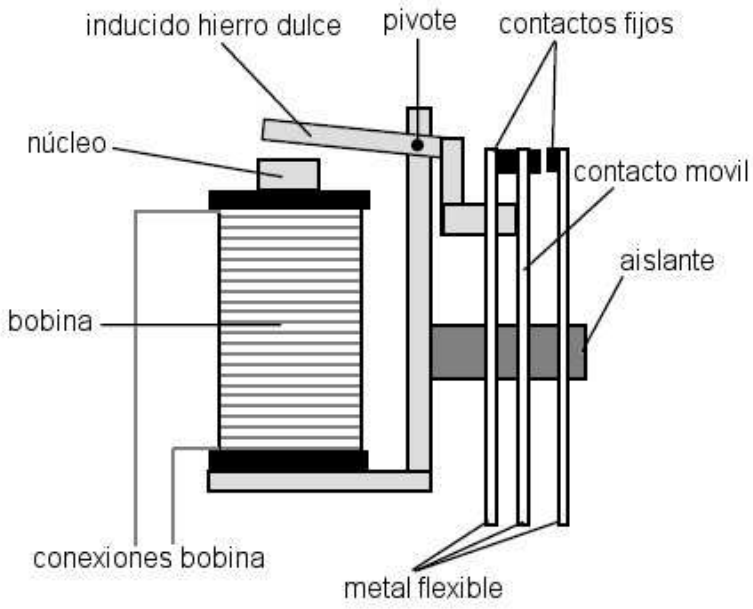


Figura 1.5. Estructura de un relé

1.2.3.1 Estructura y funcionamiento

El electroimán hace bascular la armadura al ser excitada, cerrando los contactos dependiendo de si es N.A o N.C (normalmente abierto o normalmente cerrado). Si se le aplica un voltaje a la bobina se genera un campo magnético, que provoca que los contactos hagan una conexión. Estos contactos pueden ser considerados como el interruptor, que permite que la corriente fluya entre los dos puntos que cerraron el circuito.

1.2.3.2 Tipos de relés

1.2.3.2.1 Relés electromecánicos

- **Relés de tipo armadura**: pese a ser los más antiguos siguen siendo los más utilizados en multitud de aplicaciones. Un electroimán provoca la basculación de una armadura al ser excitado, cerrando o abriendo los contactos dependiendo de si es NA (normalmente abierto) o NC (normalmente cerrado).
- **Relés de núcleo móvil**: a diferencia del anterior modelo estos están formados por un émbolo en lugar de una armadura. Debido a su mayor fuerza de atracción, se utiliza un solenoide para cerrar sus contactos. Es muy utilizado cuando hay que controlar altas corrientes
- **Relé tipo *reed* o de lengüeta**: están constituidos por una ampolla de vidrio, con contactos en su interior, montados sobre delgadas láminas de metal. Estos contactos conmutan por la excitación de una bobina, que se encuentra alrededor de la mencionada ampolla.
- **Relés polarizados o biestables**: se componen de una pequeña armadura, solidaria a un imán permanente. El extremo inferior gira dentro de los polos de un electroimán, mientras que el otro lleva una cabeza de contacto. Al excitar el electroimán, se mueve la armadura y provoca el cierre de los contactos. Si se polariza al revés, el giro será en sentido contrario, abriendo los contactos ó cerrando otro circuito.

1.2.3.2.2 Relé estado solido

Se llama relé de estado sólido a un circuito híbrido, normalmente compuesto por un optoacoplador que aísla la entrada, un circuito de disparo, que detecta el paso por cero de la corriente de línea y un triac o dispositivo similar que actúa de interruptor de potencia. Su nombre se debe a la similitud que presenta con un relé electromecánico; este dispositivo es usado generalmente para aplicaciones donde se

presenta un uso continuo de los contactos del relé que en comparación con un relé convencional generaría un serio desgaste mecánico, además de poder conmutar altos amperajes que en el caso del relé electromecánico destruirían en poco tiempo los contactos. Estos relés permiten una velocidad de conmutación muy superior a la de los relés electromecánicos.

1.2.3.2.3 Relé de corriente alterna

Cuando se excita la bobina de un relé con corriente alterna, el flujo magnético en el circuito magnético, también es alterno, produciendo una fuerza pulsante, con frecuencia doble, sobre los contactos. Es decir, los contactos de un relé conectado a la red, en algunos lugares, como varios países de Europa y Latinoamérica oscilarán a 50 Hz y en otros, como en Estados Unidos lo harán a 60 Hz. Este hecho se aprovecha en algunos timbres y zumbadores, como un activador a distancia. En un relé de corriente alterna se modifica la resonancia de los contactos para que no oscilen.

1.2.3.2.4 Relé de láminas

Este tipo de relé se utilizaba para discriminar distintas frecuencias. Consiste en un electroimán excitado con la corriente alterna de entrada que atrae varias varillas sintonizadas para resonar a sendas frecuencias de interés. La varilla que resuena acciona su contacto, las demás no. Los relés de láminas se utilizaron en aeromodelismo y otros sistemas de telecontrol.

1.2.3.3 Ventajas y usos del relé

La gran ventaja de los relés electromagnéticos 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. También

ofrecen la posibilidad de control de un dispositivo a distancia mediante el uso de pequeñas señales de control. En el caso presentado podemos ver un grupo de relés en bases interfaces que son controlado por módulos digitales programables que permiten crear funciones de temporización y contador como si de un mini PLD (Dispositivo Lógico Programable) se tratase. Con estos modernos sistemas los relés pueden actuar de forma programada e independiente lo que supone grandes ventajas en su aplicación aumentando su uso en aplicaciones sin necesidad de utilizar controles como PLD's u otros medios para comandarlos. Se puede encender por ejemplo una bombilla o motor y al encenderlo se apaga el otro motor o bombilla.

1.2.4 DISPLAY LCD [7]

La pantalla de cristal líquido o LCD (Liquid Crystal Display) es un dispositivo µControlado de visualización grafico para la presentación de caracteres, símbolos o incluso dibujos (en algunos modelos), es este caso dispone de 2 filas de 16 caracteres cada una y cada carácter dispone de una matriz de 5x7 puntos (pixels), aunque los hay de otro número de filas y caracteres. Este dispositivo está gobernado internamente por un microcontrolador Hitachi 44780 y regula todos los parámetros de presentación, este modelo es el más comúnmente usado y esta información se basará en el manejo de este u otro LCD compatible.

1.2.4.1 Características Principales

- Pantalla de caracteres ASCII, además de los caracteres Kanji y Griegos.
- Desplazamiento de los caracteres hacia la izquierda o la derecha.
- Proporciona la dirección de la posición absoluta o relativa del caracter.
- Memoria de 40 caracteres por línea de pantalla.
- Movimiento del cursor y cambio de su aspecto.

[7] http://www.forosdeelectronica.com/f24/control-display-lcd-microcontrolador-pic-201/

- Permite que el usuario pueda programar 8 caracteres.
- Conexión a un procesador usando un interfaz de 4 u 8 bits.

1.2.4.2 Funcionamiento

Para comunicarse con la pantalla LCD podemos hacerlo por medio de sus patitas de entrada de dos maneras posibles, con bus de 4 bits o con bus de 8 bits (Figura 1.6).

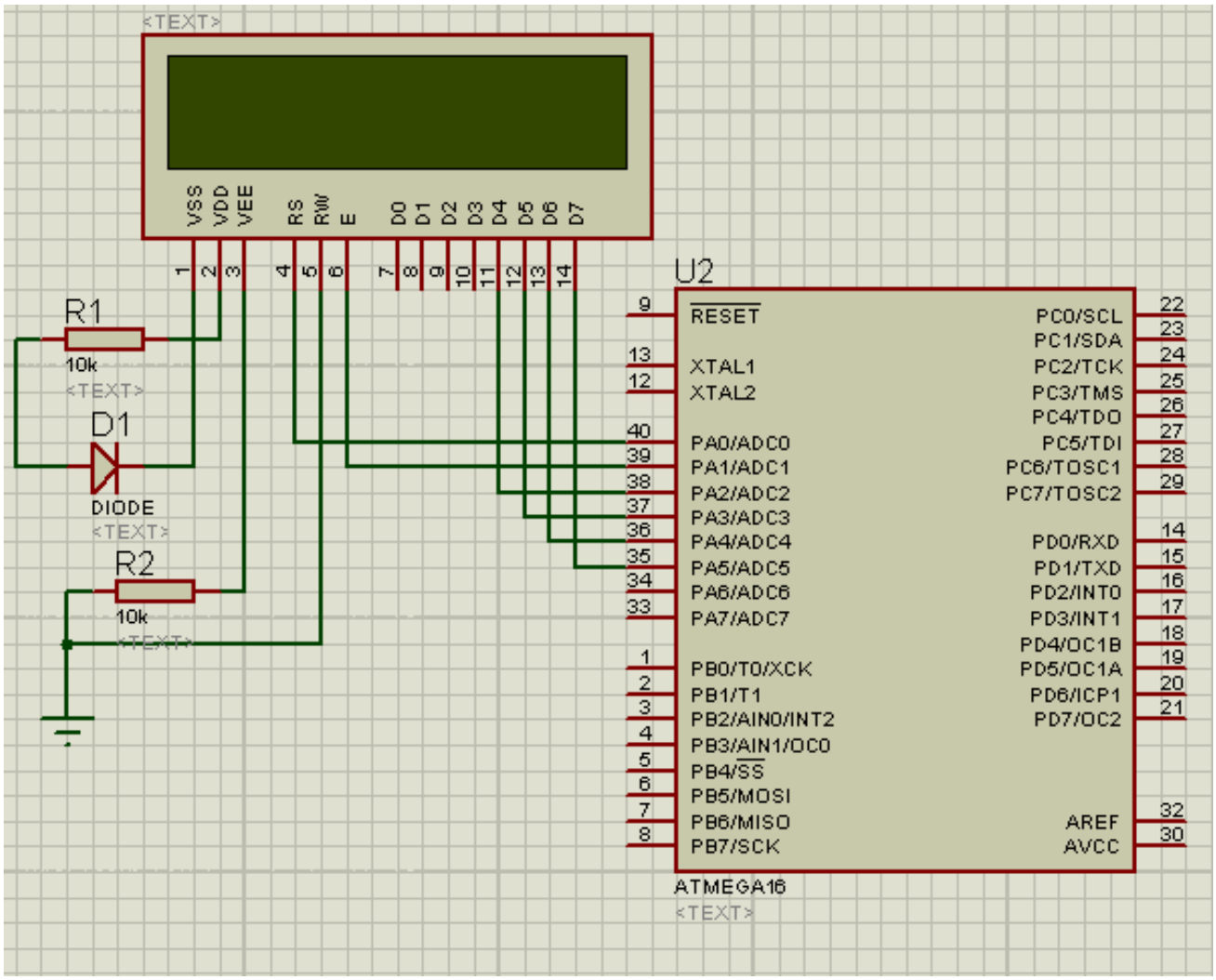


Figura 1.6. Conexionado con bus de 4 bits.

Como puede apreciarse el control de contraste se realiza al dividir la alimentación de 5V con una resistencia variable de 10K.

Las líneas de datos son triestado, esto indica que cuando el LCD no está habilitado sus entradas y salidas pasan a alta impedancia.

1.2.4.3 Descripción de pines

En la tabla 1.1 podemos apreciar la distribución de pines del Display LCD 16*2

Tabla 1.1. Distribución de pines del Display LCD 16*2

PIN N°	SIMBOLO DESCRIPCIÓN
1	Vss Tierra de alimentación GND
2	Vdd Alimentación de +5V CC
3	Vo Contraste del cristal líquido. (0 a +5V)
4	RS Selección del registro de control/registro de datos: • RS=0 Selección registro de control • RS=1 Selección registro de datos
5	R/W Señal de lectura/escritura: • R/W=0 Escritura (Write) • R/W=1 Lectura (Read)
6	E Habilitación del módulo: • E=0 Módulo desconectado • E=1 Módulo conectado
7-14	D0-D7 Bus de datos bidireccional.

1.2.5 COMUNICACIÓN SERIAL

1.2.5.1 EL PUERTO SERIE RS-232[8]

El puerto serie RS-232 es el que se emplea en las computadoras, PC, módems, conmutadores e impresoras y tiene sus inicios en los años 60´s por la EIA (Electronics Industries Association de los EE.UU), este fue creado para ofrecer una conexión entre aparatos que requieren comunicación de Datos.

Durante los últimos 30 años que este estándar ha estado en uso, los equipos han evolucionado tremendamente, pero la norma inicial RS-232 ha cambiado muy poco y los pocos que se han producido han sido debidos a la interpretación propia de algunos fabricantes.

Los ordenadores se conectan con cualquier equipo periférico, a través de sus puertos paralelo o serie, o los más recientes como el USB (Universal Serial Bus, el cual deja desbancados a los otros con 12 Mb/s), pero en éste caso solo analizaremos el puerto serie RS-232 por ser un estándar impuesto en todos los equipos informáticos.

En un ordenador puede haber varios puertos series, a los que normalmente se les denomina COM 1, COM 2, COM 3 (muchas veces los puertos serie a partir del COM 2 se denominan puertos virtuales o son debidos a ampliaciones de los puertos por tarjetas controladoras del tipo PCI), etcétera, por defecto el COM 1 suele pertenecer al ratón usando éste el IRQ 4, aunque también es posible encontrarle en el COM 2, así que lo normal es encontrarnos libre el puerto serie del COM 2 utilizando el IRQ 3.

El tener un puerto serie estándar internacionalmente, permite que distintos fabricantes produzcan aparatos que utilizando esa norma se interconectan entre sí,

[8] http://html.rincondelvago.com/puerto-serie-rs-232.html

aumentando así las posibles aplicaciones y la posibilidad de conectividad entre equipos.

Para realizar una conexión en serie de datos o información, se requiere como mínimo un cable de dos alambres, una conexión del tipo full-duplex como la de telefonía.

Si se quiere tener una comunicación bidireccional por un par de hilos y esta consiste en una serie de bits de información, se requieren otras terminales que indiquen al interface cuál de los aparatos interconectados transmite y cuál recibe, que tipo de información es, cuando el aparato receptor está listo para recibir, cuando el transmisor está listo para transmitir, a qué velocidad va ser la comunicación, etc., esto hace que el puerto serie tenga otras terminales que se usan para coordinar la comunicación entre los equipos.

Los equipos terminales de datos (conmutadores, PC, impresoras, etc.), envían señales en 0´s y 1´s lógicos binarios, que el módem debe convertir a señales analógicas y enviarlas por la línea telefónica o canal de comunicación pero también es posible que se comuniquen siempre en digital.

1.2.5.2 COMUNICACIONES SERIE ASÍNCRONAS[9]

Los datos que se envían en una comunicación serial se encuentran encapsulados en tramas como se indica en la Figura 1.7.

Figura 1.7. Encapsulamiento de los datos serie en tramas.

[9] http://www.iearobotics.com/proyectos/cuadernos/ct1/ct1.html

Primero se envía un bit de start, a continuación los bits de datos (primero el bit de mayor peso) y finalmente los bits de STOP.

El número de bits de datos y de bits de Stop es uno de los parámetros configurables, así como el criterio de paridad par o impar para la detección de errores. Normalmente, las comunicaciones serie tienen los siguientes parámetros: 1 bit de Start, 8 bits de Datos, 1 bit de Stop y sin paridad.

En la figura 1.8 se puede ver un ejemplo de la transmisión del dato binario 10011010. La línea en reposo está a nivel alto:

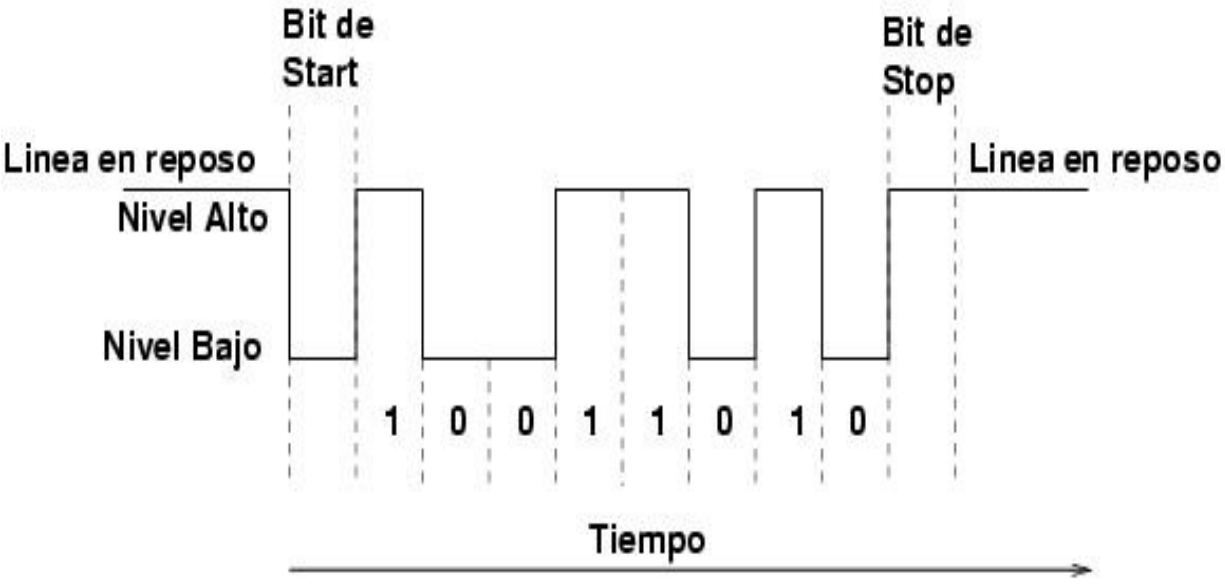


Figura 1.8. Transmisión del dato binario 10011010. La línea en reposo está a nivel alto.

1.2.5.3 NORMA RS232

La Norma RS-232 fue definida para conectar un ordenador a un modem. Además de transmitirse los datos de una forma serie asíncrona son necesarias una serie de

señales adicionales, que se definen en la norma Figura 1.9. Las tensiones empleadas están comprendidas entre +15/-15 voltios.

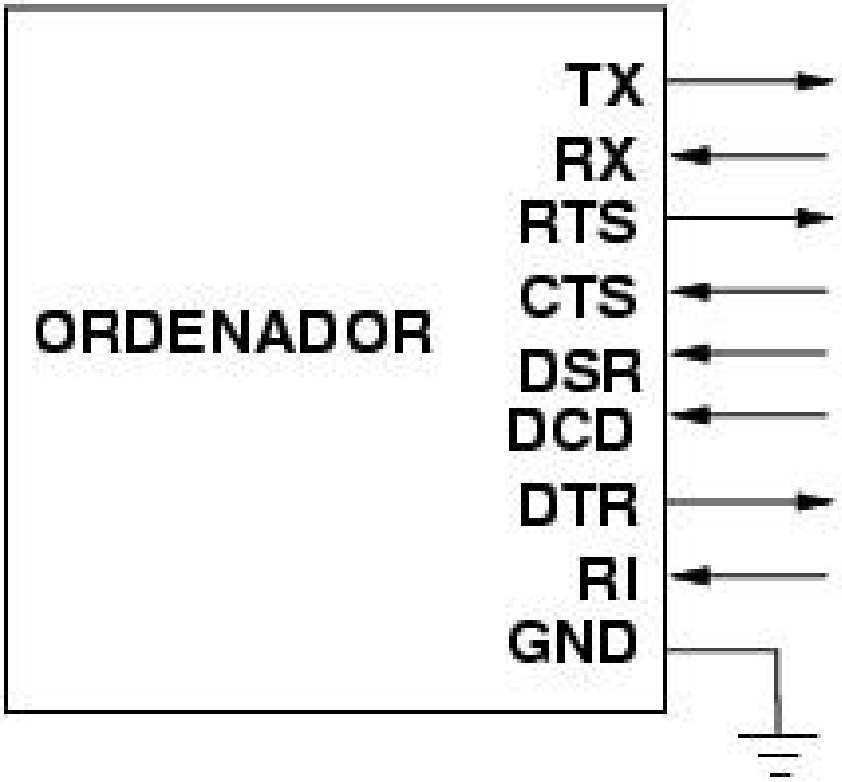


Figura 1.9. Norma RS232

1.2.6 CONEXIÓN DE UN MICROCONTROLADOR AL PUERTO SERIE DEL PC

Para conectar el PC a un microcontrolador por el puerto serie se utilizan las señales Tx, Rx y GND.

El PC utiliza la norma RS232, por lo que los niveles de tensión de los pines están comprendidos entre +15 y -15 voltios. Los microcontroladores normalmente trabajan con niveles TTL (0-5v). Es necesario por tanto intercalar un circuito que adapte los niveles (Figura 1.10).

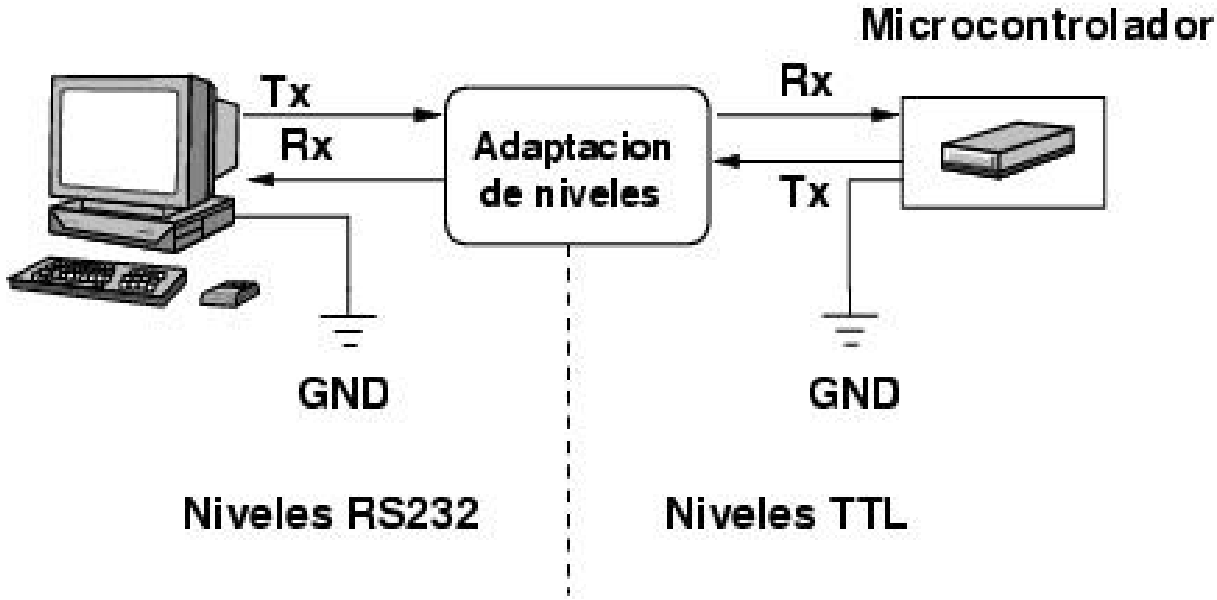


Figura 1.10. Adaptación de niveles RS232 y TTL

Uno de estos circuitos, que se utiliza mucho, es el MAX232.

1.2.6.1 EL CONECTOR DB9 DEL PC

En los PC hay conectores DB9 macho, de 9 pines, por el que se conectan los dispositivos al puerto serie. Los conectores hembra que se enchufan tienen una colocación de pines diferente, de manera que se conectan el pin 1 del macho con el pin 1 del hembra, el pin2 con el 2, etc. (Figura 1.11).

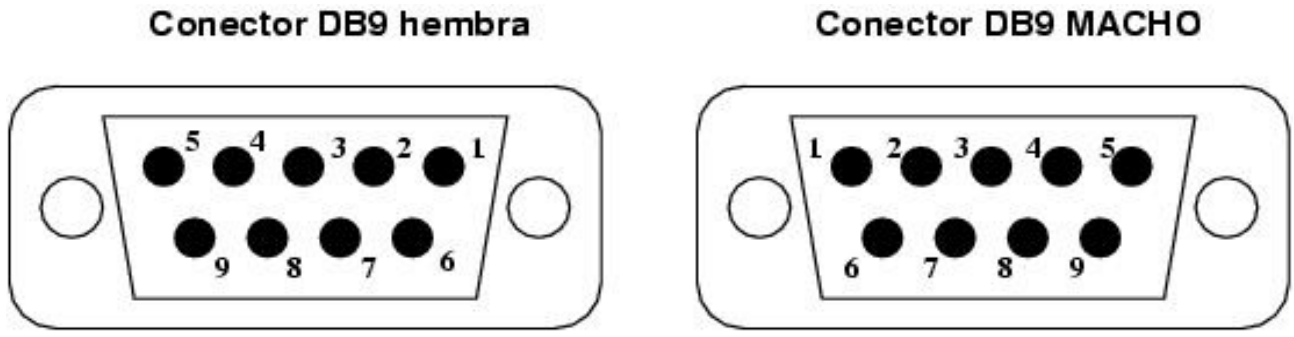


Figura 1.11. Conectores DB9 macho y hembra.

La información asociada a cada uno de los pines está en la Tabla 1.2.

Tabla 1.2. Distribución de los pines del conector DB9

NÚMERO DE PIN	SEÑAL
1	**DCD** (Data Carrier Detect)
2	**RX**
3	**TX**
4	**DTR** (Data Terminal Ready)
5	**GND**
6	**DSR** (Data Sheet Ready)
7	**RTS** (Request To Send)
8	**CTS** (Clear To Send)
9	**RI** (Ring Indicator)

1.2.6.2 EL CHIP MAX 232

Este chip permite adaptar los niveles RS232 y TTL, permitiendo conectar un PC con un microcontrolador.

Para su funcionamiento sólo es necesario este chip y 4 condensadores electrolíticos de 22 micro-faradios, utiliza un voltaje de 5V para su funcionamiento tiene 16 pines y es posible conectarle 4 líneas de entrada y cuatro líneas de salida.

El esquema es el que se muestra en la Figura 1.12.

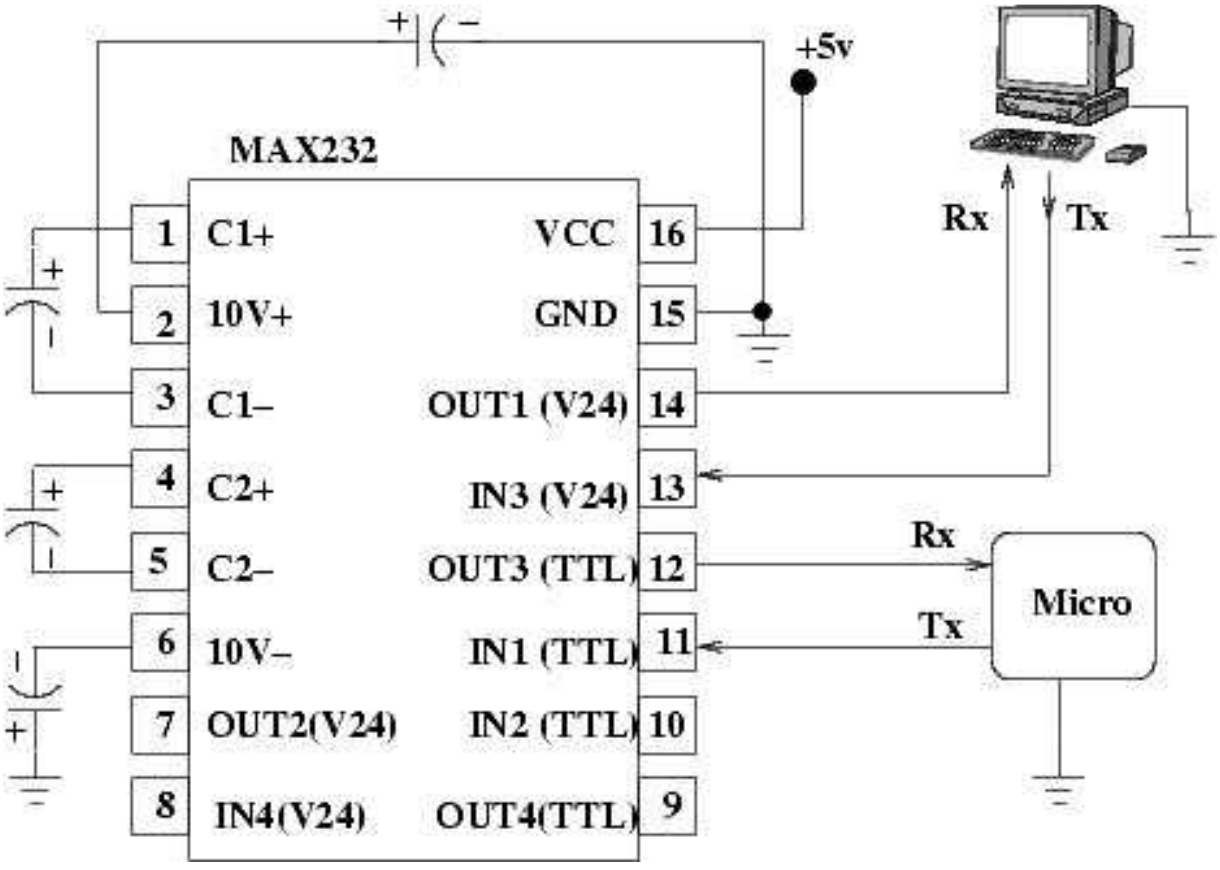


Figura 1.12. Distribución de pines del CI MAX232.

2 CAPITULO II: LENGUAJE DE PROGRAMACIÓN

2.1 LENGUAJE DE PROGRAMACIÓN BASCOM AVR[10]

La herramienta BASCOM AVR desarrollada por la empresa MCS Electronics, sirve para realizar programas de alto nivel para microcontroladores AVR, el cual posee un compilador y un ensamblador que traduce las instrucciones estructuradas en lenguaje de máquina.

En la Figura 2.1 se muestra del diagrama de bloques de la programación estructurada.

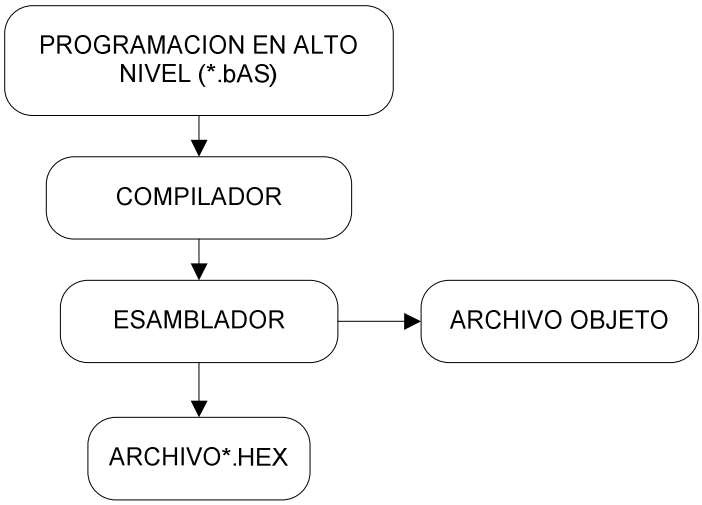


Figura 2.1. Diagrama de bloques de programación estructurada.

Luego de instalar el paquete computacional, el cual se puede conseguir como DEMO en la página del MCS Electronics, podemos apreciar la siguiente pantalla inicial que se muestra en la Figura 2.2..

[10] http://bibdigital.epn.edu.ec/bitstream/15000/2349/1/CD-3097.pdf

Figura 2.2. Ambiente del BASCOM AVR

Dentro de ella podemos ver claramente una barra de herramientas, un menú y el área de trabajo. A continuación se explicara los iconos o atajos más importantes para manejar la herramienta BASCOM AVR.

2.1.1. INICIO

Presionando NEW, nosotros podemos abrir un archivo en blanco para empezar a trabajar en nuestro proyecto (Figura 2.3).

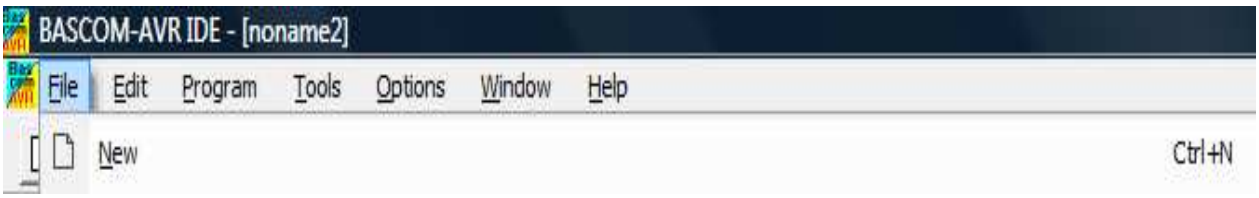


Figura 2.3. Barra de Herramientas File

2.1.2. COMPILADOR

Presionando el ícono de la barra de herramientas o F7 (Figura 2.4), nosotros podemos compilar nuestro proyecto y obtener un archive .HEX, el cual va hacer grabado en el microcontrolador.

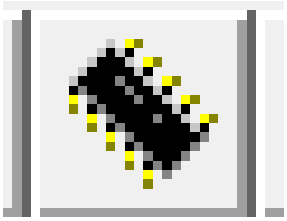


Figura 2.4. Icono del compilador BASCOM AVR.

Una vez que se ha compilado el proyecto puede aparecer el cuadro de confirmación que muestra la Figura 2.5.

Figura 2.5. Cuadro de confirmación de compilación.

En el cual se puede comprobar el porcentaje de memoria utilizada en el microcontrolador.

2.1.3. SIMULADOR

Una vez que se compila un proyecto, se puede simular con ayuda de BASCOM SIM, lo cual se realiza presionando el icono de simulación de la barra de herramientas F2 (Figura 2.6).

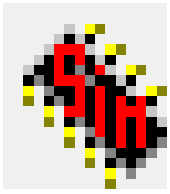


Figura 2.6. Icono del simulador BASCOM AVR.

Una vez que se presiona el simulador, aparece una pantalla donde se puede apreciar el programa principal (Figura 2.7), espacios de memoria, emuladores de comunicación serial, emuladores de LCD, etc.

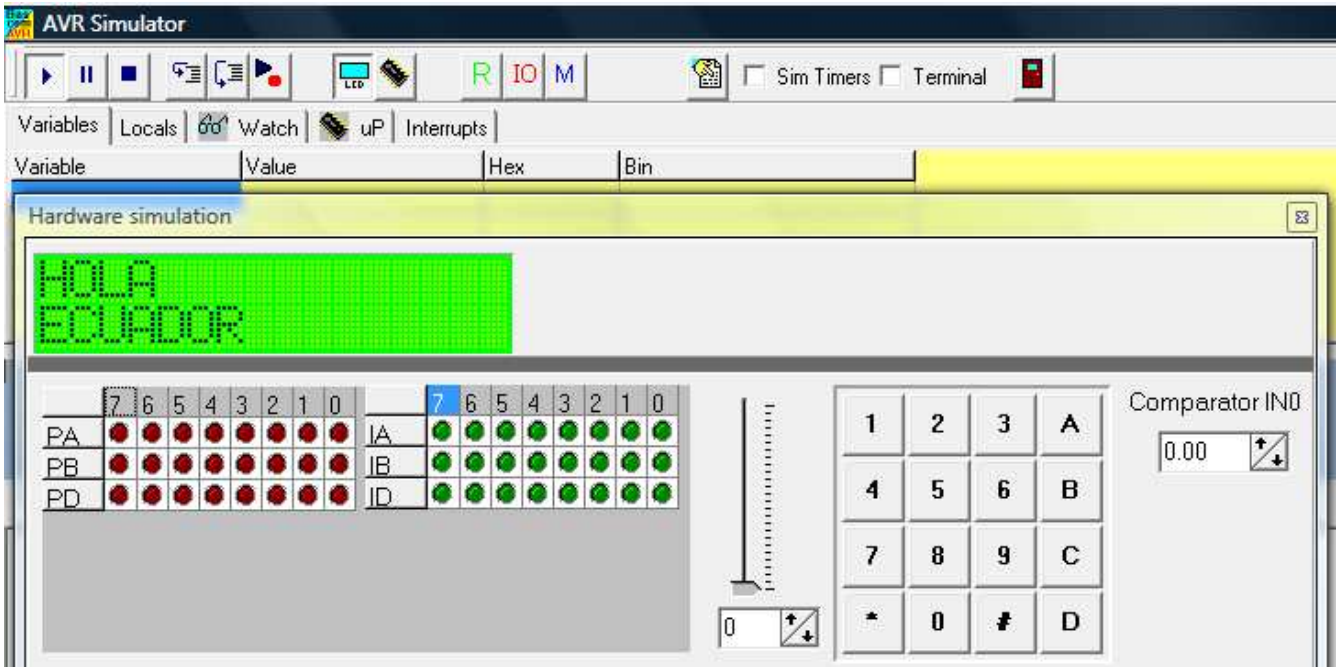


Figura 2.7. Cuadro de simulación en BASCOM AVR.

Es preferible al momento de usar este simulador, que se vaya realizando el proceso paso a paso mediante F8, con lo cual observamos una flecha azul en la parte izquierda del programa, que nos indicará el avance de la simulación

2.1.4. EMULADOR SERIAL

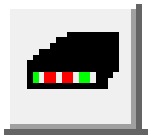
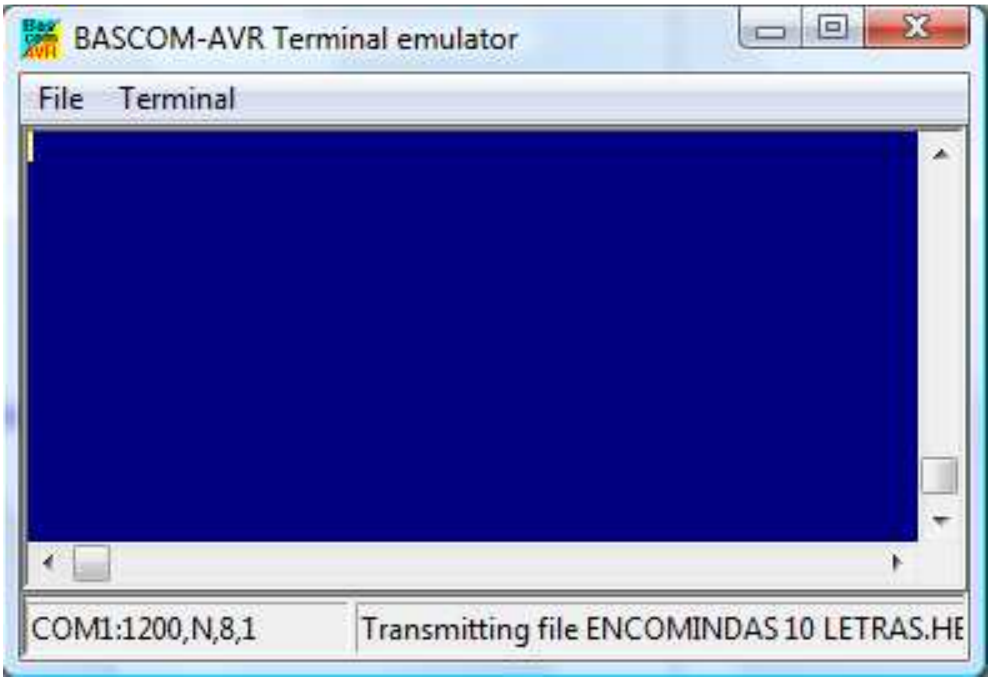


Figura 2.8. Icono del Emulador serial BASCOM AVR.

Con este icono (Figura 2.8) se puede hacer uso de un emulador de comunicación serial entre el microcontrolador y un PC, en el cual podremos observar en la Figura 2.9, que emula un terminal no inteligente, el cual recibe o transmite caracteres.



`**Figura 2.9.** Emulador de Comunicación Serial.

2.1.5. CONEXIONES PRINCIPALES

Dentro de las conexiones principales de un microcontrolador están: el programador, el oscilador, la alimentación y el reset. Para la cual se recomienda tomar en cuenta los siguientes aspectos al momento de armar un circuito (Figura 2.10).

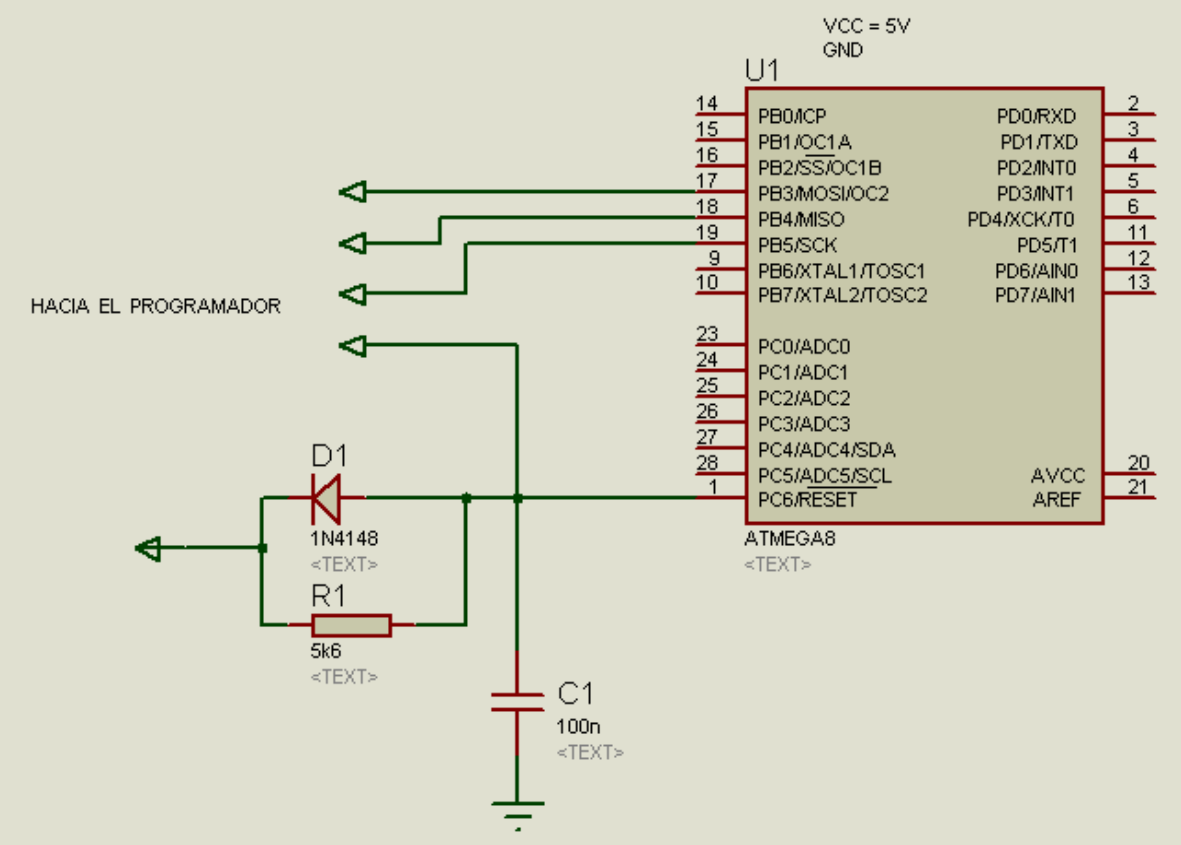


Figura 2.10. Conexiones principales del microcontrolador AVR.

2.1.6. GRABANDO EL MICROCONTROLADOR

Una vez que obtenemos nuestro archivo ".HEX", procedemos a grabar el microcontrolador, para la cual necesitamos un circuito que active la programación del microcontrolador y pase todas las instrucciones hacia la memoria de programa del mismo.

En el mercado encontramos una diversidad de circuitos grabadores de AVR, los cuales nos muestran principalmente los fusibles y el archivo a cargar en el microcontrolador.

Por ejemplo dentro de la ayuda de BASCOM, se encuentra un circuito grabador, llamado STK200-300 (IPS Programmer), el cual utiliza el puerto paralelo (DB25) para grabar el microcontrolador (Figura 2.11).

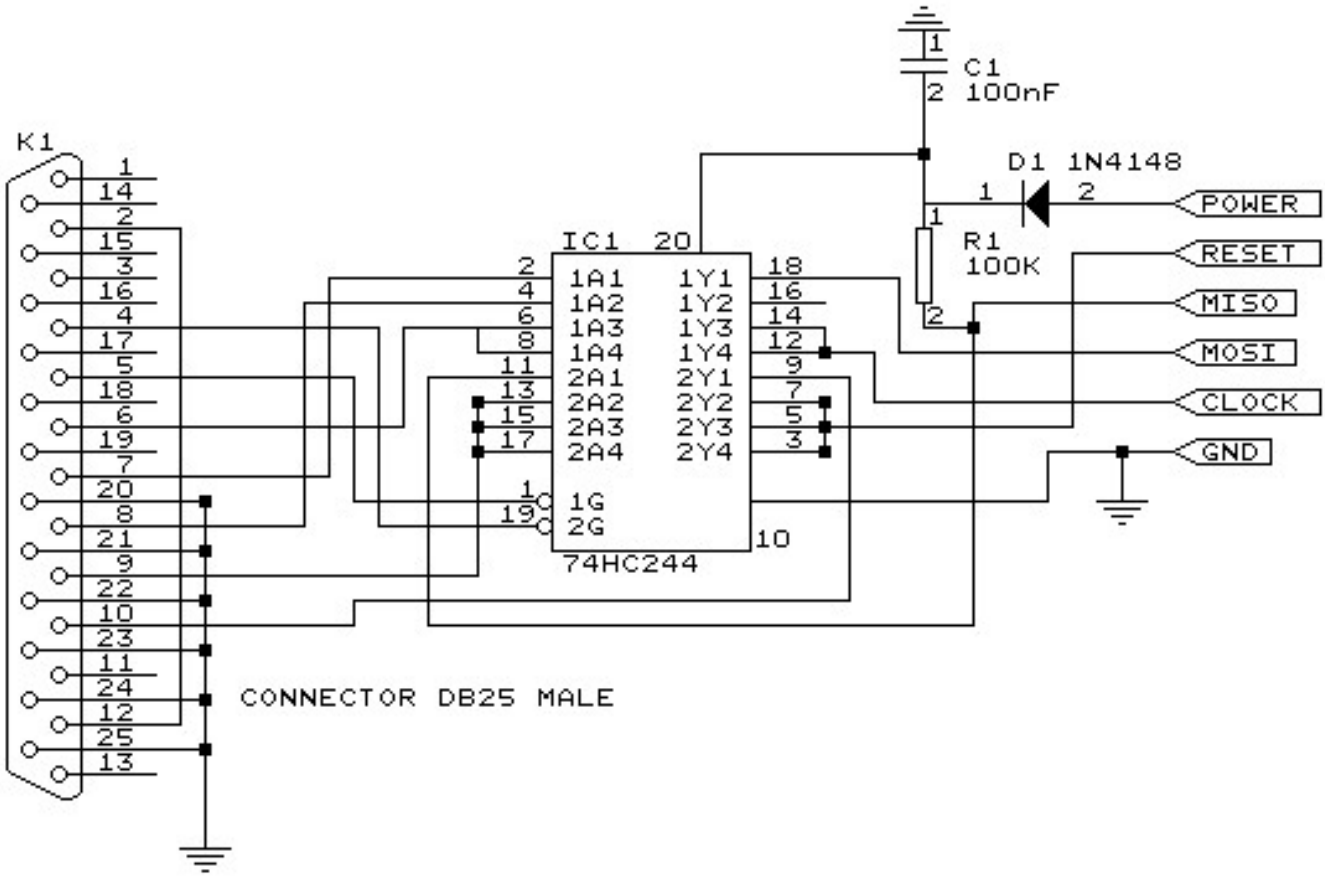


Figura 2.11. Circuito de grabación en paralelo (STK 200-300) para el microcontrolador AVR.

Además se puede interactuar con el mismo BASCOM para escribir el programa, compilar y grabar el micro controlador, ya que si se presiona F4 o el icono de la barra de herramientas, nos puede dar la opción de utilizar el programador mostrado anteriormente.

Y podemos visualizar un software propio del BASCOM, que nos permita grabar el microcontrolador, el cual se observa en la Figura 2.12.

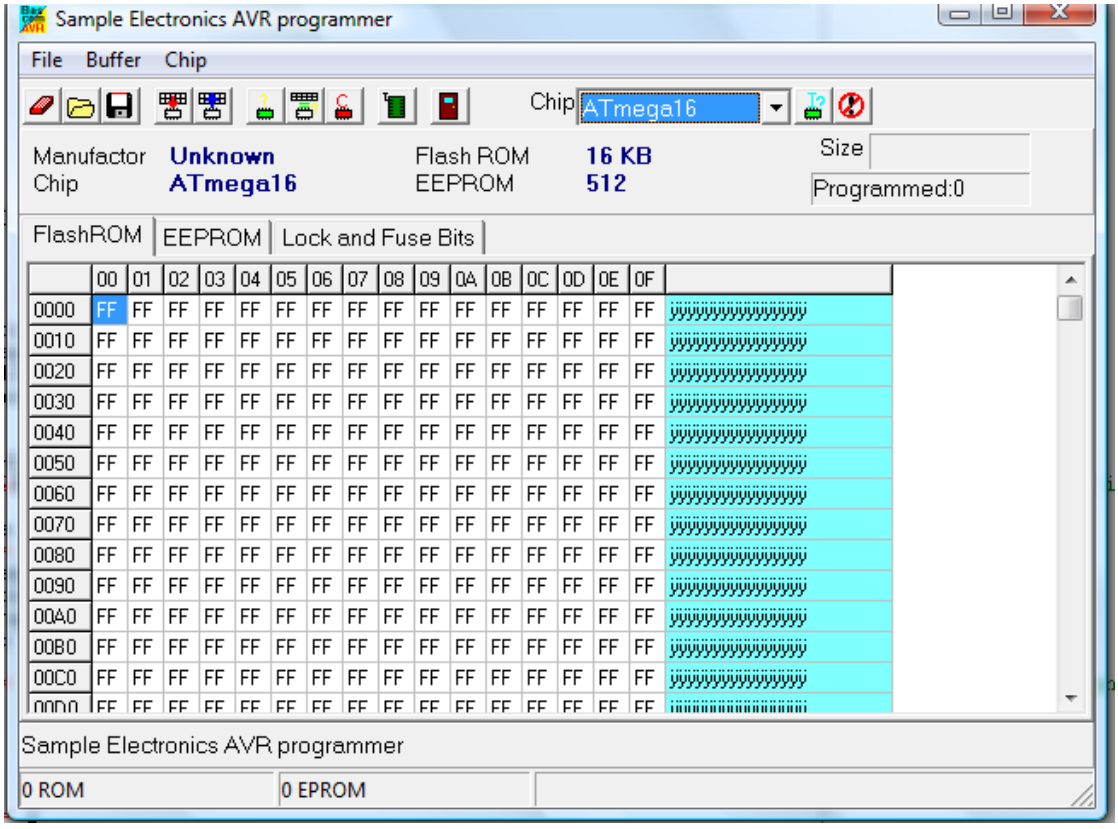


Figura 2.12. Pantalla de grabación del programa BASCOM AVR.

2.1.7. INSTRUCCIONES BÁSICAS DE BASCOM AVR

Para iniciar a descubrir cada una de las instrucciones que posee esta herramienta, empezaremos realizando la respectiva explicación de los comandos de instrucciones y un ejemplo de cada comando de instrucción.

2.1.7.1. $regfile

Esta instrucción siempre va al inicio de cualquier proyecto que realicemos, ya que esta se encarga de direccionar el respectivo microcontrolador que vamos a usar, podemos utilizar una gran variedad de microcontroladores como el ATMEGA 48, ATMEGA 16, ATMEGA 8, etc.

Por ejemplo si vamos a usar:

ATMEGA 48 —**$regfile**=”m48def.dat”

ATMEGA 16 —**$regfile**=”m16def.dat”

ATMEGA 8 —**$regfile**=”m88def.dat”

2.1.7.2. $crystal

Esta instrucción va a especificar la frecuencia de oscilación con la que va a funcionar el micro controlador

Por ejemplo:

$crystal=1000000 para 1MHz

$crystal=8000000 para 8MHz

$crystal=11059200 para 11.0592MHz

2.1.7.3. Config

Esta instrucción especifica la configuración de un pin, un puerto o un dispositivo, ya que pueden ser configurados como estradas o salida de datos.

Por ejemplo:

Config portb = output Puerto B como salida

Config pina.0 = input Pin A.0 como estrada

Config LCD = 16 * 2 LCD de 16 caracteres y 2 líneas

2.1.7.4. Wait, Waitms, Waitus

Esta instrucción sirve para crear un retardo, ya sea en segundos, milisegundos y microsegundos respectivamente.

Por ejemplo:

Wait 3 Espera 3 segundos

Waitms 700 Espera 3 segundos

Waitus 500 Espera 3 segundos

2.1.7.5. Do – Loop

Esta instrucción es un lazo cerrado, en el cual se ejecuta un conjunto de instrucciones de forma indeterminada.

2.1.7.6. Do – Loop Until

Es un lazo definido por la condición de una variable que está dentro del lazo, la cual define cuando termina de ejecutarse el conjunto de instrucciones.

Por ejemplo:

Do

A=a+1

Loop Until a= 10 Termina el lazo cuando a=10

2.1.7.7. Toggle

Este comando sirve para complementar el estado anterior de alguna variable o pin de algún puerto.

Por ejemplo:

Toggle Portb.0 Complementa el Portb.0

2.1.7.8. Dim

Dim Sirve para dimensionar el tipo de variable que se va a utilizar.

Entre los tipos de variables están en los que se muestran en la Tabla 2.1 y podemos ver la longitud de cada tipo yen casos los debemos utilizar:

Tabla 2.1. Dimensiones de variables BASCOM AVR.

TIPO	DIMENSIÓN
Bit	0 - 1
Byte	0 a 255
Word	0 a 65535
Long	-2147483648 a 2147483647
Integer	-32768 a 32767
Single	1.5×10^{-45} x 3.4×10^{38}
String	Cadena de caracteres máximo 254
Array	Matriz 65535
Double	5.0×10^{324} a 1.7×10^{308}

2.1.7.9. Alias

Sirve para dar un nombre general dentro de un proyecto, ya sea a un puerto o pin de un puerto.

Por ejemplo:

Foco **Alias** Portb.0 El Portb.0 ahora se llama foco

2.1.7.10. LCD (Display de cristal líquido)

En el caso del BASCOM AVR, podemos controlar al LCD de dos maneras:

- Por comandos
- Por configuración en cuadro de diálogo.

2.1.7.10.1. Mediante comando tenemos

2.1.7.10.1.1. Config Lcd

Sirve para configurar la clase de LCD que vamos a utilizar puede ser de 16 caracteres por dos líneas (16x2), de 20 caracteres por 4 líneas (20x4), etc

Por ejemplo:

Config Lcd=16 * 2 (Display de 16 x 2)

2.1.7.10.1.2. Config Lcdpin

Sirve para configurar los pines por los cuales se va a manejar la información la clase de LCD que vamos a utilizar, puede ser de 16 caracteres por 2 líneas (16 x 2), de 20 caracteres por 4 líneas (20x4), etc

Por ejemplo:

Config Lcdpin= Pin, Db4=Porta.4, Db5=Porta.5, Db6= Porta.6, Db7=Porta.7, E=Portc.7, Rs=Portc.6

2.1.7.10.1.3. Config lcdbus

Esta instrucción sirve para configurar cual será el modo de envió de datos, ya que puede ser hecho por 4 pines u 8 pines.

Por ejemplo:

Config lcdbus=4 (4 pines de datos)

2.1.7.10.1.4. Lcd ” ”

Sirve para escribir cualquier frase en el LCD, sin importar la localización del cursor.

Por ejemplo:

Lcd= “HOLA”

2.1.7.10.1.5. Locate x,y

Sirve para localizar el cursor en la línea y columna adecuada, para poder empezar a escribir en el LCD.

Por ejemplo:

Locate 1,1 (Localización del cursor en la fila 1, columna 1)

2.1.7.10.1.6. Shiftlcd

Sirve para mover todo el texto del LCD, ya sea para la izquierda o para la derecha, con las instrucciones:

Shiftlcd left
Shiftlcd right

2.1.7.10.2. Mediante cuadro de diálogo tenemos

BACOM AVR, nos permite interactuar con el hardware, mediante cuadros de dialogo, a los cuales podemos ingresar mediante el menú de opciones.

Como se muestra en la Figura 2.13 podemos ver el cuadro de configuración de LCD el mismo que nos aprueba la configuración de los pines que ocuparemos para realizar la comunicación de los distintos dispositivos o periféricos de un micro controlador AVR.

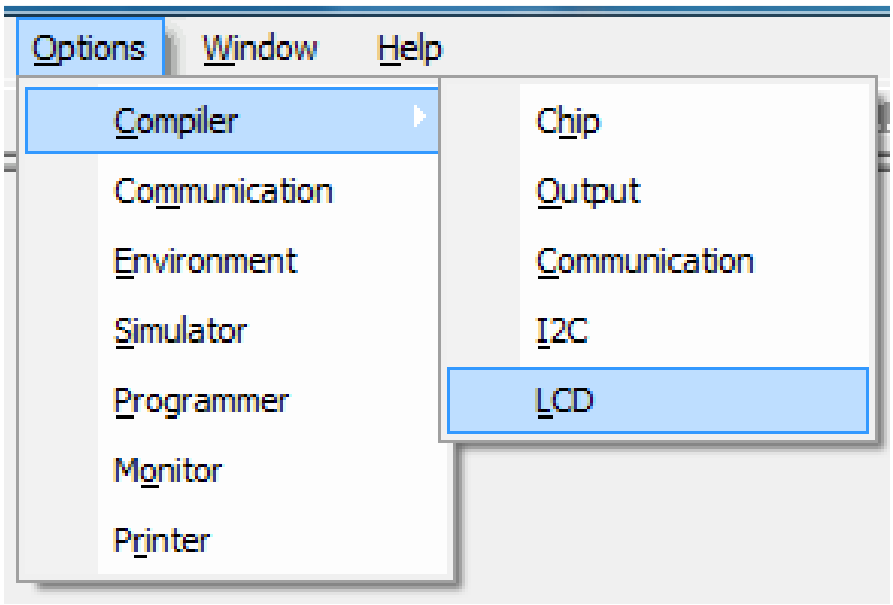


Figura 2.13. Cuadro de diálogo del BASCOM AVR para configurar un LCD.

La pantalla de la Figura 2.14 nos muestra cómo podemos configurar los pines y el tipo de LCD que utilizaremos.

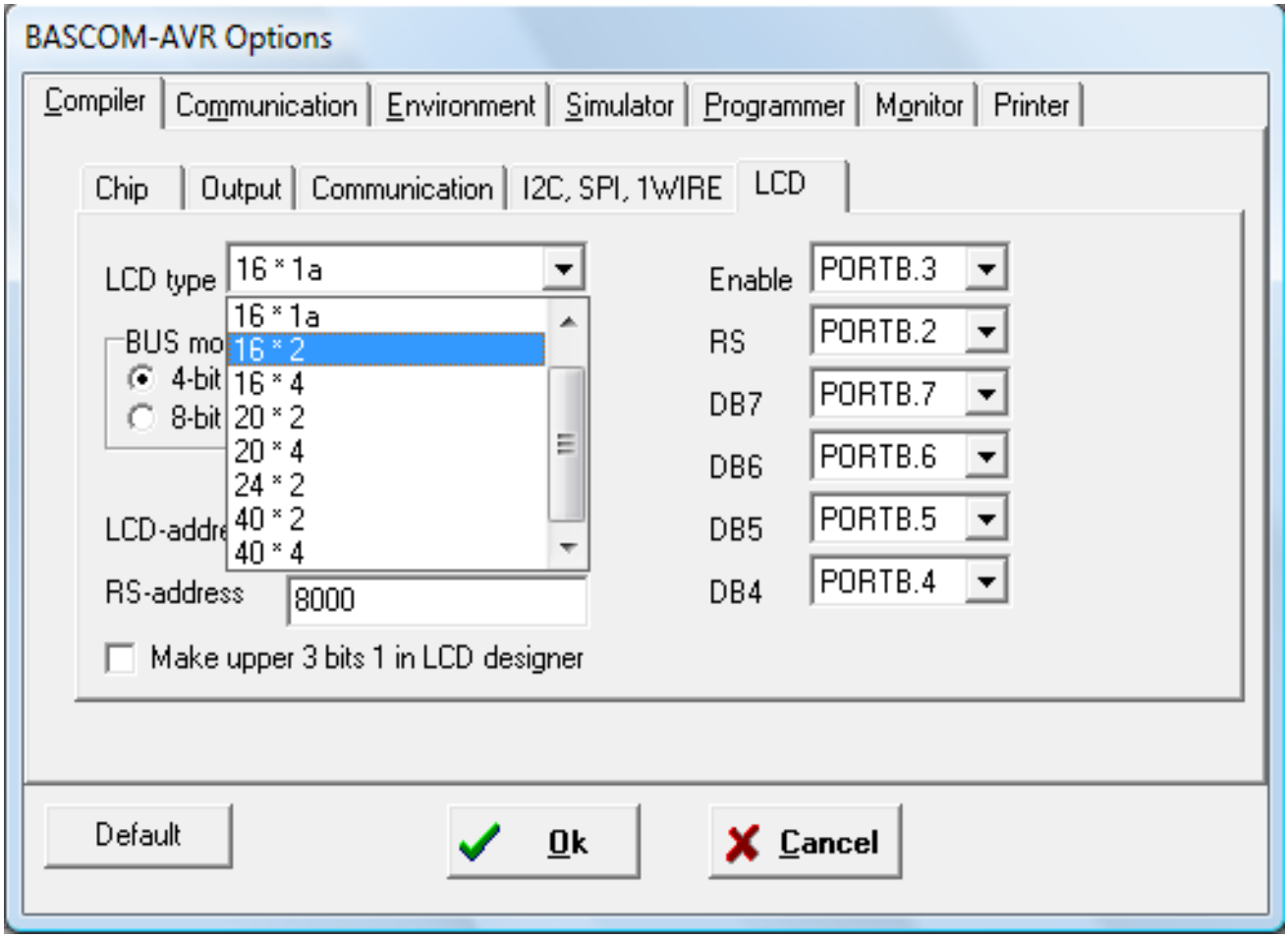


Figura 2.14. Cuadro de diálogo del BASCOM AVR para escoger el funcionamiento de un LCD.

Es recomendable realizar todas las configuraciones de dispositivos mediante código y mediante cuadros de diálogo; de esta manera nos aseguramos que la información de las interfaces de salida estén doblemente escritas y no se pierda en ningún momento.

2.1.7.11. DDRx, PORTx, PINx

DDR, PORT Y PIN son registros que nos permiten utilizar el puerto como entrada o salida de datos.

DDR : Con figura el pin como estrada o salida de datos.

PORT : Es el registro de salida de datos.

Pin : Es el registro de entrada de datos.

Las siguientes combinaciones, hacen que los pines funcionen en configuración especial, como explica a continuación.

Ddrb.x = 0 Entrada alta impedancia.

Portb.x = 0

Ddrb.x = 0 Entrada pull up.

Portb.x = 1

Ddrb.x = 1 Salida a cero (0L) 20 mA.

Portb.x = 0

Ddrb.x = 1 Salida a uno (1L) 20 mA.

Portb.x = 1

Es importante recalcar que cuando se configura un puerto como salida, se debe ocupar la palabra PORT y si se lo configura como entrada se usa la palabra PIN.

PORT = (Salida), PIN = (Entrada).

2.1.7.12. If – Them – Else

Como se indica en la Figura 2.15 son sentencias condicionales, las cuales responden a un estado de voltaje (0L, 1L), de contenido (caracteres), etc.

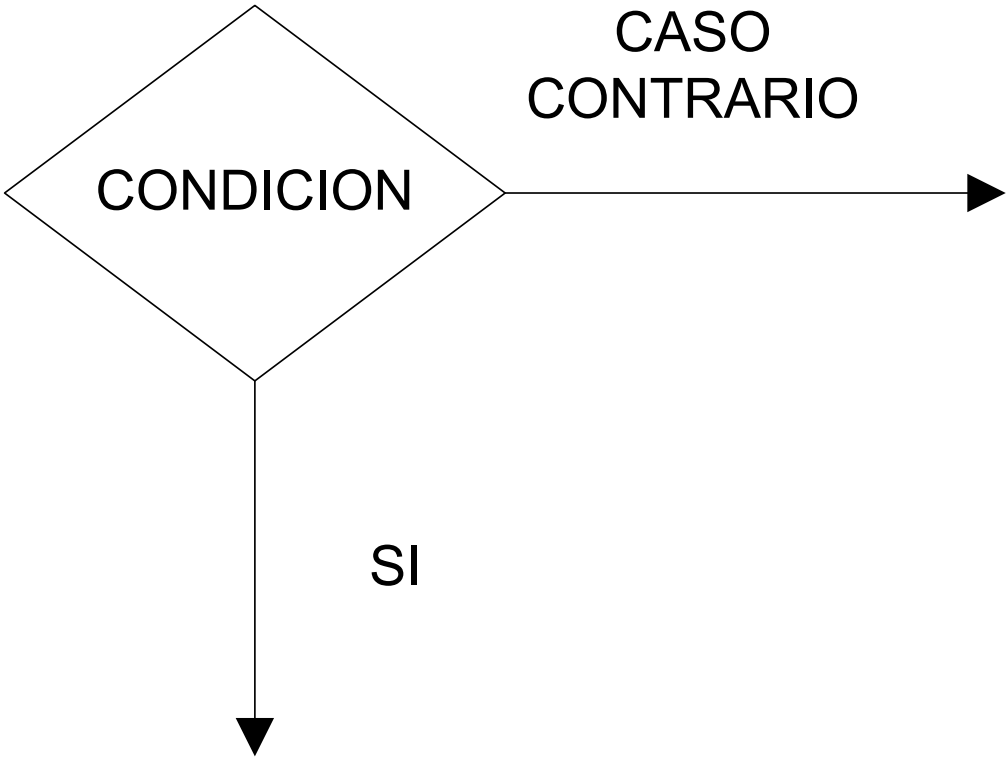


Figura 2.15. Condición lógica IF-ELSE.

2.1.7.13. For – Next

Son instrucciones de repetición, ya que se ejecutan un conjunto de instrucciones, dependiendo de una variable incremental que se encuentre dentro del lazo (Figura 2.16).

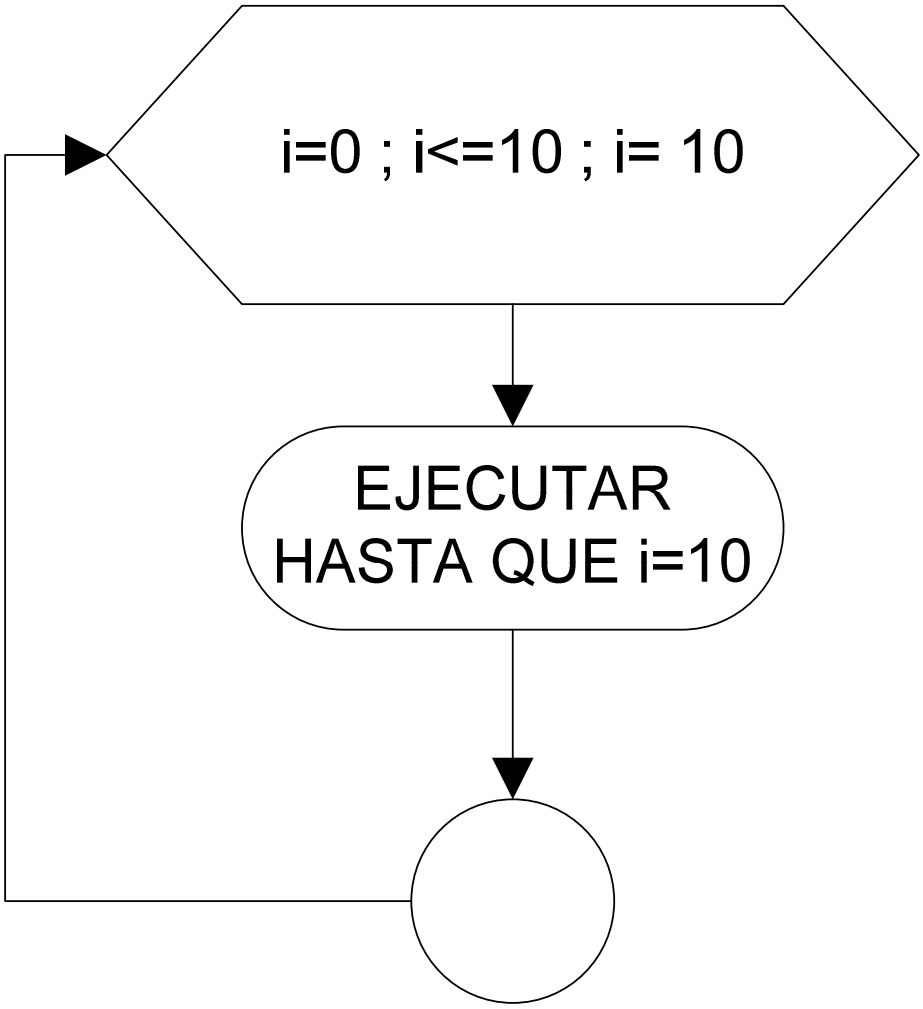


Figura 2.16. Condición de repetición FOR – NEXT.

2.1.7.14. Select – Case

Son sentencias que se pueden ejecutar, dependiendo del estado de una variable de selección.

Dentro de este esquema se puede tener un conjunto de casos que pueden ser ejecutados, dependiendo de las variables en juego (Figura 2.17).

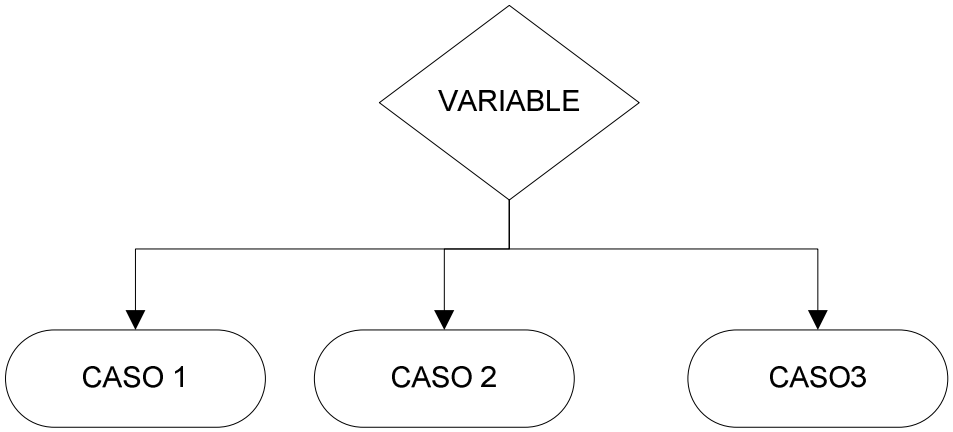


Figura 2.17. Condición de selección SELECT – CASE.

2.1.7.15. Símbolos operadores

Dentro de los operadores, pueden utilizarse los matemáticos, de relación y lógicos. Además se debe tomar en cuenta que BASCOM nos permite realizar operaciones únicamente con dos variables a la vez.

A continuación podremos observar los operadores más comunes.

Operadores Matemáticos

Suma: a=b+c

Resta: a=b-c

Multiplicación: a=b*c

División: a MOD b

Operadores Comparativos

Estos operadores nos sirven para comparar dos variables.

=	Igual	X=Y
<>	No es igual	X<>Y
<	Menor que	X<y
>	Mayor que	X>Y
<=	Menor Igual	X<=Y
>=	Mayor Igual	X>=Y

Operadores lógicos

Estos operadores son los utilizados en la lógica digital.

NOT	Complemento (Negación)
AND	Conjunción (Y)
OR	Disyunción(O)
XOR	Or Exclusiva

Representación de lógica digital

Para la representación de un número binario o hexadecimal, dentro de BASCOM AVR, es necesario anteponer el símbolo “&”. En el caso de números decimales, no es necesario anteponer ningún símbolo.

Ejemplo:

Porta= &HF9	Número Hexadecimal
Porta= &HF9	Número Hexadecimal

Porta= &HF9 Número Hexadecimal

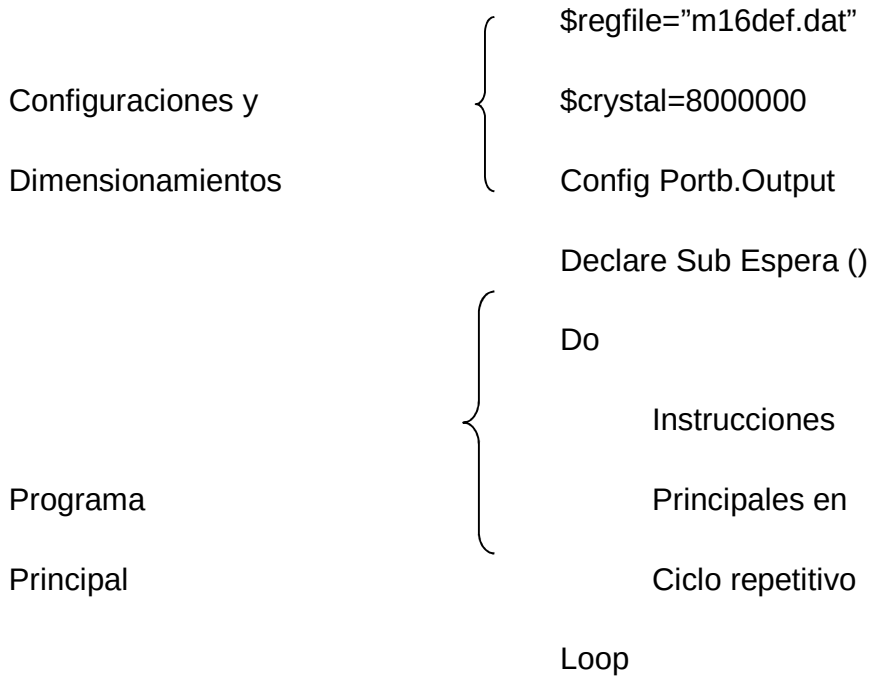
2.1.7.16. Estructura de un programa en BASIC

Este tema es muy importante tocar, ya que cuando se estructura un programa en alto nivel, es necesario llevar un orden y vinculación de las instrucciones que se realizan.

Es primordial que se tengas estructuradas 4 partes dentro de un programa en lenguaje de alto nivel.

- Configuraciones y Dimensionamientos de variables y subrutinas
- Programa principal
- Subrutinas
- Tablas de datos

El siguiente es un ejemplo de cómo se puede estructurar un programa en alto nivel, con tipos de instrucciones que se pueden realizar en su respectivo orden.



Subrutinas	{	Espera:
		Instrucciones
		De subrutina
		Return
Tablas de Datos	{	Tabla1: Data &HC0, &HF(

2.2 ESTRUCTURA DE UN PROGRAMA EN JAVA NETBEANS

NetBeans es un IDE muy completo para programar en varios lenguajes, aunque se especializa principalmente en Java. En un principio, puede que a personas que no estén familiarizadas con entornos de este tipo se les haga un poco dificultoso su manejo, pero eso es cuestión de tiempo hasta que el usuario logre familiarizarse con el entorno.

2.2.1 CÓMO CREAR UNA APLICACIÓN NUEVA

Lo primero que tenemos que hacer es crear un proyecto. Para ello, abriremos el menú **Archivo** y después pulsaremos en **Nuevo proyecto** (Figura 2.18).

Aparecerá una pantalla como la de la Figura 2.19 en la que tenemos que seleccionar qué tipo de proyecto queremos realizar.

Seleccionaremos la categoría Java y en el panel de la derecha **Java Application**. Después pulsamos **Next**.

Figura 2.18. Nuevo proyecto.

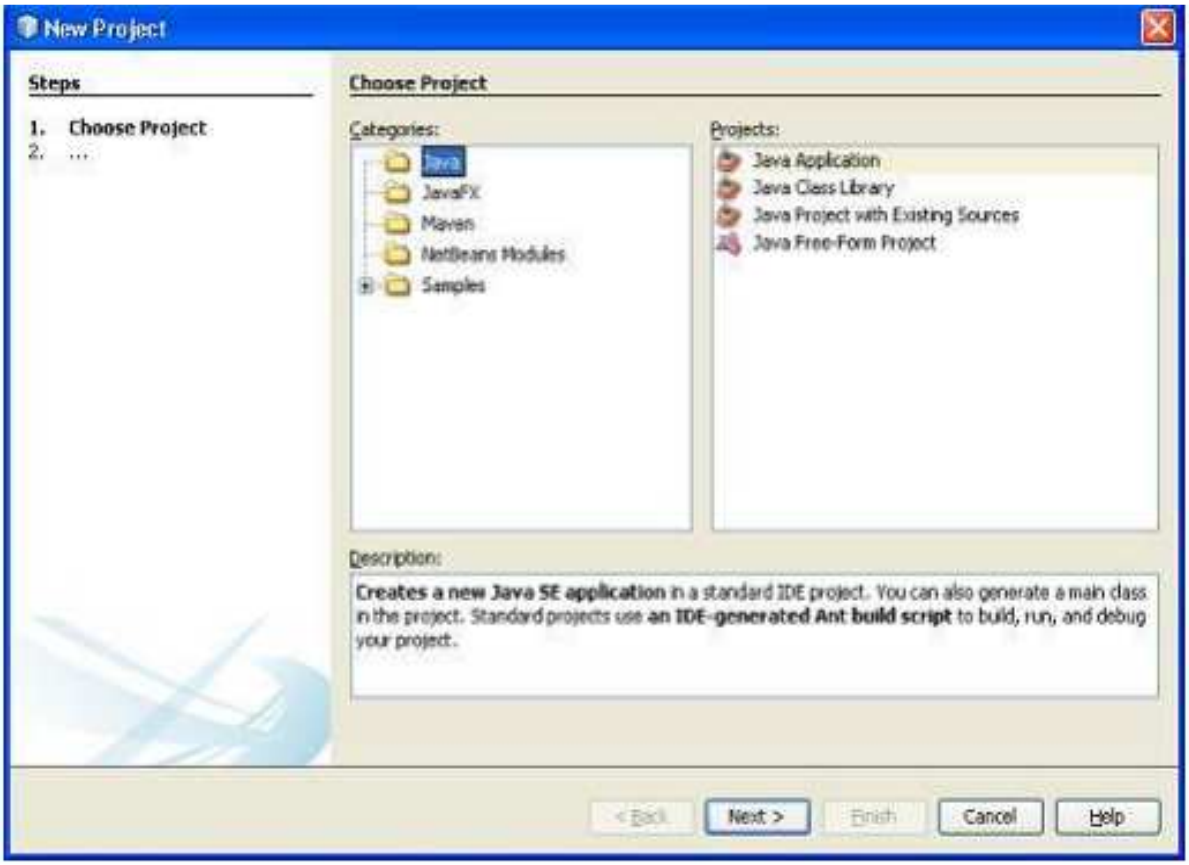


Figura 2.19. Elección el tipo de proyecto.

En la pantalla de la Figura 2.20 se nos preguntará por las propiedades generales del proyecto: nombre, la carpeta donde lo queremos guardar, si queremos utilizar una carpeta específica para guardar las librerías, si queremos crear una clase principal y si queremos establecerla como proyecto principal.

Figura 2.20. Propiedades generales del proyecto.

Las opciones de arriba las podemos configurar como queramos, se recomienda dejar la casilla de las librerías desmarcadas, y por último, como vemos en la imagen, activaremos las dos casillas de la parte inferior. Esto es necesario para que el sistema sepa que lo que vamos a escribir será lo que tenga que ejecutar cuando finalicemos.

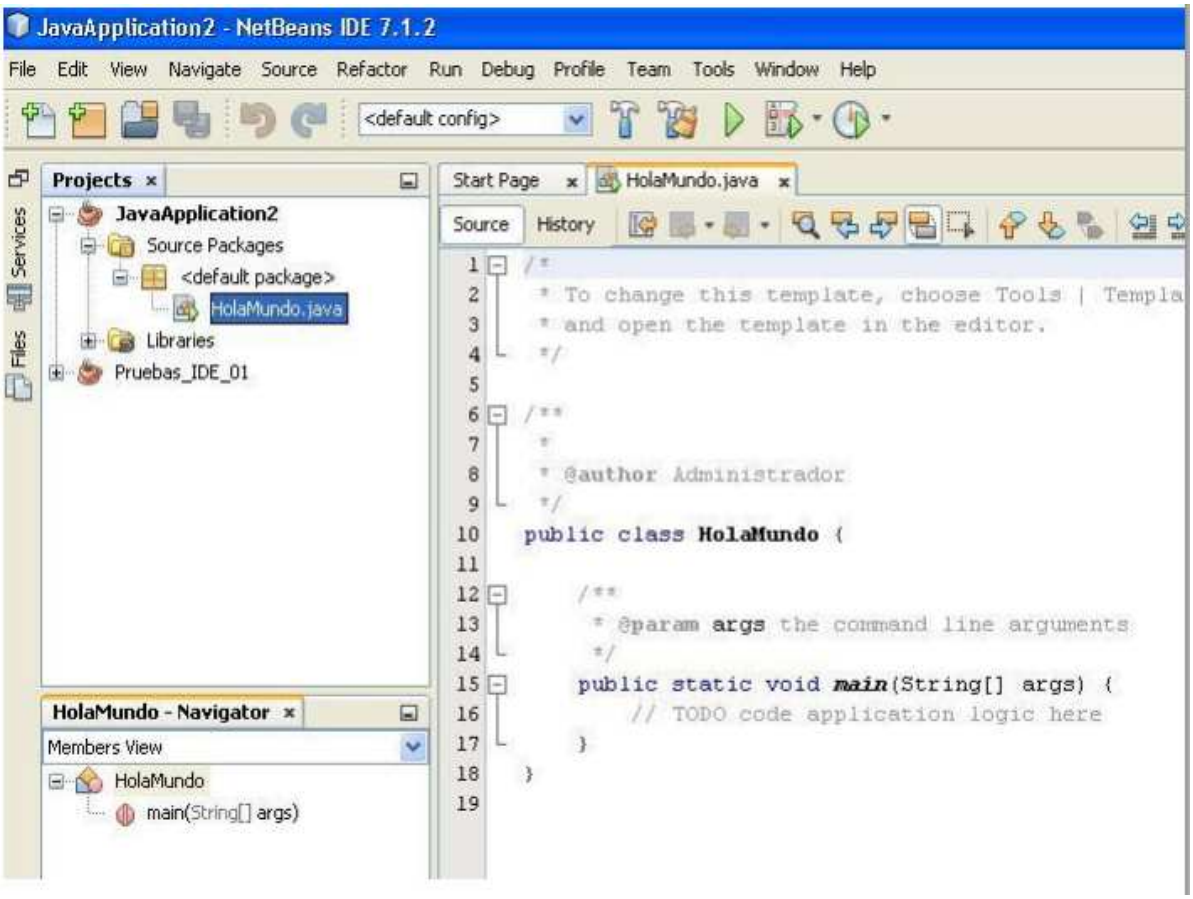


Figura 2.21. Componentes del entorno de Netbeans

En el cuadro de texto he escrito “HolaMundo”, que será la aplicación que hayamos creado. Por último pulsamos en **Finish**. (Figura 2.21).

Figura 2.22. La ventana de proyectos.

En la parte izquierda de la ventana de la Figura 2.22, aparece un pequeño recuadro en el que veremos que aparece seleccionado un pequeño objeto llamado “HolaMundo.java”. Este es el programa que hemos creado, es un archivo con la extensión.jar

Todo lo que aparece en color gris claro son comentarios (Figura 2.23).

Es muy importante que nos acostumbremos a realizar comentarios en nuestros programas. Esto tiene muchas utilidades como poder recordar mejor en un futuro qué es exactamente lo que hacía este código y también que si otra persona lo ve, pueda entender mejor cómo resuelve el problema nuestro programa.

Para hacer un comentario de varias líneas usaremos /* para iniciarlo y */ para finalizarlo. Si vamos a hacer un comentario de una sola línea podemos comenzarlo por // En la línea 11 definimos la clase holamundo, el cuerpo de la clase queda encerrado entre { }, cualquier cosa que coloquemos entre esas llaves será el código que pertenezca a la clase holamundo.

En la línea 16 se ha iniciado el método principal main. Cuando se ejecuta una aplicación, Java espera que haya un método main, ya que será lo primero que ejecute. Este método define el punto de entrada y salida de la aplicación.

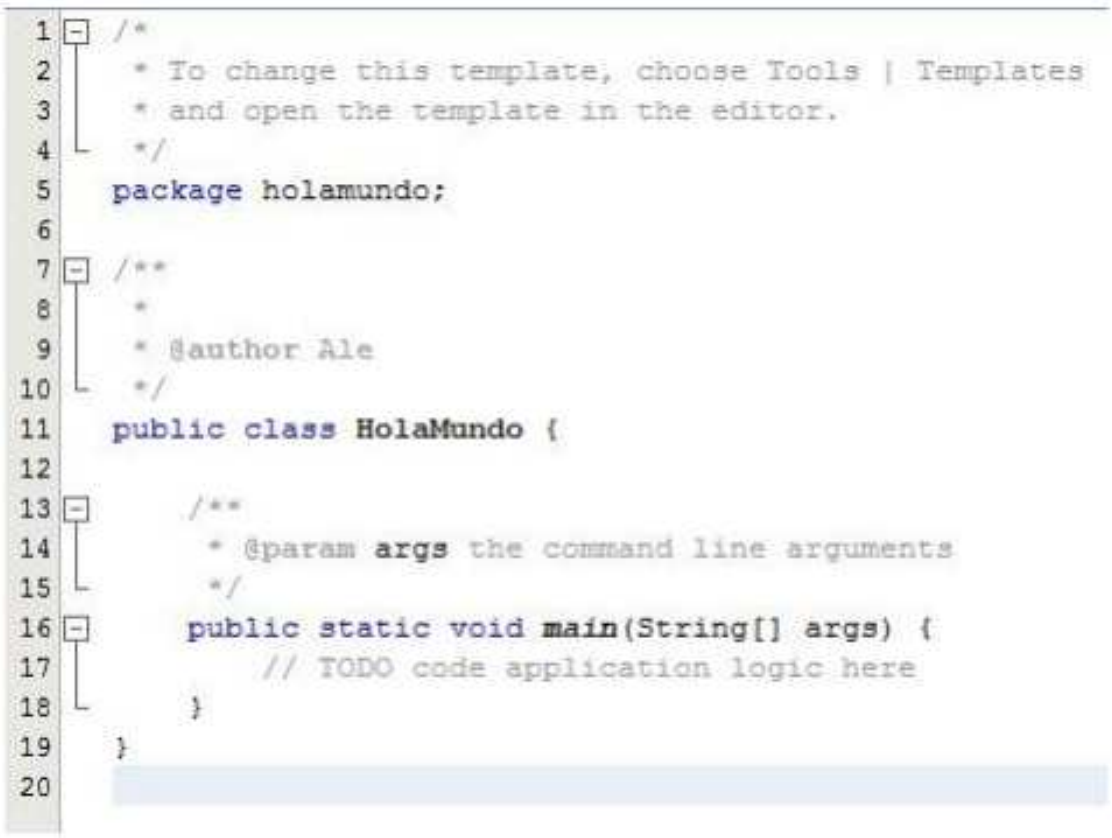


Figura 2.23. Ventana de código.

Cuando uno se inicia en un nuevo lenguaje de programación, el primer programa que hace siempre se llama “HolaMundo”. Es un programa muy simple que se utiliza para que la persona que está aprendiendo cómo funciona el sistema se familiarice con el código. La utilidad del programa no es otra que mostrar el mensaje “Hola mundo” en la pantalla.

public static void main(String[] args) { // TODO code application logic here }

Vamos a escribir lo siguiente justo después del comentario:

System.out.println("Hello World!");

Quedando nuestro programa como se muestra en la Figura 2 24.

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package holamundo;
6
7  /**
8   *
9   * @author Ale
10  */
11 public class HolaMundo {
12
13     /**
14      * @param args the command line arguments
15      */
16     public static void main(String[] args) {
17         // TODO code application logic here
18         System.out.println("Hola Mundo!");
19     }
20 }
21
```

Figura 2.24. Programa finalizado.

La orden que hemos añadido lo que hace es escribir el mensaje “Hola Mundo!” en la pantalla. println es una herramienta que utilizaremos mucho. Hay varios aspectos importantes en esta simple línea de código y que vamos a tener que respetar cuando escribamos código:

- Hemos utilizado el método println del objeto out miembro de la clase System de la biblioteca de Java.
- Después de println hemos abierto paréntesis, lo que haya dentro del paréntesis es lo que llamamos argumento que básicamente podemos decir que es el dato que necesita println para saber qué tiene que escribir.

- Los textos literales, como Hola Mundo!, tenemos que encerrarlos entre comillas.
- La instrucción termina con;

Guarda el proyecto desde el menú **File** -> **Save.**

2.2.2 COMPILANDO Y EJECUTANDO

Para poder ejecutar nuestro programa tenemos que compilarlo;

Compilar es traducir el código que hemos escrito que es el código fuente a código de bytes lenguaje de máquina para que nuestro ordenador pueda entenderlo.

Podríamos hacerlo desde la línea de comandos con esta orden:

javac HolaMundo.java

que es el nombre que le dimos al archivo.

Pero como tenemos un IDE que se va a ocupar de esas cosas por nosotros, lo aprovecharemos, cuando guardas los archivos del proyecto, NetBeans se encarga de guardar el archivo y compilarlo. Así de sencillo, así que sólo nos queda ejecutarlo.

Para ejecutar el programa podemos:

- Ir al menú Run -Run Main Project
- Pulsar la tecla F6
- Pulsar el icono correspondiente de la barra de herramientas

Después de pulsar cualquiera de las opciones anteriores, tendremos que fijarnos en el panel inferior, que es donde Java nos muestra el resultado obtenido.

En la Figura 2.25 observamos el resultado de la compilación correcta del programa.

Efectivamente aparece nuestro mensaje y si todo va bien aparecerá un mensaje en verde indicando que no hay ningún error.

Si hubiera algún error de compilación, ejecución o de otro tipo, conviene fijarse en la ventana del editor. En la parte izquierda, donde aparecen los números de línea, NetBeans nos avisará si hay algún error de sintaxis. En la Figura 2.26 se ha eliminado el ; del final de la instrucción y podemos ver claramente un símbolo de advertencia en rojo.

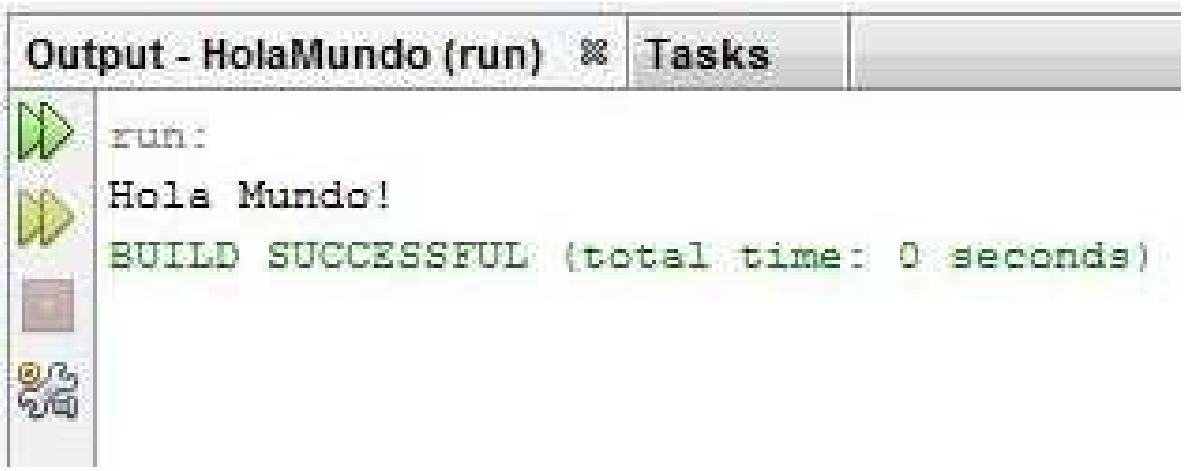


Figura 2.25. Resultado de la compilación.

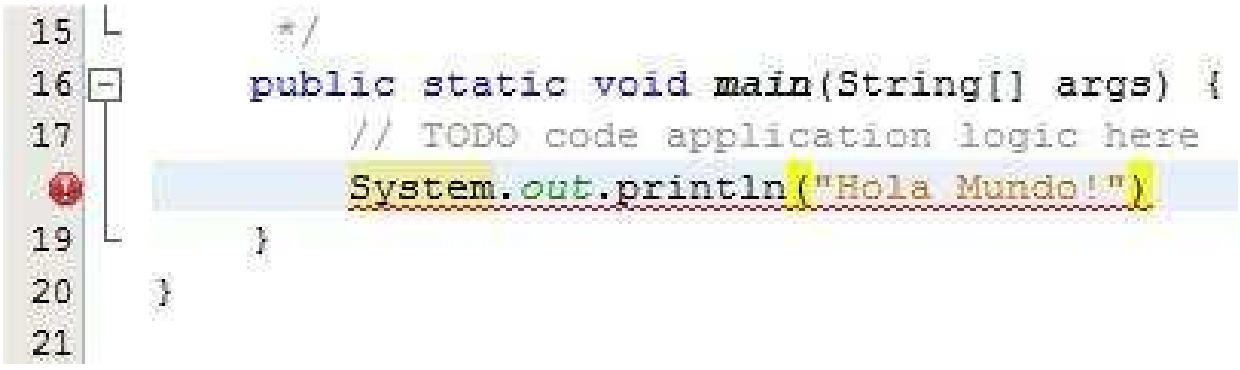


Figura 2.26. Línea de error de programación.

Si colocamos el cursor del ratón justo encima del símbolo de advertencia nos informará del error que ha encontrado:

Como vemos en la Figura 2.27, nos informa de que se esperaba un **;** y no lo ha encontrado.

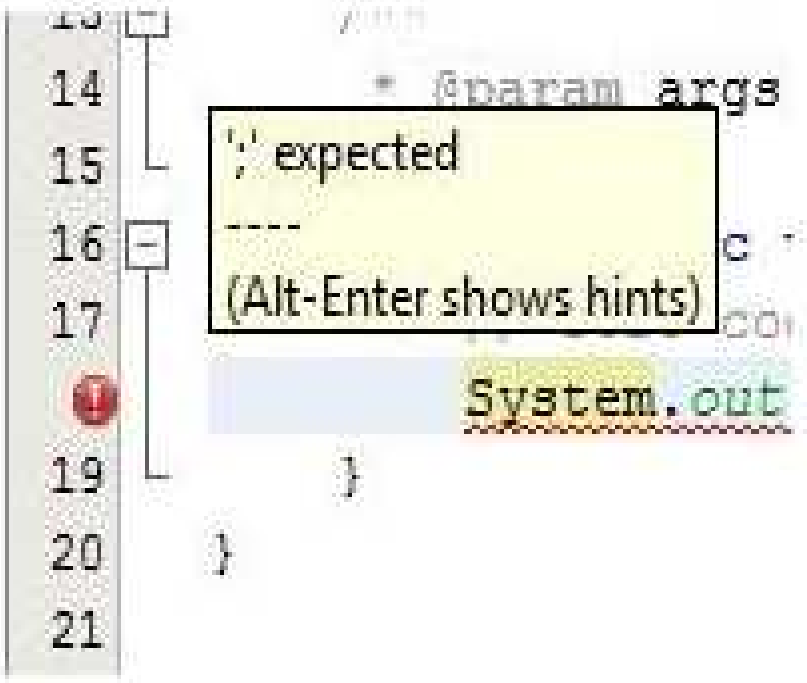


Figura 2.27. Informe de error encontrado.

El siguiente paso sería construir un fichero con extensión JAR que podamos distribuir a cualquier persona para que lo ejecute, ya lo veremos.

Vamos a escribir un par de programas básicos para irnos familiarizando con el lenguaje Java y el entorno de desarrollo Netbeans.

Vamos a realizar un pequeño programa que realice operaciones aritméticas. De modo que iniciaremos un proyecto nuevo que llamaremos AritmeticaFacil (Figura 2.28).

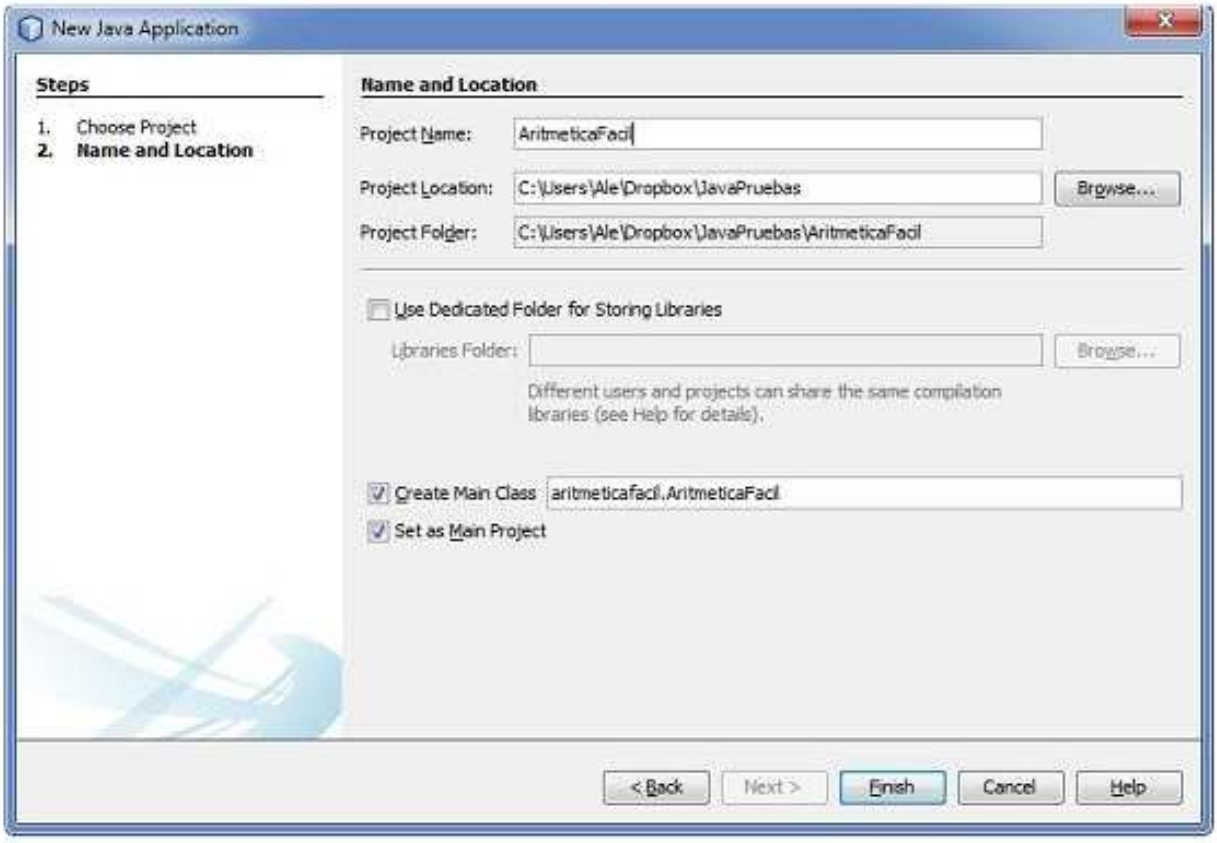


Figura 2.28. Fichero con extensión JAR.

Escribiremos todo el código dentro de main (Figura 2.29)

```
public static void main(String[] args) {
    // Este código realiza operaciones matemáticas simples
    int numero1, numero2, resultado;

    numero1=30;
    numero2=15;
```

Figura 2.29. Código dentro de main.

Como pueden ver se ha puesto un comentario debajo de main para explicar un poco qué es lo que hace el programa. Ahora puede parecer un poco infantil, pero a medida que los programas crecen, esto se hace más importante.

int numero1, numero2, resultado;

¿Qué significa esta línea? Pues es muy simple, le estamos diciendo a Java que nos tiene que hacer tres huecos en la memoria. Necesitamos tres huecos porque vamos a almacenar tres números: numero1, numero2 y resultado. La palabra int, viene de integer (entero). Esto quiere decir que el hueco que hemos reservado no será muy grande porque no nos va a hacer falta. En algunos libros dirán que esto sirve para almacenar el rango de números {-2147483648;+2147483647}, yo simplemente me limitaré a decir que el tipo de datos int nos va a servir para almacenar números no demasiado grandes y sin decimales

Para números grandes o con decimales utilizaremos otros tipos de datos y en vez de int utilizaremos otras palabras como float o string que ya analizaremos.

Esta operación se denomina declaración de variables, en casi todos los lenguajes de programación deben declararse al principio de cada clase o método.

Aunque en algunos se pueden ir declarando sobre la marcha, es considerado una muy mala costumbre de programación. Por ello, asumiremos que no se puede y siempre lo haremos al empezar.

Después nos encontramos con esto:

numero1=30;

numero2=15;

La operación que estamos haciendo aquí se llama asignación que básicamente consiste en decirle al ordenador que en número1 queremos almacenar el número 30 y en numero2 queremos almacenar el número 15.

Una vez realizada la asignación podemos utilizar numero1 y numero2 como si fueran Números tal cual. Y aquí es donde empezaremos a realizar las diferentes operaciones.

//Ahora los sumamos

resultado=numero1+numero2;

//Y mostramos el resultado de sumarlos

System.out.println(numero1+ " + " + numero2+" = " + resultado);

Este código realiza la suma y la muestra por pantalla. Analicemos las líneas, omitiré las líneas de comentario por los motivos obvios.

resultado=numero1+numero2;

Esto ya nos lo conocemos, es la operación de asignación, solo que en este caso en lugar de darle nosotros el número, estamos indicando que lo que queremos almacenar en resultado es la suma de numero1 y numero2. Tan simple como eso.

Después hacemos que salga por pantalla:

System.out.println(numero1 + " + " + numero2 + " = " + resultado);

Esto también nos lo conocemos, pero supongo que haya varias preguntas como por ejemplo: si antes el signo + sumaba, ¿por qué lo utilizamos aquí?.

A veces sucede que un mismo operador utilizado en partes concretas del programa realiza una función diferente, aunque parecida. En este caso lo que está haciendo es concatenar. Concatenar consiste en unir fragmentos de texto para que luego aparezcan todos juntos.

Fijémonos en un pequeño fragmento del argumento del método println:

numero1 + " + " + numero2

Si pusiéramos esto en el argumento del método, el resultado mostrado por pantalla sería el siguiente: *30 + 15,*

Primero escribe lo que hay guardado en la variable numero1 (fijense que la escribimos sin las comillas), después, ” + ” es un literal, es decir, un texto que queremos que aparezca tal cual.

Cuando queremos escribir un texto literal lo hacemos entre comillas, en este caso hay un espacio, el signo más y después otro espacio. Por último concatenamos la variable numero2 que contiene el número 15.

El resto del código ya podemos escribirlo, todos los bloques de código que tenemos que añadir son iguales, pero cambiaremos el operador + por – (resta), * (multiplicación) y / (división) (Figura 2.30).

Se puede probar cambiando los valores de las variables para comprobar que efectivamente funciona, únicamente hay que tener cuidado con los valores que se asignen porque si se utiliza decimales o números muy grandes, es posible que el programa genere errores.

Y el resultado al ejecutarlo sería como el que se muestra en la Figura 2.31.

```
public static void main(String[] args) {
    // Este código realiza operaciones matemáticas simples
    int numero1, numero2, resultado;

    numero1=30;
    numero2=15;

    //Ahora los sumamos
    resultado=numero1+numero2;
    //Y mostramos el resultado de sumarlos
    System.out.println(numero1+ " + " + numero2 + " = " + resultado);

    //Ahora la resta
    resultado=numero1-numero2;
    //Y mostramos el resultado de restarlos
    System.out.println(numero1 + " - " + numero2 + " = " + resultado);

    //Ahora la multiplicación
    resultado=numero1*numero2;
    //Y mostramos el resultado de la multiplicación
    System.out.println(numero1 + " * " + numero2 + " = " + resultado);

    //Y por último, la división
    resultado=numero1/numero2;
    //Y mostramos el resultado
    System.out.println(numero1 + " / " + numero2 + " = " + resultado);

    // Ya veremos cómo se puede pedir los datos al usuario
}
```

Figura 2.30. Resultado final.

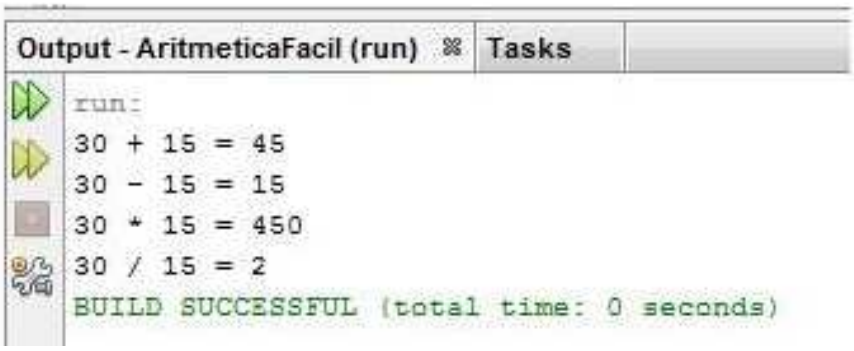


Figura 2.31. Ejecución del programa.

2.2.3 CREACIÓN DE METODOS

Vamos a comenzar un nuevo proyecto. Vamos a suponer que tenemos un parque de atracciones, tenemos varias atracciones y queremos controlarlas desde nuestro ordenador.

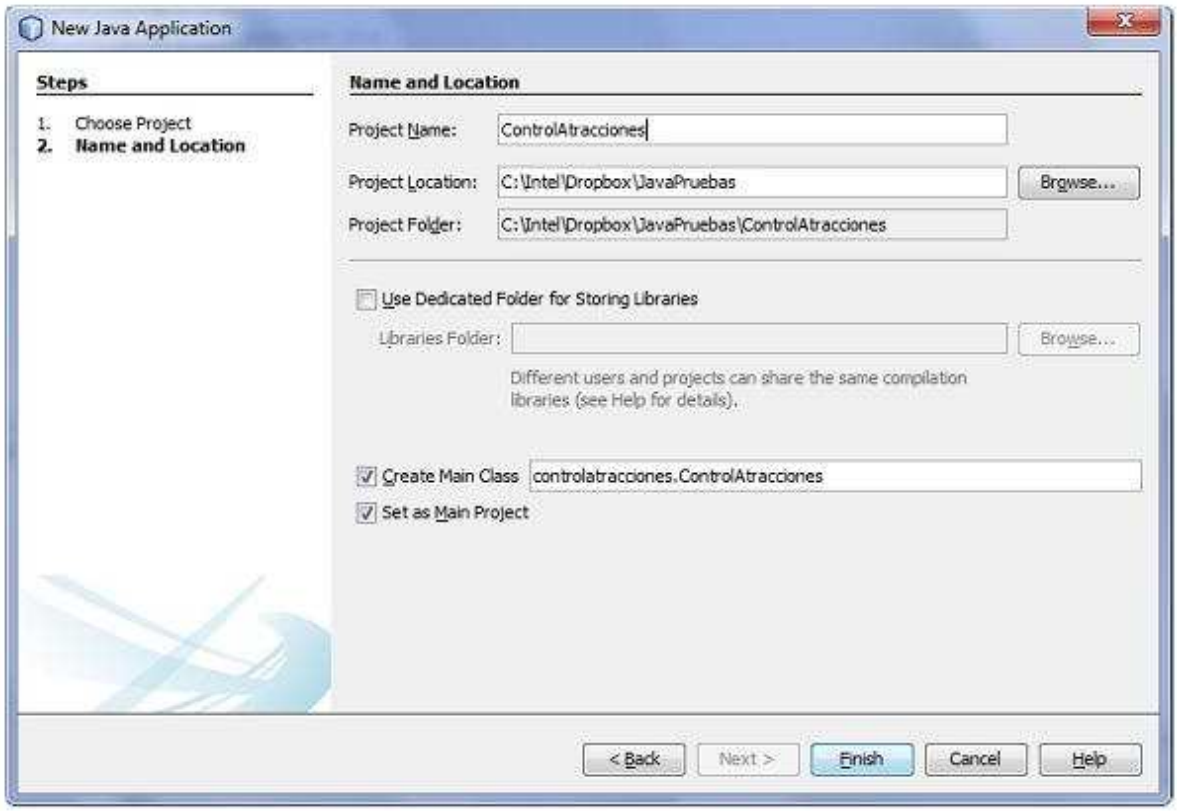


Figura 2.32. Características del proyecto.

Cada atracción será un objeto y tendremos que crear métodos para controlarlas. (Figura 2.32).

2.2.3.1 Definiendo las propiedades

Ya hemos comentado que cada atracción será un objeto, bien, pues cada atracción tiene unas características que nos interesa conocer. En nuestro caso vamos a controlar los siguientes parámetros:

- Nombre de la atracción
- Lugar donde está situada
- Encendida/Apagada
- Velocidad a la que está funcionando

También debemos definir qué cosas podemos hacer con las atracciones, en nuestro caso:

- Aumentar la velocidad
- Disminuir la velocidad
- Apagar o encender
- Consultar estado (ver un informe que indique si está encendida o apagada y su velocidad)

Todos estos puntos, es recomendable irlos definiendo en un papel y cuando tengamos claro que queremos hacer, lo llevamos al ordenador. El papel y el lápiz son herramientas importantes para un programador.

Bien, como hemos decidido que tenemos una serie de problemas que resolver y que tenemos objetos que están implicados, tenemos que crear una clase nueva. En este proyecto tendremos dos: la clase principal desde donde indicaremos las instrucciones y la clase que vamos a crear ahora en la que definiremos todas estas propiedades y métodos que hemos escrito en el papel.

NetBeans nos lo pone muy fácil, pulsamos con el botón derecho en el paquete controlatracciones, seleccionamos **New** y después **Java Class** (Figura 2.33).

En la pantalla que aparece escribiremos el nombre Atraccion como nombre de clase como en la Figura 2.34.

Ahora tendremos dos archivos con extensión .java en el proyecto: Atraccion.java y ControlAtracciones.java. En este momento deberemos tener en la pantalla, el archivo Atraccion.java abierto como en la Figura 2.35.

Debajo de **public class Atraccion** {, será donde escribamos las propiedades y métodos. Comenzaremos con las propiedades (Figura 2.36).

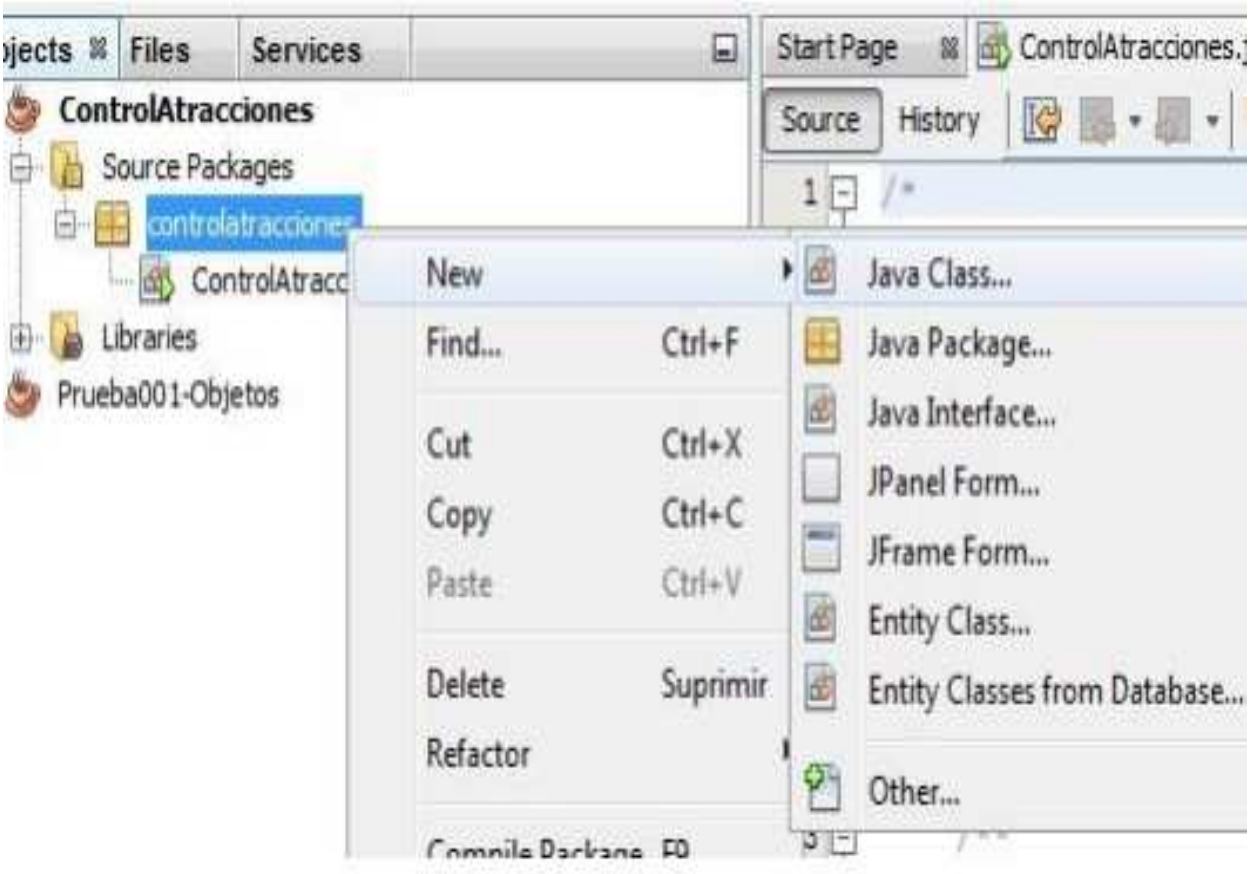


Figura 2.33. Creación de una Clase Java.

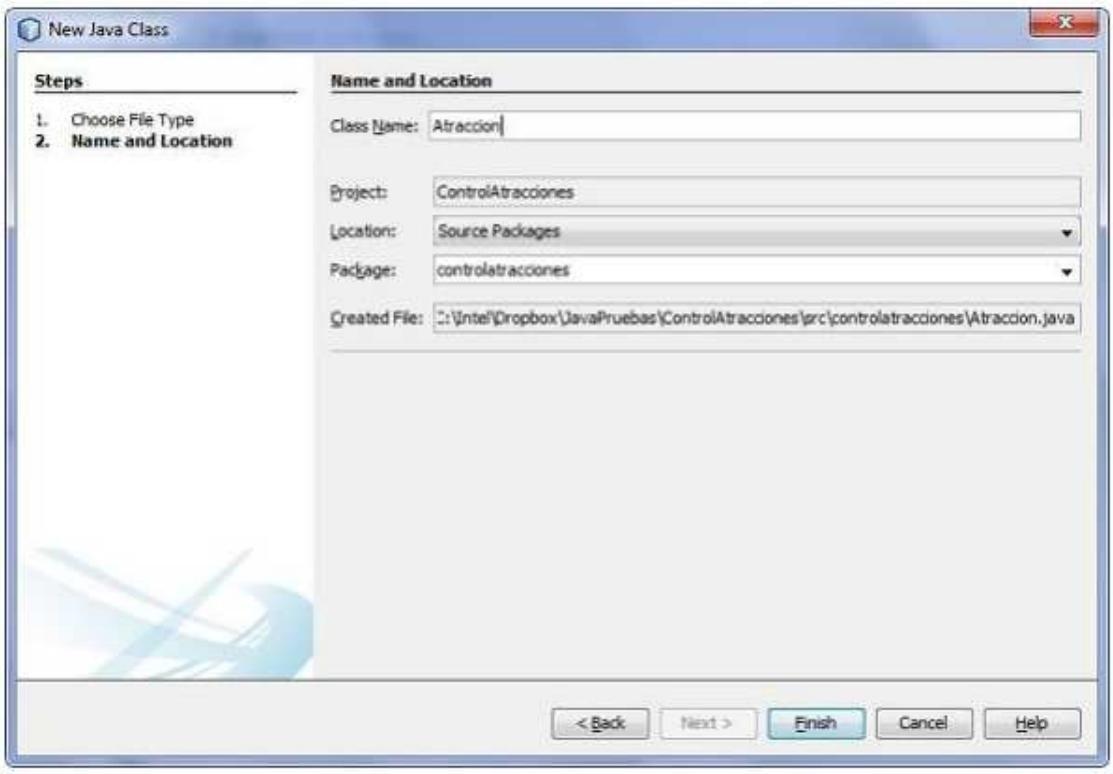


Figura 2.34. Definir el nombre de la clase.

Figura 2.35. Archivo Atraccion.java abierto.

```
public class Atraccion {

    // Las propiedades de cada atracción
    String nombre;
    String lugar;
    Boolean encendido;
    int velocidad;
```

Figura 2.36. Escribimos las propiedades y métodos.

2.2.3.2 Definición de los métodos

Debajo de las propiedades vamos a comenzar a añadir los métodos que podremos aplicar al objeto. Una de las primeras cosas que tendremos que hacer cuando lleguemos al parque todas las mañanas será encender las atracciones, de modo que crearemos un método que lo haga (Figura 2.37).
Vayamos por partes porque hay varias cosas nuevas.

void encender()

Encender es el nombre del método, así de simple. Los paréntesis vacíos indican que el método no recibe ningún dato con el que trabajar porque no le hace falta. Más adelante veremos ejemplos en los que sí le hacen falta.

Void es una palabra especial y reservada del lenguaje java. Este método sólo ejecuta una acción, no trabaja con datos. Al igual que en el párrafo anterior comentábamos que el método no necesita ningún dato para funcionar, tampoco devuelve un dato al final. Si devolviera algún dato, tendríamos que sustituir void por int, string, boolean o el tipo de datos que correspondiera.

Después tenemos una estructura condicional: if

if (encendido==true)

Que literalmente se traduce por “Si encendido vale true…”. Lo que haya debajo de esta línea se ejecutará sólo en el caso de que la condición se cumpla.

Es importante reparar en la pequeña diferencia que supone el doble igual. Recordemos en el ejemplo anterior cuando utilizábamos numero1=30; en este caso había un símbolo =, pero ahora hay dos. Cuando hay dos, lo que hace Java es comparar si lo que hay a la izquierda es igual que lo que hay a la derecha.

Un símbolo = -> Asignación

System.out.println("Esta atracción ya está funcionando.");

Esto ya sabemos lo que hace. Hemos comprobado si la atracción está encendida, en caso de que sea cierto, el programa nos dice que ya está funcionando.

else

{

encendido=true;

System.out.println("La atracción " + nombre + " se ha

encendido.");

}

Esta es la segunda parte de la estructura condicional. Ya hemos visto lo que hace el programa en caso de que ya estuviera encendido, pero si no lo está, Java ejecutará el código que haya en ELSE (sino).

Es importante fijarse en otro detalle. En la parte del if no hemos utilizado las llaves para “encerrar” el código, aquí en cambio sí. Esto es porque debajo de if sólo había una instrucción pero else tiene dos. Al haber dos instrucciones tenemos que utilizar las llaves obligatoriamente.

Si hemos llegado al else (sino), resulta que es porque la atracción no estaba funcionando, de modo que el programa la enciende:

encendido=true;

Y muestra un mensaje que incluye su nombre indicando que se ha encendido:

System.out.println("La atracción " + nombre + " se ha encendido.");

```
11  public class Atraccion {
12
13      // Las propiedades de cada atracción
14      String nombre;
15      String lugar;
16      Boolean encendido;
17      int velocidad;
18
19      //Y ahora los métodos
20
21      void encender () {
22          if (encendido==true)
23              System.out.println("Esta atracción ya está funcionando.");
24          else
25          {
26              encendido=true;
27              System.out.println("La atracción " + nombre + " se ha encendido.");
28          }
29      }
```

Figura 2.37. Creación de métodos

2.2.3.3 Apagando las atracciones

Ahora que has visto cómo se enciende la atracción, intentaremos escribir el método que permita apagar la atracción. Se puede hacer de varias formas, no hay un método único, el código que he escrito yo es el que se muestra en la Figura 2.38. Puedes dedicarte a copiarlo, pero yo te aconsejo que intentes hacerlo por tu cuenta, si ves

que no te sale míralo y estúdialo para tener una pista y después vuelve a intentarlo. Programar requiere pensar y es un ejercicio que requiere práctica.

```
21      void encender () {
22          if (encendido==true)
23              System.out.println("Esta atracción ya está funcionando.");
24          else
25          {
26              encendido=true;
27              System.out.println("La atracción " + nombre + " se ha encendido.");
28          }
29      }
30
31      void apagar () {
32          if (encendido==false)
33              System.out.println("Esta atracción ya está apagada.");
34          else
35          {
36              encendido=false;
37              System.out.println("La atracción " + nombre + " se ha apagado.");
38          }
39      }
```

Figura 2.38. Método que permite apagar la atracción.

Aun siendo un código todavía muy sencillo, es probable que haya mejores formas de resolverlo, pero ésta es funcional.

2.2.3.4 El cambio de velocidad

A estas alturas ya deberíamos ser capaces de saber, o al menos sospechar, cómo cambiar la velocidad de las atracciones.

El código de la Figura 2.39. muestra básicamente, la velocidad que tenía la atracción y su nombre, después aumenta o disminuye la velocidad y por último muestra en pantalla la velocidad actualizada.

```
41      void masrapido() {
42          System.out.println("Velocidad anterior de " + nombre + ": " + velocidad);
43          velocidad=velocidad+1;
44          System.out.println("Velocidad actual de " + nombre + ": " + velocidad);
45      }
46
47      void maslento() {
48          System.out.println("Velocidad anterior de " + nombre + ": " + velocidad);
49          velocidad=velocidad-1;
50          System.out.println("Velocidad actual de " + nombre + ": " + velocidad);
51      }
```

Figura 2.39. Código de cambio de velocidad.

2.2.3.5 Método Estado

Por último, decíamos que necesitábamos un método que nos diera información sobre el estado de la atracción. Simplemente se trata de mostrar las variables por pantalla para que las vea el usuario como lo muestra la Figura 2.40.

2.2.3.6 Volvemos al control de atracciones

Ahora que ya hemos terminado de definir las propiedades y los métodos correspondientes, sólo queda crear nuestro programa principal.

Volvemos al archivo ControlAtracciones haciendo clic en la pestaña correspondiente de la parte superior del panel tal y como lo indica la Figura 2.41.

Ahora tenemos que crear tantos objetos como atracciones tengamos en el parque. Como la clase ya está creada, sólo tendremos que decirle a Java que queremos uno nuevo.

Escribiremos lo siguiente dentro del método main:

Atraccion ExprimeCerebros=new Atraccion();

Atraccion MachacaEstomagos=new Atraccion();

Estas dos sentencias han creado dos objetos de tipo Atraccion: el ExprimeCerebros y el MachacaEstomagos.

Ahora podemos pasar a definir las propiedades:

// Establecemos las propiedades

ExprimeCerebros.nombre="ExprimeCerebros";

ExprimeCerebros.lugar="Sector oeste, plaza 21";

ExprimeCerebros.encendido=false;

ExprimeCerebros.velocidad=5;

MachacaEstomagos.nombre="MachacaEstomagos";

MachacaEstomagos.lugar="Sector este, plaza 16";

MachacaEstomagos.encendido=false;

MachacaEstomagos.velocidad=3;

Y vamos a realizar las siguientes acciones:

// Ahora mostramos la información del ExprimeCerebros

// Subiremos la velocidad, encenderemos la atracción

// y por último volveremos a mostrar su estado

ExprimeCerebros.estado();

ExprimeCerebros.masrapido();

ExprimeCerebros.encender();

ExprimeCerebros.estado();

```
53      void estado() {
54          System.out.println("Atracción: " + nombre);
55          System.out.println("Situada en: " + lugar);
56          if (encendido==true)
57              System.out.println("Está encendida");
58          else
59              System.out.println("La atracción está apagada");
60          System.out.println("Su velocidad es: " + velocidad);
61      }
62  }
63
```

Figura 2.40. Código que muestra el estado de las atracciones.

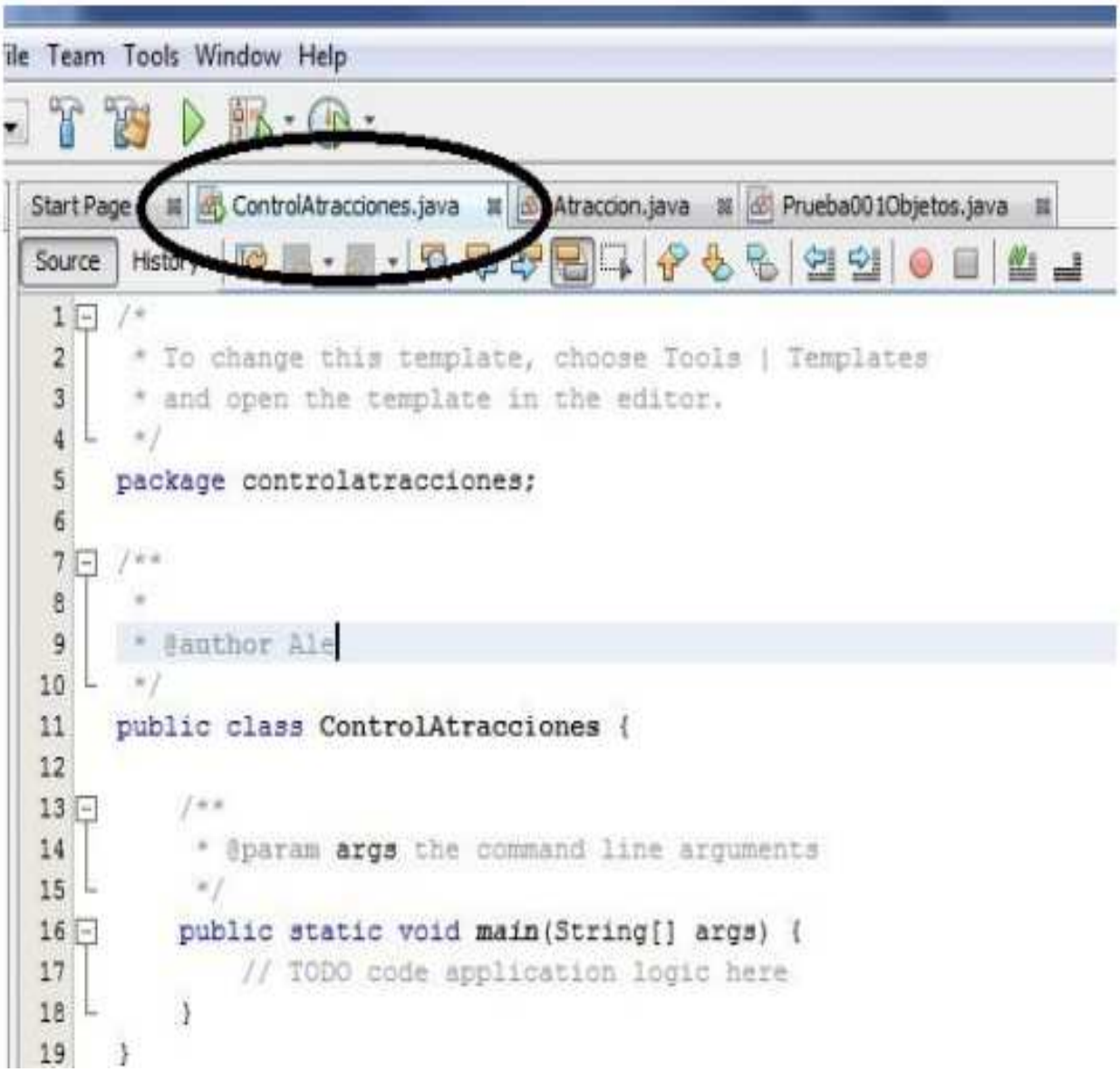


Figura 2.41. Volvemos al archivo ControlAtracciones.

Como se puede ver, una vez que se ha creado la clase, añadir los objetos y trabajar con ellos es muy sencillo. De otra manera, cada vez que quisiéramos ejecutar, por ejemplo, estado(), tendríamos que haber escrito de nuevo todo el código anterior.

Todo junto, quedará algo parecido a como se muestra en la Figura 2.42.

```
16      public static void main(String[] args) {
17          // Creamos los objetos
18          Atraccion ExprimeCerebros=new Atraccion();
19          Atraccion MachacaEstomagos=new Atraccion();
20
21          // Establecemos las propiedades
22          ExprimeCerebros.nombre="ExprimeCerebros";
23          ExprimeCerebros.lugar="Sector oeste, plaza 21";
24          ExprimeCerebros.encendido=false;
25          ExprimeCerebros.velocidad=5;
26
27          MachacaEstomagos.nombre="MachacaEstomagos";
28          MachacaEstomagos.lugar="Sector este, plaza 16";
29          MachacaEstomagos.encendido=false;
30          MachacaEstomagos.velocidad=3;
31
32          // Ahora mostramos la información del ExprimeCerebros
33          // Subiremos la velocidad, encenderemos la atracción
34          // y por último volveremos a mostrar su estado
35          ExprimeCerebros.estado();
36          ExprimeCerebros.masrapido();
37          ExprimeCerebros.encender();
38          ExprimeCerebros.estado();
39
40          //Podemos hacer lo mismo con MachacaEstómagos
41          //El código es exactamente igual, sólo
42          //cambia el nombre del objeto
43      }
44  }
```

Figura 2.42. Código completo.

Y si ejecutamos el código la salida debería ser como la Figura 2.43.

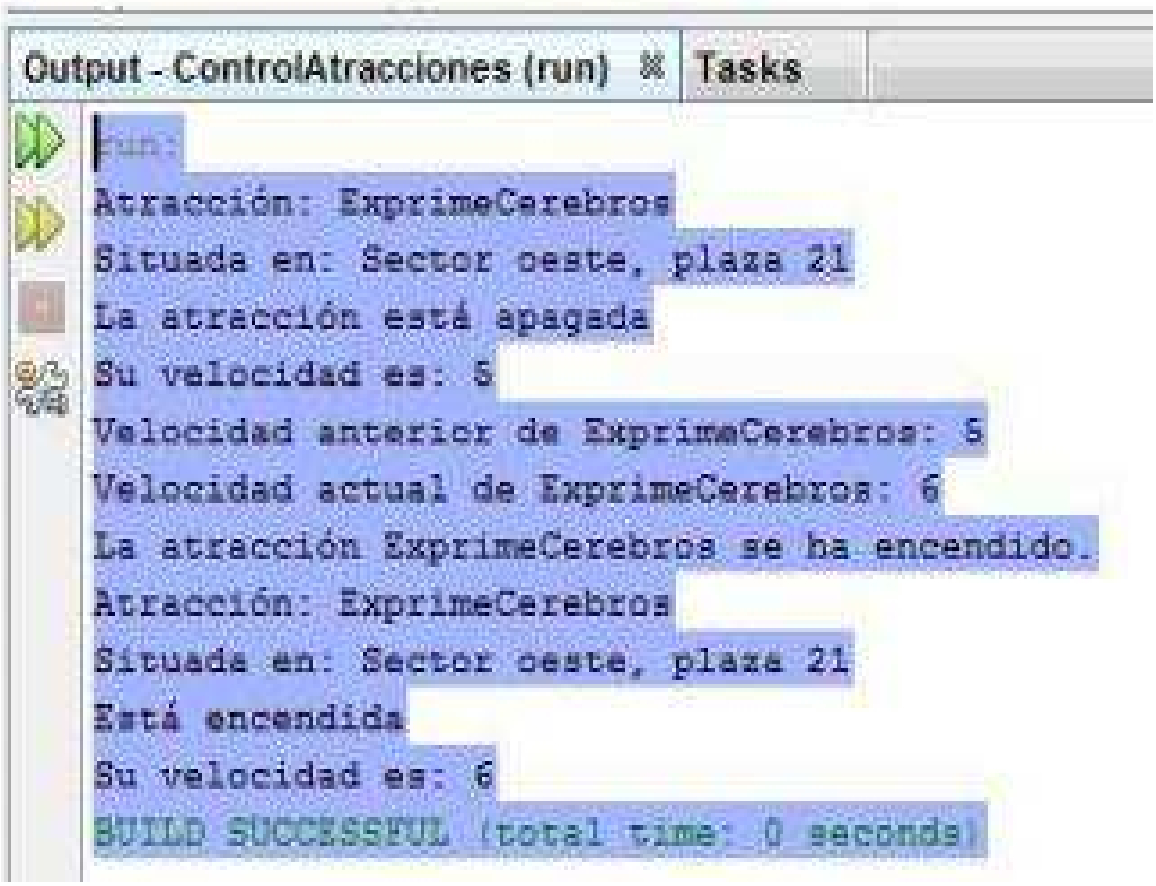


Figura 2.43. Ejecución del programa completo.

La aplicación es funcional, pero bastante mejorable. Por ejemplo, toda la información aparece junta en el mismo sitio. Podríamos hacer que cada vez que termine un método, el programa haga un salto de línea para que la información no aparezca tan seguida.

Sólo tenemos que modificar los métodos de la clase Atracción para que esto ocurra.

Añadiremos la siguiente línea al final de las instrucciones de cada método:

System.out.println("\n");

El código \n lo que hace es insertar un salto de línea. El resultado al ejecutar la aplicación ahora el que tenemos en la Figura 2.44

.

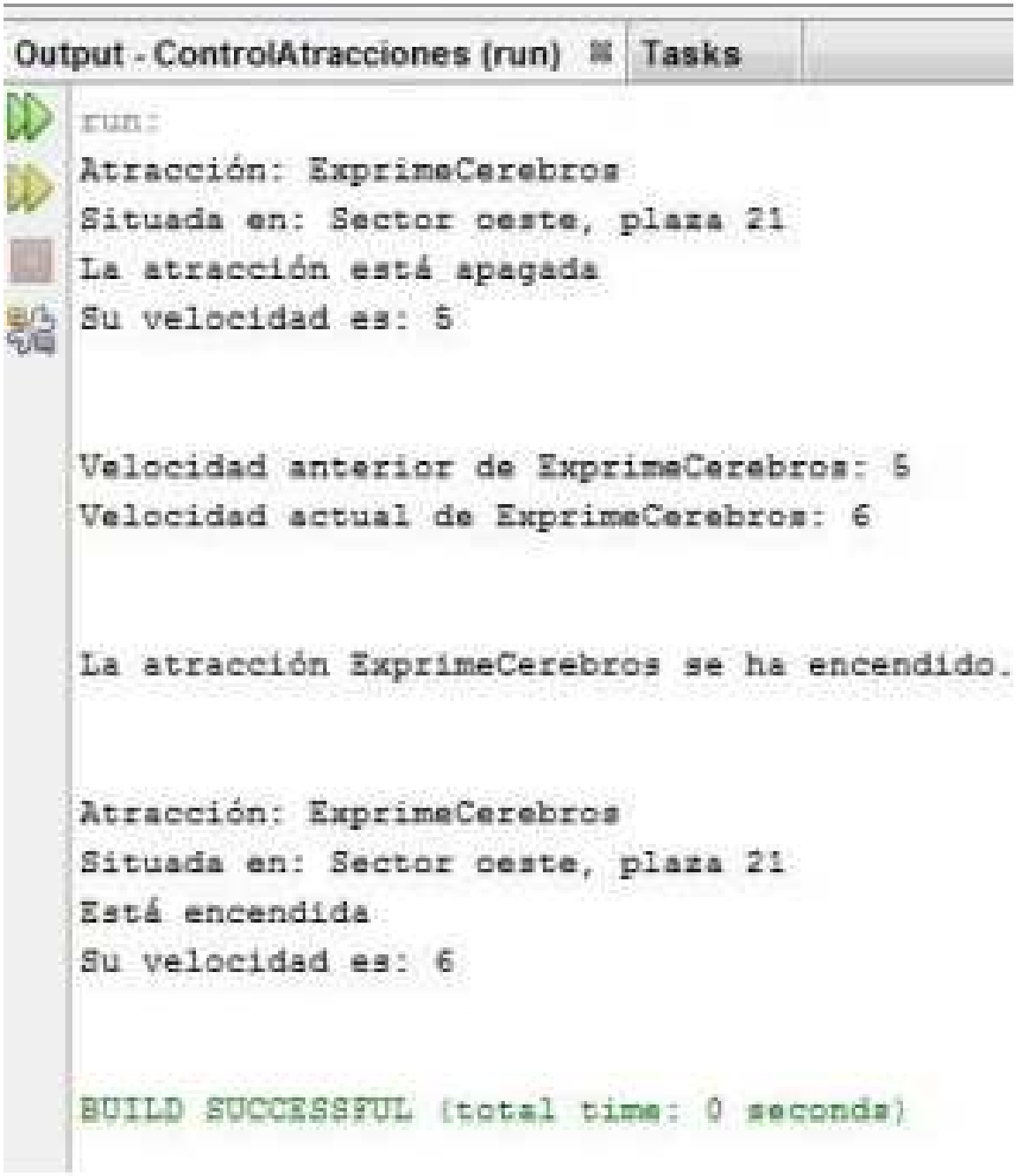


Figura 2.44. Insertar un salto de línea.

3 CAPITULO III: CONSTRUCCIÓN DEL SISTEMA

3.1 ETAPAS DEL SISTEMA

3.1.1 DESCRIPCIÓN DEL SISTEMA

El sistema en general consta de 6 etapas las cuales en conjunto permiten controlar el encendido y apagado de las luminarias de la maqueta de una vivienda y monitorear el acceso a la misma a través de puertas y ventanas utilizando un microcontrolador ATMEGA 16 por medio del puerto serial del PC, permitiendo al usuario controlar el encendido de la luces de la vivienda, a la vez que se puede detectar el ingreso no autorizado en los horarios previamente establecidos por el usuario.

A continuación se mencionan las diferentes etapas que conforman el sistema según el orden de ubicación en el sistema:

1. Etapa de fuente
2. Etapa de monitoreo
3. Etapa de control de luces y sirena
4. Etapa de monitoreo de acceso
5. Etapa de microcontrolador
6. Etapa de comunicación
7. Etapa de software

Para el correcto manejo del sistema se recomienda leer el manual del usuario adjunto, y seguir sus instrucciones paso a paso e instalar los programas necesarios para el correcto funcionamiento del mismo.

3.1.2 DIAGRAMA GENERAL DE ETAPAS DEL SISTEMA

En la Figura 3.1 se muestra el diagrama general de etapas del sistema que son necesarias para el funcionamiento del mismo.

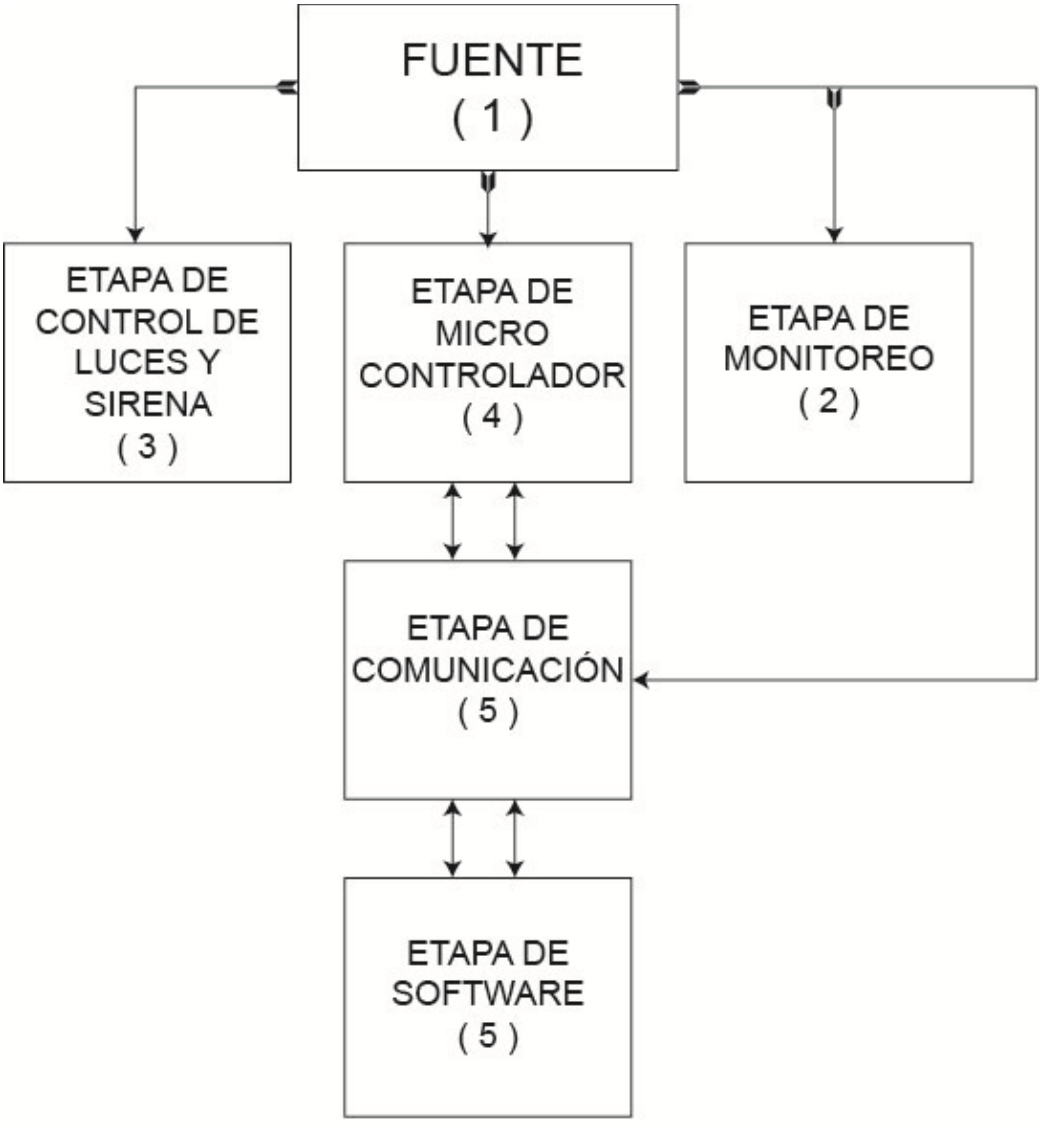


Figura 3.1. Diagrama general de etapas del sistema

3.2 ETAPA DE FUENTE

La fuente de poder del circuito consta de un transformador que es alimentado con 110 V/60 Hz a la salida del transformador tenemos 14 V / 2000 mA los cuales ingresan a un puente de diodos de 2A, que rectifica la señal, esta señal es filtrada mediante dos capacitores electrolíticos de 4700 uF y de 10 uF y otro cerámico de 100nF luego pasa a un regulador de voltaje de 12 V (7812) esta parte de la fuente sirve para alimentar los relés de 12 V, la otra etapa de la fuente consta de un capacitor electrolítico de 100 uF seguido de un regulador de voltaje de 5 V (7805) seguido de dos capacitores de 100nF y 100uF, esta parte de la fuente sirve para alimentar los circuitos integrados TTL.

Como podemos apreciar en la Figura 3.2 se muestra el diagrama de la fuente con sus respectivos componentes.

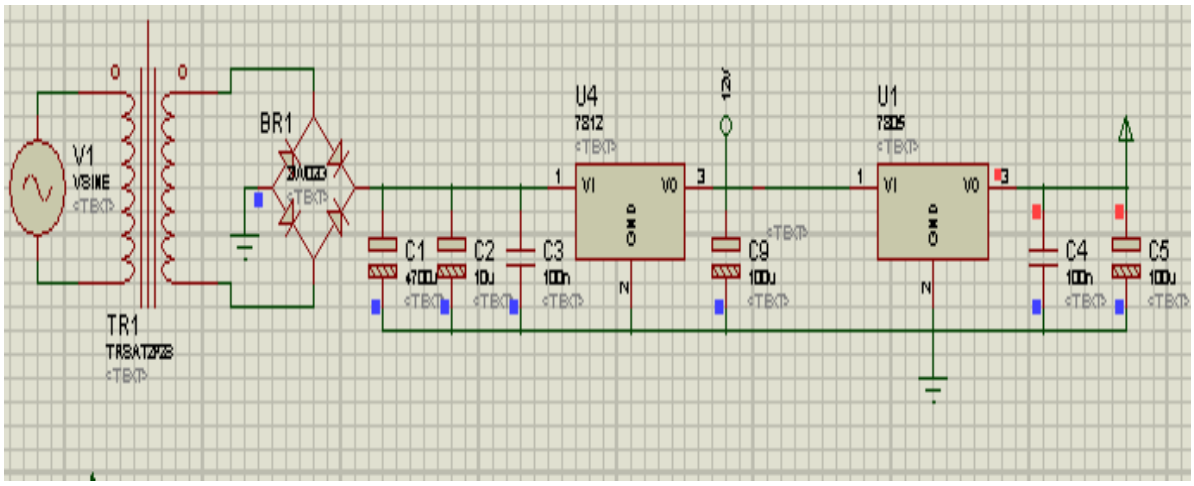


Figura 3.2. Diagrama de la etapa de fuente.

3.3 ETAPA DE MONITOREO

Esta etapa sirve para monitorear la acción que se encuentra realizando el microcontrolador y nos mostrará los mensajes en la pantalla LCD como se aprecia en la Figura 3.3.

Las entradas I nos muestran el estado de los sensores magnéticos si está en 1 significa que la puerta o ventana a la cual está conectada el sensor se encuentra abierta y si está en 0 significa que está cerrada.

Las salidas O indican el estado de las luminarias si están en 0 quiere decir que la luminaria a la cual está conectada se encuentra apagada y cuando está en 1 significa que la luminaria está encendida.

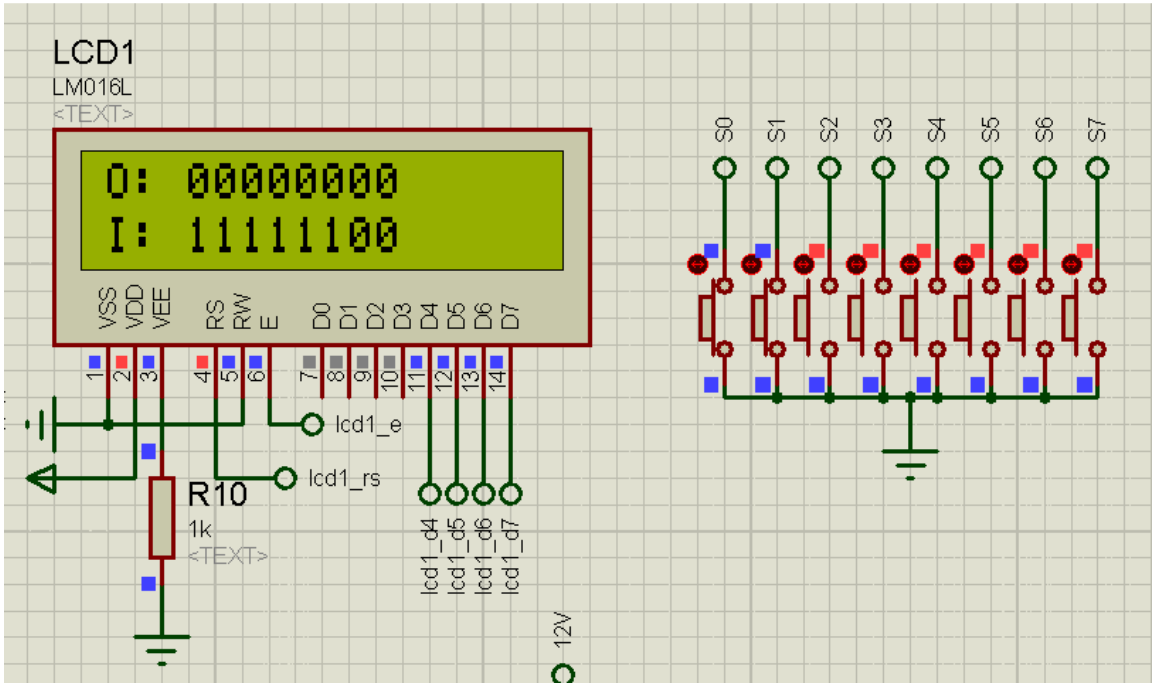


Figura 3.3. Diagrama de la etapa de monitoreo.

3.4 ETAPA DE CONTROL DE LUCES Y SIRENA

Esta etapa sirve para controlar el encendido y apagado de las luces de la vivienda para lo cual se utilizaron los 8 pines del puerto A del microcontrolador cada uno controla un foco de la casa y va conectado a la entrada del CI ULN 2803 (Tabla 3.1) dentro del cual se encuentran 8 transistores NPN Darlington (Figura 3.4).

El circuito ULN 2803 es ideal para ser empleado como interfaz entre las salidas de un PIC o cualquier integrante de las familias TTL o CMOS y dispositivos que necesitan una corriente más elevada para funcionar, todas sus salidas son a colector abierto y se dispone de un diodo para evitar las corrientes inversas.

En la Figura 3.5 podemos observar el diagrama circuital individual de este circuito integrado.

Tabla 3.1. Características del ULN 2803

CARACTERISTICAS DEL ULN 2803			
Voltaje de entrada	Voltaje CE (Max)	Corriente C (Max)	Temperatura de operación
5 V	50 V	500 mA	0° C a 70° C

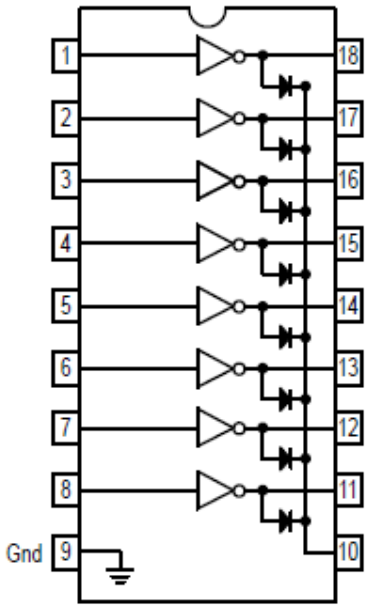


Figura 3.4. Diagrama del ULN 2803.

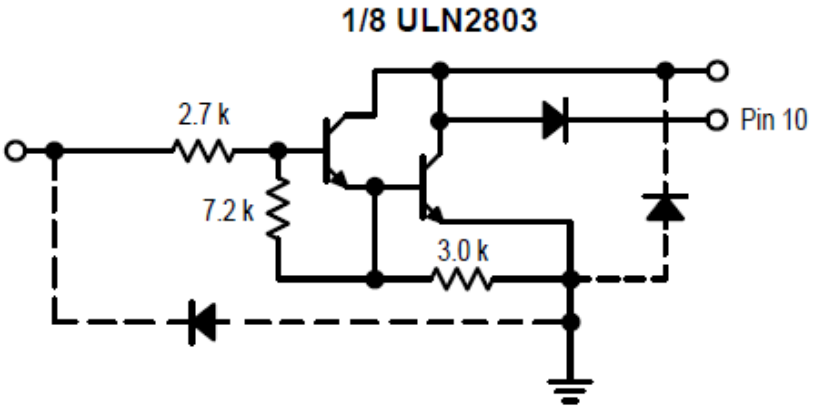


Figura 3.5. Diagrama circuital del ULN 2803.

Como podemos apreciar en la Figura 3.6, el pin 9 del ULN 2803 va conectado a GND mientras que el pin 10 (COMMON) permite el acceso a los diodos incluidos en el chip, cuya tarea es proteger los transistores del mismo frente de picos de sobretensión generados por cargas de tipo inductivo, como motores o bobinas lo conectamos a 12 V DC.

Cada salida de los ULN 2803 está conectada a un relé que controla el encendido y apagado de los focos.

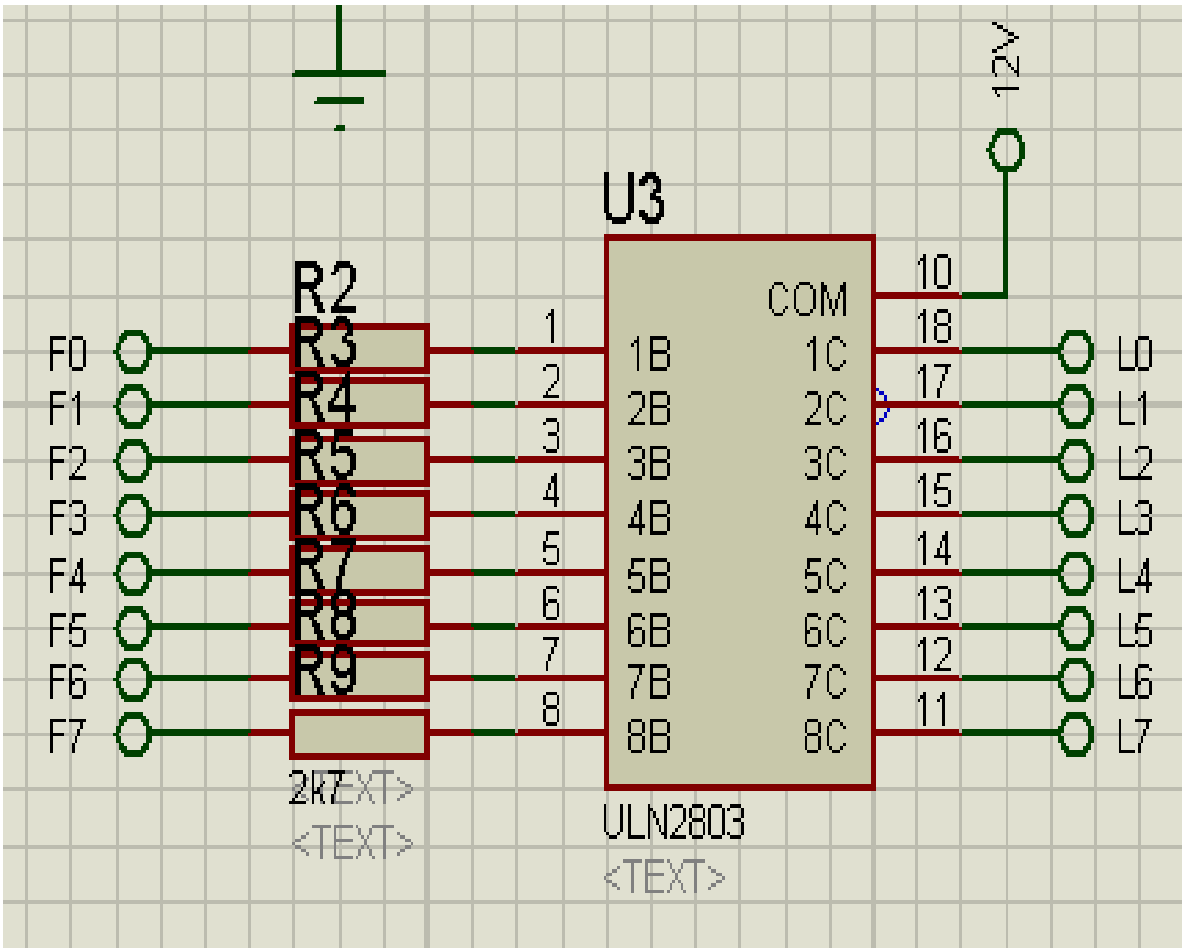


Figura 3.6. Diagrama conexión del ULN 2003 con los focos y sirena.

La conexión de los pines del microcontrolador ATMEGA 16 y del ULN 2803 la podemos observar en la tabla 3.2.

Tabla 3.2. Conexión de los pines del microcontrolador ATMEGA 16 y del ULN 2803

PIN MICROCONTROLADOR	**PIN ULN 2803**	**PIN ULN 2803**	**CONTROL**
Pin N° 40 (PORTA.0)	Pin N° 1 (IN)	Pin N° 18 (OUT)	FOCO ESTUDIO
Pin N° 39 (PORTA.1)	Pin N° 2 (IN)	Pin N° 17 (OUT)	FOCO SALA
Pin N° 38 (PORTA.2)	Pin N° 3 (IN)	Pin N° 16 (OUT)	FOCO DORMITORIO 1
Pin N° 37 (PORTA.3)	Pin N° 4 (IN)	Pin N° 15 (OUT)	FOCO SALA DE ESTAR
Pin N° 36 (PORTA.4)	Pin N° 5 (IN)	Pin N° 14 (OUT)	FOCO COCINA Y COMEDOR
Pin N° 35 (PORTA.5)	Pin N° 6 (IN)	Pin N° 13 (OUT)	FOCO DOMITORIO 2
Pin N° 34 (PORTA.6)	Pin N° 7 (IN)	Pin N° 12 (OUT)	FOCO BAÑO
Pin N° 33 (PORTA.7)	Pin N° 8 (IN)	Pin N° 11 (OUT)	SIRENA

El diagrama completo de la etapa de control lo observamos en la Figura 3.7.

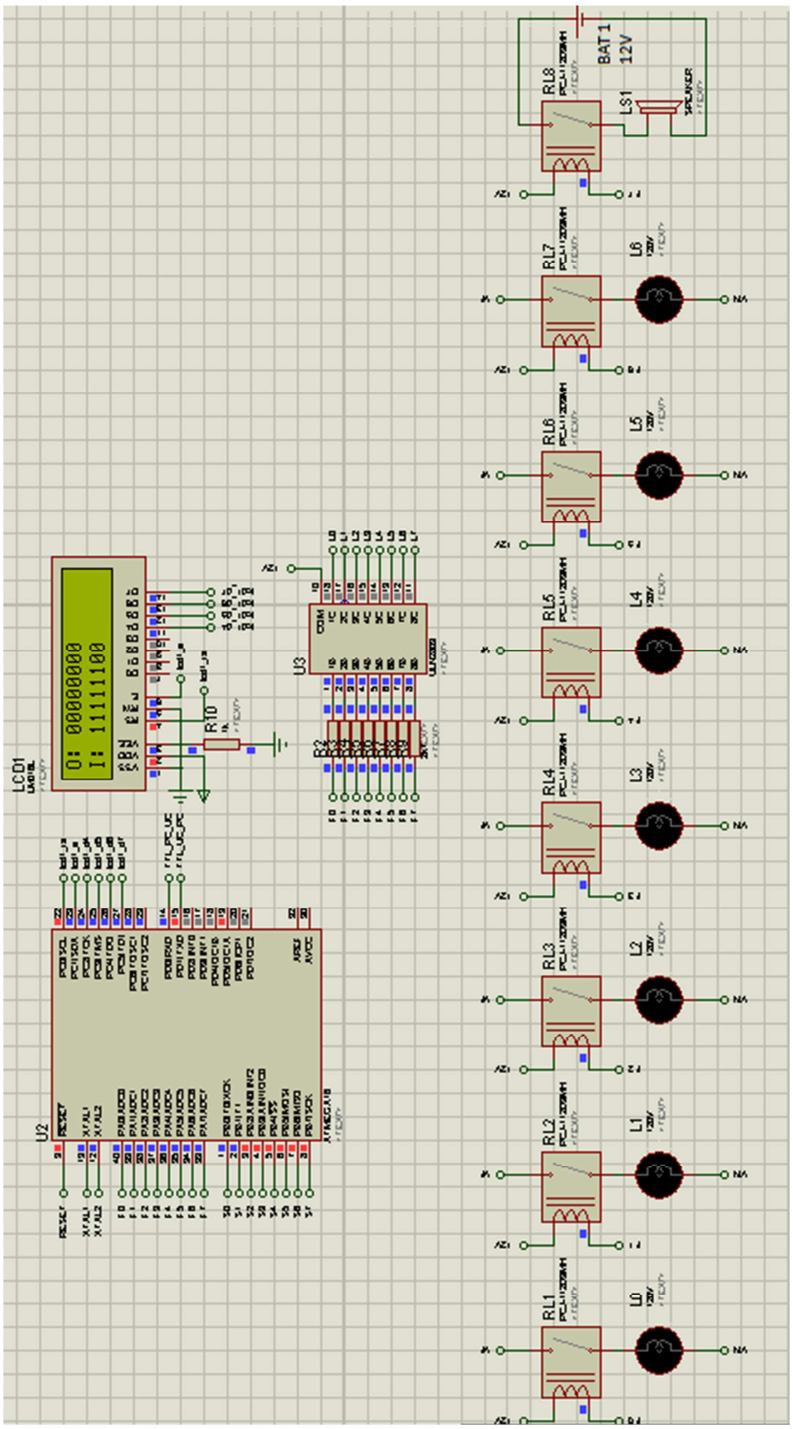


Figura 3.7. Diagrama de la etapa de control.

3.5 ETAPA DE MONITOREO DE ACCESO

Esta etapa sirve para monitorear el acceso a la vivienda a través de sensores magnéticos colocados en las puertas y ventanas de las mismas, para este propósito se usaron los 8 pines del puerto B del microcontrolador ATMEGA 16 como se muestra en la Tabla 3.3, cada PIN va conectado a un sensor magnético que funciona a 5 V y sirve como un interruptor, cuando están juntos los dos polos del sensor se cierra el circuito ON y cuando se abre una puerta o ventana de la casa se abre el circuito OFF.

Un terminal del sensor magnético está conectado a GND mientras que el otro va conectado a VCC con una resistencia de pull up de 5.6 k para asegurar que la señal no quede en un estado flotante, ya que en algunos tipos de dispositivos lógicos sino se pusieran las resistencias pull up el estado lógico 1 podría quedar con un valor de tensión intermedio entre 0 y 1.

Tabla 3.3. Conexión de los pines del microcontrolador ATMEGA 16 con los sensores.

PIN MICROCONTROLADOR	MONITOREO DE SENSORES
Pin N° 1 (PORTB.0)	PUERTA PRINCIPAL
Pin N° 2 (PORTB.1)	PUERTA TRASERA
Pin N° 3 (PORTB.2)	VENTANA ESTUDIO
Pin N° 4 (PORTB.3)	VENTANA SALA
Pin N° 5 (PORTB.4)	VENTANA DORMITORIO 1
Pin N° 6 (PORTB.5)	VENTANA COCINA
Pin N° 7 (PORTB.6)	VENTANA COMEDOR
Pin N° 8 (PORTB.7)	VENTANA DORMITORIO 2

En la figura 3.8 podemos apreciar el diagrama circuital de la etapa de monitoreo de acceso a la vivienda

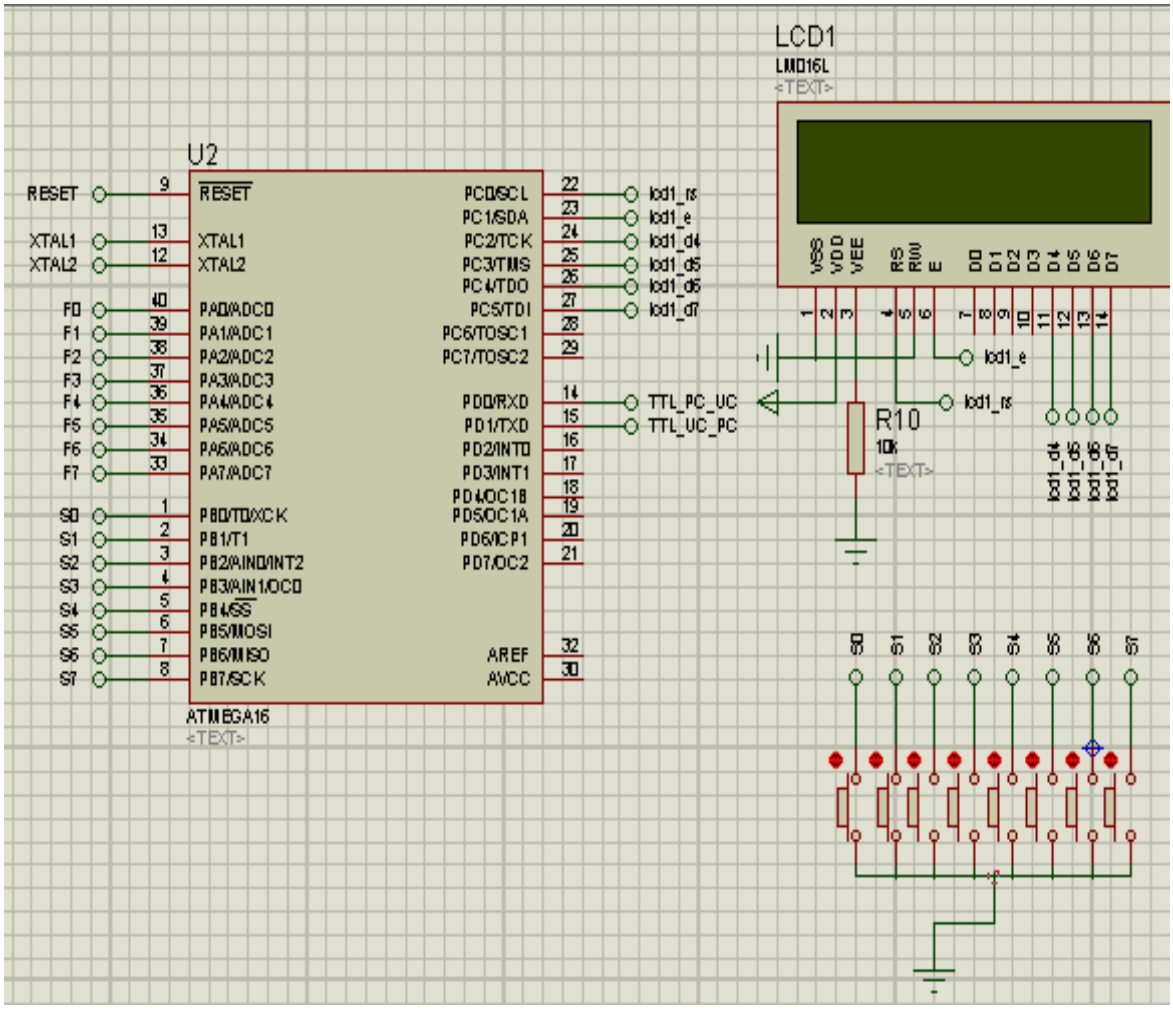


Figura 3.8. Etapa de monitoreo de acceso a la vivienda.

Si se abre una puerta o ventana en el horario programado previamente con el programa gestión casa que lo explicaremos en el manual del usuario, se activará una alarma que accionará una sirena la cual nos alerta de la presencia de un intruso en la vivienda.

3.6 ETAPA DE MICROCONTROLADOR

En la Figura 3.9 se describe la conexión de los pines del microcontrolador ATMEGA 16.

DESCRIPCION	NOMBRE	N°		N°	NOMBRE	DESCRIPCION
PUERTA PRINCIPAL	PORTB.0	1	ATMEGA 16	40	PORTA.0	FOCO ESTUDIO
PUERTA TRASERA	PORTB.1	2		39	PORTA.1	FOCO SALA
VENTANA ESTUDIO	PORTB.2	3		38	PORTA.2	FOCO DORMITORIO 1
VENTANA SALA	PORTB.3	4		37	PORTA.3	FOCO SALA DE ESTAR
VENTANA DORMITORIO 1	PORTB.4	5		36	PORTA.4	FOCO COCINA Y COMEDOR
VENTANA COCINA	PORTB.5	6		35	PORTA.5	FOCO DOMITORIO 2
VENTANA COMEDOR	PORTB.6	7		34	PORTA.6	FOCO BAÑO
VENTANA DORMITORIO 2	PORTB.7	8		33	PORTA.7	SIRENA
RESET	RESET	9		32	AREF	
5 V	VCC	10		31	GND	
0 V	GND	11		30	AVCC	
XTAL 2	XTAL2	12		29	PORTC.7	
XTAL 1	XTAL1	13		28	PORTC.6	
RECEPCION	PORTD.0	14		27	PORTC.5	RB7 (LCD)
TRANSMISION	PORTD.1	15		26	PORTC.4	RB6 (LCD)
	PORTD.2	16		25	PORTC.3	RB5 (LCD)
	PORTD.3	17		24	PORTC.2	RB4 (LCD)
	PORTD.4	18		23	PORTC.1	ENABLE (LCD)
	PORTD.5	19		22	PORTC.0	RESET (LCD)
	PORTD.6	20		21	PORTD.7	

Figura 3.9. Conexión del microcontrolador ATMEGA 16

En la figura 3.10. se muestra el diagrama de descripción de pines del conexionado del microcontrolador ATMEGA 16.

- El puerto A se utilizó como salidas a los focos.
- En el puerto B se conectaron los sensores magnéticos.
- El puerto C se utilizó para la conexión del Display LCD

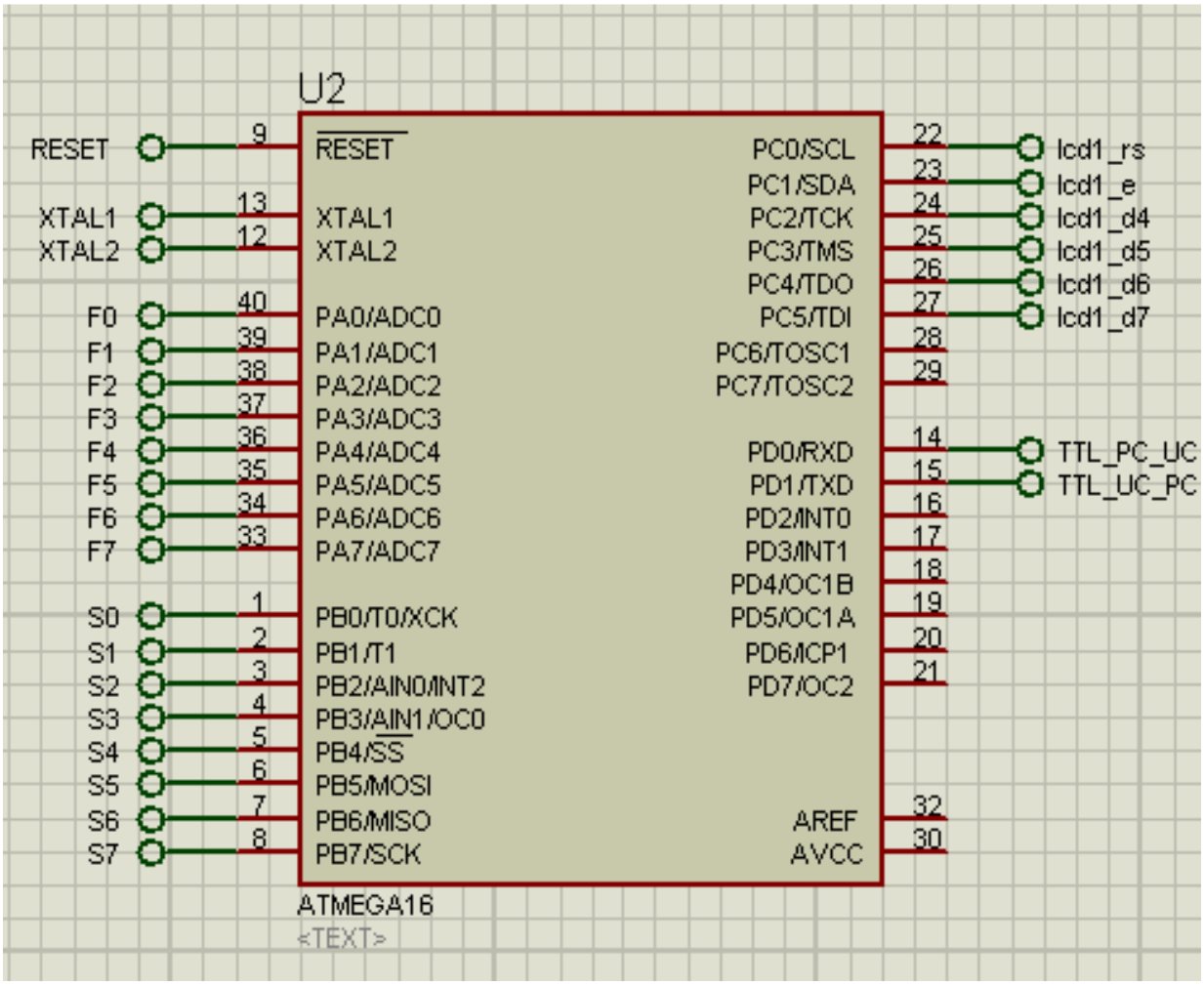


Figura 3.10. Diagrama de descripción de los pines del microcontrolador ATMEGA 16.

3.7 ETAPA DE COMUNICACIÓN

La comunicación entre el PC y el microcontrolador se la realiza utilizando el CI Max 232 el cual convierte los voltajes RS 232 que salen del computador a voltajes TTL que recibe el microcontrolador y viceversa, la configuración de este CI se la analizó en el capítulo anterior.

El microcontrolador recibe los caracteres ASSCI enviados desde la PC y realiza una acción determinada, por ejemplo si deseamos que se encienda el foco de la cocina enviamos la letra A para el foco de la sala enviamos B y así cuantas letras como focos dispongamos.

En la Figura 3.11 se muestra el diagrama de conexión de la etapa de comunicación

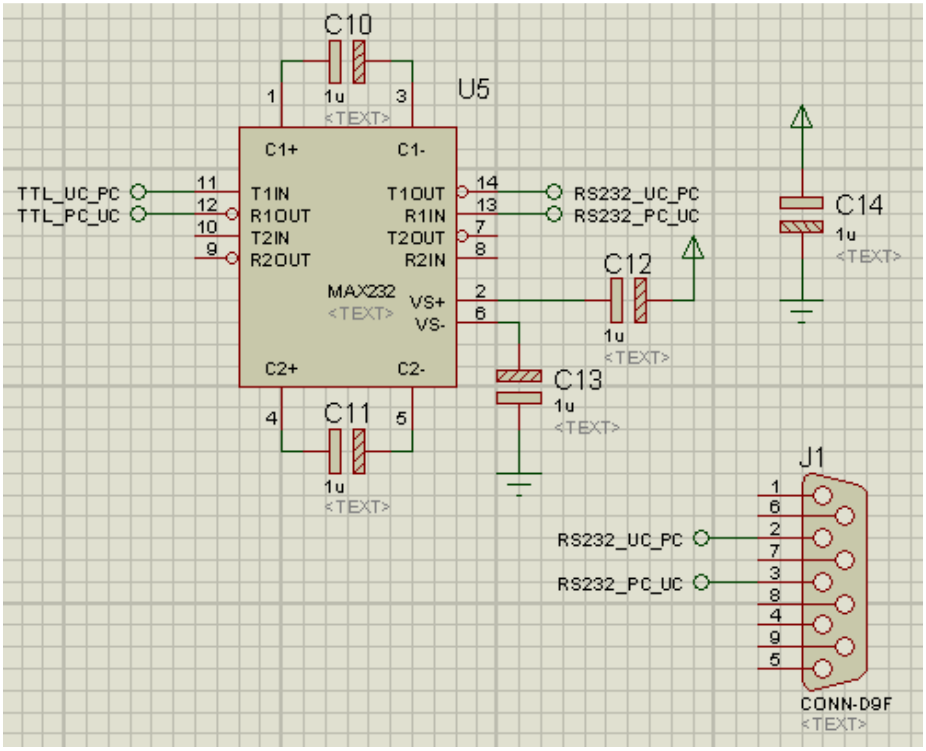


Figura 3.11. Diagrama de conexión de la etapa de comunicación.

3.8 ETAPA DE SOFTWARE

Para utilizar el programa de monitoreo y control de acceso debemos realizar los siguientes pasos:

- Instalar el Java Development Kit, el programa se encuentra en la carpeta instaladores en el CD adjunto.
- Instalar Netbeans IDE 7.1.2 que se encuentra en la carpeta instaladores en el CD adjunto.
- Instalar los archivos para el manejo del puerto serial, estos archivos se encuentran en la carpeta javaxcomm que se encuentra en el CD adjunto.

3.9 MANUAL DE USUARIO

Una vez tengamos los requisitos previos para utilizar el programa procedemos a abrir Netbeans IDE como nos muestra la Figura 3.12.

Abrimos la ubicación de nuestro proyecto que se llama gestionCasa2 que se encuentra en el CD adjunto (Figura 3.13).

Corremos el programa (Figura 3.14).

Elegimos el puerto serial al que se encuentra conectado el circuito (Figura 3.15).

Nos aparece el entorno del programa de monitoreo y acceso (Figura 3.16) en la parte izquierda tenemos el manejo de las luminarias y la sirena y en la parte derecha tenemos el monitoreo de los sensores conectados en las puertas y ventanas de nuestra maqueta.

Para configurar el horario de activación de la alarma damos click en el botón **Conf. Alarma.**

Configuramos la hora de inicio y la hora de desarme de nuestro sistema de alarma.

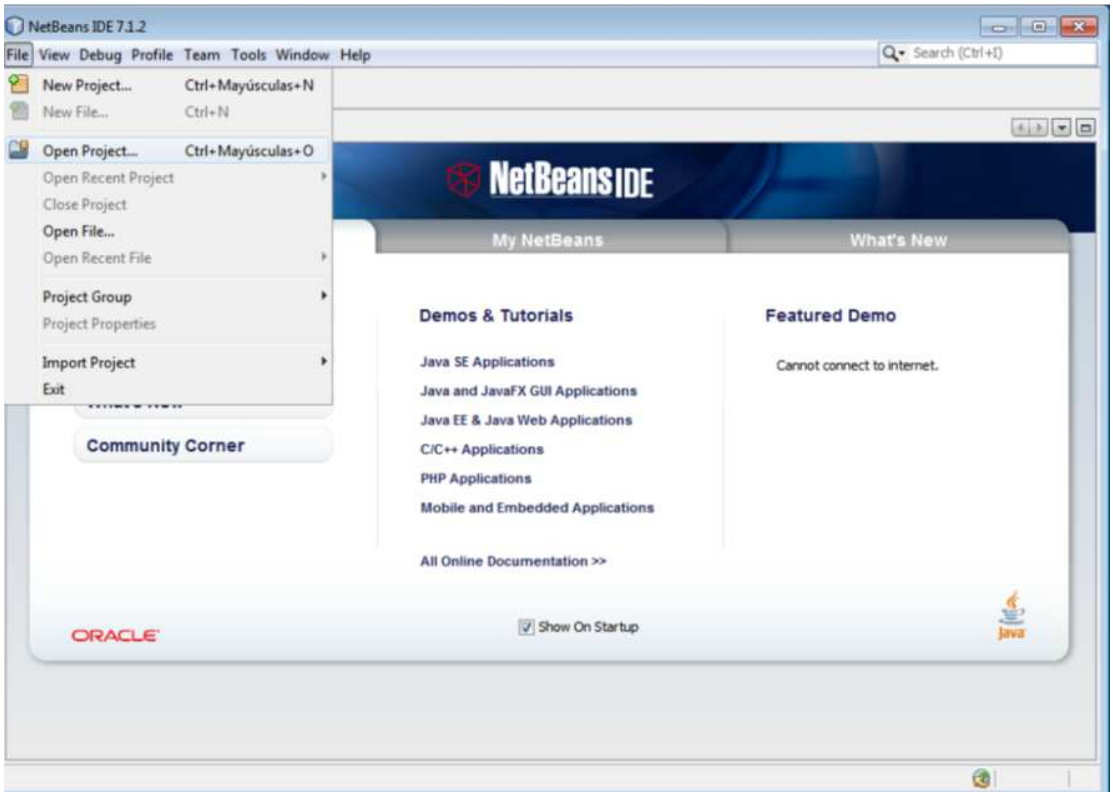


Figura 3.12. Abrimos netbeans.

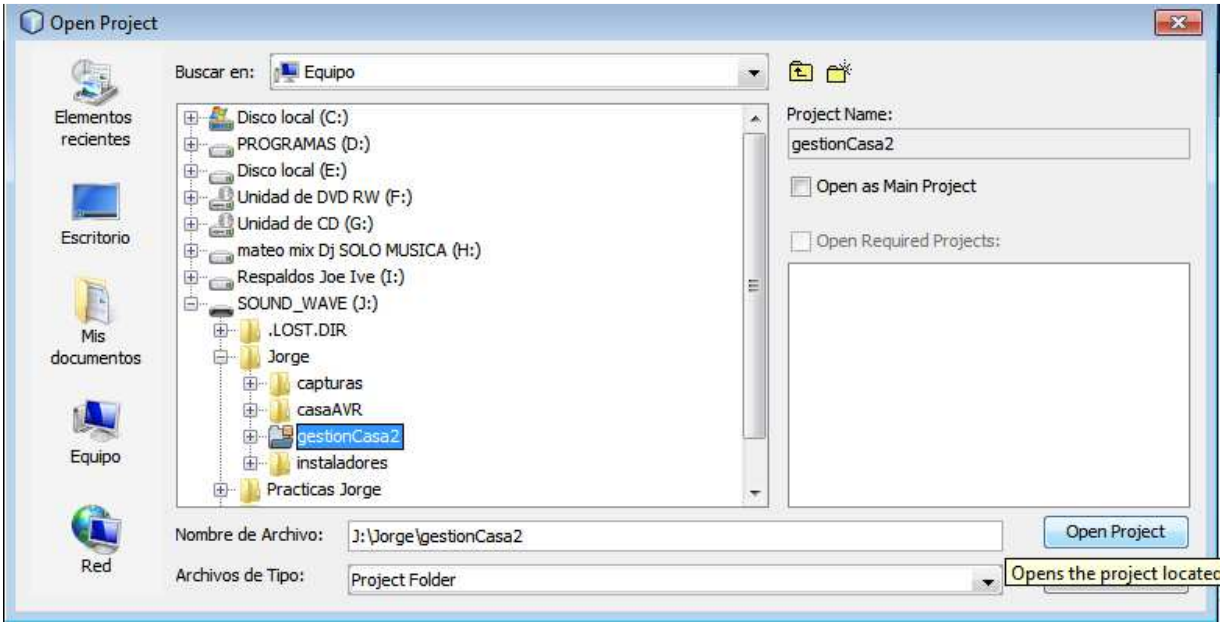


Figura 3.13. Abrimos la ubicación de nuestro proyecto.

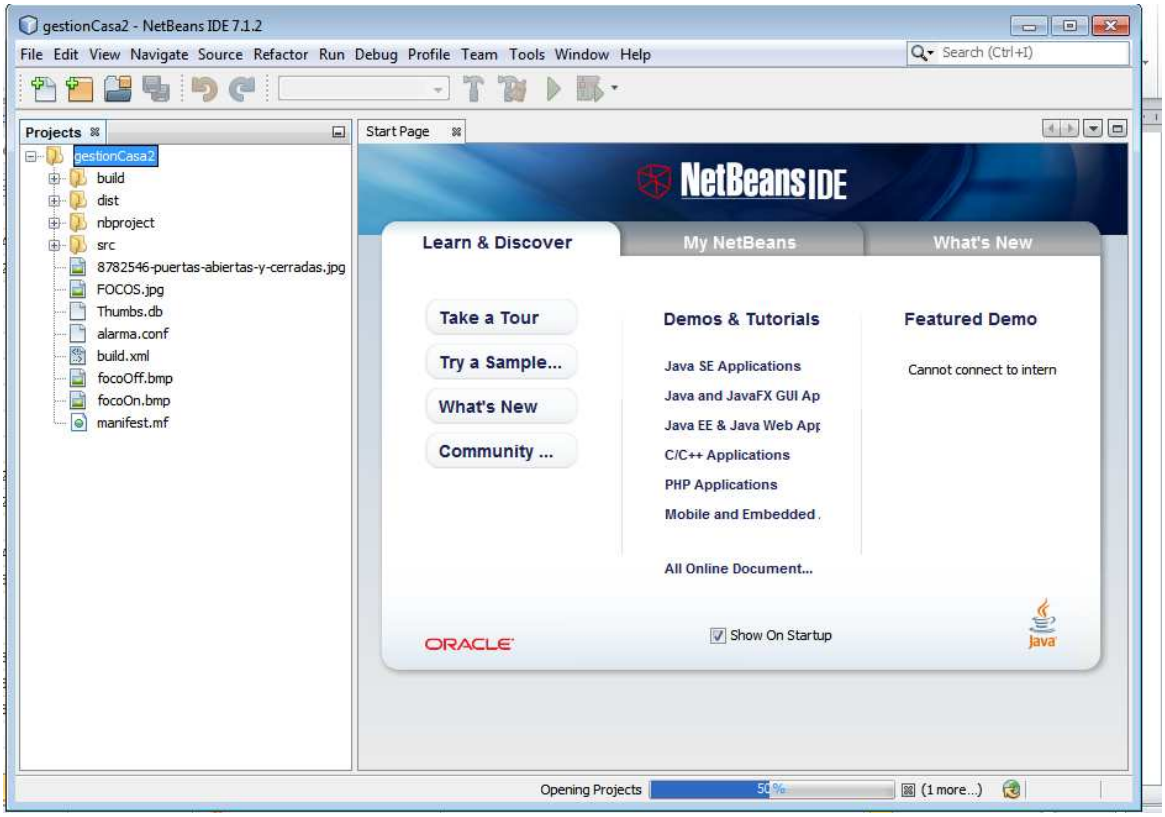


Figura 3.14. Corremos el programa.

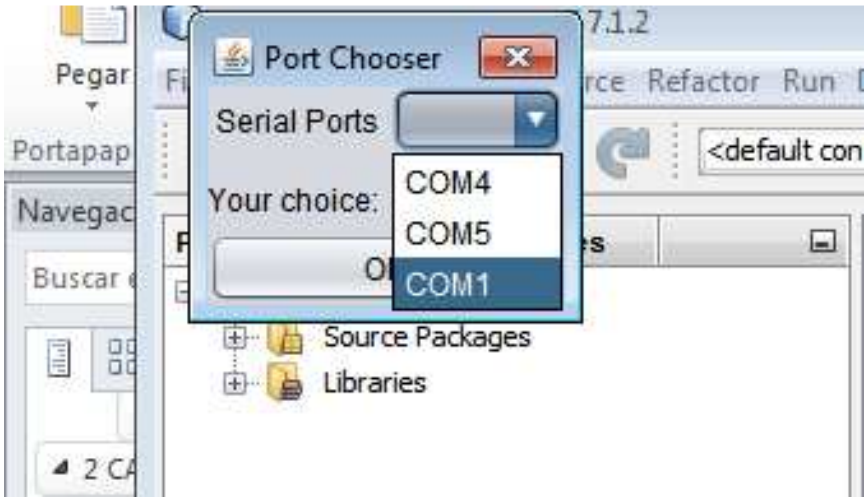


Figura 3.15. Elegimos el puerto serial.

Figura 3.16. Entorno del programa de monitoreo y acceso.

Figura 3.17. Configuración el horario de activación de la alarma.

Si la hora de nuestro PC está dentro del rango de configuración de la alarma el sistema estará armado (Figura 3.17).

Por último damos click al botón **leer entradas**.

Al dar click sobre la imagen del foco en el programa se encenderá la luminaria correspondiente en la vivienda.

Para silenciar la alarma debemos cerrar la ventana o puerta abierta y luego damos click sobre el botón de la sirena (Figura 3.18).

Figura 3.18. Desactivar la alarma.

4 CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- Mediante el desarrollo de este proyecto se cumplió el objetivo de controlar el encendido y apagado de las luces de la vivienda y monitorear el acceso irregular por puertas y ventanas a una vivienda por medio de un microcontrolador ATMEGA 16 conectado a un PC a través de un puerto serial.

- Se logró monitorear el acceso irregular a la vivienda a través de puertas y ventanas en tiempo real mediante sensores magnéticos colocados en las mismas.

- Se construyó la fuente de conversión de 120 voltios AC a 12 y 5 voltios DC utilizando los circuitos reguladores de voltaje de tres terminales 7812 y 7805 respectivamente.

- Se construyó un circuito que permite la comunicación entre la PC y el microcontrolador ATMEGA16 con la ayuda de un software llamado ISIS.

- Se realizó el programa necesario para que el microcontrolador ATMEGA 16 realice las tareas necesarias para el control de luces y el monitoreo del acceso por puertas y ventanas de la vivienda utilizando el software AVR STUDIO.

4.2 RECOMENDACIONES

- Este prototipo se lo podría implementar en un conjunto residencial para ser manejado mediante un sistema centralizado controlado por guardias de seguridad que vigilen dicho conjunto.

- Se podría añadir sensores de presencia al prototipo colocados en distintas zonas de la casa los cuales al detectar la presencia de alguien activen las luces del área en donde se encuentra la persona.

- Se podría colocar un circuito despertador adicional manejado con un teclado mediante el cual se pueda ingresar la hora en la que se desee encender las luces para de esta manera despertarse a la hora programada.

- Una manera de proteger nuestra vivienda mientras no nos encontremos en ella sería programando el encendido y apagado de las luces cada cierto tiempo para de esta manera aparentar que alguien se encuentra en la casa y así persuadir a los dueños de lo ajeno para que no ingresen a la vivienda.

REFERENCIAS BIBLIOGRÁFICAS

MICROCONTROLADOR

[1] www.rodrigog.com/pic/curso/Microcontroladores%20PIC.doc

[2] http://www.monografias.com/trabajos12/microco/microco.shtml

REGULADORES DE VOLTAJE

[3] http://www.ucontrol.com.ar/wiki/index.php?title=LM78xx

SENSORES

[4] http://es.wikipedia.org/wiki/Sensor

[5] http://www.mitecnologico.com/Main/TiposSensores

RELÉ

[6] http://es.wikipedia.org/wiki/Rel%C3%A9

DISPLAY LCD

[7] http://www.forosdeelectronica.com/f24/control-display-lcd-microcontrolador-pic-201/

COMUNICACIÓN SERIAL

[8] http://html.rincondelvago.com/puerto-serie-rs-232.html

[9] http://www.iearobotics.com/proyectos/cuadernos/ct1/ct1.html

LENGUAJE DE PROGRAMACIÓN BASCOM AVR

[10] http://bibdigital.epn.edu.ec/bitstream/15000/2349/1/CD-3097.pdf

ANEXOS

ANEXO A: PLANO DE LA MAQUETA

ANEXO B: IMÁGENES DE LA MAQUETA

ANEXO C:

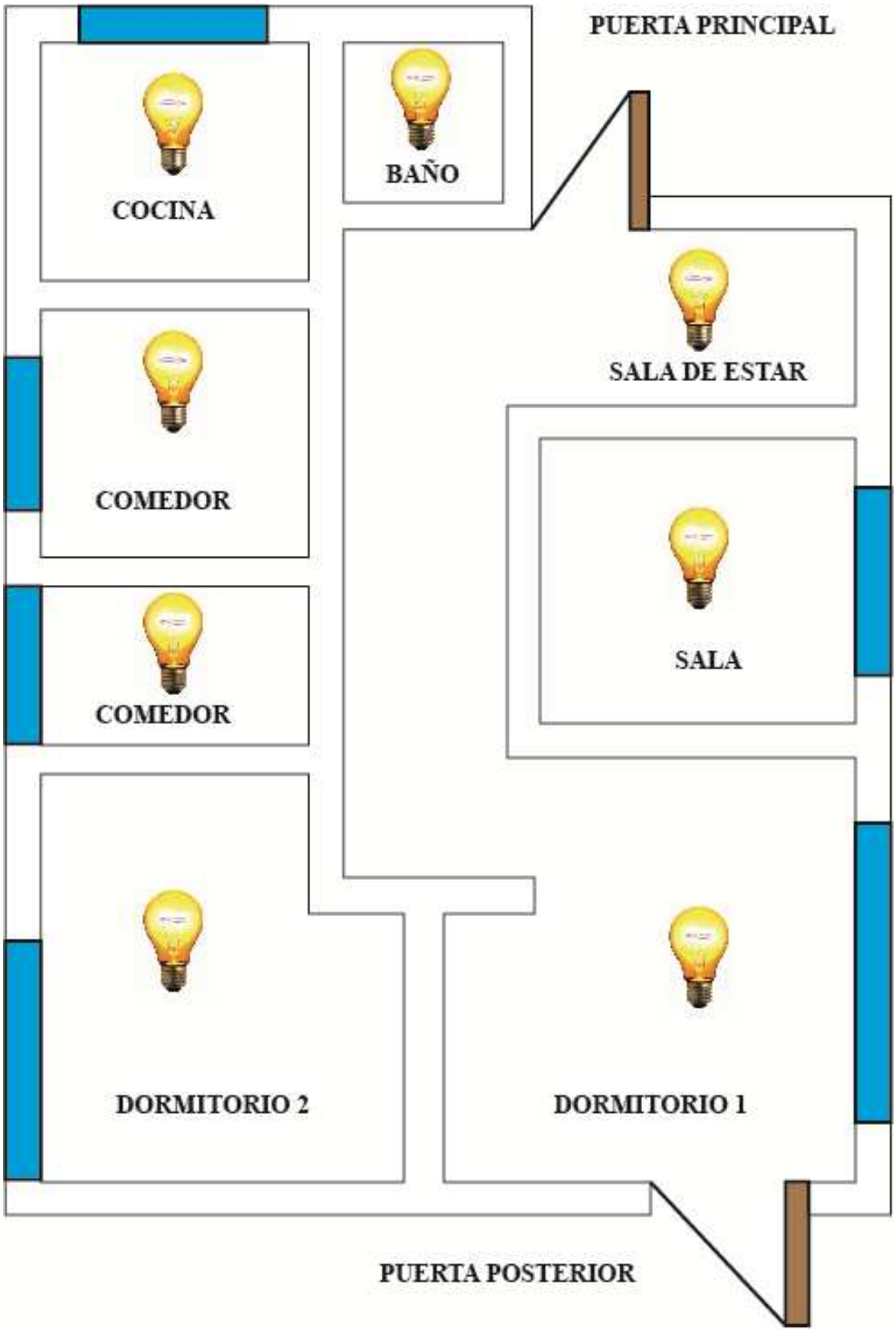
FICHA TÉCNICA DE LOS CIRCUITOS INTEGRADOS UTILIZADOS

ANEXO D: PROGRAMA DEL MICROCONTROLADOR

ANEXO E: PROGRAMA JAVA

ANEXO A

PLANO DE LA MAQUETA

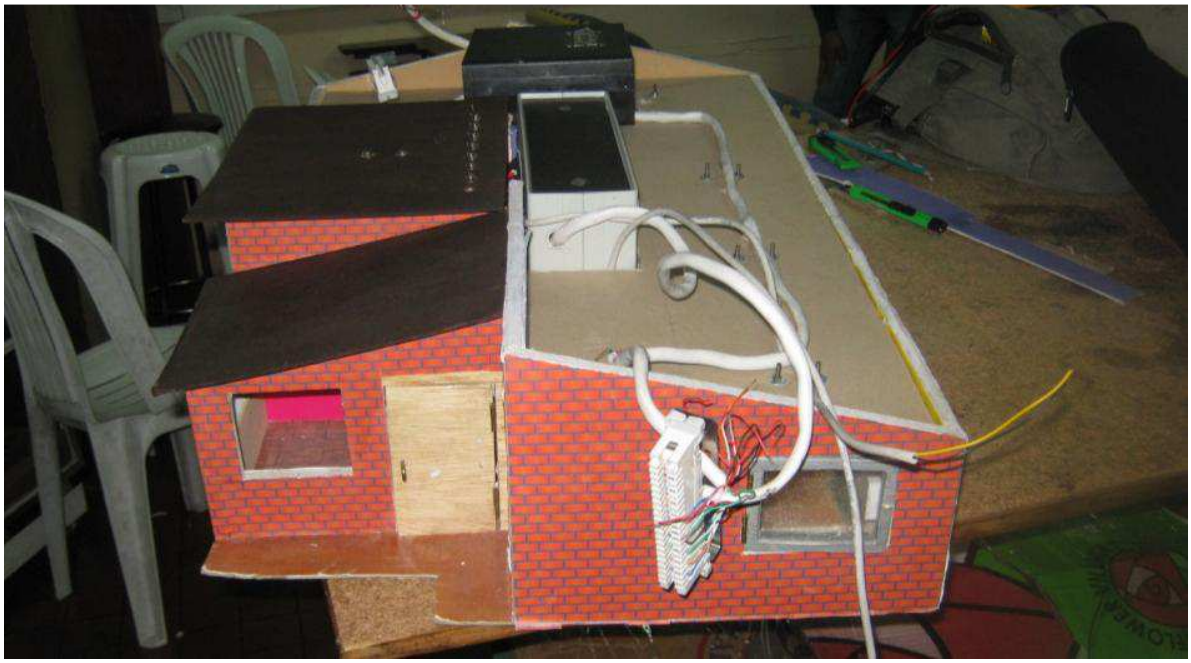

PUERTA PRINCIPAL
COCINA
BAÑO
SALA DE ESTAR
COMEDOR
SALA
COMEDOR
DORMITORIO 2
DORMITORIO 1
PUERTA POSTERIOR

ANEXO B

IMÁGENES DE LA MAQUETA

Anexo 1. Casa vista lateral izquierda

Anexo 2. Casa vista posterior.



Anexo 3. Casa vista frontal.

Anexo 4. Casa vista lateral derecha.

ANEXO C

FICHA TÉCNICA DE LOS CIRCUITOS INTEGRADOS UTILIZADOS

Features

- High-performance, Low-power Atmel® AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
  - 16 Kbytes of In-System Self-programmable Flash program memory
  - 512 Bytes EEPROM
  - 1 Kbyte Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    - 8 Single-ended Channels
    - 7 Differential Channels in TQFP Package Only
    - 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Operating Voltages
  - 2.7V - 5.5V for ATmega16L
  - 4.5V - 5.5V for ATmega16
- Speed Grades
  - 0 - 8 MHz for ATmega16L
  - 0 - 16 MHz for ATmega16
- Power Consumption @ 1 MHz, 3V, and 25°C for ATmega16L
  - Active: 1.1 mA
  - Idle Mode: 0.35 mA
  - Power-down Mode: < 1 µA

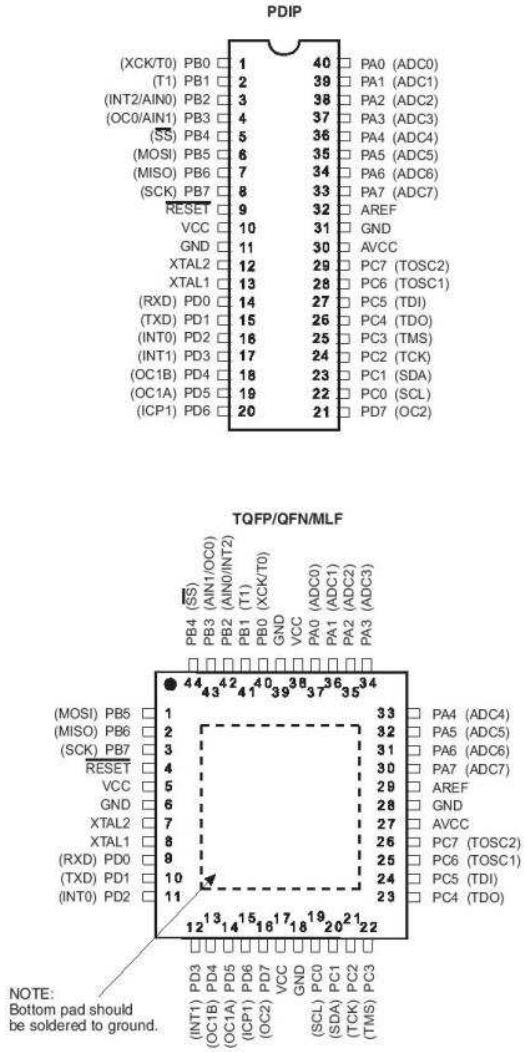
8-bit AVR® Microcontroller with 16K Bytes In-System Programmable Flash

ATmega16
ATmega16L

Rev. 2466T–AVR–07/10

Pin Configurations

Figure 1. Pinout ATmega16



Disclaimer

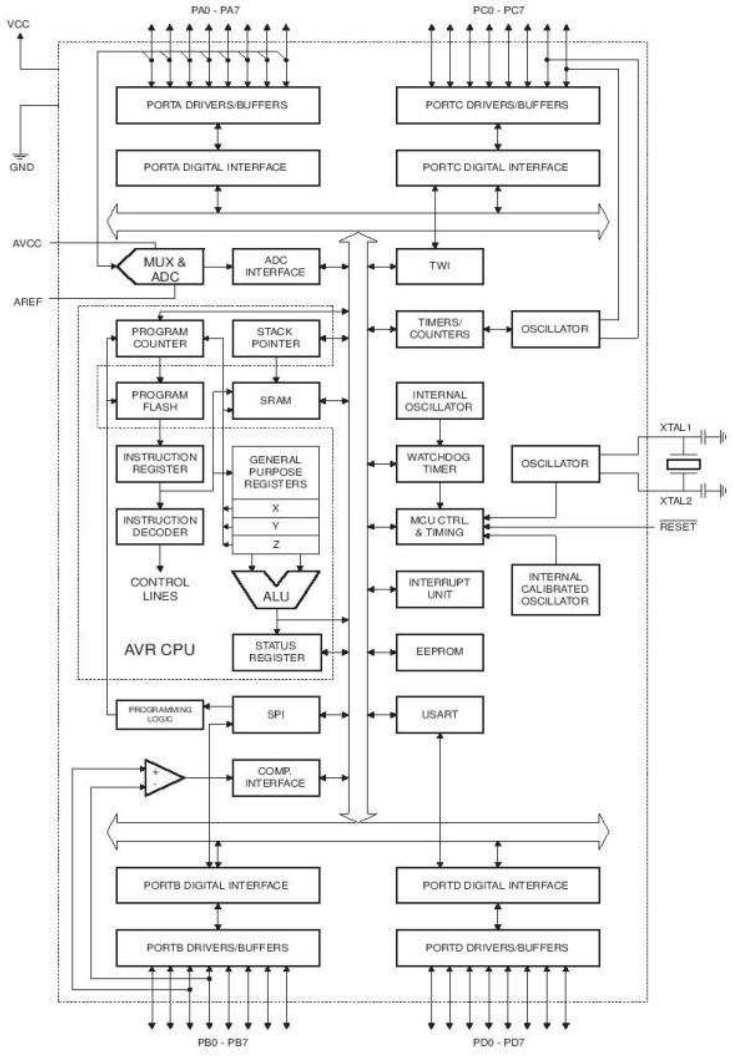
Typical values contained in this datasheet are based on simulations and characterization of other AVR microcontrollers manufactured on the same process technology. Min and Max values will be available after the device is characterized.

Overview

The ATmega16 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega16 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

Block Diagram

Figure 2. Block Diagram



The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega16 provides the following features: 16 Kbytes of In-System Programmable Flash Program memory with Read-While-Write capabilities, 512 bytes EEPROM, 1 Kbyte SRAM, 32 general purpose I/O lines, 32 general purpose working registers, a JTAG interface for Boundary-scan, On-chip Debugging support and programming, three flexible Timer/Counters with compare modes, Internal and External Interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain (TQFP package only), a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the USART, Two-wire interface, A/D Converter, SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next External Interrupt or Hardware Reset. In Power-save mode, the Asynchronous Timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption. In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run.

The device is manufactured using Atmel's high density nonvolatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega16 is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many embedded control applications.

The ATmega16 AVR is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

Pin Descriptions

VCC Digital supply voltage.

GND Ground.

Port A (PA7..PA0) Port A serves as the analog inputs to the A/D Converter.

Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pins can provide internal pull-up resistors (selected for each bit). The Port A output buffers have symmetrical drive characteristics with both high sink and source capability. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B (PB7..PB0) Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B also serves the functions of various special features of the ATmega16 as listed on page 58.

Port C (PC7..PC0) Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pins PC5(TDI), PC3(TMS) and PC2(TCK) will be activated even if a reset occurs.

Port C also serves the functions of the JTAG interface and other special features of the ATmega16 as listed on page 61.

Port D (PD7..PD0) Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port D also serves the functions of various special features of the ATmega16 as listed on page 63.

$\overline{\text{RESET}}$ Reset Input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given in Table 15 on page 38. Shorter pulses are not guaranteed to generate a reset.

XTAL1 Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

XTAL2 Output from the inverting Oscillator amplifier.

AVCC AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connected to V_{CC}, even if the ADC is not used. If the ADC is used, it should be connected to V_{CC} through a low-pass filter.

AREF AREF is the analog reference pin for the A/D Converter.

Resources

A comprehensive set of development tools, application notes and datasheets are available for download on http://www.atmel.com/avr.

Data Retention

Reliability Qualification results show that the projected data retention failure rate is much less than 1 PPM over 20 years at 85°C or 100 years at 25°C.

About Code Examples

This documentation contains simple code examples that briefly show how to use various parts of the device. These code examples assume that the part specific header file is included before compilation. Be aware that not all C Compiler vendors include bit definitions in the header files and interrupt handling in C is compiler dependent. Please confirm with the C Compiler documentation for more details.

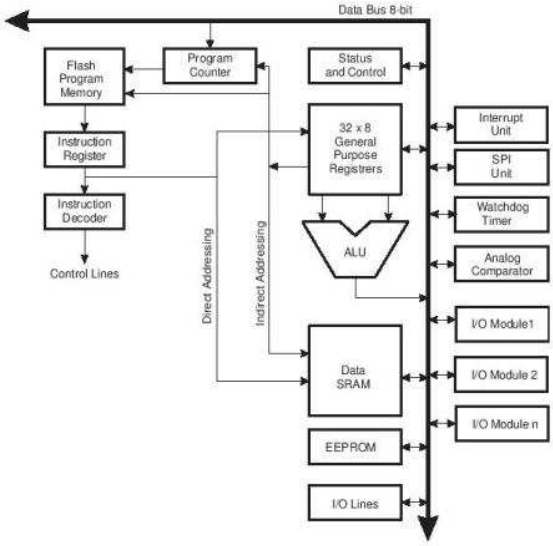
AVR CPU Core

Introduction

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

Architectural Overview

Figure 3. Block Diagram of the AVR MCU Architecture



In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory.

The fast-access Register File contains 32 × 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing – enabling efficient address calculations. One of the these address pointers can also be used as an address pointer for look up tables in Flash Program memory. These added function registers are the 16-bit X-register, Y-register, and Z-register, described later in this section.

The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation.

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16-bit or 32-bit instruction.

Program Flash memory space is divided in two sections, the Boot program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the reset routine (before subroutines or interrupts are executed). The Stack Pointer SP is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture.

The memory spaces in the AVR architecture are all linear and regular memory maps.

A flexible interrupt module has its control registers in the I/O space with an additional global interrupt enable bit in the Status Register. All interrupts have a separate interrupt vector in the interrupt vector table. The interrupts have priority in accordance with their interrupt vector position. The lower the interrupt vector address, the higher the priority.

The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, $20 - $5F.

ALU – Arithmetic Logic Unit

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See the “Instruction Set” section for a detailed description.

Status Register

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

- **Bit 6 – T: Bit Copy Storage**

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

- **Bit 5 – H: Half Carry Flag**

The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half Carry is useful in BCD arithmetic. See the "Instruction Set Description" for detailed information.

- **Bit 4 – S: Sign Bit, S = N ⊕ V**

The S-bit is always an exclusive or between the Negative Flag N and the Two's Complement Overflow Flag V. See the "Instruction Set Description" for detailed information.

- **Bit 3 – V: Two's Complement Overflow Flag**

The Two's Complement Overflow Flag V supports two's complement arithmetics. See the "Instruction Set Description" for detailed information.

- **Bit 2 – N: Negative Flag**

The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

- **Bit 1 – Z: Zero Flag**

The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

- **Bit 0 – C: Carry Flag**

The Carry Flag C indicates a carry in an arithmetic or logic operation. See the "Instruction Set Description" for detailed information.

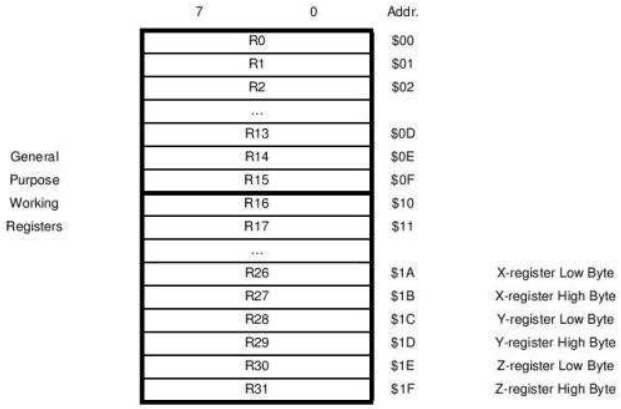
General Purpose Register File

The Register File is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the Register File:

- One 8-bit output operand and one 8-bit result input
- Two 8-bit output operands and one 8-bit result input
- Two 8-bit output operands and one 16-bit result input
- One 16-bit output operand and one 16-bit result input

Figure 4 shows the structure of the 32 general purpose working registers in the CPU.

Figure 4. AVR CPU General Purpose Working Registers



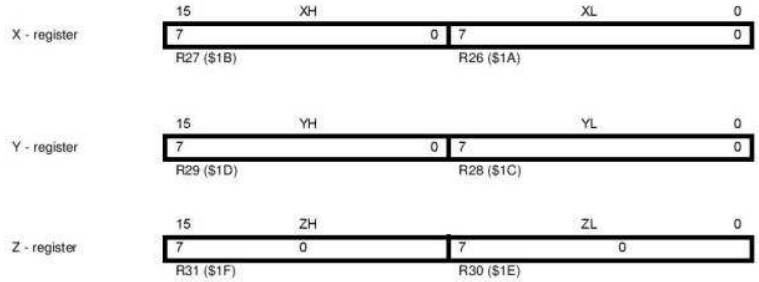
Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions.

As shown in Figure 4, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y-, and Z-pointer Registers can be set to index any register in the file.

The X-register, Y-register and Z-register

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the Data Space. The three indirect address registers X, Y, and Z are defined as described in Figure 5.

Figure 5. The X-register, Y-register, and Z-register



In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the Instruction Set Reference for details).

Stack Pointer

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer. If software reads the Program Counter from the Stack after a call or an interrupt, unused bits (15:13) should be masked out.

The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. This Stack space in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. The Stack Pointer must be set to point above $60. The Stack Pointer is decremented by one when data is pushed onto the Stack with the PUSH instruction, and it is decremented by two when the return address is pushed onto the Stack with subroutine call or interrupt. The Stack Pointer is incremented by one when data is popped from the Stack with the POP instruction, and it is incremented by two when data is popped from the Stack with return from subroutine RET or return from interrupt RETI.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Instruction Execution Timing

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock clk_{CPU}, directly generated from the selected clock source for the chip. No internal clock division is used.

Figure 6 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

Figure 6. The Parallel Instruction Fetches and Instruction Executions

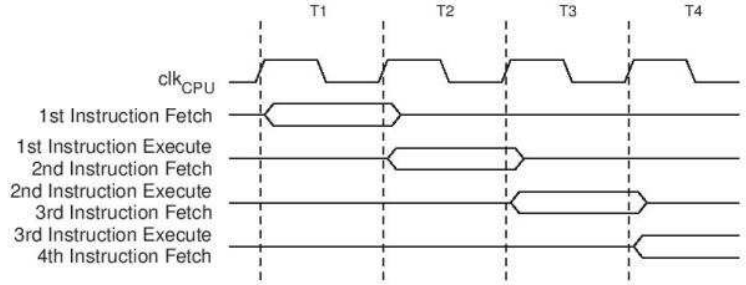
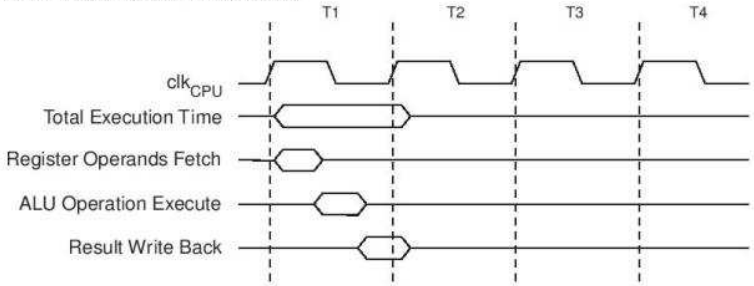


Figure 7 shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

Figure 7. Single Cycle ALU Operation



Reset and Interrupt Handling

The AVR provides several different interrupt sources. These interrupts and the separate reset vector each have a separate program vector in the program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock bits BLB02 or BLB12 are programmed. This feature improves software security. See the section "Memory Programming" on page 259 for details.

The lowest addresses in the program memory space are by default defined as the Reset and Interrupt Vectors. The complete list of vectors is shown in "Interrupts" on page 45. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request

0. The Interrupt Vectors can be moved to the start of the Boot Flash section by setting the IVSEL bit in the General Interrupt Control Register (GICR). Refer to "Interrupts" on page 45 for more information. The Reset Vector can also be moved to the start of the boot Flash section by programming the BOOTRST Fuse, see "Boot Loader Support – Read-While-Write Self-Programming" on page 246.

When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the global interrupt enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Note that the Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

Assembly Code Example

```
in  r16, SREG      ; store SREG value
cli    ; disable interrupts during timed sequence
sbi EECR, EEMWE    ; start EEPROM write
sbi EECR, EEWE
out SREG, r16      ; restore SREG value (I-bit)
```

C Code Example

```
char cSREG;
cSREG = SREG;  /* store SREG value */
/* disable interrupts during timed sequence */
_CLI();
EECR |= (1<<EEMWE); /* start EEPROM write */
EECR |= (1<<EEWE);
SREG = cSREG; /* restore SREG value (I-bit) */
```

When using the SEI instruction to enable interrupts, the instruction following SEI will be executed before any pending interrupts, as shown in this example.

Assembly Code Example

```
sei   ; set global interrupt enable
sleep ; enter sleep, waiting for interrupt
; note: will enter sleep before any pending
; interrupt(s)
```

C Code Example

```
_SEI(); /* set global interrupt enable */
_SLEEP(); /* enter sleep, waiting for interrupt */
/* note: will enter sleep before any pending interrupt(s) */
```

Interrupt Response Time

The interrupt execution response for all the enabled AVR interrupts is four clock cycles minimum. After four clock cycles the program vector address for the actual interrupt handling routine is executed. During this four clock cycle period, the Program Counter is pushed onto the Stack. The vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode.

A return from an interrupt handling routine takes four clock cycles. During these four clock cycles, the Program Counter (two bytes) is popped back from the Stack, the Stack Pointer is incremented by two, and the I-bit in SREG is set.

AVR ATmega16 Memories

This section describes the different memories in the ATmega16. The AVR architecture has two main memory spaces, the Data Memory and the Program Memory space. In addition, the ATmega16 features an EEPROM Memory for data storage. All three memory spaces are linear and regular.

In-System Reprogrammable Flash Program Memory

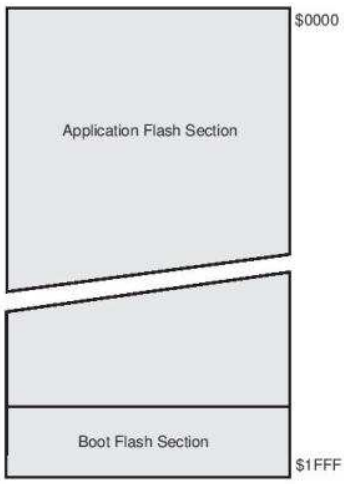
The ATmega16 contains 16 Kbytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 8K × 16. For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section.

The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega16 Program Counter (PC) is 13 bits wide, thus addressing the 8K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in "Boot Loader Support – Read-While-Write Self-Programming" on page 246. "Memory Programming" on page 259 contains a detailed description on Flash data serial downloading using the SPI pins or the JTAG interface.

Constant tables can be allocated within the entire program memory address space (see the LPM – Load Program Memory Instruction Description).

Timing diagrams for instruction fetch and execution are presented in "Instruction Execution Timing" on page 13.

Figure 8. Program Memory Map



SRAM Data Memory

Figure 9 shows how the ATmega16 SRAM Memory is organized.

The lower 1120 Data Memory locations address the Register File, the I/O Memory, and the internal data SRAM. The first 96 locations address the Register File and I/O Memory, and the next 1024 locations address the internal data SRAM.

The five different addressing modes for the data memory cover: Direct, Indirect with Displacement, Indirect, Indirect with Pre-decrement, and Indirect with Post-increment. In the Register File, registers R26 to R31 feature the indirect addressing pointer registers.

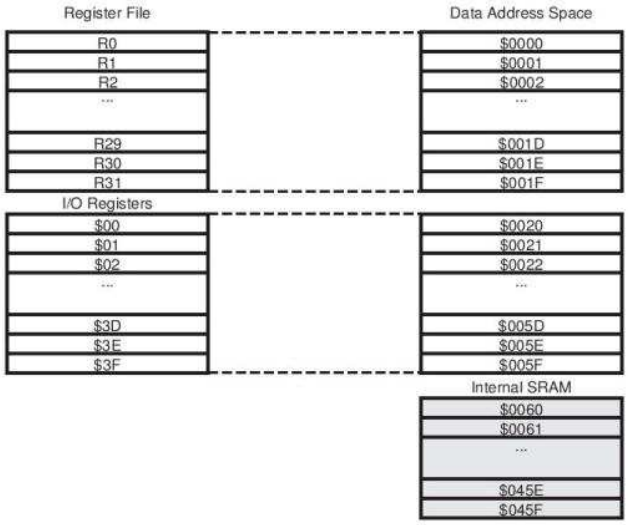
The direct addressing reaches the entire data space.

The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y-register or Z-register.

When using register indirect addressing modes with automatic pre-decrement and post-increment, the address registers X, Y, and Z are decremented or incremented.

The 32 general purpose working registers, 64 I/O Registers, and the 1024 bytes of internal data SRAM in the ATmega16 are all accessible through all these addressing modes. The Register File is described in "General Purpose Register File" on page 11.

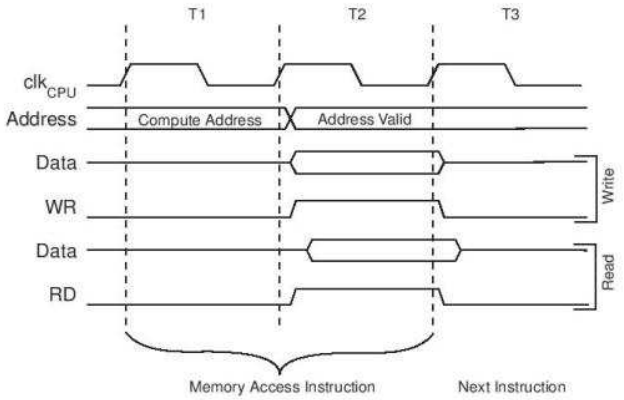
Figure 9. Data Memory Map



Data Memory Access Times

This section describes the general access timing concepts for internal memory access. The internal data SRAM access is performed in two clk_{CPU} cycles as described in Figure 10.

Figure 10. On-chip Data SRAM Access Cycles



EEPROM Data Memory

The ATmega16 contains 512 bytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

For a detailed description of SPI, JTAG, and Parallel data downloading to the EEPROM, see page 273, page 278, and page 262, respectively.

EEPROM Read/Write Access

The EEPROM Access Registers are accessible in the I/O space.

The write access time for the EEPROM is given in Table 1. A self-timing function, however, lets the user software detect when the next byte can be written. If the user code contains instructions that write the EEPROM, some precautions must be taken. In heavily filtered power supplies, V_{CC} is likely to rise or fall slowly on Power-up/down. This causes the device for some period of time to run at a voltage lower than specified as minimum for the clock frequency used. See "Preventing EEPROM Corruption" on page 22 for details on how to avoid problems in these situations.

In order to prevent unintentional EEPROM writes, a specific write procedure must be followed. Refer to the description of the EEPROM Control Register for details on this.

When the EEPROM is read, the CPU is halted for four clock cycles before the next instruction is executed. When the EEPROM is written, the CPU is halted for two clock cycles before the next instruction is executed.

The EEPROM Address Register – EEARH and EEARL

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	–	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	X	
	X	X	X	X	X	X	X	X	

- **Bits 15..9 – Res: Reserved Bits**

These bits are reserved bits in the ATmega16 and will always read as zero.

- **Bits 8..0 – EEAR8..0: EEPROM Address**

The EEPROM Address Registers – EEARH and EEARL – specify the EEPROM address in the 512 bytes EEPROM space. The EEPROM data bytes are addressed linearly between 0 and 511. The initial value of EEAR is undefined. A proper value must be written before the EEPROM may be accessed.

The EEPROM Data Register – EEDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..0 – EEDR7.0: EEPROM Data**

For the EEPROM write operation, the EEDR Register contains the data to be written to the EEPROM in the address given by the EEAR Register. For the EEPROM read operation, the EEDR contains the data read out from the EEPROM at the address given by EEAR.

The EEPROM Control Register – EECR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	X	0	

- **Bits 7..4 – Res: Reserved Bits**

These bits are reserved bits in the ATmega16 and will always read as zero.

- **Bit 3 – EERIE: EEPROM Ready Interrupt Enable**

Writing EERIE to one enables the EEPROM Ready Interrupt if the I bit in SREG is set. Writing EERIE to zero disables the interrupt. The EEPROM Ready interrupt generates a constant interrupt when EEWE is cleared.

- **Bit 2 – EEMWE: EEPROM Master Write Enable**

The EEMWE bit determines whether setting EEWE to one causes the EEPROM to be written. When EEMWE is set, setting EEWE within four clock cycles will write data to the EEPROM at the selected address If EEMWE is zero, setting EEWE will have no effect. When EEMWE has been written to one by software, hardware clears the bit to zero after four clock cycles. See the description of the EEWE bit for an EEPROM write procedure.

- **Bit 1 – EEWE: EEPROM Write Enable**

The EEPROM Write Enable Signal EEWE is the write strobe to the EEPROM. When address and data are correctly set up, the EEWE bit must be written to one to write the value into the EEPROM. The EEMWE bit must be written to one before a logical one is written to EEWE, otherwise no EEPROM write takes place. The following procedure should be followed when writing the EEPROM (the order of steps 3 and 4 is not essential):

1. Wait until EEWE becomes zero.
2. Wait until SPMEN in SPMCR becomes zero.
3. Write new EEPROM address to EEAR (optional).
4. Write new EEPROM data to EEDR (optional).
5. Write a logical one to the EEMWE bit while writing a zero to EEWE in EECR.
6. Within four clock cycles after setting EEMWE, write a logical one to EEWE.

The EEPROM can not be programmed during a CPU write to the Flash memory. The software must check that the Flash programming is completed before initiating a new EEPROM write. Step 2 is only relevant if the software contains a Boot Loader allowing the CPU to program the Flash. If the Flash is never being updated by the CPU, step 2 can be omitted. See "Boot Loader Support – Read-While-Write Self-Programming" on page 246 for details about boot programming.

Caution: An interrupt between step 5 and step 6 will make the write cycle fail, since the EEPROM Master Write Enable will time-out. If an interrupt routine accessing the EEPROM is interrupting another EEPROM Access, the EEAR or EEDR reGister will be modified, causing the interrupted EEPROM Access to fail. It is recommended to have the Global Interrupt Flag cleared during all the steps to avoid these problems.

When the write access time has elapsed, the EEWE bit is cleared by hardware. The user software can poll this bit and wait for a zero before writing the next byte. When EEWE has been set, the CPU is halted for two cycles before the next instruction is executed.

- **Bit 0 – EERE: EEPROM Read Enable**

The EEPROM Read Enable Signal – EERE – is the read strobe to the EEPROM. When the correct address is set up in the EEAR Register, the EERE bit must be written to a logic one to trigger the EEPROM read. The EEPROM read access takes one instruction, and the requested data is available immediately. When the EEPROM is read, the CPU is halted for four cycles before the next instruction is executed.

The user should poll the EEWE bit before starting the read operation. If a write operation is in progress, it is neither possible to read the EEPROM, nor to change the EEAR Register.

The calibrated Oscillator is used to time the EEPROM accesses. Table 1 lists the typical programming time for EEPROM access from the CPU.

Table 1. EEPROM Programming Time

Symbol	Number of Calibrated RC Oscillator Cycles[1]	Typ Programming Time
EEPROM write (from CPU)	8448	8.5 ms

Note: 1. Uses 1 MHz clock, independent of CKSEL Fuse setting.

The following code examples show one assembly and one C function for writing to the EEPROM. The examples assume that interrupts are controlled (for example by disabling interrupts globally) so that no interrupts will occur during execution of these functions. The examples

also assume that no Flash Boot Loader is present in the software. If such code is present, the EEPROM write function must also wait for any ongoing SPM command to finish.

Assembly Code Example

```
EEPROM_write:
  ; Wait for completion of previous write
  sbic EECR,EEWE
  rjmp EEPROM_write
  ; Set up address (r18:r17) in address register
  out  EEARH, r18
  out  EEARL, r17
  ; Write data (r16) to data register
  out  EEDR,r16
  ; Write logical one to EEMWE
  sbi  EECR,EEMWE
  ; Start eeprom write by setting EEWE
  sbi  EECR,EEWE
  ret
```

C Code Example

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
{
  /* Wait for completion of previous write */
  while(EECR & (1<<EEWE))
    ;
  /* Set up address and data registers */
  EEAR = uiAddress;
  EEDR = ucData;
  /* Write logical one to EEMWE */
  EECR |= (1<<EEMWE);
  /* Start eeprom write by setting EEWE */
  EECR |= (1<<EEWE);
}
```

The next code examples show assembly and C functions for reading the EEPROM. The examples assume that interrupts are controlled so that no interrupts will occur during execution of these functions.

Assembly Code Example

```
EEPROM_read:
  ; Wait for completion of previous write
  sbic EECR,EEWE
  rjmp EEPROM_read
  ; Set up address (r18:r17) in address register
  out  EEARH, r18
  out  EEARL, r17
  ; Start eeprom read by writing EERE
  sbi  EECR,EERE
  ; Read data from data register
  in   r16,EEDR
  ret
```

C Code Example

```
unsigned char EEPROM_read(unsigned int uiAddress)
{
  /* Wait for completion of previous write */
  while(EECR & (1<<EEWE))
    ;
  /* Set up address register */
  EEAR = uiAddress;
  /* Start eeprom read by writing EERE */
  EECR |= (1<<EERE);
  /* Return data from data register */
  return EEDR;
}
```

EEPROM Write During Power-down Sleep Mode

When entering Power-down Sleep mode while an EEPROM write operation is active, the EEPROM write operation will continue, and will complete before the Write Access time has passed. However, when the write operation is completed, the Oscillator continues running, and as a consequence, the device does not enter Power-down entirely. It is therefore recommended to verify that the EEPROM write operation is completed before entering Power-down.

Preventing EEPROM Corruption

During periods of low V_{CC}, the EEPROM data can be corrupted because the supply voltage is too low for the CPU and the EEPROM to operate properly. These issues are the same as for board level systems using EEPROM, and the same design solutions should be applied.

An EEPROM data corruption can be caused by two situations when the voltage is too low. First, a regular write sequence to the EEPROM requires a minimum voltage to operate correctly. Secondly, the CPU itself can execute instructions incorrectly, if the supply voltage is too low.

EEPROM data corruption can easily be avoided by following this design recommendation:

Keep the AVR RESET active (low) during periods of insufficient power supply voltage. This can be done by enabling the internal Brown-out Detector (BOD). If the detection level of the internal BOD does not match the needed detection level, an external low V_{CC} Reset Protection circuit can be used. If a reset occurs while a write operation is in progress, the write operation will be completed provided that the power supply voltage is sufficient.

I/O Memory

The I/O space definition of the ATmega16 is shown in "Register Summary" on page 331.

All ATmega16 I/Os and peripherals are placed in the I/O space. The I/O locations are accessed by the IN and OUT instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O Registers within the address range $00 - $1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions. Refer to the Instruction Set section for more details. When using the I/O specific commands IN and OUT, the I/O addresses $00 - $3F must be used. When addressing I/O Registers as data space using LD and ST instructions, $20 must be added to these addresses.

For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.

Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers $00 to $1F only.

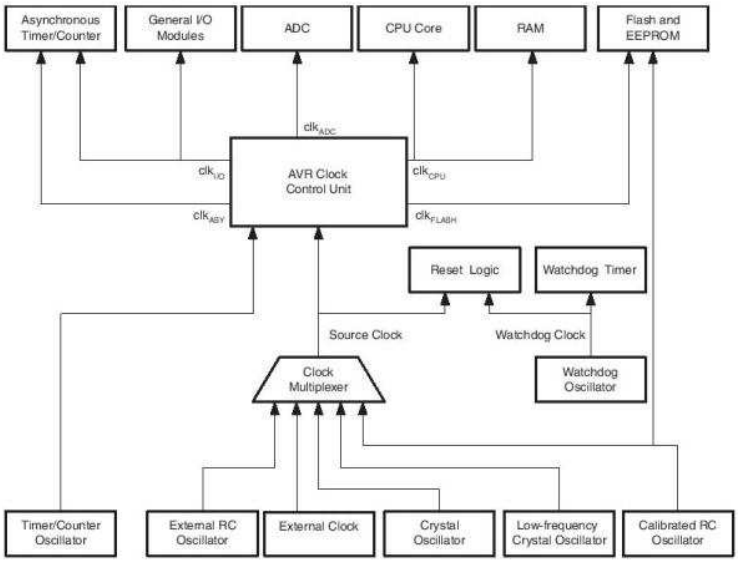
The I/O and Peripherals Control Registers are explained in later sections.

System Clock and Clock Options

Clock Systems and their Distribution

Figure 11 presents the principal clock systems in the AVR and their distribution. All of the clocks need not be active at a given time. In order to reduce power consumption, the clocks to modules not being used can be halted by using different sleep modes, as described in "Power Management and Sleep Modes" on page 32. The clock systems are detailed Figure 11.

Figure 11. Clock Distribution



CPU Clock – clk_{CPU}

The CPU clock is routed to parts of the system concerned with operation of the AVR core. Examples of such modules are the General Purpose Register File, the Status Register and the data memory holding the Stack Pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

I/O Clock – $clk_{I/O}$

The I/O clock is used by the majority of the I/O modules, like Timer/Counters, SPI, and USART. The I/O clock is also used by the External Interrupt module, but note that some external interrupts are detected by asynchronous logic, allowing such interrupts to be detected even if the I/O clock is halted. Also note that address recognition in the TWI module is carried out asynchronously when $clk_{I/O}$ is halted, enabling TWI address reception in all sleep modes.

Flash Clock – clk_{FLASH}

The Flash clock controls operation of the Flash interface. The Flash clock is usually active simultaneously with the CPU clock.

Asynchronous Timer Clock – clk_{ASY}

The Asynchronous Timer clock allows the Asynchronous Timer/Counter to be clocked directly from an external 32 kHz clock crystal. The dedicated clock domain allows using this Timer/Counter as a real-time counter even when the device is in sleep mode.

ADC Clock – clk_{ADC}

The ADC is provided with a dedicated clock domain. This allows halting the CPU and I/O clocks in order to reduce noise generated by digital circuitry. This gives more accurate ADC conversion results.

Clock Sources

The device has the following clock source options, selectable by Flash Fuse bits as shown below. The clock from the selected source is input to the AVR clock generator, and routed to the appropriate modules.

Table 2. Device Clocking Options Select[1]

Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

Note: 1. For all fuses "1" means unprogrammed while "0" means programmed.

The various choices for each clocking option is given in the following sections. When the CPU wakes up from Power-down or Power-save, the selected clock source is used to time the start-up, ensuring stable Oscillator operation before instruction execution starts. When the CPU starts from Reset, there is as an additional delay allowing the power to reach a stable level before commencing normal operation. The Watchdog Oscillator is used for timing this real-time part of the start-up time. The number of WDT Oscillator cycles used for each time-out is shown in Table 3. The frequency of the Watchdog Oscillator is voltage dependent as shown in "ATmega16 Typical Characteristics" on page 299.

Table 3. Number of Watchdog Oscillator Cycles

Typ Time-out (V_{CC} = 5.0V)	Typ Time-out (V_{CC} = 3.0V)	Number of Cycles
4.1 ms	4.3 ms	4K (4,096)
65 ms	69 ms	64K (65,536)

Default Clock Source

The device is shipped with CKSEL = "0001" and SUT = "10". The default clock source setting is therefore the 1 MHz Internal RC Oscillator with longest startup time. This default setting ensures that all users can make their desired clock source setting using an In-System or Parallel Programmer.

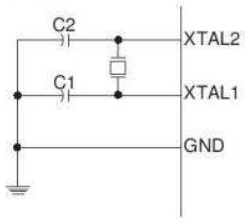
Crystal Oscillator

XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator, as shown in Figure 12. Either a quartz crystal or a ceramic resonator may be used. The CKOPT Fuse selects between two different Oscillator amplifier modes. When CKOPT is programmed, the Oscillator output will oscillate will a full rail-to-rail swing on the output. This mode is suitable when operating in a very noisy environment or when the output from XTAL2 drives a second clock buffer. This mode has a wide frequency range. When CKOPT is unprogrammed, the Oscillator has a smaller output swing. This reduces power consumption considerably. This mode has a limited frequency range and it can not be used to drive other clock buffers.

For resonators, the maximum frequency is 8 MHz with CKOPT unprogrammed and 16 MHz with CKOPT programmed. C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Some initial guidelines for

choosing capacitors for use with crystals are given in Table 4. For ceramic resonators, the capacitor values given by the manufacturer should be used.

Figure 12. Crystal Oscillator Connections



The Oscillator can operate in three different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL3..1 as shown in Table 4.

Table 4. Crystal Oscillator Operating Modes

CKOPT	CKSEL3..1	Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
1	101[1]	0.4 - 0.9	–
1	110	0.9 - 3.0	12 - 22
1	111	3.0 - 8.0	12 - 22
0	101, 110, 111	1.0 ≤	12 - 22

Note: 1. This option should not be used with crystals, only with ceramic resonators.

The CKSEL0 Fuse together with the SUT1..0 Fuses select the start-up times as shown in Table 5.

Table 5. Start-up Times for the Crystal Oscillator Clock Selection

CKSEL0	SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
0	00	258 CK[1]	4.1 ms	Ceramic resonator, fast rising power
0	01	258 CK[1]	65 ms	Ceramic resonator, slowly rising power
0	10	1K CK[2]	–	Ceramic resonator, BOD enabled
0	11	1K CK[2]	4.1 ms	Ceramic resonator, fast rising power
1	00	1K CK[2]	65 ms	Ceramic resonator, slowly rising power
1	01	16K CK	–	Crystal Oscillator, BOD enabled
1	10	16K CK	4.1 ms	Crystal Oscillator, fast rising power
1	11	16K CK	65 ms	Crystal Oscillator, slowly rising power

Notes: 1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device, and if frequency stability at start-up is not important for the application.

Low-frequency Crystal Oscillator

To use a 32.768 kHz watch crystal as the clock source for the device, the Low-frequency Crystal Oscillator must be selected by setting the CKSEL Fuses to "1001". The crystal should be connected as shown in Figure 12. By programming the CKOPT Fuse, the user can enable internal capacitors on XTAL1 and XTAL2, thereby removing the need for external capacitors. The internal capacitors have a nominal value of 36 pF.

When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in Table 6.

Table 6. Start-up Times for the Low-frequency Crystal Oscillator Clock Selection

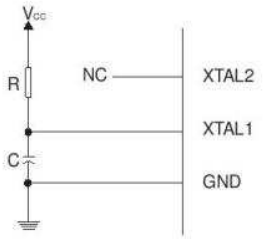
SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	1K CK(1)	4.1 ms	Fast rising power or BOD enabled
01	1K CK(1)	65 ms	Slowly rising power
10	32K CK	65 ms	Stable frequency at start-up
11	Reserved		

Note: 1. These options should only be used if frequency stability at start-up is not important for the application.

External RC Oscillator

For timing insensitive applications, the external RC configuration shown in Figure 13 can be used. The frequency is roughly estimated by the equation f = 1/(3RC). C should be at least 22 pF. By programming the CKOPT Fuse, the user can enable an internal 36 pF capacitor between XTAL1 and GND, thereby removing the need for an external capacitor. For more information on Oscillator operation and details on how to choose R and C, refer to the External RC Oscillator application note.

Figure 13. External RC Configuration



The Oscillator can operate in four different modes, each optimized for a specific frequency range. The operating mode is selected by the fuses CKSEL3..0 as shown in Table 7.

Table 7. External RC Oscillator Operating Modes

CKSEL3..0	Frequency Range (MHz)
0101	0.1 ≤ 0.9
0110	0.9 - 3.0
0111	3.0 - 8.0
1000	8.0 - 12.0

When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in Table 8.

Table 8. Start-up Times for the External RC Oscillator Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	18 CK	–	BOD enabled
01	18 CK	4.1 ms	Fast rising power
10	18 CK	65 ms	Slowly rising power
11	6 CK(1)	4.1 ms	Fast rising power or BOD enabled

Note: 1. This option should not be used when operating close to the maximum frequency of the device.

Calibrated Internal RC Oscillator

The Calibrated Internal RC Oscillator provides a fixed 1.0 MHz, 2.0 MHz, 4.0 MHz, or 8.0 MHz clock. All frequencies are nominal values at 5V and 25°C. This clock may be selected as the system-clock by programming the CKSEL Fuses as shown in Table 9. If selected, it will operate with no external components. The CKOPT Fuse should always be unpro-grammed when using this clock option. During Reset, hardware loads the calibration byte into the OSCCAL Register and thereby automatically calibrates the RC Oscillator. At 5V, 25°C and 1.0 MHz, 2.0 MHz, 4.0 MHz or 8.0 MHz Oscillator frequency selected, this calibration gives a frequency within ±3% of the nominal frequency. Using calibration methods as described in application notes available at www.atmel.com/avr it is possible to achieve ±1% accuracy at any given V_{CC} and Temperature. When this Oscillator is used as the Chip Clock, the Watchdog Oscillator will still be used for the Watchdog Timer and for the reset time-out. For more information on the pre-programmed calibration value, see the section "Calibration Byte" on page 261.

Table 9. Internal Calibrated RC Oscillator Operating Modes

CKSEL3..0	Nominal Frequency (MHz)
0001(1)	1.0
0010	2.0
0011	4.0
0100	8.0

Note: 1. The device is shipped with this option selected.

When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in Table 10. XTAL1 and XTAL2 should be left unconnected (NC).

Table 10. Start-up Times for the Internal Calibrated RC Oscillator Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10(1)	6 CK	65 ms	Slowly rising power
11	Reserved		

Note: 1. The device is shipped with this option selected.

Oscillator Calibration Register – OSCCAL

Bit	7	6	5	4	3	2	1	0	
	CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	OSCCAL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	Device Specific Calibration Value								

- **Bits 7..0 – CAL7..0: Oscillator Calibration Value**

Writing the calibration byte to this address will trim the Internal Oscillator to remove process variations from the Oscillator frequency. This is done automatically during Chip Reset. When OSCCAL is zero, the lowest available frequency is chosen. Writing non-zero values to this register will increase the frequency of the Internal Oscillator. Writing $FF to the register gives the highest available frequency. The calibrated Oscillator is used to time EEPROM and Flash access. If EEPROM or Flash is written, do not calibrate to more than 10% above the nominal frequency. Otherwise, the EEPROM or Flash write may fail. Note that the Oscillator is intended for calibration to 1.0 MHz, 2.0 MHz, 4.0 MHz, or 8.0 MHz. Tuning to other values is not guaranteed, as indicated in Table 11.

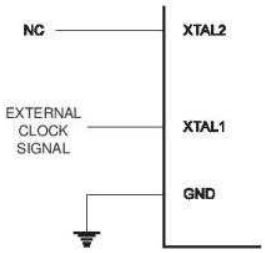
Table 11. Internal RC Oscillator Frequency Range.

OSCCAL Value	Min Frequency in Percentage of Nominal Frequency (%)	Max Frequency in Percentage of Nominal Frequency (%)
$00	50	100
$7F	75	150
$FF	100	200

External Clock

To drive the device from an external clock source, XTAL1 should be driven as shown in Figure 14. To run the device on an external clock, the CKSEL Fuses must be programmed to "0000". By programming the CKOPT Fuse, the user can enable an internal 36 pF capacitor between XTAL1 and GND.

Figure 14. External Clock Drive Configuration



When this clock source is selected, start-up times are determined by the SUT Fuses as shown in Table 12.

Table 12. Start-up Times for the External Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10	6 CK	65 ms	Slowly rising power
11	Reserved		

When applying an external clock, it is required to avoid sudden changes in the applied clock frequency to ensure stable operation of the MCU. A variation in frequency of more than 2% from one clock cycle to the next can lead to unpredictable behavior. It is required to ensure that the MCU is kept in reset during such changes in the clock frequency.

Timer/Counter Oscillator

For AVR microcontrollers with Timer/Counter Oscillator pins (TOSC1 and TOSC2), the crystal is connected directly between the pins. No external capacitors are needed. The Oscillator is optimized for use with a 32.768 kHz watch crystal. Applying an external clock source to TOSC1 is not recommended.

Note: The Timer/Counter Oscillator uses the same type of crystal oscillator as Low-Frequency Oscillator and the internal capacitors have the same nominal value of 36 pF.

Power Management and Sleep Modes

Sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application's requirements.

To enter any of the six sleep modes, the SE bit in MCUCR must be written to logic one and a SLEEP instruction must be executed. The SM2, SM1, and SM0 bits in the MCUCR Register select which sleep mode (Idle, ADC Noise Reduction, Power-down, Power-save, Standby, or Extended Standby) will be activated by the SLEEP instruction. See Table 13 for a summary. If an enabled interrupt occurs while the MCU is in a sleep mode, the MCU wakes up. The MCU is then halted for four cycles in addition to the start-up time, it executes the interrupt routine, and resumes execution from the instruction following SLEEP. The contents of the Register File and SRAM are unaltered when the device wakes up from sleep. If a Reset occurs during sleep mode, the MCU wakes up and executes from the Reset Vector.

Figure 11 on page 24 presents the different clock systems in the ATmega16, and their distribution. The figure is helpful in selecting an appropriate sleep mode.

MCU Control Register – MCUCR

The MCU Control Register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7, 5, 4 – SM2..0: Sleep Mode Select Bits 2, 1, and 0**

These bits select between the six available sleep modes as shown in Table 13.

Table 13. Sleep Mode Select

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby(1)
1	1	1	Extended Standby(1)

Note: 1. Standby mode and Extended Standby mode are only available with external crystals or resonators.

- **Bit 6 – SE: Sleep Enable**

The SE bit must be written to logic one to make the MCU enter the sleep mode when the SLEEP instruction is executed. To avoid the MCU entering the sleep mode unless it is the programmers purpose, it is recommended to write the Sleep Enable (SE) bit to one just before the execution of the SLEEP instruction and to clear it immediately after waking up.

Idle Mode

When the SM2..0 bits are written to 000, the SLEEP instruction makes the MCU enter Idle mode, stopping the CPU but allowing SPI, USART, Analog Comparator, ADC, Two-wire Serial Interface, Timer/Counters, Watchdog, and the interrupt system to continue operating. This sleep mode basically halts clk_{CPU} and clk_{FLASH}, while allowing the other clocks to run.

Idle mode enables the MCU to wake up from external triggered interrupts as well as internal ones like the Timer Overflow and USART Transmit Complete interrupts. If wake-up from the Analog Comparator interrupt is not required, the Analog Comparator can be powered down by setting the ACD bit in the Analog Comparator Control and Status Register – ACSR. This will reduce power consumption in Idle mode. If the ADC is enabled, a conversion starts automatically when this mode is entered.

ADC Noise Reduction Mode

When the SM2..0 bits are written to 001, the SLEEP instruction makes the MCU enter ADC Noise Reduction mode, stopping the CPU but allowing the ADC, the External Interrupts, the Two-wire Serial Interface address watch, Timer/Counter2 and the Watchdog to continue operating (if enabled). This sleep mode basically halts $clk_{I/O}$, clk_{CPU}, and clk_{FLASH}, while allowing the other clocks to run.

This improves the noise environment for the ADC, enabling higher resolution measurements. If the ADC is enabled, a conversion starts automatically when this mode is entered. Apart form the ADC Conversion Complete interrupt, only an External Reset, a Watchdog Reset, a Brown-out Reset, a Two-wire Serial Interface Address Match Interrupt, a Timer/Counter2 interrupt, an SPM/EEPROM ready interrupt, an External level interrupt on INT0 or INT1, or an external interrupt on INT2 can wake up the MCU from ADC Noise Reduction mode.

Power-down Mode

When the SM2..0 bits are written to 010, the SLEEP instruction makes the MCU enter Power-down mode. In this mode, the External Oscillator is stopped, while the External interrupts, the Two-wire Serial Interface address watch, and the Watchdog continue operating (if enabled). Only an External Reset, a Watchdog Reset, a Brown-out Reset, a Two-wire Serial Interface address match interrupt, an External level interrupt on INT0 or INT1, or an External interrupt on INT2 can wake up the MCU. This sleep mode basically halts all generated clocks, allowing operation of asynchronous modules only.

Note that if a level triggered interrupt is used for wake-up from Power-down mode, the changed level must be held for some time to wake up the MCU. Refer to "External Interrupts" on page 68 for details.

When waking up from Power-down mode, there is a delay from the wake-up condition occurs until the wake-up becomes effective. This allows the clock to restart and become stable after having been stopped. The wake-up period is defined by the same CKSEL Fuses that define the reset time-out period, as described in "Clock Sources" on page 25.

Power-save Mode

When the SM2..0 bits are written to 011, the SLEEP instruction makes the MCU enter Power-save mode. This mode is identical to Power-down, with one exception:

If Timer/Counter2 is clocked asynchronously, that is, the AS2 bit in ASSR is set, Timer/Counter2 will run during sleep. The device can wake up from either Timer Overflow or Output Compare event from Timer/Counter2 if the corresponding Timer/Counter2 interrupt enable bits are set in TIMSK, and the Global Interrupt Enable bit in SREG is set.

If the Asynchronous Timer is NOT clocked asynchronously, Power-down mode is recommended instead of Power-save mode because the contents of the registers in the Asynchronous Timer should be considered undefined after wake-up in Power-save mode if AS2 is 0.

This sleep mode basically halts all clocks except clk_{ASY}, allowing operation only of asynchronous modules, including Timer/Counter2 if clocked asynchronously.

Standby Mode

When the SM2..0 bits are 110 and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Standby mode. This mode is identical to Power-down with the exception that the Oscillator is kept running. From Standby mode, the device wakes up in six clock cycles.

Extended Standby Mode

When the SM2..0 bits are 111 and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Extended Standby mode. This mode is identical to Power-save mode with the exception that the Oscillator is kept running. From Extended Standby mode, the device wakes up in six clock cycles..

Table 14. Active Clock Domains and Wake Up Sources in the Different Sleep Modes

	Active Clock domains					Oscillators		Wake-up Sources					
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Osc. Enabled	INT2 INT1 INT0	TWI Address Match	Timer 2	SPM / EEPROM Ready	ADC	Other I/O
Idle			X	X	X	X	X(2)	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X	X	X	
Power Down								X(3)	X				
Power Save					X(2)		X(2)	X(3)	X	X(2)			
Standby(1)						X		X(3)	X				
Extended Standby(1)					X(2)	X	X(2)	X(3)	X	X(2)			

Notes: 1. External Crystal or resonator selected as clock source.
2. If AS2 bit in ASSR is set.
3. Only INT2 or level interrupt INT1 and INT0.

Minimizing Power Consumption

There are several issues to consider when trying to minimize the power consumption in an AVR controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as possible of the device's functions are operating. All functions not needed should be disabled. In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption.

Analog to Digital Converter

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion. Refer to "Analog to Digital Converter" on page 204 for details on ADC operation.

Analog Comparator

When entering Idle mode, the Analog Comparator should be disabled if not used. When entering ADC Noise Reduction mode, the Analog Comparator should be disabled. In the other sleep modes, the Analog Comparator is automatically disabled. However, if the Analog Comparator is set up to use the Internal Voltage Reference as input, the Analog Comparator should be disabled in all sleep modes. Otherwise, the Internal Voltage Reference will be enabled, independent of sleep mode. Refer to "Analog Comparator" on page 201 for details on how to configure the Analog Comparator.

Brown-out Detector

If the Brown-out Detector is not needed in the application, this module should be turned off. If the Brown-out Detector is enabled by the BODEN Fuse, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to "Brown-out Detection" on page 40 for details on how to configure the Brown-out Detector.

Internal Voltage Reference

The Internal Voltage Reference will be enabled when needed by the Brown-out Detector, the Analog Comparator or the ADC. If these modules are disabled as described in the sections above, the internal voltage reference will be disabled and it will not be consuming power. When turned on again, the user must allow the reference to start up before the output is used. If the reference is kept on in sleep mode, the output can be used immediately. Refer to "Internal Voltage Reference" on page 42 for details on the start-up time.

Watchdog Timer

If the Watchdog Timer is not needed in the application, this module should be turned off. If the Watchdog Timer is enabled, it will be enabled in all sleep modes, and hence, always consume power. In the deeper sleep modes, this will contribute significantly to the total current consumption. Refer to "Watchdog Timer" on page 42 for details on how to configure the Watchdog Timer.

Port Pins

When entering a sleep mode, all port pins should be configured to use minimum power. The most important thing is then to ensure that no pins drive resistive loads. In sleep modes where the both the I/O clock ($clk_{I/O}$) and the ADC clock (clk_{ADC}) are stopped, the input buffers of the device will be disabled. This ensures that no power is consumed by the input logic when not needed. In some cases, the input logic is needed for detecting wake-up conditions, and it will then be enabled. Refer to the section "Digital Input Enable and Sleep Modes" on page 54 for details on which pins are enabled. If the input buffer is enabled and the input signal is left floating or have an analog signal level close to $V_{CC}/2$, the input buffer will use excessive power.

JTAG Interface and On-chip Debug System

If the On-chip debug system is enabled by the OCDEN Fuse and the chip enter Power down or Power save sleep mode, the main clock source remains enabled. In these sleep modes, this will contribute significantly to the total current consumption. There are three alternative ways to avoid this:

- Disable OCDEN Fuse.
- Disable JTAGEN Fuse.
- Write one to the JTD bit in MCUCSR.

The TDO pin is left floating when the JTAG interface is enabled while the JTAG TAP controller is not shifting data. If the hardware connected to the TDO pin does not pull up the logic level, power consumption will increase. Note that the TDI pin for the next device in the scan chain contains a pull-up that avoids this problem. Writing the JTD bit in the MCUCSR register to one or leaving the JTAG fuse unprogrammed disables the JTAG interface.

Order this document by ULN2803/D

MOTOROLA

Octal High Voltage, High Current Darlington Transistor Arrays

ULN2803
ULN2804

OCTAL PERIPHERAL DRIVER ARRAYS

SEMICONDUCTOR TECHNICAL DATA

The eight NPN Darlington connected transistors in this family of arrays are ideally suited for interfacing between low logic level digital circuitry (such as TTL, CMOS or PMOS/NMOS) and the higher current/voltage requirements of lamps, relays, printer hammers or other similar loads for a broad range of computer, industrial, and consumer applications. All devices feature open–collector outputs and free wheeling clamp diodes for transient suppression.

The ULN2803 is designed to be compatible with standard TTL families while the ULN2804 is optimized for 6 to 15 volt high level CMOS or PMOS.

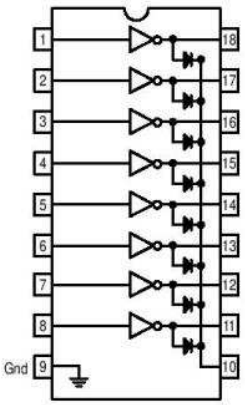
A SUFFIX
PLASTIC PACKAGE
CASE 707

MAXIMUM RATINGS (T_A = 25°C and rating apply to any one device in the package, unless otherwise noted.)

Rating	Symbol	Value	Unit
Output Voltage	V_O	50	V
Input Voltage (Except ULN2801)	V_I	30	V
Collector Current – Continuous	I_C	500	mA
Base Current – Continuous	I_B	25	mA
Operating Ambient Temperature Range	T_A	0 to +70	°C
Storage Temperature Range	T_{stg}	−55 to +150	°C
Junction Temperature	T_J	125	°C

$R_{\theta JA}$ = 55°C/W
Do not exceed maximum current limit per driver.

PIN CONNECTIONS



ORDERING INFORMATION

Device	Characteristics		
	Input Compatibility	V_{CE}(Max)/I_C(Max)	Operating Temperature Range
ULN2803A ULN2804A	TTL, 5.0 V CMOS 6 to 15 V CMOS, PMOS	50 V/500 mA	T_A = 0 to + 70°C

Rev 1

ELECTRICAL CHARACTERISTICS (T_A = 25°C, unless otherwise noted)

Characteristic		Symbol	Min	Typ	Max	Unit
Output Leakage Current (Figure 1)		I_{CEX}				µA
(V_O = 50 V, T_A = +70°C)	All Types		–	–	100	
(V_O = 50 V, T_A = +25°C)	All Types		–	–	50	
(V_O = 50 V, T_A = +70°C, V_I = 6.0 V)	ULN2802		–	–	500	
(V_O = 50 V, T_A = +70°C, V_I = 1.0 V)	ULN2804		–	–	500	
Collector–Emitter Saturation Voltage (Figure 2)		$V_{CE(sat)}$				V
(I_C = 350 mA, I_B = 500 µA)	All Types		–	1.1	1.6	
(I_C = 200 mA, I_B = 350 µA)	All Types		–	0.95	1.3	
(I_C = 100 mA, I_B = 250 µA)	All Types		–	0.85	1.1	
Input Current – On Condition (Figure 4)		$I_{I(on)}$				mA
(V_I = 17 V)	ULN2802		–	0.82	1.25	
(V_I = 3.85 V)	ULN2803		–	0.93	1.35	
(V_I = 5.0 V)	ULN2804		–	0.35	0.5	
(V_I = 12 V)	ULN2804		–	1.0	1.45	
Input Voltage – On Condition (Figure 5)		$V_{I(on)}$				V
(V_{CE} = 2.0 V, I_C = 300 mA)	ULN2802		–	–	13	
(V_{CE} = 2.0 V, I_C = 200 mA)	ULN2803		–	–	2.4	
(V_{CE} = 2.0 V, I_C = 250 mA)	ULN2803		–	–	2.7	
(V_{CE} = 2.0 V, I_C = 300 mA)	ULN2803		–	–	3.0	
(V_{CE} = 2.0 V, I_C = 125 mA)	ULN2804		–	–	5.0	
(V_{CE} = 2.0 V, I_C = 200 mA)	ULN2804		–	–	6.0	
(V_{CE} = 2.0 V, I_C = 275 mA)	ULN2804		–	–	7.0	
(V_{CE} = 2.0 V, I_C = 350 mA)	ULN2804		–	–	8.0	
Input Current – Off Condition (Figure 3) (I_C = 500 µA, T_A = +70°C)	All Types	$I_{I(off)}$	50	100	–	µA
DC Current Gain (Figure 2) (V_{CE} = 2.0 V, I_C = 350 mA)	ULN2801	h_{FE}	1000	–	–	–
Input Capacitance		C_I	–	15	25	pF
Turn–On Delay Time (50% E_I to 50% E_O)		t_{on}	–	0.25	1.0	µs
Turn–Off Delay Time (50% E_I to 50% E_O)		t_{off}	–	0.25	1.0	µs
Clamp Diode Leakage Current (Figure 6) (V_R = 50 V)	T_A = +25°C T_A = +70°C	I_R	–	–	50 100	µA
Clamp Diode Forward Voltage (Figure 7) (I_F = 350 mA)		V_F	–	1.5	2.0	V

TEST FIGURES

(See Figure Numbers in Electrical Characteristics Table)

Figure 1.

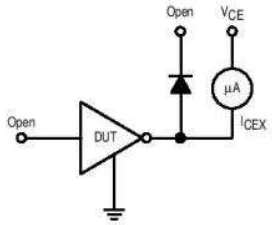


Figure 2.

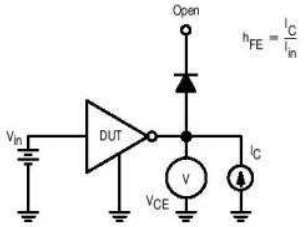


Figure 3.

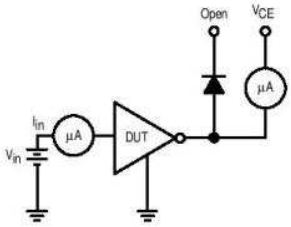


Figure 4.

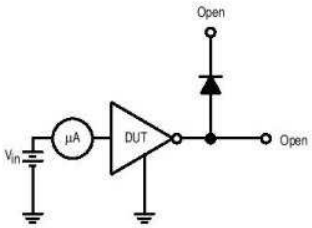


Figure 5.

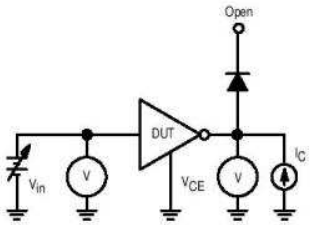


Figure 6.

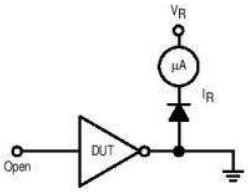
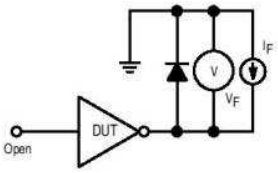


Figure 7.



TYPICAL CHARACTERISTIC CURVES – T_A = 25°C, unless otherwise noted

Output Characteristics

Figure 8. Output Current versus Saturation Voltage

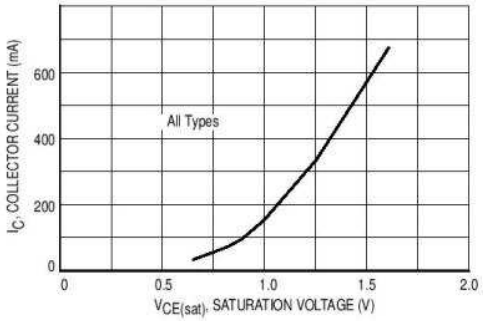
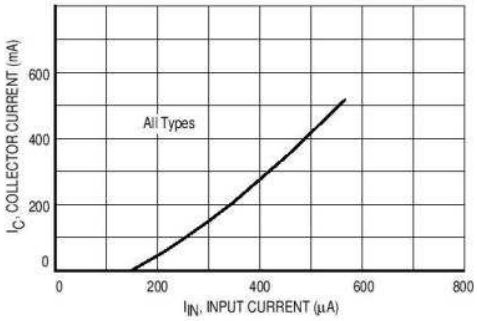


Figure 9. Output Current versus Input Current



Input Characteristics

Figure 10. ULN2803 Input Current versus Input Voltage

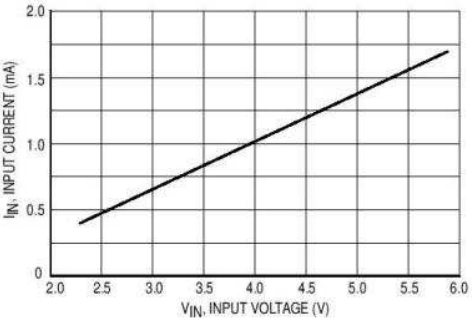


Figure 11. ULN2804 Input Current versus Input Voltage

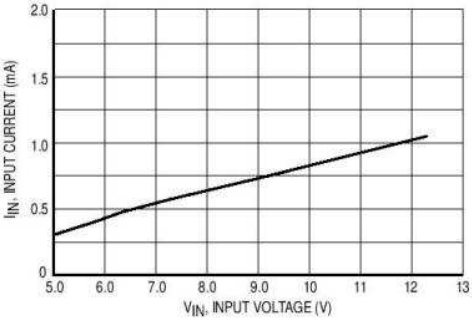
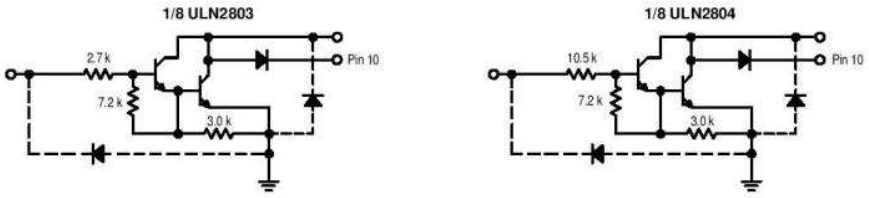


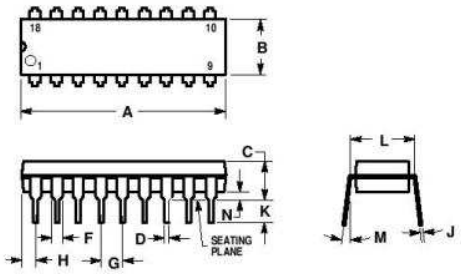
Figure 12. Representative Schematic Diagrams



ULN2803 ULN2804

OUTLINE DIMENSIONS

A SUFFIX
PLASTIC PACKAGE
CASE 707–02
ISSUE C



NOTES:
1. POSITIONAL TOLERANCE OF LEADS (D), SHALL BE WITHIN 0.25 (0.010) AT MAXIMUM MATERIAL CONDITION, IN RELATION TO SEATING PLANE AND EACH OTHER.
2. DIMENSION L TO CENTER OF LEADS WHEN FORMED PARALLEL.
3. DIMENSION B DOES NOT INCLUDE MOLD FLASH.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	22.22	23.24	0.875	0.915
B	6.10	6.60	0.240	0.260
C	3.56	4.57	0.140	0.180
D	0.36	0.56	0.014	0.022
F	1.27	1.78	0.050	0.070
G	2.54 BSC		0.100 BSC	
H	1.02	1.52	0.040	0.060
J	0.20	0.30	0.008	0.012
K	2.92	3.43	0.115	0.135
L	7.62 BSC		0.300 BSC	
M	0°	15°	0°	15°
N	0.51	1.02	0.020	0.040

How to reach us:
USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036. 1–800–441–2447 or 602–303–5454

MFAX: RMFAX0@email.sps.mot.com – TOUCHTONE 602–244–6609
INTERNET: http://Design–NET.com

JAPAN: Nippon Motorola Ltd.; Tatsumi–SPD–JLDC, 6F Seibu–Butsuryu–Center, 3–14–2 Tatsumi Koto–Ku, Tokyo 135, Japan. 03–81–3521–8315

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852–26629298

National Semiconductor

May 2000

LM78XX
Series Voltage Regulators

General Description

The LM78XX series of three terminal regulators is available with several fixed output voltages making them useful in a wide range of applications. One of these is local on card regulation, eliminating the distribution problems associated with single point regulation. The voltages available allow these regulators to be used in logic systems, instrumentation, HiFi, and other solid state electronic equipment. Although designed primarily as fixed voltage regulators these devices can be used with external components to obtain adjustable voltages and currents.

The LM78XX series is available in an aluminum TO-3 package which will allow over 1.0A load current if adequate heat sinking is provided. Current limiting is included to limit the peak output current to a safe value. Safe area protection for the output transistor is provided to limit internal power dissipation. If internal power dissipation becomes too high for the heat sinking provided, the thermal shutdown circuit takes over preventing the IC from overheating.

Considerable effort was expanded to make the LM78XX series of regulators easy to use and minimize the number of external components. It is not necessary to bypass the output, although this does improve transient response. Input bypassing is needed only if the regulator is located far from the filter capacitor of the power supply.

For output voltage other than 5V, 12V and 15V the LM117 series provides an output voltage range from 1.2V to 57V.

Features

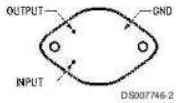
- Output current in excess of 1A
- Internal thermal overload protection
- No external components required
- Output transistor safe area protection
- Internal short circuit current limit
- Available in the aluminum TO-3 package

Voltage Range

LM7805C	5V
LM7812C	12V
LM7815C	15V

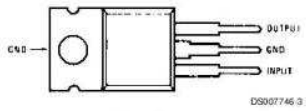
Connection Diagrams

Metal Can Package
TO-3 (K)
Aluminum



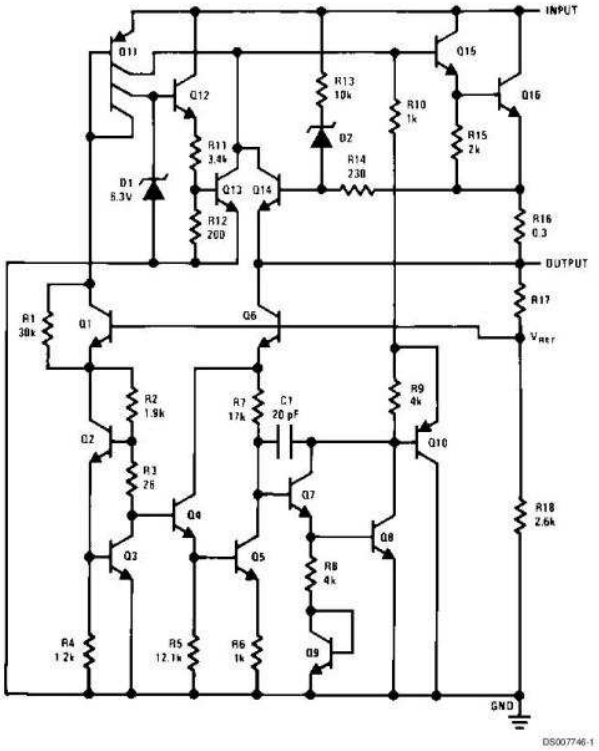
Bottom View
Order Number LM7805CK,
LM7812CK or LM7815CK
See NS Package Number KC02A

Plastic Package
TO-220 (T)



Top View
Order Number LM7805CT,
LM7812CT or LM7815CT
See NS Package Number T03B

Schematic


INPUT
Q11
Q12
Q15
Q16
R13
10k
R10
1k
D2
R14
238
R11
3.4k
R15
2k
D1
6.3V
Q13
Q14
R12
200
R16
0.3
OUTPUT
R17
R1
38k
Q1
Q6
V_{REF}
R2
1.9k
R7
17k
C1
20 pF
R9
4k
Q2
Q10
R3
26
Q7
R18
2.6k
Q4
Q8
Q3
Q5
R8
4k
R4
1.2k
R5
12.1k
R6
1k
Q9
GND
DS007746-1

Absolute Maximum Ratings (Note 3)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Input Voltage (V_O = 5V, 12V and 15V)	35V
Internal Power Dissipation (Note 1)	Internally Limited
Operating Temperature Range (T_A)	0˚C to +70˚C
Maximum Junction Temperature	
(K Package)	150˚C
(T Package)	150˚C
Storage Temperature Range	−65˚C to +150˚C
Lead Temperature (Soldering, 10 sec.)	
TO-3 Package K	300˚C
TO-220 Package T	230˚C

Electrical Characteristics LM78XXC (Note 2)

0˚C ≤ T_J ≤ 125˚C unless otherwise noted.

Output Voltage				5V			12V			15V			Units
Input Voltage (unless otherwise noted)				10V			19V			23V			
Symbol	Parameter	Conditions		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
V_O	Output Voltage	Tj = 25˚C, 5 mA ≤ I_O ≤ 1A		4.8	5	5.2	11.5	12	12.5	14.4	15	15.6	V
		P_D ≤ 15W, 5 mA ≤ I_O ≤ 1A		4.75		5.25	11.4		12.6	14.25		15.75	V
		V_{MIN} ≤ V_{IN} ≤ V_{MAX}		(7.5 ≤ V_{IN} ≤ 20)			(14.5 ≤ V_{IN} ≤ 27)			(17.5 ≤ V_{IN} ≤ 30)			V
ΔV_O	Line Regulation	I_O = 500 mA	Tj = 25˚C		3	50		4	120		4	150	mV
			ΔV_{IN}	(7 ≤ V_{IN} ≤ 25)			14.5 ≤ V_{IN} ≤ 30)			(17.5 ≤ V_{IN} ≤ 30)			V
			0˚C ≤ Tj ≤ +125˚C			50			120			150	mV
			ΔV_{IN}	(8 ≤ V_{IN} ≤ 20)			(15 ≤ V_{IN} ≤ 27)			(18.5 ≤ V_{IN} ≤ 30)			V
		I_O ≤ 1A	Tj = 25˚C			50			120			150	mV
			ΔV_{IN}	(7.5 ≤ V_{IN} ≤ 20)			(14.6 ≤ V_{IN} ≤ 27)			(17.7 ≤ V_{IN} ≤ 30)			V
			0˚C ≤ Tj ≤ +125˚C			25			60			75	mV
			ΔV_{IN}	(8 ≤ V_{IN} ≤ 12)			(16 ≤ V_{IN} ≤ 22)			(20 ≤ V_{IN} ≤ 26)			V
ΔV_O	Load Regulation	Tj = 25˚C	5 mA ≤ I_O ≤ 1.5A		10	50		12	120		12	150	mV
			250 mA ≤ I_O ≤ 750 mA			25			60			75	mV
		5 mA ≤ I_O ≤ 1A, 0˚C ≤ Tj ≤ +125˚C				50			120			150	mV
I_Q	Quiescent Current	I_O ≤ 1A	Tj = 25˚C			8			8			8	mA
			0˚C ≤ Tj ≤ +125˚C			8.5			8.5			8.5	mA
ΔI_Q	Quiescent Current Change	5 mA ≤ I_O ≤ 1A				0.5			0.5			0.5	mA
		Tj = 25˚C, I_O ≤ 1A				1.0			1.0			1.0	mA
		V_{MIN} ≤ V_{IN} ≤ V_{MAX}		(7.5 ≤ V_{IN} ≤ 20)			(14.8 ≤ V_{IN}≤ 27)			(17.9 ≤ V_{IN} ≤ 30)			V
		I_O ≤ 500 mA, 0˚C ≤ Tj ≤ +125˚C				1.0			1.0			1.0	mA
		V_{MIN} ≤ V_{IN} ≤ V_{MAX}		(7 ≤ V_{IN} ≤ 25)			(14.5 ≤ V_{IN}≤ 30)			(17.5 ≤ V_{IN} ≤ 30)			V
V_N	Output Noise Voltage	T_A =25˚C, 10 Hz ≤ f ≤ 100 kHz			40			75			90		µV
$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	f = 120 Hz	I_O ≤ 1A, Tj = 25˚C or	62	80		55	72		54	70		dB
			I_O ≤ 500 mA 0˚C ≤ Tj ≤ +125˚C	62			55			54			dB
		V_{MIN} ≤ V_{IN} ≤ V_{MAX}		(8 ≤ V_{IN} ≤ 18)			(15 ≤ V_{IN} ≤ 25)			(18.5 ≤ V_{IN} ≤ 28.5)			V
R_O	Dropout Voltage	Tj = 25˚C, I_{OUT} = 1A			2.0			2.0			2.0		V
	Output Resistance	f = 1 kHz			8			18			19		mΩ

Electrical Characteristics LM78XXC (Note 2) (Continued)

$0^{\circ}C \le T_J \le 125^{\circ}C$ unless otherwise noted.

Output Voltage			5V			12V			15V			Units
Input Voltage (unless otherwise noted)			10V			19V			23V			
Symbol	Parameter	Conditions	Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
	Short-Circuit Current	Tj = 25˚C		2.1			1.5			1.2		A
	Peak Output Current	Tj = 25˚C		2.4			2.4			2.4		A
	Average TC of V_{OUT}	0˚C ≤ Tj ≤ +125˚C, I_O = 5 mA		0.6			1.5			1.8		mV/˚C
V_{IN}	Input Voltage Required to Maintain Line Regulation	Tj = 25˚C, $I_O \le 1A$		7.5		14.6			17.7			V

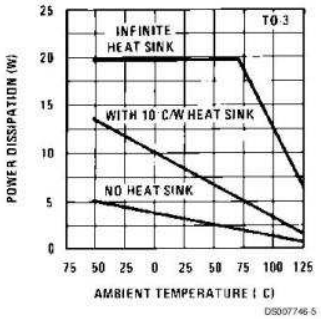
Note 1: Thermal resistance of the TO-3 package (K, KC) is typically 4˚C/W junction to case and 35˚C/W case to ambient. Thermal resistance of the TO-220 package (T) is typically 4˚C/W junction to case and 50˚C/W case to ambient.

Note 2: All characteristics are measured with capacitor across the input of 0.22 µF, and a capacitor across the output of 0.1 µF. All characteristics except noise voltage and ripple rejection ratio are measured using pulse techniques ($t_w \le 10$ ms, duty cycle ≤ 5%). Output voltage changes due to changes in internal temperature must be taken into account separately.

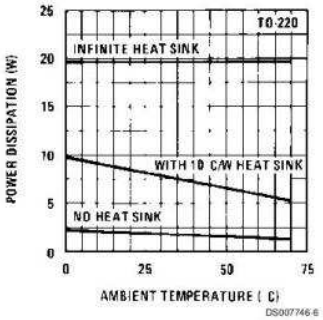
Note 3: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. For guaranteed specifications and the test conditions, see Electrical Characteristics.

Typical Performance Characteristics

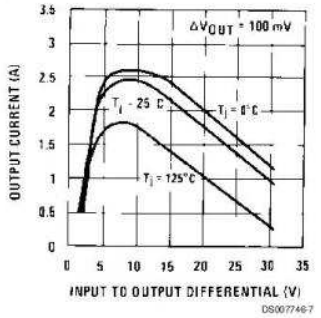
Maximum Average Power Dissipation



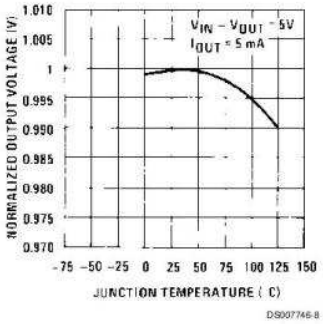
Maximum Average Power Dissipation



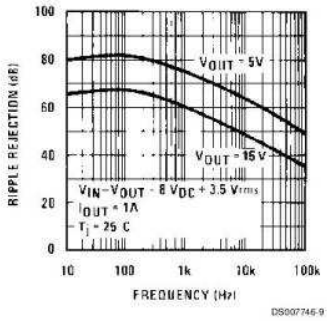
Peak Output Current



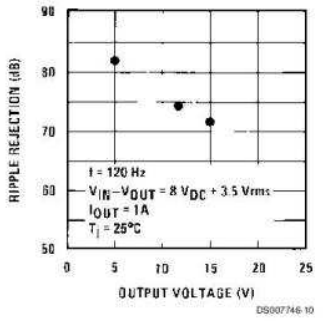
Output Voltage (Normalized to 1V at T_J = 25°C)



Ripple Rejection

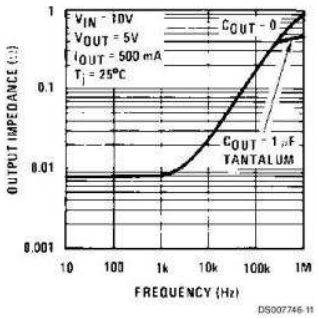


Ripple Rejection

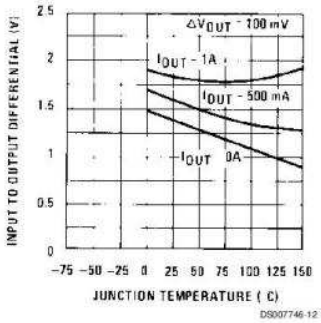


Typical Performance Characteristics (Continued)

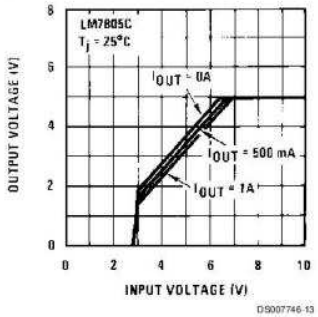
Output Impedance



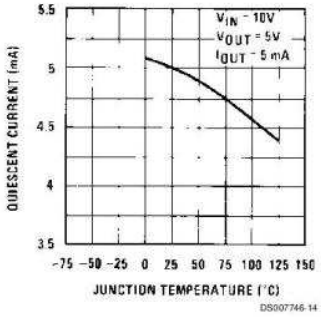
Dropout Voltage



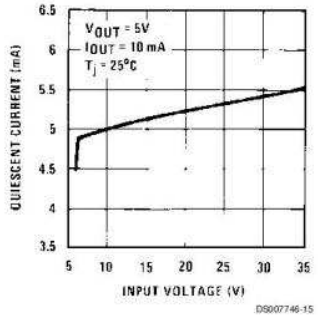
Dropout Characteristics



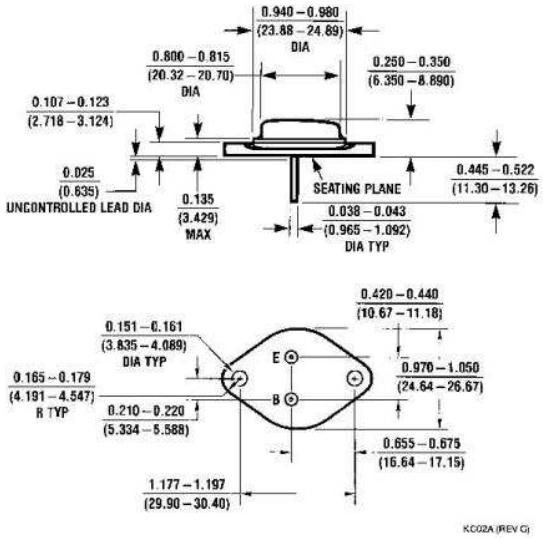
Quiescent Current



Quiescent Current

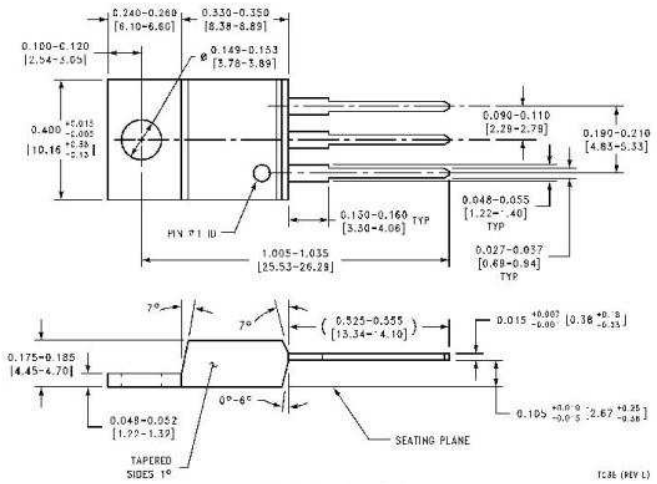


Physical Dimensions inches (millimeters) unless otherwise noted



Aluminum Metal Can Package (KC)
Order Number LM7805CK, LM7812CK or LM7815CK
NS Package Number KC02A

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



TO-220 Package (T)
Order Number LM7805CT, LM7812CT or LM7815CT
NS Package Number T03B

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
Americas
Tel: 1-800-272-9959
Fax: 1-800-737-7018
Email: support@nsc.com
www.national.com

National Semiconductor Europe
Fax: +49 (0) 180-530 85 86
Email: europe.support@nsc.com
Deutsch Tel: +49 (0) 69 9508 6208
English Tel: +44 (0) 870 24 0 2171
Français Tel: +33 (0) 1 41 91 8790

National Semiconductor Asia Pacific Customer Response Group
Tel: 65-2544466
Fax: 65-2504466
Email: ap.support@nsc.com

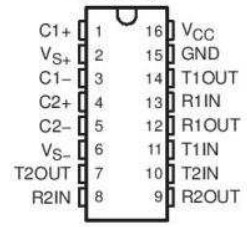
National Semiconductor Japan Ltd.
Tel: 81-3-5639-7560
Fax: 81-3-5639-7507

MAX232, MAX232I
DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047L – FEBRUARY 1989 – REVISED MARCH 2004

- Meets or Exceeds TIA/EIA-232-F and ITU Recommendation V.28
- Operates From a Single 5-V Power Supply With 1.0-μF Charge-Pump Capacitors
- Operates Up To 120 kbit/s
- Two Drivers and Two Receivers
- ±30-V Input Levels
- Low Supply Current . . . 8 mA Typical
- ESD Protection Exceeds JESD 22
  - 2000-V Human-Body Model (A114-A)
- Upgrade With Improved ESD (15-kV HBM) and 0.1-μF Charge-Pump Capacitors is Available With the MAX202
- Applications
  - TIA/EIA-232-F, Battery-Powered Systems, Terminals, Modems, and 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 TIA/EIA-232-F voltage levels from a single 5-V supply. Each receiver converts TIA/EIA-232-F inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V, a typical hysteresis of 0.5 V, and can accept ±30-V inputs. Each driver converts TTL/CMOS input levels into TIA/EIA-232-F 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 of 25	MAX232N	MAX232N
	SOIC (D)	Tube of 40	MAX232D	MAX232
		Reel of 2500	MAX232DR	
	SOIC (DW)	Tube of 40	MAX232DW	MAX232
		Reel of 2000	MAX232DWR	
	SOP (NS)	Reel of 2000	MAX232NSR	MAX232
–40°C to 85°C	PDIP (N)	Tube of 25	MAX232IN	MAX232IN
	SOIC (D)	Tube of 40	MAX232ID	MAX232I
		Reel of 2500	MAX232IDR	
	SOIC (DW)	Tube of 40	MAX232IDW	MAX232I
		Reel of 2000	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

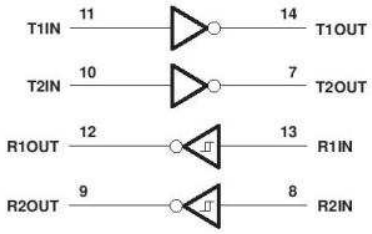
H = high level, L = low level

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 Notes 2 and 3): D package . . . . . 73°C/W
DW package . . . . . 57°C/W
N package . . . . . 67°C/W
NS package . . . . . 64°C/W
Operating virtual junction temperature, T_J . . . . . 150°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.

NOTES: 1. All voltages are with respect to network GND.
2. Maximum power dissipation is a function of $T_J(max)$, θ_{JA}, and T_A. The maximum allowable power dissipation at any allowable ambient temperature is $P_D = (T_J(max) - T_A)/\theta_{JA}$. Operating at the absolute maximum T_J of 150°C can affect reliability.
3. 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 4 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 4: 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 4)

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 4: 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 4)

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 4: 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 4)

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

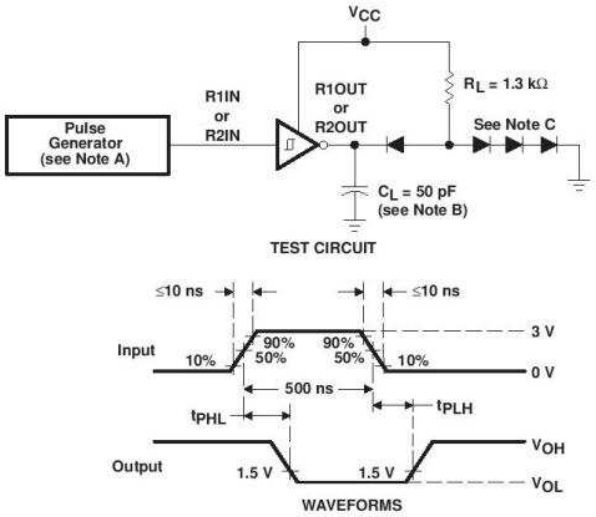
† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ The algebraic convention, in which the least-positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels only.
NOTE 4: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

switching characteristics, V_{CC} = 5 V, T_A = 25°C (see Note 4 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 4: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

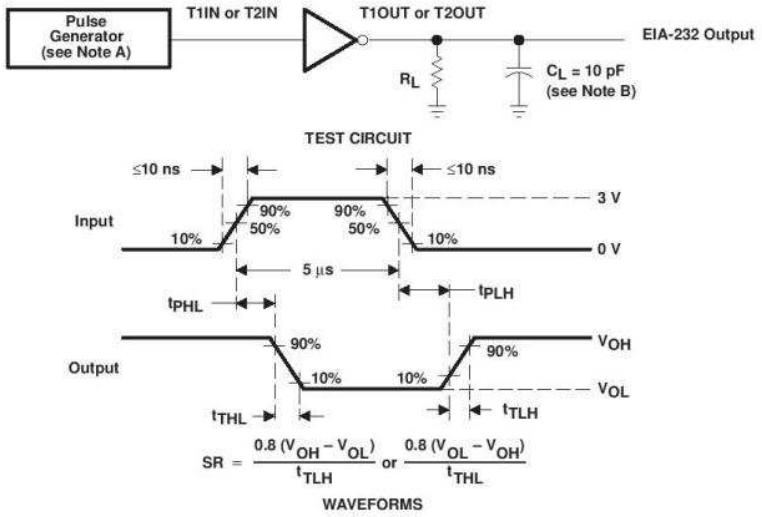
PARAMETER MEASUREMENT INFORMATION



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

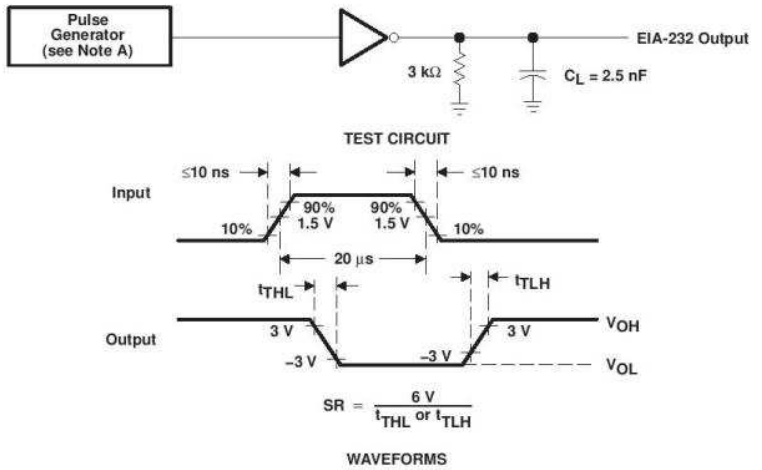
Figure 1. Receiver Test Circuit and Waveforms for t_{PHL} and t_{PLH} Measurements

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.

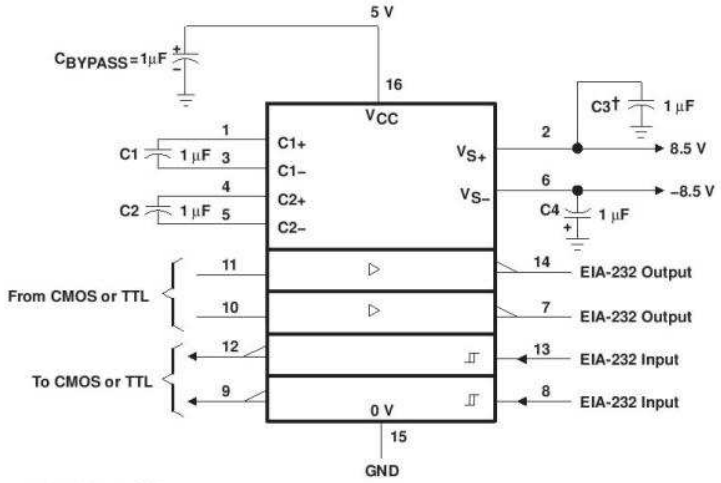
Figure 2. Driver Test Circuit and Waveforms for t_{PHL} and t_{PLH} Measurements (5-µs Input)



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.

NOTES: A. Resistor values shown are nominal.

B. Nonpolarized ceramic capacitors are acceptable. If polarized tantalum or electrolytic capacitors are used, they should be connected as shown. In addition to the 1-μF capacitors shown, the MAX202 can operate with 0.1-μF capacitors.

Figure 4. Typical Operating Circuit

TEXAS INSTRUMENTS
www.ti.com

PACKAGE OPTION ADDENDUM

18-Jul-2006

PACKAGING INFORMATION

Orderable Device	Status (1)	Package Type	Package Drawing	Pins	Package Qty	Eco Plan (2)	Lead/Ball Finish	MSL Peak Temp (3)
MAX232D	ACTIVE	SOIC	D	16	40	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232DE4	ACTIVE	SOIC	D	16	40	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232DR	ACTIVE	SOIC	D	16	2500	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232DRE4	ACTIVE	SOIC	D	16	2500	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232DW	ACTIVE	SOIC	DW	16	40	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232DWE4	ACTIVE	SOIC	DW	16	40	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232DWR	ACTIVE	SOIC	DW	16	2000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232DWRE4	ACTIVE	SOIC	DW	16	2000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232ID	ACTIVE	SOIC	D	16	40	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232IDE4	ACTIVE	SOIC	D	16	40	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232IDR	ACTIVE	SOIC	D	16	2500	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232IDRE4	ACTIVE	SOIC	D	16	2500	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232IDW	ACTIVE	SOIC	DW	16	40	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232IDWE4	ACTIVE	SOIC	DW	16	40	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232IDWG4	ACTIVE	SOIC	DW	16	40	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232IDWR	ACTIVE	SOIC	DW	16	2000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232IDWRE4	ACTIVE	SOIC	DW	16	2000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232IDWRG4	ACTIVE	SOIC	DW	16	2000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232IN	ACTIVE	PDIP	N	16	25	Pb-Free (RoHS)	CU NIPDAU	N / A for Pkg Type
MAX232INE4	ACTIVE	PDIP	N	16	25	Pb-Free (RoHS)	CU NIPDAU	N / A for Pkg Type
MAX232N	ACTIVE	PDIP	N	16	25	Pb-Free (RoHS)	CU NIPDAU	N / A for Pkg Type
MAX232NE4	ACTIVE	PDIP	N	16	25	Pb-Free (RoHS)	CU NIPDAU	N / A for Pkg Type
MAX232NSR	ACTIVE	SO	NS	16	2000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
MAX232NSRE4	ACTIVE	SO	NS	16	2000	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM

(1) The marketing status values are defined as follows:

ACTIVE: Product device recommended for new designs.
LIFEBUY: TI has announced that the device will be discontinued, and a lifetime-buy period is in effect.
NRND: Not recommended for new designs. Device is in production to support existing customers, but TI does not recommend using this part in a new design.
PREVIEW: Device has been announced but is not in production. Samples may or may not be available.
OBSOLETE: TI has discontinued the production of the device.

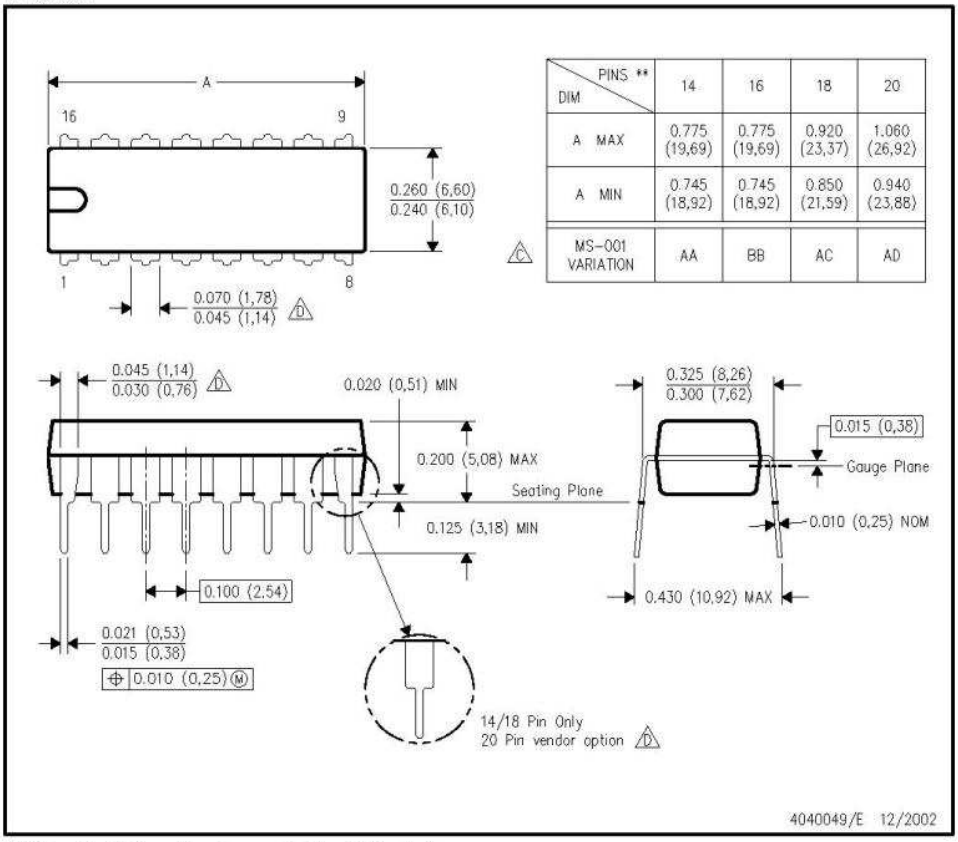
(2) Eco Plan - The planned eco-friendly classification: Pb-Free (RoHS), Pb-Free (RoHS Exempt), or Green (RoHS & no Sb/Br) - please check http://www.ti.com/productcontent for the latest availability information and additional product content details.
TBD: The Pb-Free/Green conversion plan has not been defined.
Pb-Free (RoHS): TI's terms "Lead-Free" or "Pb-Free" mean semiconductor products that are compatible with the current RoHS requirements for all 6 substances, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, TI Pb-Free products are suitable for use in specified lead-free processes.
Pb-Free (RoHS Exempt): This component has a RoHS exemption for either 1) lead-based flip-chip solder bumps used between the die and package, or 2) lead-based die adhesive used between the die and leadframe. The component is otherwise considered Pb-Free (RoHS compatible) as defined above.
Green (RoHS & no Sb/Br): TI defines "Green" to mean Pb-Free (RoHS compatible), and free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material)

(3) MSL, Peak Temp. -- The Moisture Sensitivity Level rating according to the JEDEC industry standard classifications, and peak solder temperature.

N (R–PDIP–T**)

16 PINS SHOWN

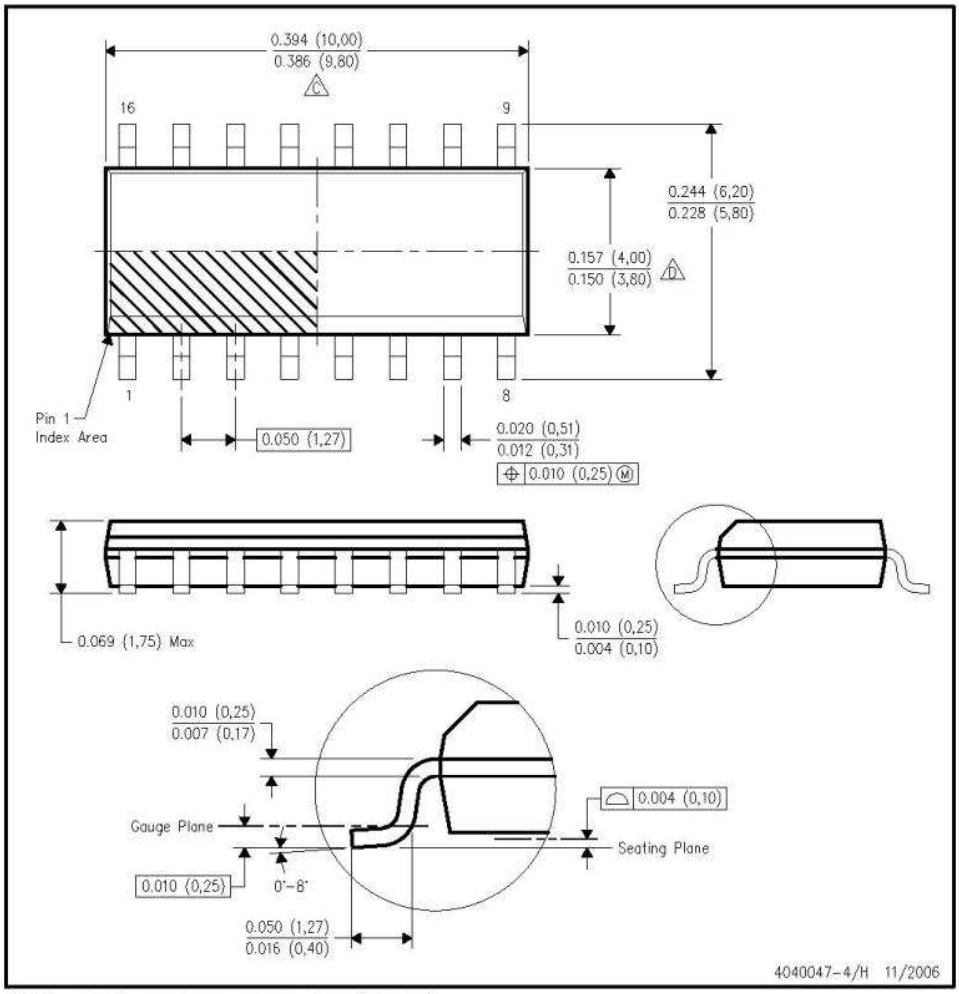
PLASTIC DUAL–IN–LINE PACKAGE



DIM \ PINS **	14	16	18	20
A MAX	0.775 (19,69)	0.775 (19,69)	0.920 (23,37)	1.060 (26,92)
A MIN	0.745 (18,92)	0.745 (18,92)	0.850 (21,59)	0.940 (23,88)
MS–001 VARIATION	AA	BB	AC	AD

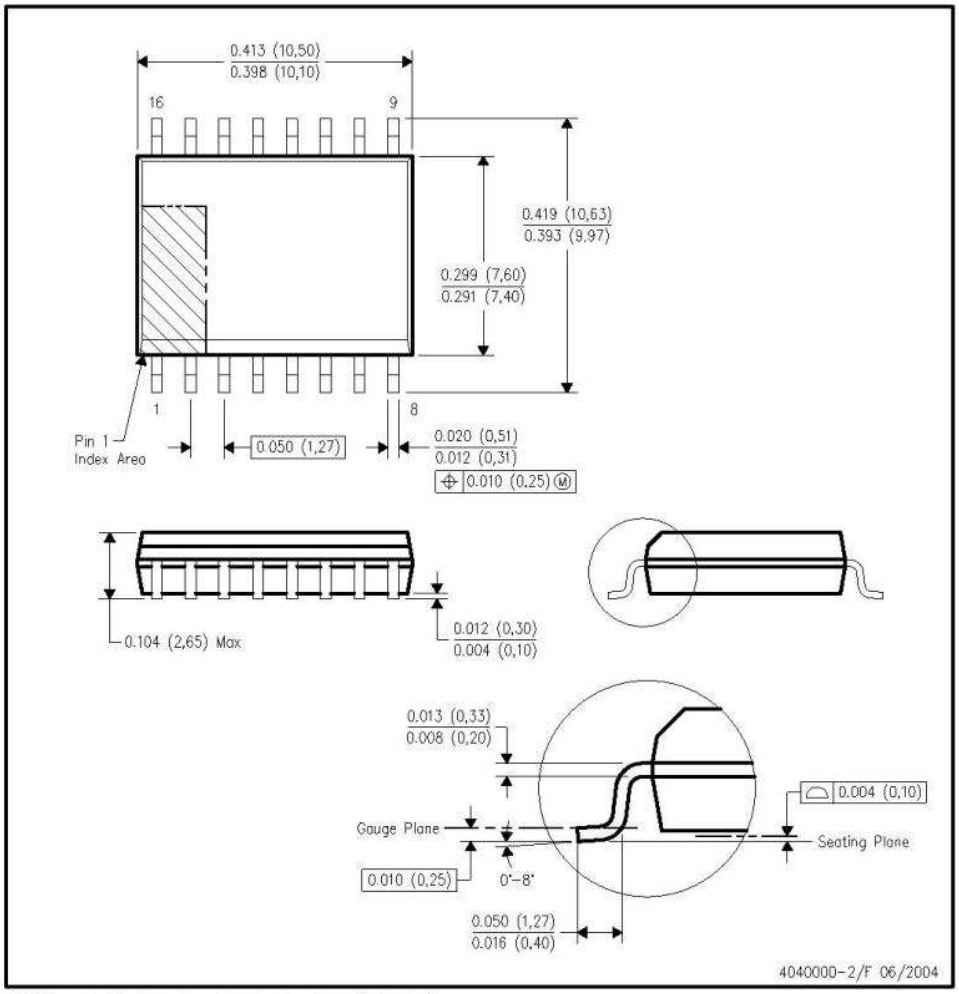
NOTES:
A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS–001, except 18 and 20 pin minimum body length (Dim A).
D. The 20 pin end lead shoulder width is a vendor option, either half or full width.

D (R–PDSO–G16) PLASTIC SMALL–OUTLINE PACKAGE



NOTES:
A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Body length does not include mold flash, protrusions, or gate burrs. Mold flash, protrusions, or gate burrs shall not exceed .006 (0,15) per end.
D. Body width does not include interlead flash. Interlead flash shall not exceed .017 (0,43) per side.
E. Reference JEDEC MS–012 variation AC.

DW (R–PDSO–G16) **PLASTIC SMALL–OUTLINE PACKAGE**

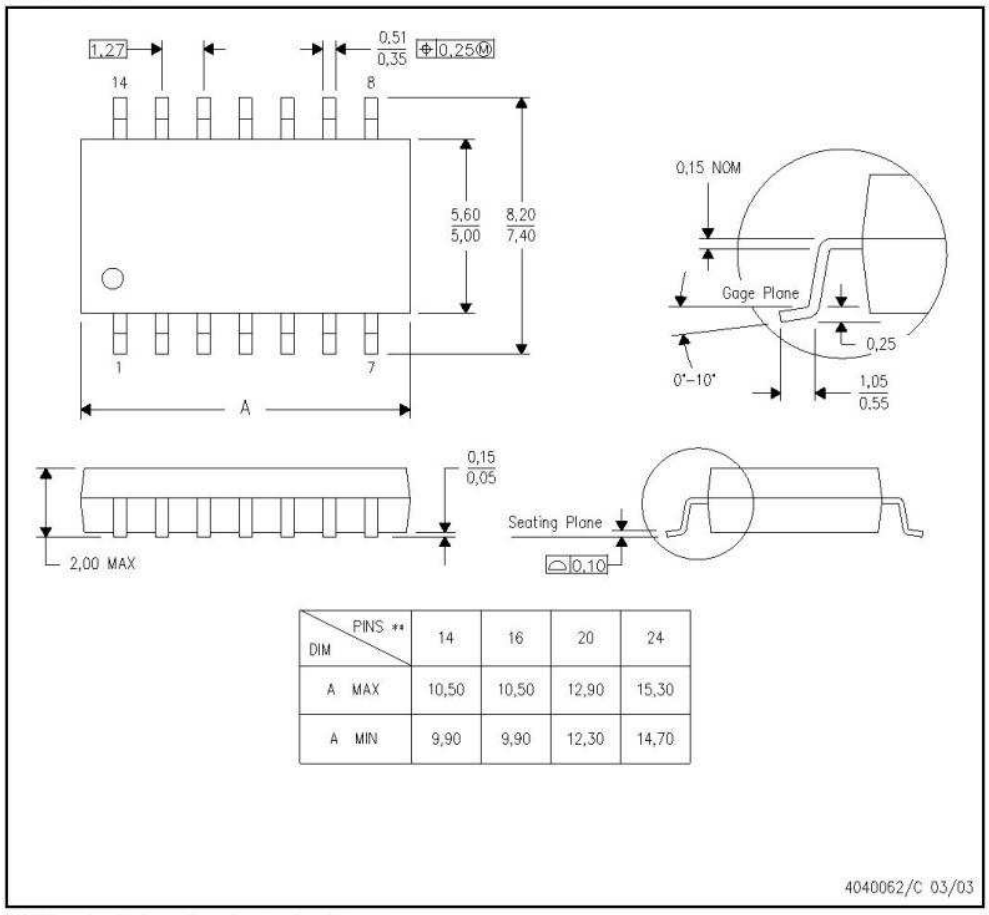


NOTES:
A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Body dimensions do not include mold flash or protrusion not to exceed 0.006 (0,15).
D. Falls within JEDEC MS–013 variation AA.

MECHANICAL DATA

NS (R-PDSO-G)** **PLASTIC SMALL-OUTLINE PACKAGE**

14-PINS SHOWN



DIM \ PINS **	14	16	20	24
A MAX	10,50	10,50	12,90	15,30
A MIN	9,90	9,90	12,30	14,70

NOTES:
A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.
C. Body dimensions do not include mold flash or protrusion, not to exceed 0,15.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

ANEXO D

PROGRAMA DEL MICROCONTROLADOR

```
.include        "m16def.inc"


.equ    lcd_ddr         =       ddrc

.equ    lcd_port        =       portc                   ;LCD
Data port

.equ    lcd_in          =       pinc


.equ    lcd_rs          =       0
        ;LCD Register Select

.equ    lcd_en          =       1
        ;LCD Enable

.equ    lcd_d4          =       2

.equ    lcd_d5          =       3

.equ    lcd_d6          =       4

.equ    lcd_d7          =       5


.def    temp0           =       r16
        ;usado como auxiliar para todo el programa en general

.def    temp1           =       r17
        ;usado como auxiliar para todo el programa en general

.def    temp2           =       r18
        ;usado como auxiliar para todo el programa en general


.def    flag            =       r19

.def    focos           =       r24

.def    sensores        =       r25


.equ    mSeg20                  =       55292;55500
        ;constante para un retardo de 20 milisegundos

.equ    mSeg15                  =       41468;41472
        ;constante para un retardo de 15 milisegundos

.equ    mSeg10                  =       27644;27648
        ;constante para un retardo de 10 milisegundos

.equ    uSeg480                 =       1323;1325
                ;constante para un retardo de 480 microsegundos
```

```
.equ    uSeg50                  =       134;135
        ;constante para un retardo de 50 microsegundos

.equ    uSeg20                  =       51;51
        ;constante para un retardo de 20 microsegundos

.equ    uSeg19                  =       49
                ;constante para un retardo de 19 microsegundos

.equ    uSeg18                  =       46
                ;constante para un retardo de 18 microsegundos

.equ    uSeg17_5                =       44;44.384
        ;constante para un retardo de 17.5 microsegundos

.equ    uSeg17                  =       43;43
        ;constante para un retardo de 17 microsegundos

.equ    uSeg16                  =       40
                ;constante para un retardo de 16 microsegundos

.equ    uSeg15                  =       38;38
        ;constante para un retardo de 15 microsegundos

.equ    uSeg10                  =       24;25
        ;constante para un retardo de 10 microsegundos


.dseg

lcd_status:             .byte       1
        ;display on/off, cursor on/off, blinks on/off==> 0b00001dcb

lcd_salida:             .byte       1
        ;valor que va al puerto

;*****************************************************************************************

;**********************************Inicio del programa**********************************

;*****************************************************************************************


.cseg

                        .org        0x0000
        ;primera direccion, donde el micro comienza a leer

                        jmp             reset


                        .org        0x0000C

                        jmp             TIM1_COMPA
                ;Timer1 CompareA Handler
```

```
                .org    0x0016
                jmp     USART_RXC
        ;USART RX Complete Handler

                .org    0x001A
                jmp     USART_TXC
        ;USART TX Complete Handler

                ;Configuracion del funcionamiento del uC
                .org    0x002A
reset:  ldi     temp0,high(ramend)
        ;inicializo el puntero del stack
                out     sph,temp0
                ldi     temp0,low(ramend)
                out     spl,temp0

                ldi     temp0,10
sup_retardo:
                call    wait_20m
                dec     temp0
                brne    sup_retardo

                ldi     temp0,0b11111111
                out     ddra,temp0
                out     portb,temp0
                out     ddrc,temp0
                ldi     temp0,0b00100010
```

```
                out             ddrd,temp0

                ldi             temp0,high(143)
                out             ubrrh,temp0
                ldi             temp0,low(143)
                out             ubrrl,temp0
                ldi             temp0,0b00000010
                out             ucsra,temp0
                ldi             temp0,0b10011000
                out             ucsrb,temp0
                ldi             temp0,0b10000110
                out             ucsrc,temp0

                ldi             temp0,0b01000000
                out             tccr1a,temp0
                ldi             temp0,0b00001101
                out             tccr1b,temp0
                ldi             temp0,high(2700)
                out             ocr1ah,temp0
                ldi             temp0,low(2700)
                out             ocr1al,temp0
                ldi             temp0,0b00010000
                out             timsk,temp0
                clr             focos

int_on:         sei
        ;enciendo las interrupciones

;*************************************************************************************
;**********************************Programa Principal*********************************
```

```
;*****************************************************************************

main:
                        cpi             flag,1
                        brne            main
                        clr             flag
                        call            setup_lcd
                        //ser           focos
FocosLcd:
                        ldi             temp0,0x83
                ;situo el cursor sobre el decimo primer caracter de la primera linea
                        call            lcd_command
                        out             porta,focos
Foco7:          mov                     temp0,focos
                        andi            temp0,0x80
                        brne            PC+6
                        ldi             temp0,'0'
                        call            lcd_senddata
                        jmp             PC+5
                        ldi             temp0,'1'
                        call            lcd_senddata
Foco6:          mov                     temp0,focos
                        andi            temp0,0x40
                        brne            PC+6
                        ldi             temp0,'0'
                        call            lcd_senddata
                        jmp             PC+5
                        ldi             temp0,'1'
                        call            lcd_senddata
Foco5:          mov                     temp0,focos
```

```
                andi    temp0,0x20
                brne    PC+6
                ldi             temp0,'0'
                call    lcd_senddata
                jmp             PC+5
                ldi             temp0,'1'
                call    lcd_senddata
Foco4:          mov             temp0,focos
                andi    temp0,0x10
                brne    PC+6
                ldi             temp0,'0'
                call    lcd_senddata
                jmp             PC+5
                ldi             temp0,'1'
                call    lcd_senddata
Foco3:          mov             temp0,focos
                andi    temp0,0x08
                brne    PC+6
                ldi             temp0,'0'
                call    lcd_senddata
                jmp             PC+5
                ldi             temp0,'1'
                call    lcd_senddata
Foco2:          mov             temp0,focos
                andi    temp0,0x04
                brne    PC+6
                ldi             temp0,'0'
                call    lcd_senddata
                jmp             PC+5
                ldi             temp0,'1'
```

```
                    call    lcd_senddata
Foco1:      mov             temp0,focos
                    andi    temp0,0x02
                    brne    PC+6
                    ldi             temp0,'0'
                    call    lcd_senddata
                    jmp             PC+5
                    ldi             temp0,'1'
                    call    lcd_senddata
Foco0:      mov             temp0,focos
                    andi    temp0,0x01
                    brne    PC+6
                    ldi             temp0,'0'
                    call    lcd_senddata
                    jmp             PC+5
                    ldi             temp0,'1'
                    call    lcd_senddata


SensoresLcd:
                    ldi             temp0,0xC3
          ;situo el cursor sobre el decimo primer caracter de la primera linea
                    call    lcd_command
                    in              sensores,pinb
                    out             udr,sensores

Sens7:      mov             temp0,sensores
                    andi    temp0,0x80
                    brne    PC+6
                    ldi             temp0,'0'
```

```
                call    lcd_senddata
                jmp     PC+5
                ldi     temp0,'1'
                call    lcd_senddata
Sens6:  mov     temp0,sensores
                andi    temp0,0x40
                brne    PC+6
                ldi     temp0,'0'
                call    lcd_senddata
                jmp     PC+5
                ldi     temp0,'1'
                call    lcd_senddata
Sens5:  mov     temp0,sensores
                andi    temp0,0x20
                brne    PC+6
                ldi     temp0,'0'
                call    lcd_senddata
                jmp     PC+5
                ldi     temp0,'1'
                call    lcd_senddata
Sens4:  mov     temp0,sensores
                andi    temp0,0x10
                brne    PC+6
                ldi     temp0,'0'
                call    lcd_senddata
                jmp     PC+5
                ldi     temp0,'1'
                call    lcd_senddata
Sens3:  mov     temp0,sensores
                andi    temp0,0x08
```

```
                brne    PC+6
                ldi     temp0,'0'
                call    lcd_senddata
                jmp     PC+5
                ldi     temp0,'1'
                call    lcd_senddata
Sens2:  mov     temp0,sensores
                andi    temp0,0x04
                brne    PC+6
                ldi     temp0,'0'
                call    lcd_senddata
                jmp     PC+5
                ldi     temp0,'1'
                call    lcd_senddata
Sens1:  mov     temp0,sensores
                andi    temp0,0x02
                brne    PC+6
                ldi     temp0,'0'
                call    lcd_senddata
                jmp     PC+5
                ldi     temp0,'1'
                call    lcd_senddata
Sens0:  mov     temp0,sensores
                andi    temp0,0x01
                brne    PC+6
                ldi     temp0,'0'
                call    lcd_senddata
                jmp     PC+5
                ldi     temp0,'1'
                call    lcd_senddata
```

```
                    ldi         temp0,0x0d
                    out         udr,temp0
                    jmp         main
;-----------------------------------------------------------------------------

TIM1_COMPA:
                    ldi         flag,1
                    reti
;-----------------------------------------------------------------------------

USART_RXC:
                    in          focos,udr
                    reti
;-----------------------------------------------------------------------------

USART_TXC:
                    reti
;-----------------------------------------------------------------------------

wait_20m:           ldi         yh,high(mSeg20)
        ;subrutina de retardo de 40mSeg
                    ldi         yl,low(mSeg20)
        ;copio la constante correspondiente
                    jmp         wait_us
            ;salto al retardo
wait_15m:           ldi         yh,high(mSeg15)
        ;subrutina de retardo de 30mSeg
                    ldi         yl,low(mSeg15)
        ;copio la constante correspondiente
                    jmp         wait_us
            ;salto al retardo
wait_10m:           ldi         yh,high(mSeg10)
        ;subrutina de retardo de 10mSeg
```

```
                    ldi         yl,low(mSeg10)
          ;copio la constante correspondiente

                    jmp         wait_us
               ;salto al retardo

wait_480u:          ldi         yh,high(uSeg480)
     ;subrutina de retardo de 480uSeg

                    ldi         yl,low(uSeg480)
          ;copio la constante correspondiente

                    jmp         wait_us
               ;salto al retardo

wait_50u:           ldi         yh,high(uSeg50)
     ;subrutina de retardo de 50uSeg

                    ldi         yl,low(uSeg50)
          ;copio la constante correspondiente

                    nop
               ;retardo de ajuste

                    nop
               ;retardo de ajuste

                    jmp         wait_us
               ;salto al retardo

wait_20u:           ldi         yh,high(uSeg20)
     ;subrutina de retardo de 20uSeg

                    ldi         yl,low(uSeg20)
          ;copio la constante correspondiente

                    jmp         wait_us
               ;salto al retardo

wait_19u:           ldi         yh,high(uSeg19)
     ;subrutina de retardo de 20uSeg

                    ldi         yl,low(uSeg19)
          ;copio la constante correspondiente

                    jmp         wait_us
               ;salto al retardo

wait_18u:           ldi         yh,high(uSeg18)
     ;subrutina de retardo de 20uSeg

                    ldi         yl,low(uSeg18)
          ;copio la constante correspondiente

                    jmp         wait_us
               ;salto al retardo
```

```
wait_17u5:          ldi         yh,high(uSeg17_5)
                ;subrutina de retardo de 20uSeg

                    ldi         yl,low(uSeg17_5)
                ;copio la constante correspondiente

                    jmp         wait_us
                ;salto al retardo

wait_17u:           ldi         yh,high(uSeg17)
        ;subrutina de retardo de 20uSeg

                    ldi         yl,low(uSeg17)
                ;copio la constante correspondiente

                    jmp         wait_us
                ;salto al retardo

wait_16u:           ldi         yh,high(uSeg16)
        ;subrutina de retardo de 20uSeg

                    ldi         yl,low(uSeg16)
                ;copio la constante correspondiente

                    jmp         wait_us
                ;salto al retardo

wait_15u:           ldi         yh,high(uSeg15)
        ;subrutina de retardo de 20uSeg

                    ldi         yl,low(uSeg15)
                ;copio la constante correspondiente

                    jmp         wait_us
                ;salto al retardo

wait_10u:           ldi         yh,high(uSeg10)
        ;subrutina de retardo de 10uSeg

                    ldi         yl,low(uSeg10)
                ;copio la constante correspondiente

wait_us:            sbiw        y,1

                    brne        wait_us

                    ret


;**********************************Subrutinas del LCD************************************

setup_lcd:          call        lcd_init                    ;subrutina par
ainicializar la pantalla lcd

                    call        lcd_cursor_off              ;apago
el cursor del lcd
```

```
                    ldi             temp0,0x80
                    call            lcd_command
        ;situo el cursor sobre el primer caracter de la primera linea
                    ldi             temp0,'O'
                    call            lcd_senddata
        ;escribo la letra T
                    ldi             temp0,':'
                    call            lcd_senddata
        ;escribo dos puntos
                    ldi             temp0,' '
                    call            lcd_senddata
        ;escribo un espacio


                    ldi             temp0,0xC0
            ;situo el cursor sobre el decimo primer caracter de la primera linea
                    call            lcd_command
                    ldi             temp0,'I'
                    call            lcd_senddata
        ;escribo la letra N
                    ldi             temp0,':'
                    call            lcd_senddata
        ;escribo dos puntos
                    ldi             temp0,' '
                    call            lcd_senddata
        ;escribo un espacio
                    ret
;************************************inicializacion**************************************


lcd_init:           call            lcd_reset
                        clr             temp0
                        sts             lcd_status,temp0
                        ldi             temp0,0x28
        ;Function set: 4bits, 2 Line, 8-bit, 5x7 dots
```

```
                    call    lcd_command                 ;envie
el comando
                    call    lcd_on
                    call    lcd_cursor_on
                    call    lcd_cls
                    call    lcd_entry
                    ldi             temp0,0x80
                    call    lcd_command
                    ret
        ;Return from routine
;*************************************Reseteo del LCD*****************************************

lcd_reset:          ldi     temp0,255
                    out     lcd_port,temp0
                    call    wait_10m
                    call    wait_10m
                    ldi     temp0,0x03
                    cbi     lcd_port,LCD_rs     ;Selected command
register
                    bst     temp0,3
                    bld     temp1,lcd_d7
                    bst     temp0,2
                    bld     temp1,lcd_d6
                    bst     temp0,1
                    bld     temp1,lcd_d5
                    bst     temp0,0
                    bld     temp1,lcd_d4
                    out     lcd_port,temp1
                    sbi     lcd_port,lcd_en     ;Enable H->L
                    call    wait_50u
                    cbi     lcd_port,lcd_en
```

```
                call    wait_15m
                call    wait_15m
                sbi             lcd_port,lcd_en     ;Enable H->L
                call    wait_50u
                cbi             lcd_port,lcd_en
                call    wait_10m
                sbi             lcd_port,lcd_en     ;Enable H->L
                call    wait_50u
                cbi             Lcd_port,lcd_en
                call    wait_10m
                ldi             temp0,0x02                          ;indico
4 lineas de datos
                cbi             lcd_port,lcd_rs     ;Selected command register
                bst             temp0,3
                bld             temp1,lcd_d7
                bst             temp0,2
                bld             temp1,lcd_d6
                bst             temp0,1
                bld             temp1,lcd_d5
                bst             temp0,0
                bld             temp1,lcd_d4
                out             lcd_port,temp1
                sbi             lcd_port,lcd_en     ;Enable H->L
                call    wait_50u
                cbi             Lcd_port,lcd_en
                call    wait_10m
                ret
;*************************************Comandos del
LCD*************************************
```

```
lcd_on:             lds     temp0,lcd_status
                    ori     temp0,0b00001100
                    sts     lcd_status,temp0
                    jmp     lcd_command

lcd_off:        lds     temp0,lcd_status
                    andi    temp0,0b11111011
                    sts     lcd_status,temp0
                    jmp     lcd_command

lcd_cursor_on:      lds     temp0,lcd_status
                    ori     temp0,0b00001010
                    sts     lcd_status,temp0
                    jmp     lcd_command

lcd_cursor_off:     lds     temp0,lcd_status
                    andi    temp0,0b11111101
                    sts     lcd_status,temp0
                    jmp     lcd_command

lcd_blink_on:       lds     temp0,lcd_status
                    ori     temp0,0b00001101
                    sts     lcd_status,temp0
                    jmp     lcd_command

lcd_blink_off:      lds     temp0,lcd_status
                    andi    temp0,0b11111110
                    sts     lcd_status,temp0
                    jmp     lcd_command
```

```
lcd_entry:          ldi     temp0,0x06          ;Entry mode,
auto increment with no shift, puede

        ;tener valores de 0x04, 0x05, 0x06, 0x07
                    jmp     lcd_command

lcd_home:           ldi     TEMP0,0X02
                    jmp     lcd_command

lcd_shift_r:        ldi     temp0,0x1c
                    jmp     lcd_command

lcd_shift_l:        ldi     temp0,0x18
                    jmp     lcd_command

lcd_cursor_r:       ldi     temp0,0x14
                    jmp     lcd_command

lcd_cursor_l:       ldi     temp0,0x10
                    jmp     lcd_command

lcd_cls:            ldi     TEMP0,0X01          ;Clear LCD
                    jmp     lcd_command
;**********************************Envio de comandos al
LCD**********************************

LCD_command:        clr     temp1
                    cbi     lcd_port,lcd_rs     ;Selected command register
                    bst     temp0,7
                    bld     temp1,lcd_d7
                    bst     temp0,6
```

```
                bld     temp1,lcd_d6
                bst     temp0,5
                bld     temp1,lcd_d5
                bst     temp0,4
                bld     temp1,lcd_d4
                out     lcd_port,temp1
                sbi     lcd_port,lcd_en     ;Enable H->L
                call    wait_50u
                cbi     lcd_port,lcd_en
                bst     temp0,3
                bld     temp1,lcd_d7
                bst     temp0,2
                bld     temp1,lcd_d6
                bst     temp0,1
                bld     temp1,lcd_d5
                bst     temp0,0
                bld     temp1,lcd_d4
                out     lcd_port,temp1
                sbi     lcd_port,lcd_en     ;Enable H->L
                call    wait_50u
                cbi     lcd_port,lcd_en
                call    wait_480u                   ;8ms
Wait for LCD to process the command
                call    wait_480u                   ;Wait
for LCD to process the command
                call    wait_480u                   ;Wait
for LCD to process the command
                call    wait_480u                   ;Wait
for LCD to process the command

                        ret
        ;Return from command routine
```

```
;************************************Envio de datos al
LCD************************************

LCD_senddata:       clr     temp1
                    bst     temp0,7
                    bld     temp1,lcd_d7
                    bst     temp0,6
                    bld     temp1,lcd_d6
                    bst     temp0,5
                    bld     temp1,lcd_d5
                    bst     temp0,4
                    bld     temp1,lcd_d4
                    out     lcd_port,temp1          ;mueve  a
lcd_port los 4 bits mas significativos
                    sbi     lcd_port,lcd_rs     ;Selected data register
                    sbi     lcd_port,lcd_en     ;Enable H->L
                    nop
                    nop
                    cbi     Lcd_port,lcd_en
                    bst     temp0,3
                    bld     temp1,lcd_d7
                    bst     temp0,2
                    bld     temp1,lcd_d6
                    bst     temp0,1
                    bld     temp1,lcd_d5
                    bst     temp0,0
                    bld     temp1,lcd_d4
                    out     lcd_port,temp1          ;mueve  a
lcd_port los 4 bits menos significativos
                    sbi     lcd_port,lcd_rs     ;Selected data register
                    sbi     lcd_port,lcd_en     ;Enable H->L
```

```
                    nop
                    nop
                    cbi         lcd_port,lcd_en
                    call    wait_480u                   ;Wait
for LCD to process the command
                    call    wait_480u                   ;Wait
for LCD to process the command
                    call    wait_480u                   ;Wait
for LCD to process the command
                    call    wait_480u                   ;Wait
for LCD to process the command
                    ret                     ;Return from busy routine
;**********************************Envio de strings al
LCD**********************************

lcd_print_sub:      lpm     temp0,z+
                    cpi         temp0,0
                    breq    lcd_print_exit
                    rcall   lcd_senddata            ;send first char
                    rjmp    lcd_print_sub           ;jump back to send the
next character
lcd_print_exit:     ret                                     ;End of routine
;*****************************************************************************************
;**                                                                  **
;*****************************************Tablas*****************************************
;**                                                                  **
;*****************************************************************************************
CLR_:
.db                 "@CLR",13,10,0,0
```

ANEXO E

PROGRAMA JAVA

```
package front;
import inout.HiloFile;
import inout.HiloRS232;
import inout.TxRxRS232;
import java.io.File;


public class Main extends javax.swing.JFrame {

    /** Creates new form Main */
    private TxRxRS232 serial = new TxRxRS232();
    private javax.swing.JLabel[] entradas = new javax.swing.JLabel[9];
    private byte focos = 0;
    private byte focos1 = 0;
    private int hInicio = 0;
    private int hFin = 0;
    private HiloFile hFile = new HiloFile();
    private HiloRS232 inSerial;// = new HiloRS232(serial,entradas,botonSirena);

    public void setInicio (int hInicio)
    {
        this.hInicio = hInicio;
    }

    public void setFin (int hFin)
    {
        this.hFin = hFin;
    }

    public Main() {
```

```
    initComponents();
    serial.setup();

    entradas[0] = jLabelPuertaEntrada;
    entradas[1] = jLabelPuertaSalida;
    entradas[2] = jLabelVentanaSala1;
    entradas[3] = jLabelVentanaSala2;
    entradas[4] = jLabelVentanaDormitorio;
    entradas[5] = jLabelVentanaCocina;
    entradas[6] = jLabelVentanaComedor1;
    entradas[7] = jLabelVentanaComedor2;
    entradas[8] = jLabelEstado;
    inSerial = new HiloRS232(serial,entradas,jButtonSirena);

    entradas[0].setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/ventanaCerrada.PNG")));
    entradas[1].setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/ventanaCerrada.PNG")));
    entradas[2].setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/ventanaCerrada.PNG")));
    entradas[3].setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/puertaCerrada.PNG")));
    entradas[4].setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/ventanaCerrada.PNG")));
    entradas[5].setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/ventanaCerrada.PNG")));
    entradas[6].setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/puertaCerrada.PNG")));
    entradas[7].setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/ventanaCerrada.PNG")));

    File lock = new File(".\\file.lock");
    if (lock.exists())
       lock.delete();
    hFile.setLabel(jLabelEstado);
    hFile.start();
}
```

```
/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jButtonConectar = new javax.swing.JButton();
    jPanel1 = new javax.swing.JPanel();
    jButtonFoco0 = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();
    jButtonFoco1 = new javax.swing.JButton();
    jLabel2 = new javax.swing.JLabel();
    jButtonFoco2 = new javax.swing.JButton();
    jLabel3 = new javax.swing.JLabel();
    jButtonFoco3 = new javax.swing.JButton();
    jLabel4 = new javax.swing.JLabel();
    jButtonFoco4 = new javax.swing.JButton();
    jLabel5 = new javax.swing.JLabel();
    jButtonFoco5 = new javax.swing.JButton();
    jLabel6 = new javax.swing.JLabel();
    jButtonFoco6 = new javax.swing.JButton();
    jLabel7 = new javax.swing.JLabel();
    jButtonSirena = new javax.swing.JButton();
    jLabel8 = new javax.swing.JLabel();
    jPanel2 = new javax.swing.JPanel();
    jLabelPuertaEntrada = new javax.swing.JLabel();
    jLabelPuertaSalida = new javax.swing.JLabel();
```

```
        jLabelVentanaSala1 = new javax.swing.JLabel();
        jLabelVentanaSala2 = new javax.swing.JLabel();
        jLabelVentanaDormitorio = new javax.swing.JLabel();
        jLabelVentanaCocina = new javax.swing.JLabel();
        jLabelVentanaComedor1 = new javax.swing.JLabel();
        jLabelVentanaComedor2 = new javax.swing.JLabel();
        jLabelEstado = new javax.swing.JLabel();
        jButtonAlarma = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jButtonConectar.setText("Leer Entradas");
        jButtonConectar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonConectarActionPerformed(evt);
            }
        });

        jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Luces"));

        jButtonFoco0.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG"))); //
NOI18N
        jButtonFoco0.setAlignmentY(0.0F);
        jButtonFoco0.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
        jButtonFoco0.setIconTextGap(0);
        jButtonFoco0.setMargin(new java.awt.Insets(0, 0, 0, 0));
        jButtonFoco0.setRequestFocusEnabled(false);
        jButtonFoco0.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonFoco0ActionPerformed(evt);
```

```
        }
    });

    jLabel1.setText("Estudio");

    jButtonFoco1.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG"))); //
NOI18N
    jButtonFoco1.setAlignmentY(0.0F);
    jButtonFoco1.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    jButtonFoco1.setIconTextGap(0);
    jButtonFoco1.setMargin(new java.awt.Insets(0, 0, 0, 0));
    jButtonFoco1.setRequestFocusEnabled(false);
    jButtonFoco1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonFoco1ActionPerformed(evt);
        }
    });

    jLabel2.setText("Sala");

    jButtonFoco2.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG"))); //
NOI18N
    jButtonFoco2.setAlignmentY(0.0F);
    jButtonFoco2.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    jButtonFoco2.setIconTextGap(0);
    jButtonFoco2.setMargin(new java.awt.Insets(0, 0, 0, 0));
    jButtonFoco2.setRequestFocusEnabled(false);
    jButtonFoco2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonFoco2ActionPerformed(evt);
```

```
        }
    });

    jLabel3.setText("Sala de Estar");

    jButtonFoco3.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG"))); //
NOI18N
    jButtonFoco3.setAlignmentY(0.0F);
    jButtonFoco3.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    jButtonFoco3.setIconTextGap(0);
    jButtonFoco3.setMargin(new java.awt.Insets(0, 0, 0, 0));
    jButtonFoco3.setRequestFocusEnabled(false);
    jButtonFoco3.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonFoco3ActionPerformed(evt);
        }
    });

    jLabel4.setText("Dormitorio 1");

    jButtonFoco4.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG"))); //
NOI18N
    jButtonFoco4.setAlignmentY(0.0F);
    jButtonFoco4.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    jButtonFoco4.setIconTextGap(0);
    jButtonFoco4.setMargin(new java.awt.Insets(0, 0, 0, 0));
    jButtonFoco4.setRequestFocusEnabled(false);
    jButtonFoco4.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonFoco4ActionPerformed(evt);
```

```
        }
    });

    jLabel5.setText("Dormitorio 2");

    jButtonFoco5.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG"))); //
NOI18N
    jButtonFoco5.setAlignmentY(0.0F);
    jButtonFoco5.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    jButtonFoco5.setIconTextGap(0);
    jButtonFoco5.setMargin(new java.awt.Insets(0, 0, 0, 0));
    jButtonFoco5.setRequestFocusEnabled(false);
    jButtonFoco5.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonFoco5ActionPerformed(evt);
        }
    });

    jLabel6.setText("Cocina y Comedor");

    jButtonFoco6.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG"))); //
NOI18N
    jButtonFoco6.setAlignmentY(0.0F);
    jButtonFoco6.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    jButtonFoco6.setIconTextGap(0);
    jButtonFoco6.setMargin(new java.awt.Insets(0, 0, 0, 0));
    jButtonFoco6.setRequestFocusEnabled(false);
    jButtonFoco6.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonFoco6ActionPerformed(evt);
```

```
        }
    });

    jLabel7.setText("Sirena");

    jButtonSirena.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/alarmaOff.png"))); //
NOI18N
    jButtonSirena.setAlignmentY(0.0F);
    jButtonSirena.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
    jButtonSirena.setIconTextGap(0);
    jButtonSirena.setMargin(new java.awt.Insets(0, 0, 0, 0));
    jButtonSirena.setMaximumSize(new java.awt.Dimension(70, 71));
    jButtonSirena.setMinimumSize(new java.awt.Dimension(70, 71));
    jButtonSirena.setRequestFocusEnabled(false);
    jButtonSirena.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jButtonSirenaActionPerformed(evt);
        }
    });

    jLabel8.setText("Baño");

    javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
    jPanel1.setLayout(jPanel1Layout);
    jPanel1Layout.setHorizontalGroup(
        jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                .addGroup(jPanel1Layout.createSequentialGroup()
```

```
                .addComponent(jLabel7)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jButtonSirena, javax.swing.GroupLayout.PREFERRED_SIZE, 71,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(jLabel1)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jButtonFoco0, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                            .addComponent(jLabel4)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                                .addComponent(jLabel8)
                                .addComponent(jLabel6)))
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jButtonFoco2, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jButtonFoco4, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jButtonFoco6, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE))))
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(62, 62, 62)
                        .addComponent(jLabel2))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
```

```
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

                        .addComponent(jLabel5, javax.swing.GroupLayout.Alignment.TRAILING)

                        .addComponent(jLabel3, javax.swing.GroupLayout.Alignment.TRAILING))))

                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

                    .addComponent(jButtonFoco3, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)

                    .addComponent(jButtonFoco1, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)

                    .addComponent(jButtonFoco5, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE))))

            .addGap(31, 31, 31))

        );

        jPanel1Layout.setVerticalGroup(

            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

            .addGroup(jPanel1Layout.createSequentialGroup()

                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

                    .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

                        .addGroup(jPanel1Layout.createSequentialGroup()

                            .addGap(40, 40, 40)

                            .addComponent(jButtonFoco1, javax.swing.GroupLayout.PREFERRED_SIZE, 73,
javax.swing.GroupLayout.PREFERRED_SIZE))

                        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()

                            .addContainerGap()

                            .addComponent(jButtonFoco0, javax.swing.GroupLayout.PREFERRED_SIZE, 73,
javax.swing.GroupLayout.PREFERRED_SIZE)))

                    .addGroup(jPanel1Layout.createSequentialGroup()

                        .addGap(69, 69, 69)

                        .addComponent(jLabel1))

                    .addGroup(jPanel1Layout.createSequentialGroup()
```

```
                .addGap(71, 71, 71)
                .addComponent(jLabel2)))
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(79, 79, 79)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel3)
                    .addComponent(jLabel4)))
                .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(51, 51, 51)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jButtonFoco2, javax.swing.GroupLayout.PREFERRED_SIZE, 73,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jButtonFoco3, javax.swing.GroupLayout.PREFERRED_SIZE, 73,
javax.swing.GroupLayout.PREFERRED_SIZE))))
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(jPanel1Layout.createSequentialGroup()
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel5)
                    .addComponent(jLabel6))
                .addGap(82, 82, 82))
                .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(52, 52, 52)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jButtonFoco4, javax.swing.GroupLayout.PREFERRED_SIZE, 73,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jButtonFoco5, javax.swing.GroupLayout.PREFERRED_SIZE, 73,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 53,
Short.MAX_VALUE)))
```

```
            .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addGap(33, 33, 33)
                    .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jLabel7)
                        .addComponent(jLabel8)))
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addGap(3, 3, 3)
                    .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jButtonFoco6, javax.swing.GroupLayout.PREFERRED_SIZE, 73,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jButtonSirena, javax.swing.GroupLayout.PREFERRED_SIZE, 73,
javax.swing.GroupLayout.PREFERRED_SIZE))))
            .addContainerGap())
        );

        jPanel2.setBorder(javax.swing.BorderFactory.createTitledBorder("Sensores"));
        jPanel2.setPreferredSize(new java.awt.Dimension(316, 453));

        jLabelPuertaEntrada.setText("Ventana Sala");

        jLabelPuertaSalida.setText("Ventana Dormitorio 1");

        jLabelVentanaSala1.setText("Ventana Estudio");

        jLabelVentanaSala2.setText("Puerta Trasera");

        jLabelVentanaDormitorio.setText("Ventana Dormitorio 2");

        jLabelVentanaCocina.setText("Ventana Cocina");
```

```
jLabelVentanaComedor1.setText("Puerta Principal");

jLabelVentanaComedor2.setText("Ventana  Comedor");

jLabelEstado.setFont(new java.awt.Font("Tahoma", 0, 48)); // NOI18N
jLabelEstado.setText("Sistema Armado");

javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
jPanel2.setLayout(jPanel2Layout);
jPanel2Layout.setHorizontalGroup(
   jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
   .addGroup(jPanel2Layout.createSequentialGroup()
      .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(jPanel2Layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
               .addComponent(jLabelVentanaSala1)
               .addComponent(jLabelVentanaComedor1)
               .addComponent(jLabelPuertaSalida))
            .addGap(12, 12, 12)
            .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
               .addComponent(jLabelVentanaSala2)
               .addComponent(jLabelPuertaEntrada)
               .addComponent(jLabelVentanaCocina)))
         .addGroup(jPanel2Layout.createSequentialGroup()
            .addContainerGap()
            .addComponent(jLabelVentanaComedor2)
            .addGap(18, 18, 18)
            .addComponent(jLabelVentanaDormitorio))
```

```
                .addGroup(jPanel2Layout.createSequentialGroup()
                    .addGap(39, 39, 39)
                    .addComponent(jLabelEstado)))
            .addContainerGap(51, Short.MAX_VALUE))
        );
        jPanel2Layout.setVerticalGroup(
            jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addGap(43, 43, 43)
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabelVentanaSala2)
                    .addComponent(jLabelVentanaComedor1))
                .addGap(73, 73, 73)
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabelVentanaSala1)
                    .addComponent(jLabelPuertaEntrada))
                .addGap(68, 68, 68)
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabelVentanaCocina)
                    .addComponent(jLabelPuertaSalida))
                .addGap(74, 74, 74)
                .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabelVentanaComedor2)
                    .addComponent(jLabelVentanaDormitorio))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(jLabelEstado)
                .addContainerGap())
        );
```

```
        jButtonAlarma.setText("Conf. Alarma");
        jButtonAlarma.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonAlarmaActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE, 446,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                    .addComponent(jButtonConectar, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jButtonAlarma, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap(127, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
```

```
                .addGroup(layout.createSequentialGroup()
                    .addGap(12, 12, 12)
                    .addComponent(jButtonConectar)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                    .addComponent(jButtonAlarma))
                .addComponent(jPanel2, javax.swing.GroupLayout.DEFAULT_SIZE, 525, Short.MAX_VALUE)
                .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
            .addContainerGap(23, Short.MAX_VALUE))
        );

        jPanel1.getAccessibleContext().setAccessibleName("");
        jPanel1.getAccessibleContext().setAccessibleDescription("");
        jPanel2.getAccessibleContext().setAccessibleName("");
        jPanel2.getAccessibleContext().setAccessibleDescription("");

        pack();
    }// </editor-fold>

    private void jButtonConectarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:

        inSerial.start();
        jButtonConectar.setText("Conectado");
        jButtonConectar.setEnabled(false);
    }

    private void jButtonFoco0ActionPerformed(java.awt.event.ActionEvent evt) {

        if(jButtonFoco0.getIcon().toString().contains("focoOff"))
```

```
    {
        jButtonFoco0.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOn.PNG")));
        focos |= 0x01;
    }
    else
    {
        jButtonFoco0.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG")));
        focos &= 0xFE;
    }

    String cadena= (new StringBuffer().append((char)focos)).toString();
    //serial.send(cadena);
    serial.sendByte(focos);
}

private void jButtonFoco1ActionPerformed(java.awt.event.ActionEvent evt) {

    if(jButtonFoco1.getIcon().toString().contains("focoOff"))
    {
        jButtonFoco1.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOn.PNG")));
        focos |= 0x02;
    }
    else
    {
        jButtonFoco1.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG")));
        focos &= 0xFD;
    }
    //String cadena= (new StringBuffer().append((char)focos)).toString();
    serial.sendByte(focos);
}
```

```
private void jButtonFoco2ActionPerformed(java.awt.event.ActionEvent evt) {

    if(jButtonFoco2.getIcon().toString().contains("focoOff"))
    {
        jButtonFoco2.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOn.PNG")));
        focos |= 0x04;
    }
    else
    {
        jButtonFoco2.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG")));
        focos &= 0xFB;
    }
    //String cadena= (new StringBuffer().append((char)focos)).toString();
    serial.sendByte(focos);
}

private void jButtonFoco3ActionPerformed(java.awt.event.ActionEvent evt) {

    if(jButtonFoco3.getIcon().toString().contains("focoOff"))
    {
        jButtonFoco3.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOn.PNG")));
        focos |= 0x08;
    }
    else
    {
        jButtonFoco3.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG")));
        focos &= 0xF7;
    }
    //String cadena= (new StringBuffer().append((char)focos)).toString();
```

```
    serial.sendByte(focos);
}

private void jButtonFoco4ActionPerformed(java.awt.event.ActionEvent evt) {

    if(jButtonFoco4.getIcon().toString().contains("focoOff"))
    {
        jButtonFoco4.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOn.PNG")));
        focos |= 0x10;
    }
    else
    {
        jButtonFoco4.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG")));
        focos &= 0xEF;
    }
    //String cadena= (new StringBuffer().append((char)focos)).toString();
    serial.sendByte(focos);
}

private void jButtonFoco5ActionPerformed(java.awt.event.ActionEvent evt) {

    if(jButtonFoco5.getIcon().toString().contains("focoOff"))
    {
        jButtonFoco5.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOn.PNG")));
        focos |= 0x20;
    }
    else
    {
        jButtonFoco5.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG")));
        focos &= 0xDF;
```

```
    }
    //String cadena= (new StringBuffer().append((char)focos)).toString();
    serial.sendByte(focos);
}

private void jButtonFoco6ActionPerformed(java.awt.event.ActionEvent evt) {

    if(jButtonFoco6.getIcon().toString().contains("focoOff"))
    {
        jButtonFoco6.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOn.PNG")));
        focos |= 0x40;
    }
    else
    {
        jButtonFoco6.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/focoOff.PNG")));
        focos &= 0xBF;
    }
    //String cadena= (new StringBuffer().append((char)focos)).toString();
    serial.sendByte(focos);
}

private void jButtonSirenaActionPerformed(java.awt.event.ActionEvent evt) {

    if(jButtonSirena.getIcon().toString().contains("alarmaOff"))
    {
        jButtonSirena.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/alarmaOn.PNG")));
        focos |= 0x80;
    }
    else
    {
```

```
            jButtonSirena.setIcon(new javax.swing.ImageIcon(getClass().getResource("/front/alarmaOff.PNG")));
            focos &= 0x7F;
        }
        //cadena = (new StringBuffer().append((char)focos)).toString();
        serial.sendByte(focos);
        //serial.send("hola");
    }

    private void jButtonAlarmaActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        Alarma alarma = new Alarma();
        alarma.setVisible(true);
        //this.setVisible(false);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
```

```
            }
        }
    } catch (ClassNotFoundException ex) {
        java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
    } catch (InstantiationException ex) {
        java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
    } catch (IllegalAccessException ex) {
        java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
    }
    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {

        @Override
        public void run() {
            new Main().setVisible(true);
        }
    });
}



// Variables declaration - do not modify
private javax.swing.JButton jButtonAlarma;
private javax.swing.JButton jButtonConectar;
```

```
    private javax.swing.JButton jButtonFoco0;
    private javax.swing.JButton jButtonFoco1;
    private javax.swing.JButton jButtonFoco2;
    private javax.swing.JButton jButtonFoco3;
    private javax.swing.JButton jButtonFoco4;
    private javax.swing.JButton jButtonFoco5;
    private javax.swing.JButton jButtonFoco6;
    private javax.swing.JButton jButtonSirena;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabelEstado;
    private javax.swing.JLabel jLabelPuertaEntrada;
    private javax.swing.JLabel jLabelPuertaSalida;
    private javax.swing.JLabel jLabelVentanaCocina;
    private javax.swing.JLabel jLabelVentanaComedor1;
    private javax.swing.JLabel jLabelVentanaComedor2;
    private javax.swing.JLabel jLabelVentanaDormitorio;
    private javax.swing.JLabel jLabelVentanaSala1;
    private javax.swing.JLabel jLabelVentanaSala2;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    // End of variables declaration
}
```