ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería Eléctrica y Computación

"PROPUESTA DE NUEVAS PRÁCTICAS DE LA MATERIA MICROCONTROLADORES UTILIZANDO LA PLACA ARDUINO MEGA 2560 COMO COMPLEMENTO Y MEJORA DEL CURSO ACTUAL"

INFORME DE PROYECTO DE GRADUACIÓN

Previa a la obtención del título de

INGENIERO EN ELECTRICIDAD ESPECIALIZACIÓN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL

Presentada por

Diego Andres Sierra Cevallos

Guayaquil Ecuador

2014

AGRADECIMIENTO

A Dios y a la virgen por darme la fuerza para lograr este gran pasó en mi vida, a mis Padres por apoyarme durante toda mi carrera y a Mafer quien fue de gran ayuda para culminar este proyecto.

Diego Andres Sierra Cevallos

DEDICATORIA

A mi Familia, quienes desearon este logro tanto como yo, a las personas que fueron participes de mi crecimiento personal y profesional, que con su apoyo y consejos, me ayudaron a terminar con esta etapa de mi vida.

Diego Andres Sierra Cevallos

TRIBUNAL DE SUSTENTACIÓN

_____

Ph. D.Boris Xavier Vintimilla Burgos

PRESIDENTE

_____

Ing. Carlos Valdivieso A.

DIRECTOR DEL PROYECTO DE GRADUACIÓN

_____

Ing. Ronald Ponguillo I.

MIEMBRO PRINCIPAL DEL TRIBUNAL

DECLARACION EXPRESA

¨La responsabilidad del contenido de este Proyecto de Graduación, me corresponde exclusivamente; y el patrimonio intelectual de la misma, a la **Escuela Superior Politécnica del Litoral**¨

(Reglamento de Graduación de la ESPOL)

Diego Andres Sierra Cevallos

RESUMEN

Este proyecto consiste en la elaboración de cinco prácticas que se pretenden utilizar en la materia de Microcontroladores. Estas prácticas se basan en la utilización de la tarjeta Arduino Mega 2560.

En la primera práctica se usarán un teclado matricial 4x4 y un LCD 16x2, con los cuales los estudiantes aprenderán a hacer diversas interfaces para sus proyectos y posteriormente podrán usar lo aprendido en las siguientes prácticas.

En la segunda práctica se hará uso de un motor DC y se aprenderá cómo controlar su velocidad, sentido de giro y posteriormente realizar una secuencia.

En la tercera práctica se usará un motor de paso y un servomotor, y para estos dos tipos de motores se aprenderá cómo controlar la posición de giro y también se realizará una secuencia usando los dos tipos de motores.

Para la cuarta práctica se usará un LCD 20x4 y un sensor de temperatura TMP 102 los cuales trabajan con la comunicación I2C. Con esto los estudiantes aprenderán cómo usar diferentes dispositivos I2C.

Para la quinta práctica se usaran los módulos XBee, con los cuales se aprenderá a cómo enviar y recibir información inalámbricamente.

ÍNDICE GENERAL

Pag.

RESUMEN……..……...I

ÍNDICE GENERAL……..III

ÍNDICE DE FIGURAS……..……VIII

ÍNDICE DE TABLAS……..…X

INTRODUCCIÓN……XI

CAPÍTULO 1

1. ANÁLISIS DEL PROYECTO……...…….1

    1.1. OBJETIVOS……..……..1

    1.2. JUSTIFICACIÓN……..……..2

    1.3. VIABILIDAD DE LA INVESTIGACIÓN……...2

    1.4. CONSECUENCIAS DE LA INVESTIGACIÓN……..…….3

CAPÍTULO 2

2. BASE TEÓRICA PRÁCTICA……...4

2.1. FILOSOFÍA DE LOS SISTEMAS ARDUINO……..……4

2.2. CARACTERÍSTICAS DEL ARDUINO MEGA 2560……..7

2.3. PROGRAMACIÓN DE TARJETA ARDUINO MEGA 2560……..9

2.4. SIMULACIÓN DE TARJETAS ARDUINO……10

2.5. MATERIALES DEL LABORATORIO……11

CAPÍTULO 3

3. IMPLEMENTACIÓN CON ARDUINO……...13

3.1. TECLADO MATRICIAL Y LCD……...14

3.1.1. PREPARACIÓN PARA LA PRÁCTICA 1……..……...14

3.1.1.1. LCD 16x2……..……....15

3.1.1.2. TECLADO MATRICIAL 4X4……19

3.1.1.3. LED INFRARROJO EMISOR Y RECEPTOR……21

3.1.2. PRACTICA 1: USO DEL TECLADO MATRICIAL Y LCD……23

3.1.2.1. OBJETIVOS……....23

3.1.2.2. TEST DE EDUCACIÓN VIAL……..……...24

3.1.2.3. INGRESO DE CONTRASEÑA…….……..……35

3.1.2.4. CONTEO CON SENSOR DE PROXIMIDAD INFRARROJO……42

3.2. MOTORES DC……..48

3.2.1. PREPARACIÓN PARA LA PRÁCTICA 2……..……...48

3.2.1.1. MOTOR DC……49

3.2.1.2. PUENTE H (L293)……..……..51

3.2.2. PRÁCTICA 2: USO DE LOS MOTORES DC……53

3.2.2.1. OBJETIVOS……..……54

3.2.2.2. CONTROL DE VELOCIDAD ANALÓGICO DE UN MOTOR DC……54

3.2.2.3. SECUENCIA DE GIRO DE UN MOTOR DC……57

3.2.2.4. CAMBIO DE GIRO DE UN MOTOR DC MEDIANTE UN SENSOR DE PROXIMIDAD INFRARROJO……64

3.3. MOTORES DE PASO (STEPPER) Y SERVOMOTORES……..…73

3.3.1. PREPARACIÓN PARA LA PRÁCTICA 3……...…...74

3.3.1.1. SERVOMOTOR……..……..…74

3.3.1.2. MOTOR DE PASO (STEPPER)……..……77

3.3.1.3. INTEGRADO ULN2003……81

3.3.2. PRÁCTICA 3: USO DE LOS MOTORES DE PASO (STEPPER) Y SERVOMOTORES……..…..84
3.3.2.1. OBJETIVOS……..…..85
3.3.2.2. CONTROL DE POSICIÓN DE UN SERVOMOTOR…...….85
3.3.2.3. CONTROL DE POSICIÓN DE UN MOTOR STEPPER…..88
3.3.2.4. SECUENCIA CON UN SERVOMOTOR Y MOTOR STEPPER……..…..95
3.4. COMUNICACIÓN I2C……103
3.4.1. PREPARACIÓN PARA LA PRÁCTICA 4……..……103
3.4.2. PRÁCTICA 4: IMPLEMENTACIÓN DE LA COMUNICACIÓN I2C……....111
3.4.2.1. OBJETIVOS……..…111
3.4.2.2. SENSOR DE TEMPERATURA CON COMUNICACIÓN I2C……..…..111
3.5. COMUNICACIÓN XBEE……115
3.5.1. PREPARACIÓN PARA LA PRÁCTICA 5……..….115
3.5.2. PRÁCTICA 5: USO DE LA COMUNICACIÓN XBEE……127
3.5.2.1. OBJETIVOS……..…127
3.5.2.2. IMPLEMENTACIÓN PRÁCTICA DE LA COMUNICACIÓN XBEE……..…..127

3.6. PROPUESTA DE FUTUROS PROYECTOS DE FIN DE CURSO…….130

CAPÍTULO 4

4. PRUEBAS UTILIZANDO ARDUINO……..134

4.1. PRUEBAS CON TECLADO MATRICIAL Y LCD……..134

4.2. PRUEBAS CON MOTORES DC……..137

4.3. PRUEBAS CON MOTORES DE PASO (STEPPER) Y SERVOMOTORES…….138

4.4. PRUEBAS UTILIZANDO LA COMUNICACIÓN I2C……141

4.5. PRUEBAS UTILIZANDO LA COMUNICACIÓN XBEE……...142

CONCLUSIONES

RECOMENDACIONES

ANEXOS

ÍNDICE DE FIGURAS

Pag.

Figura 3.1 Pines del LCD 16x2……..……15

Figura 3.2 Conexión del Arduino con Pantalla LCD 16x2……..……...…18

Figura 3.3 Pines de Teclado Matricial 4x4…….……...19

Figura 3.4 Conexión entre Arduino y Teclado Matricial 4x4……..…21

Figura 3.5 Partes de un Diodo…….……...22

Figura 3.6 Conexión leds infrarrojos al Arduino…….……..22

Figura 3.7 Partes del motor DC…….……..…50

Figura 3.8 Conexión de motor DC con puente H completo……..….51

Figura 3.9 Conexión Arduino + L293 + Motor DC……..…53

Figura 3.10 Partes de un Servomotor……....75

Figura 3.11 Control de posición (PWM) de un Servomotor……..……76

Figura 3.12 Conexión Arduino con Servomotor…….……...77

Figura 3.13 Motor unipolar de 6 terminales y motor bipolar……78

Figura 3.14 Conexiones con colores de motor de paso 28byj-48……80

Figura 3.15 Pines del integrado ULN2003……..82

Figura 3.16 Circuito ULN2003 + Bobinas de motor de paso……..…….…83

Figura 3.17 Diagrama de conexión Arduino Mega + ULN2003 + motor de paso 28byj-48……..…..…..83

Figura 3.18 Conexión entre master y esclavos con protocolo I2C……..…104

Figura 3.19 Representación grafica de comunicación I2C……..……106

Figura 3.20 TMP102…….....107

Figura 3.21 Conexión Arduino mega + TMP103 + LCD con I2c……..…..110

Figura 3.22 Niveles de jerarquía en comunicación Zigbee…..……116

Figura 3.23……..……..122

Figura 3.24……..……..…123

Figura 3.25……..……..…123

Figura 3.26……..……..…125

Figura 3.27 Proyecto XBee 2……..……..…126

ÍNDICE DE TABLAS

Pag.

Tabla 3.1 Conexión entre Arduino y LCD……..……...….18

Tabla 3.2 Conexión entre Arduino y Teclado Matricial……...20

Tabla 3.3 Conexión entre Arduino, L293, Motor DC y Fuente Externa…….…...52

Tabla 3.4 Conexión entre Arduino y Servomotor…….77

Tabla 3.5 Secuencia de motor de paso Unipolar……..……...…79

Tabla 3.6 Secuencia de motor de paso Bipolar……..……79

Tabla 3.7 Secuencia de motor de paso 28byj-48…….……...81

Tabla 3.8 Arduino Mega + ULN2003 + motor de paso 28byj-48……84

Tabla 3.9 Direcciones del TMP102……..……108

Tabla 3.10 Datos de temperatura con formato de 12 bits del TMP102……..109

Tabla 3.11 Tabla de pines Practica con I2C……..……...110

Tabla 3.12 Trama receptor API y Transmisor AT……..118

Tabla 3.13 Trama Transmisor API y Receptor AT……120

INTRODUCCIÓN

En este proyecto se utiliza el Arduino Mega 2560 para un conjunto de prácticas que se consideran esenciales para el aprendizaje actual de microcontroladores.

Arduino es una plataforma de software y hardware libre que consta de una placa con un microcontrolador y un entorno de desarrollo. Existen algunos modelos de estas placas, cada uno con diferentes características, siendo los modelos Arduino UNO y Arduino MEGA los ideales para personas que desean iniciarse en el mundo de la electrónica y los microcontroladores.

Las tarjetas Arduino son ideales para el aprendizaje, ya que son de fácil programación y además existe gran cantidad de información sobre ellas, ejemplos de cómo realizar diferentes tipos de proyectos y también tienen librerías para casi todo, lo cual facilita mucho el trabajo.

Es importante recalcar que los microcontroladores pueden ser usados en cualquier aplicación que requiera una toma de decisiones, ya sea con supervisión o sin supervisión de un humano. Por lo que están incluidos en casi todos los aparatos eléctricos y electrónicos que nos rodean, como un celular, un televisor, un microondas, un juguete de un niño, como también en aparatos médicos y hasta en armas de defensa. Por esta razón es importante aprender a cómo hacer uso de ellos.

Son estas las razones por las cuales se decidió bajo la supervisión del coordinador de la materia de Microcontroladores realizar este proyecto de graduación. El cual consiste en llevar a cabo cinco prácticas utilizando la placa Arduino Mega. Las cuales en un futuro pueden ser incluidas en la materia de Microcontroladores.

CAPÍTULO 1

ANÁLISIS DEL PROYECTO

En este capítulo se revisarán objetivos, justificación, viabilidad y consecuencias del proyecto de graduación a tratar.

1.1. Objetivos:

- Incluir en la materia de Laboratorio de Microcontroladores de la ESPOL la realización de 5 prácticas con sus correspondientes ejercicios.

- Elaborar prácticas para el Laboratorio de Microcontroladores utilizando tarjetas Arduino y componentes compatibles adicionales.
- Desarrollar en los estudiantes las destrezas necesarias para la realización de proyectos con Arduino.

1.2. Justificación:

El desarrollo alcanzado por los microcontroladores Arduino mediante el uso de herramientas de software y hardware libre, se ha constituido como una línea educativa de gran valor, que merece ser incorporada en el plan de estudios de la ESPOL.

Existen numerosas universidades en el mundo que han incluido a las tarjetas Arduino en sus programas de estudio, como por ejemplo la Universidad de Chile o la Universidad de la República de Uruguay y otras de más renombre como Standford, Carnegie Mellon y el MIT.

1.3. Viabilidad de la Investigación:

La realización de este proyecto es viable ya que Arduino posee una gran cantidad de librerías ya desarrolladas que facilitan al usuario la

elaboración de proyectos. Además la tarjeta Arduino Mega 2560 es muy popular y se la encuentra en el mercado alrededor de los 40 usd. Los demás materiales como protoboard, motores, sensores y cables también se los encuentran fácilmente.

1.4. Consecuencias de la investigación:

Los estudiantes del Laboratorio aprenderán a utilizar la tarjeta Arduino Mega 2560 y obtendrán un mayor interés por las herramientas basadas en microcontroladores. Para que de esta forma estén mejor preparados para resolver problemas en sus carreras profesionales.

CAPÍTULO 2

BASE TEÓRICA PRÁCTICA

En este capítulo se revisará la base teórica referente a la tarjeta Arduino Mega 2560. Con la cual se realizarán cinco prácticas para la materia Microcontroladores. Por último se mencionarán los elementos con los cuales se realizarán las prácticas.

2.1. FILOSOFÍA DE LOS SISTEMAS ARDUINO

Arduino es plataforma de software y hardware libre, la cual está basada en una placa con un microcontrolador que contiene entradas y salidas, digitales y analógicas y una plataforma de desarrollo [1]. Lo que significa que tanto el software como la información del diseño de las

placas están disponibles de forma gratuita para que las personas puedan, construirlas, mejorarlas, crear un nuevo modelo, crear nuevos programas, librerías e incluso vender las placas. Esto es lo que ha hecho que Arduino gane cada vez más fans, ya que por ser de código y hardware libre los precios de las placas son asequibles y para tener una referencia aquí en el Ecuador una placa Arduino Mega está costando alrededor de los 42 usd.

Arduino fue inventado en el año 2005 por Massimo Banzi quien en ese entonces era un estudiante del instituto IVRAE en Italia. Arduino fue creado por la necesidad que tenían los estudiantes de computación y electrónica de ese instituto de adquirir conocimientos pero con materiales que tuvieran bajo costo, ya que una placa de microcontroladores era muy cara en ese entonces. Posteriormente se integro un grupo de trabajo con el que se mejoro el prototipo y se incluyo un programa con un lenguaje de programación [2].

¿Por qué usar Arduino?

Hay muchas razones por las cuales los usuarios prefieren las tarjetas Arduino. A continuación pongo una lista dando a conocer las razones por las cuales yo prefiero trabajar con Arduino:

- **Software y Hardware libres**, esto sin duda llama mucho la atención ya que el programa es muy fácil de conseguir y sobre todo es gratis.

- **Asequible**. Esto es muy importante ya que como estudiante no siempre se tienen los recursos para poder comprar equipos muy caros.
- **Fácil uso**. Con esto me refiero a que la placa (Arduino Mega) tiene puertos de entradas y salidas bien ordenadas y numeradas que dan una facilidad al usuario para realizar conexiones.
- **Fácil programación**. Aunque el lenguaje de programación para Arduino es Processing, este deriva de Java el cual a su vez deriva de C y C++. Lo que hace que cualquier persona que tenga un concepto básico de C o C++ pueda programar muy fácilmente una placa Arduino.
- **Cantidad de sensores y actuadores**. Existen un sinnúmero de sensores y actuadores que pueden funcionar directa o indirectamente con Arduino, lo permite que se puedan realizar de la misma manera un sinnúmero de proyectos de diferentes tipos.
- **Cantidad de Shields.** Los shields son placas que se acoplan directamente a los puertos de salida y existen de diferentes tipos, como por ejemplo de drivers para motores, de protoboard, de XBee, etc y son muy útiles a la hora de realizar cierto tipo de proyectos.
- **Librerías.** Estas son muy importantes ya que facilitan mucho el trabajo de programación. Existen muchas librerías para Arduino y debido a que cada vez se hace más popular mucha gente hace sus propias

librerías y las comparte con el resto del mundo. Así de esta manera no es de sorprenderse si se encuentra una librería para cada sensor y actuador que se quiera usar con Arduino.

- **Información.** Para mi esta es la más importante. Dada la popularidad de Arduino se puede encontrar muchísima información de cómo conectar y hacer que funcionen diferentes tipos de sensores y actuadores. Además de que existen foros y tutoriales que ayudan muchísimo en realizar un proyecto o simplemente resolver un problema.

2.2. CARACTERÍSTICAS DEL ARDUINO MEGA 2560

Existen muchas placas Arduino que tienen diferentes especificaciones y tamaños. Para realizar las prácticas que se dictaran en la materia de microcontroladores se uso un Arduino Mega 2560 debido a que tiene más entradas y salidas que su antecesor el Arduino UNO además que es de fácil uso. A continuación se muestra una breve descripción de las características del Arduino Mega 2560.

- **Microcontrolador ATmega2560.** Este es el cerebro y el que le da todas las características al resto de la placa. Este microcontrolador es de la marca Atmel, funciona con 5V, tiene 256 KB de memoria flash, 4KB de EEPROM, 8KB de RAM, 16MHz frecuencia de reloj, 86 I/O de propósito general, con 16 canales de entradas analógico digital de 10 bits [3].
  - **Voltaje de alimentación 7V-12V.** Con limites de 6V a 20V

- **Voltaje de operación 5V.** Dado que el microcontrolador funciona a 5V, las salidas también funciona a 5V.
- **16 entradas analógicas.** Cada entrada analógica tiene una resolución de 10 bits, esto es importante pues para poder usar estas entradas sabemos que vamos a tener valores de 0 a 1023.
- **54 Pines digitales de E/S.** De los cuales 15 (Pines 2-13, 44, 46) proveen salida PWM con 8 bits de resolución por lo que para estos se manejan valores de 0 a 255.
- **Memoria Flash de 256 KB.** Es aquí donde se almacena el código y 8 KB están destinados para el gestor de arranque.
- **SRAM de 8 KB**
- **EEPROM de 4 KB**
- **Velocidad de reloj de 16 MHz**
- **3Pin salida de 5V**
- **Pin salida de 3.3V**
- **5 Pines de Tierra**
- **4 puertos para comunicación serial (UART).** Son 4 pares de pines Rx y Tx
- **2 puertos para comunicación I2C.** Son 2 Pares de pines con SDA y SCl
- **SPI.** Los siguientes pines soportan la comunicación SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS) [4].

2.3. PROGRAMACIÓN DE TARJETA ARDUINO MEGA 2560

La programación es parte importante para poder llevar a cabo un proyecto con Arduino ya que sin ella no se podrían realizar. Arduino tiene un lenguaje de programación llamado wiring, el cual es un lenguaje de alto nivel que fue diseñado e implementado con Arduino para que artistas, diseñadores y personas de todo tipo, ya sean principiantes o avanzados puedan entrar en el mundo de la electrónica muy fácilmente. El lenguaje de programación wiring proviene del lenguaje processing, el cual a su vez proviene de Java y por ultimo de C/C++. Es por eso que el lenguaje wiring es muy parecido a C y reclama ser más fácil [5].

Para realizar la programación de las tarjetas Arduino es necesario descargar el programa Arduino IDE el cual se lo puede encontrar en la página web de Arduino.

Existe una estructura básica que hay que seguir para poder programar las tarjetas y consta de dos partes.

La primera se llama setup() y esta es la parte encargada de la configuración de los pines como entradas o salidas, según se requiera. También es en donde se inicializan algunas librerías y así mismo se utiliza para establecer el estado inicial de las salidas.

La segunda es loop() y es aquí donde se escribe toda la lógica del programa. Esta función como sugiere el nombre se ejecuta de forma

cíclica, lo que permite que el programa responda continuamente a los eventos que se produzcan en la placa [6].

Estas dos funciones son las principales y necesarias para que los programas funcionen, por esto siempre se deben incluir.

2.4. SIMULACIÓN DE TARJETAS ARDUINO

En el mundo de la electrónica siempre es importante realizar simulaciones antes de llevar un proyecto a la realidad, ya que de esta forma podemos corregir errores que después serían difíciles de reparar.

Existen algunos tipos de simuladores y aunque aun no hay un simulador oficial para Arduino aquí enlisto tres herramientas que pueden ayudar para el curso de Microcontroladores:

- **Fritzing.** Es un software libre que permite graficar proyectos usando las tarjetas Arduino, sin embargo no es un simulador en sí, pero es muy útil al momento de documentar un proyecto, ya que sus gráficos son muy parecidos a los objetos reales.
- **Virtual Breadboard.** Este es un simulador que soporta las tarjetas Arduino y es gratis, sin embargo la librería de elementos de este simulador es limitada y no abarca todos los temas con los que Arduino trabaja.

- **Proteus.** Este simulador es muy famoso y muy utilizado ya que tiene una librería muy extensa de integrados y demás artículos electrónicos además de que permite realizar circuitos impresos. Para poder realizar simulaciones de Arduino con Proteus es necesario primero descargar una librería que contenga las placas Arduino y luego instalarla. Como el caso del simulador anterior la librería de Proteus no tiene todos los elementos necesarios para abarcar todos los temas que se pueden tratar con Arduino.

2.5. MATERIALES DEL LABORATORIO

Para poder realizar las prácticas de laboratorio se selecciono junto con el coordinador de la materia de microcontroladores los materiales necesarios a utilizar.

A continuación la lista de los materiales para el laboratorio de Microcontroladores:

- Arduino Mega 2560 R3
- LCD 16x2
- Teclado matricial 4x4
- Integrado L293
- Servomotor Hitec HS-311
- Integrado ULN2003

- Motor stepper 28byj-48
- Sensor de temperatura TMP102
- 2 Xbee serie 2
- Xbee shield
- Xbee Explorer Regulated
- LCD I2C 20x4
- Cables para puente
- Fuente de poder 5V 2A
- Fuente de poder 12V 2A
- Conectores 12V macho y hembra
- Potenciómetro de 10K
- Trimer de 10K
- Leds rojo y verde
- Resistencias de 220, 1K y 220K ohmios
- Pulsadores
- Leds infrarrojos, uno emisor y otro receptor
- Protoboard
- Cable UTP

CAPÍTULO 3

IMPLEMENTACIÓN CON ARDUINO

Con Arduino se pueden implementar un sinnúmero de aplicaciones interactivas o autónomas, dado que puede leer datos de una gran variedad de sensores y controlar de la misma manera muchos motores y actuadores; la limitante para realizar cualquier proyecto con Arduino es solo la imaginación del usuario [7].

En este capítulo se detalla tanto la teoría como la programación y conexión de las cinco prácticas realizadas con Arduino. Las cuales tratan los siguientes temas:

- TECLADO MATRICIAL Y LCD
- MOTORES DC
- MOTORES DE PASO (STEPPER) Y SERVOMOTORES
- COMUNICACIÓN I2C
- COMUNICACIÓN XBEE

Para poder realizar las siguientes prácticas con Arduino es necesario instalar el programa Arduino IDE. Este lo podemos encontrar en la página principal de Arduino. Estas prácticas se realizaron con la versión 1.0.5 del programa.

3.1. TECLADO MATRICIAL Y LCD

En esta sección se demuestra el uso de un teclado matricial 4x4 y una pantalla LCD 16x2 con la tarjeta Arduino Mega.

3.1.1. PREPARACION PARA LA PRACTICA 1

Para poder realizar la práctica 1 es necesario tener conocimientos acerca del funcionamiento y conexión de los siguientes elementos:

- **LCD 16x2**
- **TECLADO MATRICIAL 4x4**

- **LED INFRARROJO EMISOR Y RECEPTOR**

3.1.1.1. LCD 16x2

Las pantallas de cristal líquido o LCD (Liquid Crystal Display) son dispositivos de visualización, y sirven para realizar la presentación de caracteres y símbolos. La pantalla LCD 16x2 puede contener un máximo de 32 caracteres y cada carácter está conformado por una matriz de 5X7 puntos que se denominan pixeles [8].

La pantalla LCD 16x2 tiene una distribución de pines como se muestra en la Figura 3.1

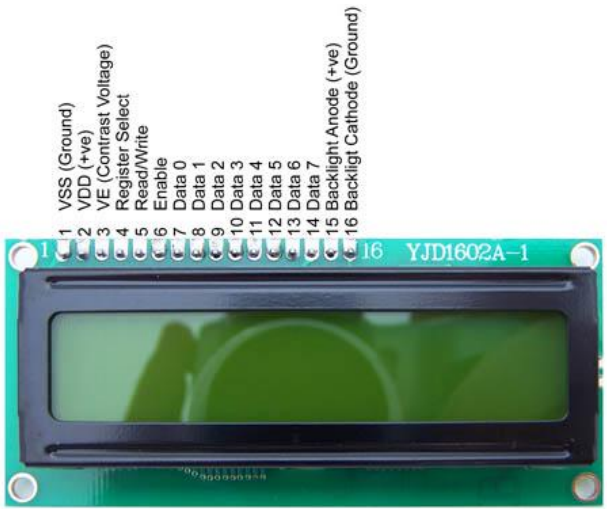


Figura 3.1 Pines del LCD 16x2 [http://www.protostack.com/blog/2010/03/character-lcd-displays-part-1/]

Donde los pines son:

1. VSS (Tierra)
2. Vcc (5v)
3. VE (Pin de contraste que se lo regula con un potenciómetro de 10K)
4. RS (Pin de selección de registro)
5. RW (Pin de escritura o lectura)
6. E (Pin de habilitación)
7. D0 (Dato 0)
8. D1 (Dato 1)
9. D2 (Dato 2)
10. D3 (Dato 3)
11. D4 (Dato 4)
12. D5 (Dato 5)
13. D6 (Dato 6)
14. D7 (Dato 7)
15. VA (Voltaje Ánodo 5v para luz trasera)
16. VK (Voltaje Cátodo Tierra para luz trasera)

El pin 4 (RS) corresponde al pin de selección de registro. Cuando RS es 0 el dato que en ese instante esta en el bus

pertenece a un registro de control/instrucción y cuando RS es 1 el dato pertenece a un registro de datos o un caracter.

El pin 5 (RW) nos permite hacer una lectura (cuando es 1) y una escritura (cuando es 0) en el LCD. Por esta razón en los proyectos que se realizaron se mantiene este pin conectado a tierra.

El pin 6 (E) corresponde al pin de habilitación en el cual si es 0 el LCD no recibe datos y si es 1 es apto para escribir o leer datos desde el LCD [9].

La conexión del LCD con Arduino la haremos como se muestra en la figura 3.2. Los pines conectados del Arduino con el LCD se muestran en la tabla 3.1.

Tabla 3.1 Conexión entre Arduino y LCD

Arduino	**LCD**
5V	Vcc (Proto)
GND	GND (Proto)
36 (Pin Digital)	RS
37 (Pin Digital)	Enable
38 (Pin Digital)	D4
39 (Pin Digital)	D5
40 (Pin Digital)	D6
41 (Pin Digital)	D7

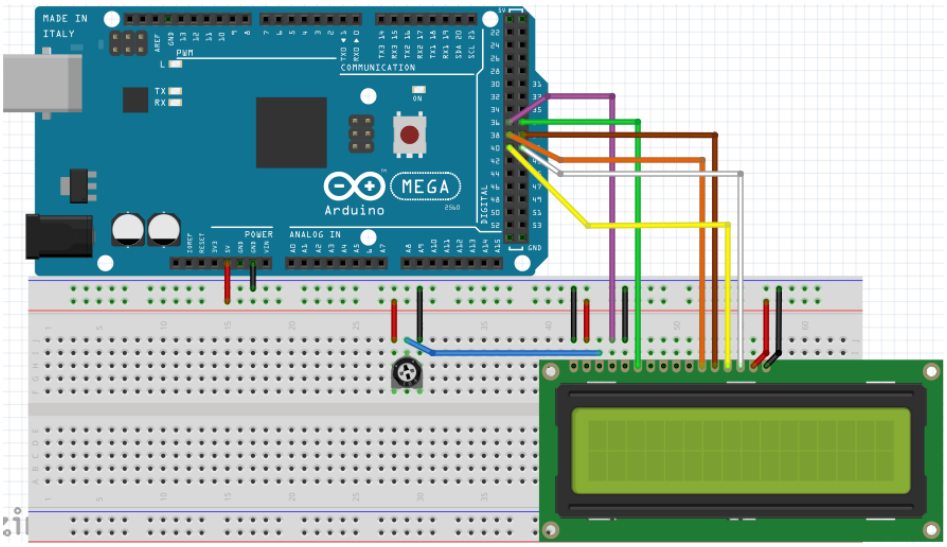


Figura 3.2 Conexión del Arduino con Pantalla LCD 16x2

3.1.1.2. TECLADO MATRICIAL 4x4

Un teclado matricial 4x4 no es más que una matriz de pulsadores que tiene 8 pines de conexión, estos se dividen en 4 que representan las filas y los otros 4 que representan las columnas. Esto tiene como intención el ahorro de pines ya que si se usara un pin por cada pulsador necesitaríamos 16 pines [10]. En la Figura 3.3 se muestra un teclado matricial 4x4 como el usado en los proyectos que se realizaron:

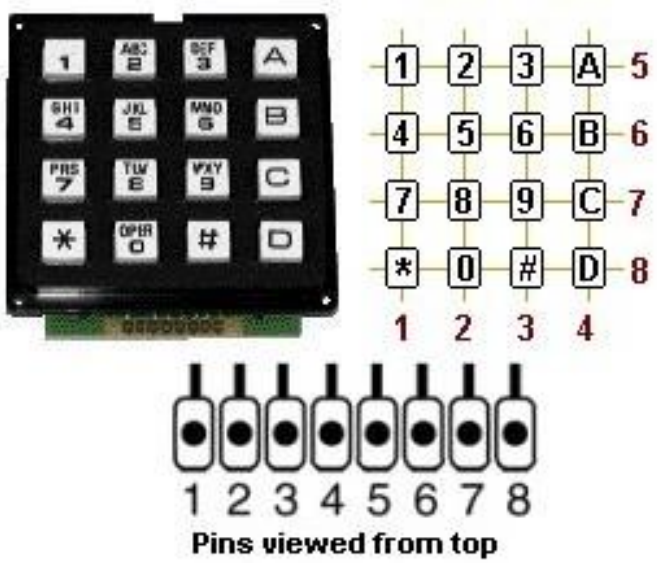


Figura 3.3 Pines de Teclado Matricial 4x4
[http://talkingelectronics.com/pay/PIC/PIC-Page21.html]

Donde los pines del 1 al 4 representan a las columnas y los pines del 5 al 8 representan las filas.

La conexión que se realizó entre el Arduino y el teclado matricial se muestra en la tabla 3.2 y en la Figura 3.4

Tabla 3.2 Conexión entre Arduino y Teclado Matricial

Arduino	Teclado Matricial
44 (Pin Digital)	1
45 (Pin Digital)	2
46 (Pin Digital)	3
47 (Pin Digital)	4
48 (Pin Digital)	5
49 (Pin Digital)	6
50 (Pin Digital)	7
51 (Pin Digital)	8

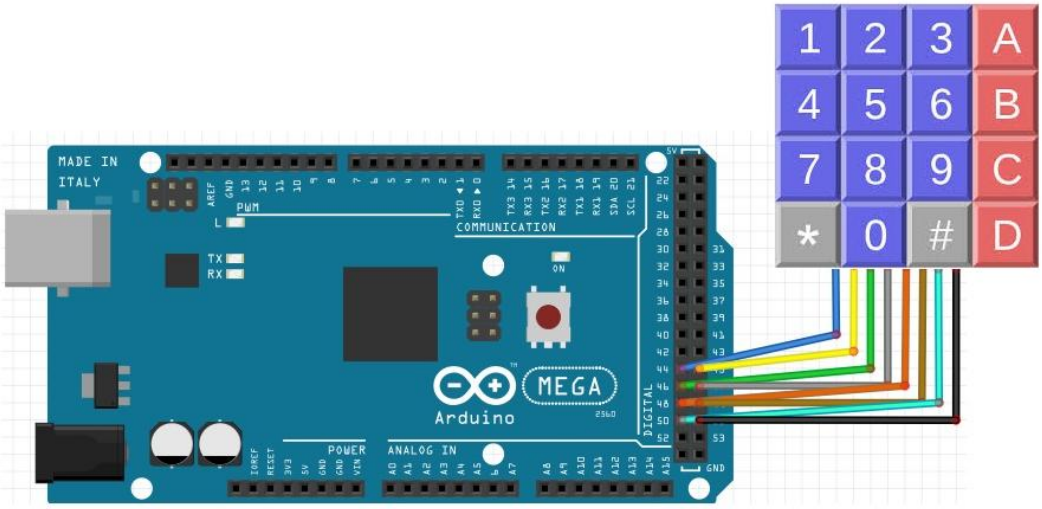


Figura 3.4 Conexión entre Arduino y Teclado Matricial 4x4

[http://talkingelectronics.com/pay/PIC/PIC-Page21.html

3.1.1.3. LED INFRARROJO EMISOR Y RECEPTOR

Un led infrarrojo es un tipo específico de diodo emisor de luz, que produce luz en el rango de espectro infrarrojo, el cual no es visible para el ojo humano.

Para el proyecto Conteo con Sensor de Proximidad Infrarrojo se usó 4 leds emisores infrarrojo y un led receptor.

Como sabemos el ánodo (+) de un diodo es la pata más larga y el cátodo (-) es la pata más corta, como se muestra en la figura 3.5

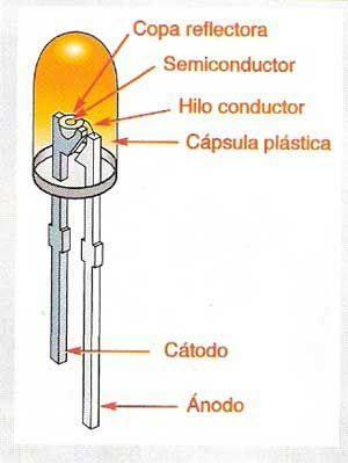


Figura 3.5 Partes de un Diodo

Sabiendo esto procedemos a realizar la conexión que se indica en la figura 3.6

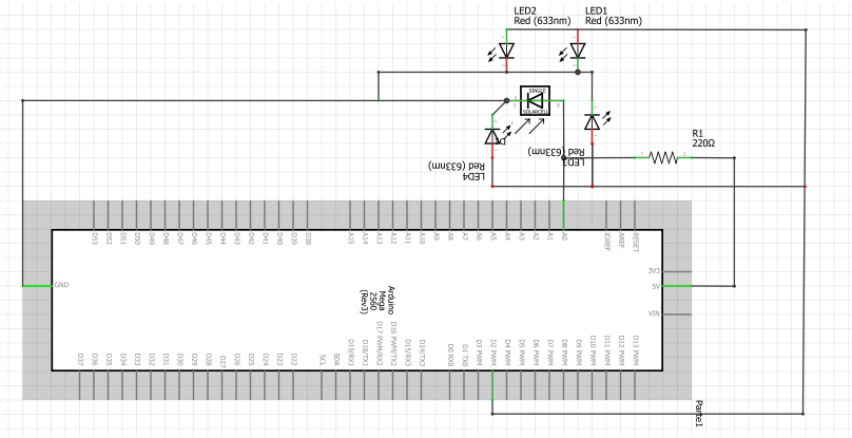


Figura 3.6 Conexión leds infrarrojos al Arduino

3.1.2. PRACTICA 1: USO DEL TECLADO MATRICIAL Y LCD

En esta práctica se hizo uso de la placa Arduino Mega, un teclado matricial 4x4 y una pantalla LCD 16x2 como los materiales principales.

Consta de 3 proyectos, los cuales son:

- TEST DE EDUCACIÓN VIAL
- INGRESO DE CONTRASEÑA
- CONTEO CON SENSOR DE PROXIMIDAD INFRARROJO

Antes de realizar esta práctica es importante agregar la librería Keypad.h, la cual es necesaria para poder hacer la programación del teclado matricial. Esta librería se la puede encontrar en la página principal de Arduino, donde también se encontraran indicaciones a seguir para incorporar esta librería.

3.1.2.1. OBJETIVOS

- Familiarizarse con el uso de las librerías LiquidCrystal.h (LCD) y Keypad.h (Teclado Matricial)

- Realizar interfaces de usuario usando un LCD 16x2 y un teclado matricial 4x4

3.1.2.2. TEST DE EDUCACION VIAL

Este proyecto se basa en el uso de un teclado matricial 4x4 y una pantalla LCD 16x2 con Arduino para tomar un test sencillo sobre educación vial.

A continuación se muestra el código de este proyecto:

```
#include <LiquidCrystal.h>   // Libreria para usar el LCD
#include <Keypad.h>          // Libreria para usar el teclado matricial
                             // descargar de http://playground.arduino.cc/Code/Keypad

int LEDverde = 52;  // Pin 52 (Digital) asignado al LED VERDE
int LEDrojo = 53;   // Pin 53 (Digital) asignado al LED ROJO

char Boton;    //  Variable tipo char a la cual se le asigna la tecla presionada
int x = 0;     //  Bandera para salir de lazo while
```

```
const byte filas = 4;        // Se define el numero de filas del teclado matricial

const byte columnas = 4;     // Se define el numero de columnas del teclado matricial


char keys[filas][columnas] = // Se definen los caracteres asociados a cada tecla del teclado

{

  {'1','2','3','A'},

  {'4','5','6','B'},

  {'7','8','9','C'},

  {'*','0','#','D'}

};


byte PinsFil[filas] = {44,45,46,47};         // Se definen los pines para las filas del teclado

byte PinsCol[columnas] = {48,49,50,51};     // Se definen los pines para las columnas del teclado


Keypad keypad = Keypad(makeKeymap(keys), PinsFil, PinsCol, filas, columnas); // Se crea el teclado con la informacion adquirida

LiquidCrystal lcd(36, 37, 38, 39, 40, 41); // Declaracion de pines para uso del LCD
```

```
void setup()

{

  lcd.begin(16, 2);               //Inicializacion del LCD, se declara que el LCD usado es
de 16 x 2



  pinMode(LEDrojo, OUTPUT);      // Declaracion de LEDrojo como salida

  pinMode(LEDverde, OUTPUT);     // Declaracion de LEDverde como salida

  digitalWrite(LEDrojo, LOW);    // Setea el valor de LEDrojo a 0

  digitalWrite(LEDverde, LOW);   // Setea el valor de LEDverde a 0

}



void loop()

{

  Bienvenida();

  Pregunta1();

  Pregunta2();

  Pregunta3();

}



void Bienvenida()  // Subrutina que indica el tiulo del proyecto

{
```

```
  int i=0;                          // Se declara la variable i en 0
  lcd.setCursor(3,0);                  // coloca el cursor en 3,0 (columna 4, fila 1 del
LCD)
  lcd.print("EXAMEN COMISION DE TRANSITO");   // Imprime el mensaje
  for (i=0;i < 17 ; i++)     // Lazo for para que la pantalla se desplace hacia la
izquierda cadda 500 ms
  {
    lcd.scrollDisplayLeft();        // Desplaza la pantalla hacia la izquierda
    delay (500);                            // Retardo de 500 ms antese del siguiente
desplazamiento
  }
  delay(1000);                      // Retardo de 1s
  lcd.clear();                      // Limpia la pantalla
}

void Correcto()  // Subrutina que indica que la respuesta es correcta
{
  lcd.clear ();                     // Limpia la pantalla
  digitalWrite(LEDverde, HIGH);     // Asigna Vcc para que el led verde se encienda
  lcd.setCursor(0,0);                  // coloca el cursor en 0,0 (columna 1, fila 1 del
LCD)
  lcd.print("Presiono :");          // Imprime el mensaje
```

```
  lcd.setCursor(13,0);                // coloca el cursor en 13,0 (columna 14, fila 1 del
LCD)
  lcd.print(Boton);                  // Imprime el caracter de la tecla presionada
  lcd.setCursor(2,1);                   // coloca el cursor en 2,1 (columna 3, fila 2 del
LCD)
  lcd.print("Resp Correcta");        // Imprime el mensaje
  delay(5000);                       // Retardo de 5000 ms antes de apagar el led rojo
  digitalWrite(LEDverde, LOW);       // Apaga el led verde
  x = 1;                                // Se asigna el valor de 1 a x para salir de lazo
while
}

void Incorrecto()  // Subrutina que indica que la respuesta es incorrecta
{
  lcd.clear ();                      // Limpia la pantalla
  digitalWrite(LEDrojo, HIGH);       // Asigna Vcc para que el led rojo se encienda
  lcd.setCursor(0,0);                   // coloca el cursor en 0,0 (columna 1, fila 1 del
LCD)
  lcd.print("Presiono :");           // Imprime el mensaje
  lcd.setCursor(13,0);                // coloca el cursor en 13,0 (columna 14, fila 1 del
LCD)
  lcd.print(Boton);                  // Imprime el caracter de la tecla presionada
```

```
  lcd.setCursor(3,1);                  // coloca el cursor en 3,1 (columna 4, fila 2 del
LCD)
  lcd.print("INCORRECTO");           // Imprime el mensaje


  delay(5000);                        // Retardo de 5000 ms antes de apagar el led rojo
  digitalWrite(LEDrojo, LOW);         // Apaga el led rojo
  x = 1;                               // Se asigna el valor de 1 a x para salir de lazo
while
}


void Pregunta1()   // Subrutina que realiza la primera pregunta
{
  lcd.clear();                         // Limpia la pantalla
  lcd.setCursor(3,0);                   // coloca el cursor en 3,0 (columna 4, fila 1 del
LCD)
  lcd.print("CON LA LUZ VERDE UD :");  // Imprime el mensaje (pregunta)
  while(Boton != '*')                   // Mientras Boton no sea igual a * se realiza lo
siguiente (para seguir hay que presionar *)
  {
     Boton = keypad.getKey();          // Obtiene la tecla presionada
    lcd.scrollDisplayLeft();           // Mueve la pantalla hacia la izquierda
    delay (300);                       // Retardo de 300 ms
```

```
  }
  delay(1000);                          // Retardo de 1s


  lcd.clear();                          // Limpia la pantalla
  lcd.setCursor(3,0);                    // coloca el cursor en 3,0 (columna 4, fila 1 del
LCD)
  lcd.print("A)Para     B)Continua");  // Imprime el mensaje (respuestas)
  lcd.setCursor(3,1);                    // coloca el cursor en 3,1 (columna 4, fila 2 del
LCD)
  lcd.print("C)Pita     D)Baja del carro");  // Imprime el mensaje (respuestas)
  x = 0;                                // Bandera igual a 0
  while(x == 0 )                         // Mientras la bandera sea igual a 0 se realiza lo
siguiente
  {
    Boton = keypad.getKey();            // Obtiene la tecla presionada
    lcd.scrollDisplayLeft();            // Mueve la pantalla hacia la izquierda
    delay (300);                        // Retardo de 300 ms
    if(Boton == 'B')                          // Si el boton presionado es B entonces la
respuesta es correcta
    {
     Correcto();                        // Se llama a la subrutina correcto
    }
```

```
    else if(Boton == 'A'|| Boton == 'C' || Boton == 'D')  // Si el boton presionado es A
, C o D entonces la respuesta es incorrecta
    {
     Incorrecto();                      // Se llama a la subrutina incorrecto
    }
  }
  delay(1000);                          // Retardo de 1000 ms
  lcd.clear();                          // Limpia la pantalla
}

void Pregunta2()                        // Subrutina que realiza la segunda pregunta
{
  lcd.clear();                          // Limpia la pantalla
  lcd.setCursor(3,0);                    // coloca el cursor en 3,0 (columna 4, fila 1 del
LCD)
  lcd.print("CON LA LUZ ROJA UD :");    // Imprime el mensaje (pregunta)
  while(Boton != '*')                    // Mientras Boton no sea igual a * se realiza lo
siguiente (para seguir hay que presionar *)
  {
    Boton = keypad.getKey();            // Obtiene la tecla presionada
    lcd.scrollDisplayLeft();            // Mueve la pantalla hacia la izquierda
```

```
    delay (300);                        // Retardo de 300 ms
  }
  delay(1000);                          // Retardo de 1s
  lcd.clear();                          // Limpia la pantalla
  lcd.setCursor(3,0);                    // coloca el cursor en 3,0 (columna 4, fila 1 del
LCD)
  lcd.print("A)Para     B)Continua");  // Imprime el mensaje (respuestas)
  lcd.setCursor(3,1);                    // coloca el cursor en 3,1 (columna 4, fila 2 del
LCD)
  lcd.print("C)Pita     D)Baja del carro"); // Imprime el mensaje (respuestas)
  x = 0;                                // Bandera igual a 0
  while(x == 0)                          // Mientras la bandera sea igual a 0 se realiza lo
siguiente
  {
     Boton = keypad.getKey();           // Obtiene la tecla presionada
    lcd.scrollDisplayLeft();            // Mueve la pantalla hacia la izquierda
    delay (300);                        // Retardo de 300 ms
    if(Boton == 'A')                          // Si el boton presionado es A entonces la
respuesta es correcta
    {
     Correcto();                        // Se llama a la subrutina correcto
    }
```

```
    else if(Boton == 'B'|| Boton == 'C' || Boton == 'D' )  // Si el boton presionado es B
, C o D entonces la respuesta es incorrecta
    {
     Incorrecto();                      // Se llama a la subrutina incorrecto
    }
  }
  delay(1000);                          // Retardo de 1000 ms
  lcd.clear();                          // Limpia la pantalla
}



void Pregunta3()                        // Subrutina que realiza la tercera pregunta
{
  lcd.clear();                          // Limpia la pantalla
  lcd.setCursor(2,0);                    // coloca el cursor en 2,0 (columna 3, fila 1 del
LCD)
  lcd.print("CON LA LUZ AMARILLA UD :");  // Imprime el mensaje (pregunta)
  while(Boton != '*')                    // Mientras Boton no sea igual a * se realiza lo
siguiente (para seguir hay que presionar *)
  {
    Boton = keypad.getKey();            // Obtiene la tecla presionada
```

```
    lcd.scrollDisplayLeft();           // Mueve la pantalla hacia la izquierda
    delay (300);                       // Retardo de 300 ms
  }
  delay(1000);                         // Retardo de 1000 ms
  lcd.clear();                         // Limpia la pantalla
  lcd.setCursor(3,0);                   // coloca el cursor en 3,0 (columna 4, fila 1 del
LCD)
  lcd.print("A)Para     B)Continua");  // Imprime el mensaje (respuestas)
  lcd.setCursor(3,1);                   // coloca el cursor en 3,1 (columna 4, fila 2 del
LCD)
  lcd.print("C)Pita     D)Baja del carro");  // Imprime el mensaje (respuestas)
  x = 0;                               // Bandera igual a 0
  while(x == 0)                         // Mientras la bandera sea igual a 0 se realiza lo
siguiente
  {
    Boton = keypad.getKey();          // Obtiene la tecla presionada
    lcd.scrollDisplayLeft();           // Mueve la pantalla hacia la izquierda
    delay (300);                       // Retardo de 300 ms
    if(Boton == 'A')                        // Si el boton presionado es A entonces la
respuesta es correcta
    {
     Correcto();                       // Se llama a la subrutina correcto
```

```
    }
    else if(Boton == 'B'|| Boton == 'C' || Boton == 'D' )  // Si el boton presionado es B
, C o D entonces la respuesta es incorrecta
    {
     Incorrecto();                    // Se llama a la subrutina incorrecto
    }
  }
  delay(1000);                        // Retardo de 1000 ms
  lcd.clear();                        // Limpia la pantalla
}
```

3.1.2.3. INGRESO DE CONTRASEÑA

Este proyecto simula una cerradura electrónica, la cual solo puede ser abierta cuando la contraseña ingresada es la correcta. El led verde indica que la contraseña es correcta y el led rojo indica que la contraseña es incorrecta.

A continuación se muestra el código del proyecto:

```
#include <LiquidCrystal.h>   // Libreria para usar el LCD
#include <Keypad.h>          // Libreria para usar el teclado matricial
                             // descargar de http://playground.arduino.cc/Code/Keypad
```

```
int LEDverde = 52;           // Asignacion de pin 52 a led verde

int LEDrojo = 53;            // Asignacion de pin 53 a led rojo


char* Codigo = "1234";       // Variable tipo char donde se guarda el codigo

int n = strlen(Codigo);        // Variable tipo entero donde se guarda la longitud del
codigo

int PosicionActual = 0;


int Correcto = 0;


const byte filas = 4;        // Se define el numero de filas del teclado matricial

const byte columnas = 4;     // Se define el numero de columnas del teclado matricial


char keys[filas][columnas] = // Se definen los caracteres asociados a cada tecla del
teclado

{

  {'1','2','3','A'},

  {'4','5','6','B'},

  {'7','8','9','C'},

  {'*','0','#','D'}

};
```

```
byte PinsFil[filas] = {44,45,46,47};         // Se definen los pines para las filas del teclado

byte PinsCol[columnas] = {48,49,50,51};     // Se definen los pines para las columnas del teclado


Keypad keypad = Keypad(makeKeymap(keys), PinsFil, PinsCol, filas, columnas); // Se crea el teclado

                                                                                //    con  la informacion adquirida

LiquidCrystal lcd(36, 37, 38, 39, 40, 41); // Declaracion de pines para uso del LCD


void setup()

{

  lcd.begin(16, 2);               //Inicializacion del LCD, se declara que el LCD usado es de 16 x 2


  PantallaPrincipal();           // Llamada a la subrutina PantallaPrincipal


  pinMode(LEDrojo, OUTPUT);      // Declaracion de LEDrojo como salida

  pinMode(LEDverde, OUTPUT);     // Declaracion de LEDverde como salida
```

```
  digitalWrite(LEDrojo, LOW);    // Setea el valor de LEDrojo a 0
  digitalWrite(LEDverde, LOW);   // Setea el valor de LEDverde a 0
}

void loop()
{
  int l;                         // variable tipo int para lazo for

  char Boton = keypad.getKey();  // Se guarda en Boton la tecla presionada

 if (int (Boton) != 0)           // Si una tecla es presionada sucede lo siguiente
 {
    lcd.setCursor(2,1);          // coloca el cursor en 2,1 (columna 3, fila 2 del LCD)

    for (l=0; l<=PosicionActual; l++) // Para cada tecla presionada se escribe un
asterisco en el LCD
    {
      lcd.print("*");
    }
```

```
    if (Boton == Codigo[PosicionActual] && Correcto == PosicionActual) // Si la tecla presionada es igual a la letra
                                                                       // que corresponde dicha posicion y el numero
                                                                       //   de   aciertos (Correcto) es igual a la posicion
                                                                       //         actual, entonces:
      {
        Correcto++;             // Correcto se incrementa
        PosicionActual++;       // PosicionActual se incrementa
        if (PosicionActual == n && Correcto == n)   // Si PosicionActual y elnumero de aciertos (Correcto)es igual
                                                    // al numero de caracteres que tiene la clave, entonce:
        {
          CodigoValido();       // Se llama a la subrutina CodigoValido
          PosicionActual = 0;   // Se asigna valor 0
          Correcto = 0;         // Se asigna valor 0
        }
      }
     else                           // Si la tecla presionada no es igual a la letra que corresponde dicha posicion
```

```
      {                           // y el numero de aciertos (Correcto) no es igual a la posicion actual, entonces:
        PosicionActual++;       // PosicionActual se incrementa
        if (PosicionActual == n && Correcto != n) // Si PosicionActual es igual a n y correcto diferente de n
        {
          CodigoInvalido();     // Se llama a la subrutina CodigoValido
          PosicionActual = 0;   // Se asigna valor 0
          Correcto = 0;         // Se asigna valor 0
        }
      }
  }
}
void PantallaPrincipal()  // Subrutina que escribe el mensaje principal
{
  lcd.clear();                          // Limpia la pantalla
  lcd.setCursor(0,0);                     // coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
  lcd.print("Entre el codigo secreto");  // Imprime el mensaje
}
void CodigoValido() // Subrutina que se encarga de escribir que el codigo fue correcto y enciende el led de acceso
```

```
{
  delay (800);                        // Retardo de 800 ms antes de borrar la pantalla

  digitalWrite(LEDverde, HIGH);       // Asigna Vcc para que el led verde se encienda

  lcd.clear();                        // Limpia la pantalla
  lcd.setCursor(0,0);                    // coloca el cursor en 0,0 (columna 1, fila 1 del
LCD)
  lcd.print("ACCESO PERMITIDO!");     // Imprime el mensaje
  lcd.setCursor(0,1);                    // coloca el cursor en 0,1 (columna 1, fila 2 del
LCD)
  lcd.print("CODIGO CORRECTO");       // Imprime el mensaje

  delay(2000);                        // Retardo de 2000 ms antes de apagar el led
  digitalWrite(LEDverde, LOW);        // Se apaga el led verde
  PantallaPrincipal();                // Llama a la subrutina PantallaPrincipal
}

void CodigoInvalido()// Subrutina que se encarga de escribir que el codigo fue incorrecto
y enciende el led rojo
{
```

```
  delay (800);                      // Retardo de 800 ms antes de borrar la pantalla
  digitalWrite(LEDrojo, HIGH);      // Asigna Vcc para que el led rojo se encienda


  lcd.clear ();                     // Limpia la pantalla
  lcd.setCursor(0,0);                  // coloca el cursor en 0,0 (columna 1, fila 1 del
LCD)
  lcd.print("*Acceso Negado!*");    // Imprime el mensaje
  lcd.setCursor(0,1);                  // coloca el cursor en 0,1 (columna 1, fila 2 del
LCD)
  lcd.print("Codigo Invalido");     // Imprime el mensaje


  delay(2000);                      // Retardo de 2000 ms antes de apagar el led
  digitalWrite(LEDrojo, LOW);       // Se apaga el led rojo
  PantallaPrincipal();              // Llama a la subrutina PantallaPrincipal
}
```

3.1.2.4. CONTEO CON SENSOR DE PROXIMIDAD INFRARROJO

Este proyecto indica en pantalla cuantas veces un objeto se acerca a un sensor de proximidad armado con 4 leds infrarrojo emisores y un led receptor. Para poder iniciar

el conteo hay que presionar el boton A y para encerar el contador se presiona el botón B.

A continuación se muestra el código del proyecto:

```
#include <LiquidCrystal.h>   // Libreria para usar el LCD
#include <Keypad.h>          // Libreria para usar el teclado matricial
                             // descargar de http://playground.arduino.cc/Code/Keypad

char Boton;    //  Variable tipo char a la cual se le asigna la tecla presionada
const byte filas = 4;        // Se define el numero de filas del teclado matricial
const byte columnas = 4;     // Se define el numero de columnas del teclado matricial

char keys[filas][columnas] = // Se definen los caracteres asociados a cada tecla del teclado
{
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
```

```
byte PinsFil[filas] = {44,45,46,47};         // Se definen los pines para las filas del teclado

byte PinsCol[columnas] = {48,49,50,51};     // Se definen los pines para las columnas del teclado


Keypad keypad = Keypad(makeKeymap(keys), PinsFil, PinsCol, filas, columnas); // Se crea el teclado con la informacion adquirida


int Emisor = 33;    // Pin 33 (Digital) asignado a los 4 leds emisores (IR)

int Receptor = A2;  // Pin A2 (Entrada Analogica) asignado al led receptor (IR)


int Sensor ;        // Variable donde se guardan los datos que obtiene el receptor

int contador = 0;   // Variable donde se guarda el conteo de las interrupciones


int bandera = 0;    // Bandera que permite que empiece el conteo o no


LiquidCrystal lcd (36, 37, 38, 39, 40, 41); // Declaracion de pines para uso del LCD


void setup()

{
```

```
  pinMode(Emisor,OUTPUT);      // Declaracion de Emisor como salida
  digitalWrite(Emisor,LOW);    // Escritura de voltaja bajo (0) en Emisor

  lcd.begin(16, 2);             //Inicializacion del LCD, se declara que el LCD usado es de 16 x 2
  Serial.begin(9600);          //Inicializacion del Monitor Serial
}

void loop ()
{
    Boton = keypad.getKey();           //Obtiene la tecla presionada

    digitalWrite(Emisor,HIGH);          // Sen encienden los leds emisores para enviar la señal IR
    delay(2);                          // Retardo minimo para poder leer los valores
    Sensor = analogRead(Receptor);     // Se asignan a Sensor los datos recibidos
    digitalWrite(Emisor,LOW);          // Sen apagan los leds emisores
    delay(2);

    Serial.println (Sensor);              // Imprime los datos del Sensor en el Monitor Serial
```

```
    if(Boton == 'A')                    // Si la tecla presionada es A la bandera es 1
    {
      bandera = 1;                      // Bandera toma el valor de 1
    }

    else if(Boton == 'B')               // Si la tecla presionada es B la bandera es 0
    {
      bandera = 0;                      // Bandera toma el valor de 0
    }

    if (bandera == 1)                       // Si la bandera es 1 se procede a contar
interrupciones
    {
       lcd.clear ();                // Se limpia la pantalla
       lcd.setCursor(0,0);           // se coloca el cursor en 0,0 (columna 1, fila 1 del
LCD)
       lcd.print ("CONTEO :  ");    // Se imprime "CONTEO :  " en LCD
       lcd.print(contador);         // Se imprime el valor de contador en LCD
```

```
      if (Sensor < 1011)    // Si el sensor detecta un obstaculo (rango menor de 1010) se
realiza lo siguiente :
      {
        lcd.clear ();               // Se limpia la pantalla
        lcd.setCursor(0,0);          // se coloca el cursor en 0,0 (columna 1, fila 1 del
LCD)
        lcd.print ("CONTEO :  ");   // Se imprime "CONTEO :  " en LCD
        contador++;                 // Se incrementa el valor de contador
        lcd.print(contador);        // Se imprime el valor de contador en LCD
        delay (500);                // Retardo de 500 ms antes de realizar otra cuenta
      }
      delay (20);            // Retardo de 20 ms antes de imprimir pantalla otra vez
    }
    else
    {
       lcd.clear ();                // Se limpia la pantalla
       lcd.setCursor(0,0);           // se coloca el cursor en 0,0 (columna 1, fila 1 del
LCD)
       lcd.print ("Presione A");    // Se imprime "Presione A" en LCD
       lcd.setCursor(0,1);           // se coloca el cursor en 0,1 (columna 1, fila 2 del
LCD)
       lcd.print ("para empezar");  // Se imprime "para empezar" en LCD
```

```
   contador = 0;

   delay (20);                  // Retardo de 10 ms

}
```

3.2. MOTORES DC

En esta sección se revisará el uso de los motores DC controlados por la placa Arduino Mega 2560 a través del integrado L293 (Puente H).

3.2.1. PREPARACIÓN PARA LA PRÁCTICA 2

Para poder realizar la práctica 2 es necesario tener conocimientos acerca del funcionamiento y conexión de los siguientes elementos:

- Motor DC
- Integrado L293

3.2.1.1. Motor DC

Un motor de corriente directa (CD), también denominado motor de corriente continua es una maquina que convierte la energía eléctrica en energía mecánica [11].

Consta de cinco partes principales, las cuales son:

Armadura o rotor.

Conmutador o colector.

Cepillos o escobillas.

Eje.

Imán de campo o estator

En la figura 3.7 se pueden apreciar las partes del tipo de motor que se usó en esta práctica.

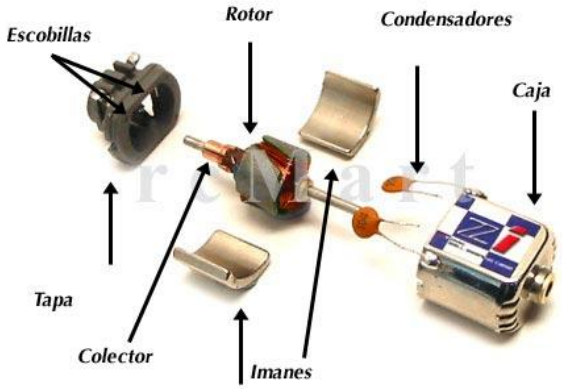


Figura 3.7 Partes del motor DC [http://www.insflug.org/shots/motorz.jpg].

Cuando la corriente pasa por las bobinas de la armadura, estas generan un campo magnético que interactúan con el campo magnético del estator creando así la torsión.

Existen variaciones en la construcción de este tipo de motores, estas son:

Estator bobinado

Imán permanente

Sin escobillas

En este caso se uso un motor DC con imanes permanentes, del tipo que se encuentran en los juguetes.

3.2.1.2. Puente H (L293)

Un puente H es un circuito muy conocido en el área de la electrónica y es comúnmente usado para manejar motores DC. Para efectuar esta práctica se uso el integrado L293, el cual consta de cuatro medios puentes H, los cuales se pueden usar independientemente (Medio Puente H + Motor DC = Un Sentido de Giro) o formar dos puentes H completos (Puente H + Motor DC = Dos Sentidos de Giro) [12]. En la figura 3.8 se muestra como se usa el integrado L293 para controlar motores con medio puente H o con un puente H completo.

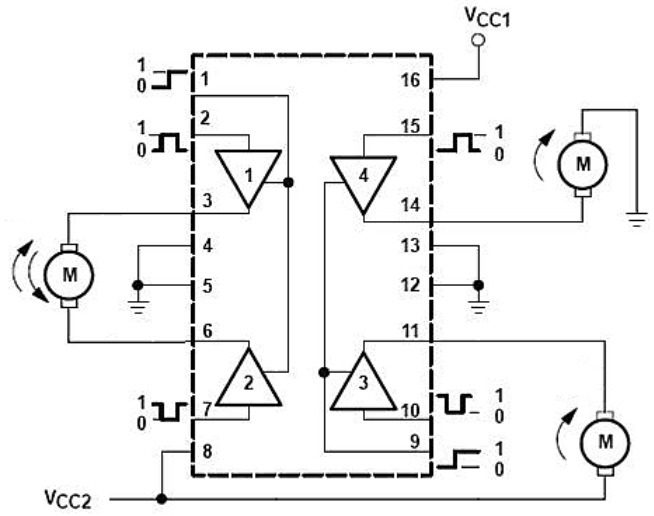


Figura 3.8 Conexión de motor DC con puente H completo (lado izquierdo) y con medio puente H (lado derecho).

La conexión que se uso para esta práctica es la de puente H completo. Las conexiones realizadas para esta práctica entre el Arduino, el integrado L293 y el motor DC se muestran en la tabla 3.3 y la figura3.9.

Tabla 3.3 Conexión entre Arduino, L293, Motor DC y Fuente Externa

Arduino	**L293**	**Motor DC**	**Fuente externa**
5v	9, 16	-	-
GND	12, 13	-	GND
8 (Pin Digital PWM)	15	-	-
9 (Pin Digital PWM)	10	-	-
-	8	-	Vcc
-	11	Positivo	-
-	14	Negativo	-

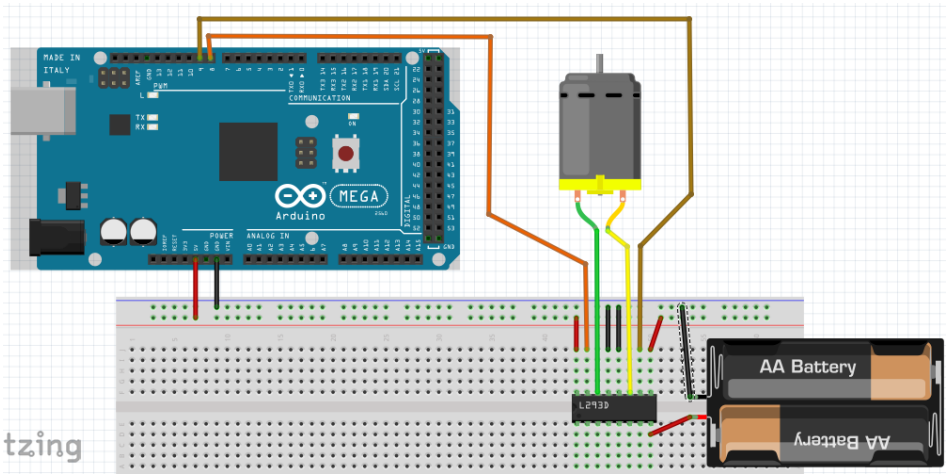


Figura 3.9 Conexión Arduino + L293 + Motor DC

3.2.2. PRÁCTICA 2: USO DE LOS MOTORES DC

En esta práctica se hizo uso de la placa Arduino Mega, un integrado L293 (Puente H) y un motor DC como los materiales principales. También se hizo uso de un teclado matricial 4x4, un LCD de 16x2 y un potenciómetro de 10K. Las conexiones de estos últimos se los puede ver en la preparación para la practica 1.

La práctica 2 consta de 3 proyectos, los cuales son:

- CONTROL DE VELOCIDAD ANALÓGICO DE UN MOTOR DC
- SECUENCIA DE GIRO DE UN MOTOR DC

- CAMBIO DE GIRO DE UN MOTOR DC MEDIANTE UN SENSOR DE PROXIMIDAD INFRARROJO

3.2.2.1. OBJETIVOS

- Familiarizarse con el uso de entradas analógicas
- Familiarizarse con el uso de salidas con PWM para control de velocidad de los motores DC
- Realizar el control de velocidad y sentido de giro de un motor DC

3.2.2.2. CONTROL DE VELOCIDAD ANALÓGICO DE UN MOTOR DC

Este proyecto se basa en el control de velocidad de un motor DC. Para lograr esto se uso un potenciómetro el cual va conectado a una entrada analógica del Arduino Mega y mediante las salidas PWM conectadas a un puente H (L293) se controla la velocidad del motor. Los datos de velocidad del motor se muestran en pantalla (LCD).

A continuación se muestra el código de este proyecto:

```
#include <LiquidCrystal.h> //Libreria para LCD


const int motorpin1 = 8;   //Se asigna al pin 8 (PWM) el control 1 del motor


const int motorpin2 = 9;   //Se asigna al pin 8 (PWM) el control 1 del motor


const int Pot = A0;        //Se asigna al pin A0 (Entrada analogica) el potenciometro


int velocidad = 0;         //variable para dar la velocidad al motor

int valor = 0;             //variable que recoge los datos del potenciometro

int velimp = 0;            //variable para imprimir en pantalla la velocidad del motor


LiquidCrystal lcd(36, 37, 38, 39, 40, 41); // Declaracion de pines para uso del LCD


void setup ()

{


  lcd.begin(16,2);     //Inicializacion del LCD, se declara que el LCD usado es de 16 x 2

  lcd.clear();         //Limpia la pantalla
```

```
  pinMode(motorpin1,OUTPUT);       // Declaracion del control 1 como salida

  pinMode(motorpin2,OUTPUT);       // Declaracion del control 2 como salida

}

void loop ()

{
  lcd.setCursor(0,0);    // coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
  lcd.print("VELOCIDAD MOTOR:");  // Imprime en pantalla el mensaje
  lcd.setCursor(3,1);             // coloca el cursor en 3,0 (columna 4, fila 2 del LCD)
  valor = analogRead (Pot);       // Obtiene valores 0 - 1023 del potenciometro
  velimp = map(valor, 15, 1010, 0, 100);  //Convierte los valores obtenidos por valor (0
- 1023)
                                          //a  una  escala  de  0  -  100  para  poder  ser
impreso en LCD

  velocidad = map(valor, 0, 1023, 0, 255);//Convierte los valores obtenidos por valor (0
- 1023)
```

```
                                        //a una escala de 0 - 255 que son los rangos de
salida
                                        //PWM para luego ser asignados al motor


  lcd.print(velimp);    // Imprime en pantalla la velocidad que tiene el motor
  lcd.print("%");       // Imprime en pantalla el signo %
  lcd.print("   ");     // Imprime en pantalla espacios


  analogWrite(motorpin1,velocidad);      //Se asigna al control del motro 1 el valor de
velocidad
  digitalWrite(motorpin2,LOW);           //Se asigna al control del motro 2 un valor bajo
(GND)


  delay (50);    //retardo de 50 ms
}
```

3.2.2.3. SECUENCIA DE GIRO DE UN MOTOR DC

Este proyecto se basa en cómo controlar el sentido de giro y la velocidad de un motor DC. Para empezar la secuencia es necesario presionar el botón A del teclado

matricial y a continuación se mostrara en pantalla (LCD) el sentido de giro y velocidad del motor.

A continuación se muestra el código de este proyecto:

```
#include <LiquidCrystal.h>  //Libreria para LCD
#include <Keypad.h>          // Libreria para usar el teclado matricial
                             // descargar de http://playground.arduino.cc/Code/Keypad


char Boton;    //  Variable tipo char a la cual se le asigna la tecla presionada
const byte filas = 4;        // Se define el numero de filas del teclado matricial
const byte columnas = 4;     // Se define el numero de columnas del teclado matricial


char keys[filas][columnas] = // Se definen los caracteres asociados a cada tecla del
teclado
{
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
```

```
byte PinsFil[filas] = {44,45,46,47};       // Se definen los pines para las filas del
teclado

byte PinsCol[columnas] = {48,49,50,51};    // Se definen los pines para las columnas del
teclado


Keypad keypad = Keypad(makeKeymap(keys), PinsFil, PinsCol, filas, columnas); // Se crea
el teclado con la informacion adquirida


int IN1 = 8;  //Se asigna al pin 8 (PWM) el control 1 del motor

int IN2 = 9;   //Se asigna al pin 9 (PWM) el control 2 del motor


int velocidad;

int bandera = 0;    // Bandera que permite que empiece la secuencia o no

LiquidCrystal lcd (36, 37, 38, 39, 40, 41); // Declaracion de pines para uso del LCD


void setup ()

{


  pinMode (IN1, OUTPUT);  // Declaracion del control 1 como salida

  pinMode (IN2, OUTPUT);  // Declaracion del control 2 como salida
```

```
  lcd.begin(16, 2);       //Inicializacion del LCD, se declara que el LCD usado es de 16
x 2



}
void loop()
{
  Boton = keypad.getKey();           //Obtiene la tecla presionada


  if(Boton == 'A')                   // Si la tecla presionada es A la bandera es 1
  {


    bandera = 1;                     // Bandera toma el valor de 1
  }


  if (bandera == 1)                // Si la bandera es iguan a 1 se procede con secuencia
  {
    derecha_50 ();
    paro ();
    izquierda_50 ();
    paro ();
    bandera = 0;
```

```
  }
  else
  {
    lcd.clear();                  // Limpia la pantalla
    lcd.setCursor(0,0);           // coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
    lcd.print ("Presione A");     // Imprime en pantalla el mensaje
    lcd.setCursor(0,1);           // coloca el cursor en 0,0 (columna 1, fila 2 del LCD)
    lcd.print ("Para Empezar");   // Imprime en pantalla el mensaje
    delay (50);                   // Retardo de 50 ms
  }
}
void derecha_50 () // Subrutina que hace girar al motor en sentido horario
                   // hasta una velocidad de 50%
{
  velocidad = 0;      // Se asigna una velocidad de 0%
  for (int i = 0; i < 130; i=i+2) //Lazo for que hace que aumente la velocidad de a poco
  {
    lcd.clear();                  // Limpia la pantalla
    lcd.setCursor(0,0);           // coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
    lcd.print ("Giro Derecha");   // Imprime en pantalla el mensaje
```

```
    lcd.setCursor(0,1);           // coloca el cursor en 0,0 (columna 1, fila 2 del LCD)
    lcd.print ("Velocidad : ");   // Imprime en pantalla el mensaje
    analogWrite (IN1, i);         // Se asigna al control del motro 1 el valor de
velocidad
    digitalWrite (IN2, LOW);      //Se asigna al control del motro 2 un valor bajo (GND)

    velocidad = map (i, 0, 255, 0, 100);  //Convierte los valores de la variable i (0 -
255)
                                          //a una escala de 0 - 100 para poder
imprimirlos
                                          //en pantalla
    lcd.print(velocidad);         // Imprime en pantalla la velocidad que tiene el motor
    lcd.print("%   ");            // Imprime en pantalla el signo % con espacios
    delay (50);                   // retardo de 50 ms entre iteracion
  }
  delay (3000);                   //retardo de 3s para mantener velocidad maxima
}

void izquierda_50 () // Subrutina que hace girar al motor en sentido antihorario
                  // hasta una velocidad de 50%
{
```

```
  velocidad = 0;      // Se asigna una velocidad de 0%
  for (int i = 0; i < 130; i=i+2) //Lazo for que hace que aumente la velocidad de a poco
  {
    lcd.clear();                  // Limpia la pantalla
    lcd.setCursor(0,0);           // coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
    lcd.print ("Giro Izquierda");   // Imprime en pantalla el mensaje
    lcd.setCursor(0,1);           // coloca el cursor en 0,0 (columna 1, fila 2 del LCD)
    lcd.print ("Velocidad : ");   // Imprime en pantalla el mensaje
    analogWrite (IN2, i);         // Se asigna al control del motro 2 el valor de
velocidad
    digitalWrite (IN1, LOW);      //Se asigna al control del motro 1 un valor bajo (GND)

    velocidad = map (i, 0, 255, 0, 100);  //Convierte los valores de la variable i (0 -
255)
                                          //a una escala de 0 - 100 para poder
imprimirlos
                                          //en pantalla
    lcd.print(velocidad);         // Imprime en pantalla la velocidad que tiene el motor
    lcd.print("%   ");            // Imprime en pantalla el signo % con espacios
    delay (50);                   // retardo de 50 ms entre iteracion
  }
```

```
  delay (3000);                    //retardo de 3s para mantener velocidad maxima
}


void paro ()
{
  lcd.clear();                     // Limpia la pantalla
  lcd.setCursor(0,0);              // coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
  lcd.print ("Motor Detenido");    // Imprime en pantalla el mensaje
  lcd.setCursor(0,1);              // coloca el cursor en 0,0 (columna 1, fila 2 del LCD)
  lcd.print ("Velocidad : ");      // Imprime en pantalla el mensaje
  analogWrite (IN1, LOW);          //Se asigna al control del motro 1 un valor bajo (GND)
  digitalWrite (IN2, LOW);         //Se asigna al control del motro 2 un valor bajo (GND)
  lcd.print("0%");                 // Imprime en pantalla el mensaje "0%"
  delay (1500);                    // retardo de 1.5s entre para detener el motor
}
```

3.2.2.4. CAMBIO DE GIRO DE UN MOTOR DC MEDIANTE UN SENSOR DE PROXIMIDAD INFRARROJO

Este proyecto se basa en el control de giro de un Motor DC mediante un sensor de proximidad infrarrojo. Cuando un objeto pasa cerca del sensor de proximidad el motor DC

para y hace un cambio de giro. Los datos de sentido de giro y velocidad del motor son mostrados en pantalla (LCD).

A continuación se muestra el código de este proyecto:

```
#include <LiquidCrystal.h>

int Control1 = 8;   // Pin 9 (PWM) control 1 para el motor DC

int Control2 = 9;   // Pin 8 (PWM) control 2 para el motor DC




int Emisor = 33;    // Pin 33 (Digital) asignado a los 4 leds emisores (IR)

int Receptor = A2;  // Pin A2 (Entrada Analogica) asignado al led receptor (IR)


int Sensor ;        // Variable donde se guardan los datos que obtiene el receptor

int contador = 0;   // Variable donde se guarda el conteo de las interrupciones


int i;              // Variable usada para asignar velocidad al motor

int velocidad;      // Variable usada para mostrar en LCD la velocidad en %


LiquidCrystal lcd (36, 37, 38, 39, 40, 41); // Declaracion de pines para uso del LCD
```

```
void setup ()

{

  pinMode (Control1, OUTPUT);  // Declaracion de Control1 como salida

  pinMode (Control2, OUTPUT);  // Declaracion de Control2 como salida



  pinMode(Emisor,OUTPUT);      // Declaracion de Emisor como salida

  digitalWrite(Emisor,LOW);    // Escritura de voltaja bajo (0) en Emisor


//Notese que Receptor no esta declarado como entrada, esto se debe a que por default estos pines solo son de entrada


  lcd.begin(16, 2);             //Inicializacion del LCD, se declara que el LCD usado es de 16 x 2

  Serial.begin (9600);

  delay (3000);       //retardo de 3 s antes de iniciar el programa

}

void loop()

{

    if ((contador % 2) == 0)  // Si el numero que contiene contador es par se ejecuta Direccion1
```

```
    {
     Direccion1();
    }

    else                        // Si el numero que contiene contador es impar se ejecuta
Direccion2
    {
      Direccion2();
    }
}

void IR ()    // Subrutina para enviar y recibir la informacion de los leds
{
    digitalWrite(Emisor,HIGH);             // Sen encienden los leds emisores para enviar
la señal IR
    delay(1);                             // Retardo minimo para poder leer los valores
    Sensor = analogRead(Receptor);        // Se asignan a Sensor los datos recibidos
    digitalWrite(Emisor,LOW);             // Sen apagan los leds emisores
    delay(1);
```

```
    Serial.println (Sensor);              // Imprime los datos del Sensor en el Monitor Serial


    if (Sensor < 1009)                        // Si el sensor detecta un obstaculo (rango menor 990 de 1023) se realiza lo siguiente :
    {
      contador++;                          // Aumenta la cuenta del contador
    //  delay (500);                         // Lo mantiene encendido 500 ms
    }
}


void Direccion1 ()      // Subrutina que imprime en el LCD y en el motor el sentido de giro y velocidad del motor
{
  lcd.clear();                    // se limpia la pantalla
  lcd.setCursor(0,0);               // se coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
  lcd.print ("Giro Derecha");     // Se imprime "Giro Derecha" en LCD
  lcd.setCursor(0,1);               // se coloca el cursor en 0,1 (columna 1, fila 2 del LCD)
  lcd.print ("Velocidad: ");      // Se imprime "Velocidad: " en LCD
```

```
  i = 0;                          // Se inicializa la velocidad del motor en 0


  while ((i < 256) && ((contador % 2) == 0)) // Mientras se cumpla que velocidad del motor sea menor que 256 (255 maxima)
  {                                          // y que el numero de contador sea par se realiza lo siguiente :


    IR ();                         // Se llama a la subrutina IR () para enviar y recibir valores


    lcd.clear();                   // se limpia la pantalla
    lcd.setCursor(0,0);             // se coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
    lcd.print ("Giro Derecha");    // Se imprime "Giro Derecha" en LCD
    lcd.setCursor(0,1);             // se coloca el cursor en 0,1 (columna 1, fila 2 del LCD)
    lcd.print ("Velocidad: ");     // Se imprime "Velocidad: " en LCD


    analogWrite (Control1, i);      // Se le da a contol 1 la velocidad dependiendo del valor de i para que contole al motor
    digitalWrite (Control2, LOW);  // Se mantiene el valor de control 2 bajo y de esta forma el sentido de giro del motor es hacia la derecha
```

```
    velocidad = map(i, 0, 255, 0, 100) ; // Cambia el rango de velocidad de (0 - 255) a
(0 - 100)
    lcd.print(velocidad);             // Se imprime en el LCD la velocidad del motor en
porcentaje
    lcd.print("%   ");              // Se imprime el signo de porcentaje
    delay (50);                     // Retardo de 50 ms


    i = i + 2;                      // Se incrementa la velocidad en 2


    if (i >= 255)                   // Condicion if para mantener la velocidad maxima
    {
     i = 255;
    }
  }
  delay (500);                     //Retardo de 2s para tener mantener la velocidad maxima
  digitalWrite (Control1, LOW);     // Se asigna a Control1 un valor bajo y con esto el
motor se detiene
  lcd.setCursor(0,1);               // se coloca el cursor en 0,1 (columna 1, fila 2 del
LCD)
  lcd.print ("Velocidad: 0% ");      // Se imprime "Velocidad: " en LCD
  delay (2000);                    //Retardo de 2s para tener mantener el motor detenido
}
```

```
void Direccion2 ()  // Subrutina que imprime en el LCD y en el motor el sentido de giro y velocidad del motor
{
  lcd.clear();                      // se limpia la pantalla
  lcd.setCursor(0,0);               // se coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
  lcd.print ("Giro Izquierda");     // Se imprime "Giro Izquierda" en LCD
  lcd.setCursor(0,1);               // se coloca el cursor en 0,1 (columna 1, fila 2 del LCD)
  lcd.print ("Velocidad: ");        // Se imprime "Velocidad: " en LCD

  i=0;                              // Se inicializa la velocidad del motor en 0

  while ((i < 256) && ((contador % 2) != 0))  // Mientras se cumpla que velocidad del motor sea menor que 256 (255 maxima)
  {                                           // y que el numero de contador no sea par se realiza lo siguiente :

    IR();                                 // Se llama a la subrutina IR () para enviar y recibir valores
```

```
    lcd.clear();                    // se limpia la pantalla

    lcd.setCursor(0,0);             // se coloca el cursor en 0,0 (columna 1, fila 1 del
LCD)

    lcd.print ("Giro Izquierda");   // Se imprime "Giro Izquierda" en LCD

    lcd.setCursor(0,1);             // se coloca el cursor en 0,1 (columna 1, fila 2 del
LCD)

    lcd.print ("Velocidad: ");      // Se imprime "Velocidad: " en LCD


    analogWrite (Control2, i);      // Se le da a contol 2 la velocidad dependiendo del
valor de i para que contole al motor

    digitalWrite (Control1, LOW);   // Se mantiene el valor de control 1 bajo y de esta
forma el sentido de giro del motor es hacia la Izquierda


    velocidad = map(i, 0, 255, 0, 100) ; // Cambia el rango de velocidad de (0 - 255) a
(0 - 100)

    lcd.print(velocidad);             // Se imprime en el LCD la velocidad del motor en
porcentaje

    lcd.print("%   ");              // Se imprime el signo de porcentaje

    delay (50);                     // Retardo de 50 ms


     i = i + 2;                     // Se incrementa la velocidad en 2
```

```
    if (i >= 255)                    // Condicion if para mantener la velocidad maxima
    {
     i = 255;
    }
  }
  delay (500);                           //Retardo de 2s para tener mantener la velocidad
maxima
  digitalWrite (Control2, LOW);      // Se asigna a Control2 un valor bajo y con esto el
motor se detiene
  lcd.setCursor(0,1);                // se coloca el cursor en 0,1 (columna 1, fila 2 del
LCD)
  lcd.print ("Velocidad: 0% ");      // Se imprime "Velocidad: " en LCD
  delay (2000);                      //Retardo de 2s para tener mantener el motor detenido
}
```

3.3. MOTORES DE PASO (STEPPER) Y SERVOMOTORES

En esta sección se revisará el uso de los motores de paso (Stepper) como también de servomotores controlados por la placa Arduino Mega 2560.

3.3.1. PREPARACIÓN PARA LA PRÁCTICA 3

Para poder realizar la práctica 3 es necesario tener conocimientos acerca del funcionamiento y conexión de los siguientes elementos:

- Servomotor
- Motor de paso
- ULN2003 (Arreglo Darlington de transistores)

3.3.1.1. Servomotor

Un servomotor es un aparato parecido a un motor DC, el cual tiene la capacidad de ubicarse en cualquier posición angular dentro de un rango dado y mantenerse en esa posición. El servomotor está constituido principalmente por un motor DC, un juego de engranajes, una tarjeta controladora y un potenciómetro como se muestra en la figura 3.10 [13].

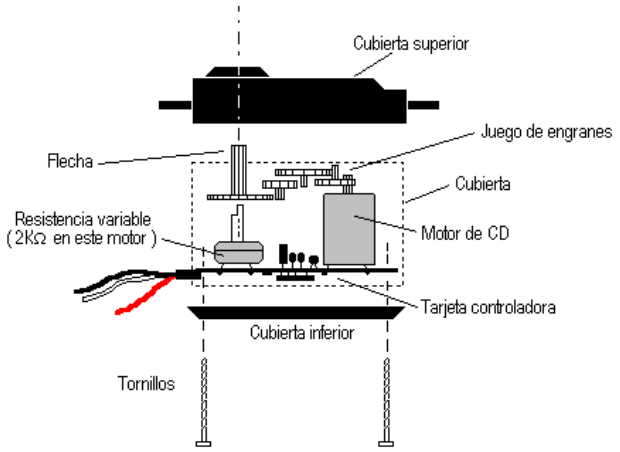


Figura 3.10 Partes de un Servomotor [http://www.info-ab.uclm.es/labelec/solar/electronica/elementos/servomotor.htm]

Los servomotores constan de tres cables, el rojo que es Vcc (5V), el negro que es tierra y el blanco o amarillo que es el cable de control.

Para poder realizar el control de un servomotor se usa la modulación de ancho de pulsos (PWM). A medida que el ancho de pulso aumenta la posición angular va aumentando como se puede apreciar en la figura 3.11

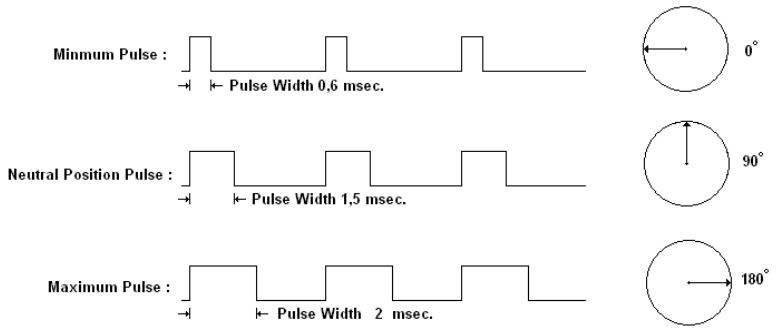


Figura 3.11 Control de posición (PWM) de un Servomotor
[http://www.ahmetozkurt.net/robotics2005/14/hitechs_dosyalar/image006.gif]

Por lo general los servos vienen con un rango de 0 a 180 grados de movimiento, aunque algunos pueden ser modificables para que giren 360 grados. Cada marca y modelo de servomotor es diferente, por lo tanto es recomendable revisar las especificaciones para poder hacer uso de este.

Las conexiones que se realizaron para controlar a un servomotor con el Arduino Mega se pueden apreciar en la tabla 3.4 y la figura 3.12

Tabla 3.4 Conexión entre Arduino y Servomotor

Arduino	Servomotor
5V	Positivo (Rojo)
GND	Negativo (Negro)
2 (Pin Digital PWM)	Control (Amarillo)

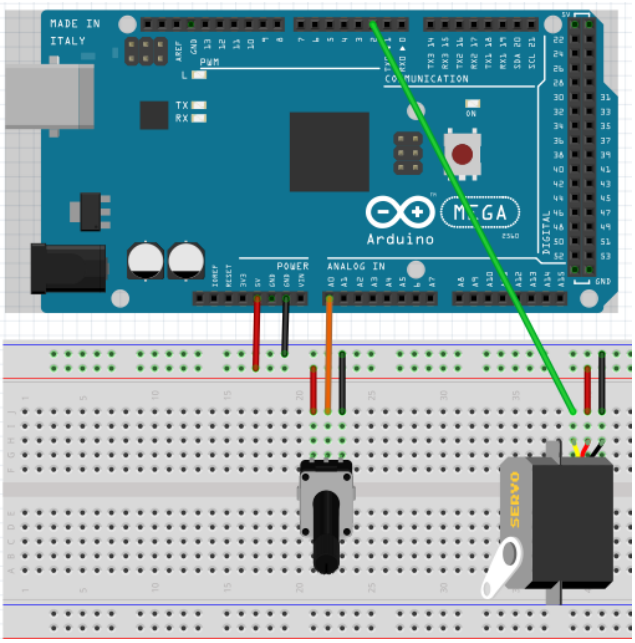


Figura 3.12 Conexión Arduino con Servomotor

3.3.1.2. Motor de Paso (Stepper)

Un motor de paso a paso, también conocido como PaP o Stepper es un dispositivo electromecánico que permite la

conversión de pulsos eléctricos en la rotación de pequeños movimientos angulares. Este tipo de motores son usados por su gran precisión y los pasos pueden variar desde 90° a tan solo 1.8° dependiendo del motor.

Existen dos tipos de motores de paso, el primero se llama unipolar y el segundo bipolar. Ambos tienen dos bobinas, con la diferencia de que el primero usa 5 o más cables para su control y el segundo usa tan solo 4. En la figura 3.13 se aprecia la diferencia en los terminales. El primero es un motor de pasos unipolar con 6 terminales y el segundo es un motor bipolar con 4 terminales [14].

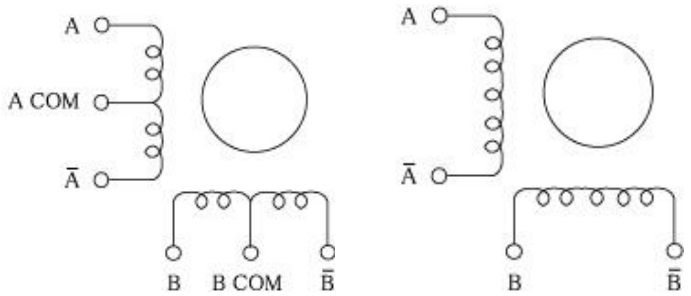


Figura 3.13 Motor unipolar de 6 terminales y motor bipolar [http://www.nmbtc.com/step-motors/engineering/winding-diagram-and-switching-sequence/]

La manera en que se controla cada motor cambia dependiendo del tipo y de tipo de torque y medida de paso que se quiera obtener. A medida de una simple

demostración en la tabla 3.5 y la tabla 3.6 se muestran secuencias para los diferentes tipos de motores de paso.

Tabla 3.5 Secuencia de motor de paso Unipolar

PASO	**A**	**B**	**¬A**	**¬B**	**A - COM B**
1	-	-	0	0	+
2	0	-	-	0	+
3	0	0	-	-	+
4	-	0	0	-	+

Tabla 3.6 Secuencia de motor de paso Bipolar

PASO	**A**	**B**	**¬A**	**¬B**
1	+	+	-	-
2	-	+	+	-
3	-	-	+	+
4	+	-	-	+

Para realizar esta práctica se hizo uso de un motor de pasos unipolar de 5 terminales. El modelo de este motor es el 28byj-48, el cual funciona con 5V y tiene 5.625° por paso y un sistema de engranaje de reducción de 1:64, lo cual hace que para dar una revolución completa se tenga que dar 4096 pasos. Este motor además tiene la particularidad que necesita una secuencia de 8 pasos para que funcione, la cual debe ser cumplida al pie de la letra. En la figura 3.14 se puede mostrar cómo están conectados y que color tienen los cables y en la tabla 3.7 se puede apreciar la secuencia a seguir para el motor de paso 28byj-48 [15].

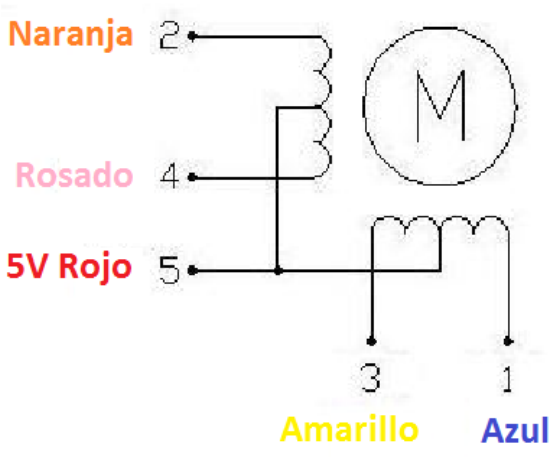


Figura 3.14 Conexiones con colores de motor de paso 28byj-48 [http://www.4tronix.co.uk/arduino/Stepper-Motors.php]

Tabla 3.7 Secuencia de motor de paso 28byj-48

COLOR DE CABLE	8 PASOS							
	1	2	3	4	5	6	7	8
NARANJA	-	-						-
AMARILLO		-	-	-				
ROSADO				-	-	-		
AZUL						-	-	-

3.3.1.3. Integrado ULN2003

El ULN2003 es un integrado que contiene un arreglo de transistores en configuración Darlington y es usado para manejar motores de paso. Este integrado funciona con 5V y soporta una corriente de 500ma en cada pata. En la figura 3.15 se muestran los pines de este integrado [16].

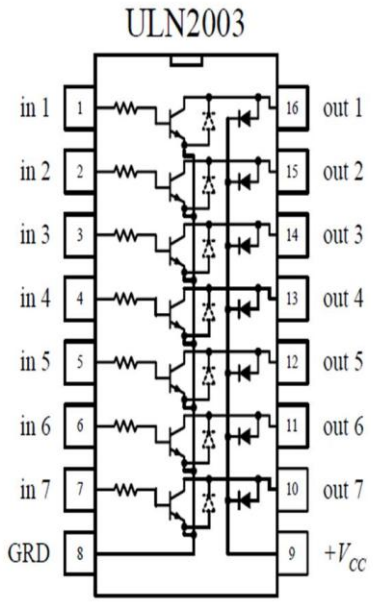


Figura 3.15 Pines del integrado ULN2003

[http://forum.clubedohardware.com.br/uploads/monthly_07_2014/post-650643-0-94029900-1405978848.jpg]

En la figura 3.16 se muestra como funciona circuito entre el ULN2003 y un motor de paso unipolar. Como se puede apreciar al poner la base del transistor Q1 en alto permite que este ultimo deje pasar corriente desde Vcc, pasando por las boninas, el transistor y luego a tierra completando así el circuito.

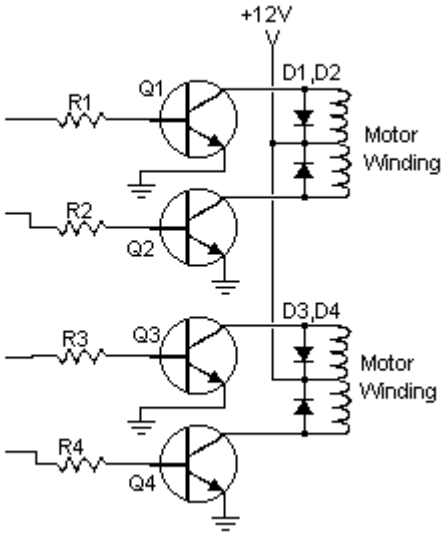


Figura 3.16 Circuito ULN2003 + Bobinas de motor de paso [http://www.societyofrobots.com/member_tutorials/node/28].

La figura 3.17 Muestra las conexiones hechas para entre el Arduino Mega, el integrado ULN2003 y el motor de paso 28byj-48.

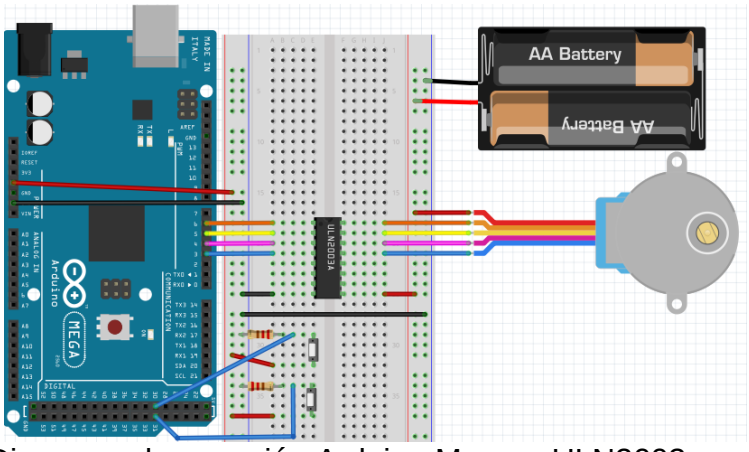


Figura 3.17 Diagrama de conexión Arduino Mega + ULN2003 + motor de paso 28byj-48.

En la Tabla 3.8 se muestran los pines de conexión usados entre el Arduino Mega, el integrado ULN2003 y el motor de paso 28byj-48.

Tabla 3.8 Arduino Mega + ULN2003 + motor de paso 28byj-48

Arduino	**ULN2003**	**28byj-48**	**Fuente externa 5V**
Pin6	Pin1	Naranja	
Pin5	Pin2	amarillo	
Pin4	Pin3	rosado	
Pin3	Pin4	Azul	
	Pin8	Rojo	5V
GND	GND		GND

3.3.2. PRÁCTICA 3: USO DE LOS MOTORES DE PASO (STEPPER) Y SERVOMOTORES

En esta práctica se hizo uso de la placa Arduino Mega, un servomotor Hitec HS-311, el integrado ULN2003 y un motor de paso 28byj-48 como los materiales principales. Como materiales secundarios se usaron un LCD 16x2 Botoneras y resistencias.

Esta práctica consta de 3 proyectos, los cuales son:

- CONTROL DE POSICIÓN DE UN SERVOMOTOR
- CONTROL DE POSICIÓN DE UN MOTOR STEPPER
- SECUENCIA CON UN SERVOMOTOR Y MOTOR STEPPER

3.3.2.1. OBJETIVOS

- Familiarizarse con la librería Servo.h
- Aprender a controlar la posición de giro de un servomotor con la placa Arduino Mega
- Aprender a controlar la posición de giro de un motor de paso con la placa Arduino Mega

3.3.2.2. CONTROL DE POSICIÓN DE UN SERVOMOTOR

Este proyecto se basa en el control de un servomotor a través de un potenciómetro y la placa Arduino Mega. A medida de que se gira el potenciómetro el ángulo del servomotor va aumentando, mientras que en pantalla se muestra cual es la posición angular con respecto a la posición inicial.

A continuación se muestra el código de este proyecto:

```
#include <LiquidCrystal.h>   // Libreria para usar el LCD

#include <Servo.h>           // Libreria especial para servo


LiquidCrystal lcd (36, 37, 38, 39, 40, 41); // Declaracion de pines para uso del LCD


Servo servo1;    // Se declara el nombre del servo


const int pot = A0;    // Se asigna pin al Potenciometro

const int servo = 7;   // Se asigna el pin de control del servo

const int pulsomin = 620; //minimo valor en us equivalente a 0°

const int pulsomax = 2400; //maximo valor en us equivalente a 180°


int valor;    //variable que recibe valores del potenciometro

int angulo;   //variable que obtiene valores de angulo


void setup()

{
```

```
  lcd.begin(16, 2);           //Inicializacion del LCD, se declara que el LCD usado es de
16 x 2

  servo1.attach (servo, pulsomin, pulsomax); //Se ingresan los valores que tiene el servo
a usar

}



void loop()

{

  valor = analogRead(pot); //Se obtienen valores del potenciometro

  angulo = map (valor, 0, 1023, 0, 185); //Se convierten valores a grados

  servo1.write (angulo);       // Se asigna al servo el numero de grados para que este se
mueva



  lcd.setCursor(0,0);         // se coloca el cursor en 0,0 (columna 1, fila 1 del LCD)

  lcd.print ("ANGULO : ");    // Se imprime el mensaje en LCD

  lcd.print(angulo);          // Se imprime el valor del angulo en LCD

  lcd.print("gr    ");        // Se imprime el mensaje en LCD

  delay (20);                 // retardo de 20 ms

}
```

3.3.2.3. CONTROL DE POSICIÓN DE UN MOTOR STEPPER

Este proyecto se basa en el control de un motor de paso por medio de dos botoneras y la placa Arduino Mega. Una botonera hace que el motor pueda girar en sentido horario y la otra botonera hace que el motor pueda girar en sentido anti horario mientras en un LCD se muestra la posición angular con respecto al punto de partida.

A continuación se muestra el código de este proyecto:

```
#include <LiquidCrystal.h>   // Libreria para usar el LCD

LiquidCrystal lcd (36, 37, 38, 39, 40, 41); // Declaracion de pines para uso del LCD


//declaracion de pines del motor stepper 28BYJ48

int motorPin1 = 3;    // Azul

int motorPin2 = 4;    // Rosado

int motorPin3 = 5;    // Amarillo

int motorPin4 = 6;    // Naranja


const int I = 30;     //declaracion de pin para boton izquierdo
```

```
const int D = 31;     //declaracion de pin para boton derecho


int grados;         //variable para imprimir en pantalla grados


int izquierda;      //Variable que guarda valores de I

int derecha;        //Variable que guarda valores de D


int velocidadStepper = 1200;  //variable asigna la velocidad del stepper

int conteo;          //cuenta los pasos dados

int ultimoval;       // se asigna los valores de conteo




//Variable que contiene la secuencia a seguir por el stepper

int secuencia[8] = {B01000, B01100, B00100, B00110, B00010, B00011, B00001, B01001};


////////////////////////////////////////////////////////////////////////////////

void setup()

{

  lcd.begin(16, 2);     //Inicializacion del LCD, se declara que el LCD usado es de 16 x
2
```

```
  pinMode(I, INPUT);  // declara el pin del boton izquierdo como entrada
  pinMode(D, INPUT);  // declara el pin del boton derecho como entrada

  // declara los pines de los motores como salida
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);

 // Serial.begin(9600);     //Inicializacion del puerto serial
}

////////////////////////////////////////////////////////////////////////////
void loop()
{
  izquierda = digitalRead (I); //lee la entrada I y lo asigna a izquierda
  derecha = digitalRead (D);   //lee la entrada D y lo asigna a derecha

  if (derecha == HIGH)  //si el boton derecha es presionado entonces
  {
```

```
    horario();          //se llama a sbrutina horario
    imprimir1();        //se llama a subrutina imprimir1
  }
  else if (izquierda == HIGH)  //si el boton izquierda es presionado entonces
  {
    antihorario();    //se llama a sbrutina antihorario
    imprimir1();      //se llama a subrutina imprimir1
  }
  else              //si ningun boton es presionado entonces
  {
    detener();      //se llama a sbrutina detener
    imprimir2();    //se llama a sbrutina imprimir2
    conteo = 0;     //se asigna 0 a conteo
  }
}

////////////////////////////////////////////////////////////////////////////

void antihorario() //Subrutina que hace que el motor gire en sentido antihorario
{
```

```
  for(int i = 0; i < 8; i++) //lazo para poder lanzar la secuencia en sentido antihorario
  {                          //va de 0-7 porque la secuencia es de 8 pasos
    conteo++;              //cuenta cada paso que se da
    ultimoval = conteo;    //obtiene el ultimo conteo de pasos
    setSalida(i);          //llama a la subrutina setSalida y asigna los valores de i
                           //para que esta asigne los valores correctos a los pines
    delayMicroseconds(velocidadStepper);  //retardo que permite controlar la velocidad
  }
}


void horario() //Subrutina que hace que el motor gire en sentido horario
{
  for(int i = 7; i >= 0; i--) //lazo para poder lanzar la secuencia en sentido horario
  {                           //va de 7-0 porque la secuencia es de 8 pasos
    conteo++;              //cuenta cada paso que se da
    ultimoval = conteo;    //obtiene el ultimo conteo de pasos
    setSalida(i);          //llama a la subrutina setSalida y asigna los valores de i
                           //para que esta asigne los valores correctos a los pines
    delayMicroseconds(velocidadStepper);  //retardo que permite controlar la velocidad
  }
```

```
}

void setSalida(int salida)  //subrutina que permite dar el valor correcto a las salidas
{                           //primero obtiene un valor binario del arreglo secuencia
                            //{B01000, B01100, B00100, B00110, B00010, B00011, B00001,
B01001}
                            //luego obtiene un bit y lo asigna a la salida
  digitalWrite(motorPin1, bitRead(secuencia[salida], 0));
  digitalWrite(motorPin2, bitRead(secuencia[salida], 1));
  digitalWrite(motorPin3, bitRead(secuencia[salida], 2));
  digitalWrite(motorPin4, bitRead(secuencia[salida], 3));
}

void detener()  //subrutina que para los pulsos despues de haber soltado un boton
{
  digitalWrite(motorPin1, LOW);  //asigna al min un nivel logico bajo
  digitalWrite(motorPin2, LOW);  //asigna al min un nivel logico bajo
  digitalWrite(motorPin3, LOW);  //asigna al min un nivel logico bajo
  digitalWrite(motorPin4, LOW);  //asigna al min un nivel logico bajo
}
```

```
void imprimir1() //subrutina imprime en el lcd los grados mientras un boton sea presionado
{
  grados = map (conteo, 0, 4096, 0, 360); //convierte un valor de 0-4096 pasos que tiene
                                          //una revolucion a un rango de 0-360 grados
  lcd.setCursor(0,0);        // se coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
  lcd.print ("ANGULO : ");   // Se imprime el mensaje en LCD
  lcd.print(grados);         // Se imprime el numero de grados que ha girado
  lcd.print("gr    ");       // Se imprime el mensaje en LCD
}

void imprimir2() ////subrutina imprime en el lcd los grados mientras el stepper esta detenido
{
  grados = map (ultimoval, 0, 4096, 0, 360); //convierte un valor de 0-4096 pasos que tiene
                                             //una revolucion a un rango de 0-360 grados
  lcd.setCursor(0,0);        // se coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
  lcd.print ("ANGULO : ");   // Se imprime el mensaje en LCD
  lcd.print(grados);         // Se imprime el numero de grados que ha girado
  lcd.print("gr    ");       // Se imprime el mensaje en LCD
}
```

3.3.2.4. SECUENCIA CON UN SERVOMOTOR Y MOTOR STEPPER

Este proyecto se basa en una secuencia programada usando un servomotor y un motor de paso. Se presiona un botón para comenzar la secuencia, luego el motor de paso gira 180°, después de esto el servomotor gira 180° y regresa a su posición inicial y por último el motor de paso también regresa a su posición inicial para esperar de nuevo un pulso en la botonera. En todo este proceso se muestra siempre en pantalla que motor está actuando.

A continuación se muestra el código de este proyecto:

```
#include <LiquidCrystal.h>   // Libreria para usar el LCD

#include <Servo.h>           // Libreria especial para servo


LiquidCrystal lcd (36, 37, 38, 39, 40, 41); // Declaracion de pines para uso del LCD


//////////////////////////////////////////////////////////////////////
```

```
Servo servo1;    // Se declara el nombre del servo


const int servo = 7;   // Se asigna el pin de control del servo

const int pulsomin = 620; //minimo valor en us equivalente a 0°

const int pulsomax = 2400; //maximo valor en us equivalente a 180°


int valor;    //variable que recibe valores del potenciometro


////////////////////////////////////////////////////////////////////////////

//declaracion de pines del motor stepper 28BYJ48

int motorPin1 = 3;    // Azul

int motorPin2 = 4;    // Rosado

int motorPin3 = 5;    // Amarillo

int motorPin4 = 6;    // Naranja

const int I = 30;     //declaracion de pin para boton izquierdo

const int D = 31;     //declaracion de pin para boton derecho


int boton;        //Variable que guarda valores de D


int velocidadStepper = 1200;  //variable asigna la velocidad del stepper
```

```
int conteo = 0;          //cuenta los pasos dados


//Variable que contiene la secuencia a seguir por el stepper
int secuencia[8] = {B01000, B01100, B00100, B00110, B00010, B00011, B00001, B01001};


////////////////////////////////////////////////////////////////////////////
void setup()
{
  lcd.begin(16, 2);     //Inicializacion del LCD, se declara que el LCD usado es de 16 x
2
  servo1.attach (servo, pulsomin, pulsomax); //Se ingresan los valores que tiene el servo
a usar


  pinMode(I, INPUT);  // declara el pin del boton izquierdo como entrada
  pinMode(D, INPUT);  // declara el pin del boton derecho como entrada


  // declara los pines de los motores como salida
  pinMode(motorPin1, OUTPUT);
  pinMode(motorPin2, OUTPUT);
  pinMode(motorPin3, OUTPUT);
  pinMode(motorPin4, OUTPUT);
```

```
  servo1.write (0);      // Se asigna al servo el numero de grados para que este se mueva


}


////////////////////////////////////////////////////////////////////////////
void loop()
{
  boton = digitalRead (D);   //lee la entrada D y lo asigna a derecha


  if (boton == HIGH)  //si el boton derecha es presionado entonces
  {
    imprimir1();
    while (conteo <= 2048)
    {
      horario();          //se llama a sbrutina horario
    }


    conteo = 0;
```

```
    imprimir2();
    servo1.write (90);       // Se asigna al servo el numero de grados para que este se
mueva

    delay (1000);

    servo1.write (0);       // Se asigna al servo el numero de grados para que este se
mueva

    delay (1000);
    imprimir1();
    while (conteo <= 2048)
    {
      antihorario();          //se llama a sbrutina horario
    }

    conteo = 0;

  }

  else              //si ningun boton es presionado entonces
```

```
  {
    detener();      //se llama a sbrutina detener
    lcd.clear ();
  }
}

////////////////////////////////////////////////////////////////////////////

void antihorario() //Subrutina que hace que el motor gire en sentido antihorario
{
  for(int i = 0; i < 8; i++) //lazo para poder lanzar la secuencia en sentido antihorario
  {                          //va de 0-7 porque la secuencia es de 8 pasos
    conteo++;              //cuenta cada paso que se da

    setSalida(i);          //llama a la subrutina setSalida y asigna los valores de i
                           //para que esta asigne los valores correctos a los pines
    delayMicroseconds(velocidadStepper);  //retardo que permite controlar la velocidad
  }
}
```

```
void horario() //Subrutina que hace que el motor gire en sentido horario
{
  for(int i = 7; i >= 0; i--) //lazo para poder lanzar la secuencia en sentido horario
  {                           //va de 7-0 porque la secuencia es de 8 pasos
    conteo++;             //cuenta cada paso que se da

    setSalida(i);         //llama a la subrutina setSalida y asigna los valores de i
                          //para que esta asigne los valores correctos a los pines
    delayMicroseconds(velocidadStepper);  //retardo que permite controlar la velocidad
  }
}

void setSalida(int salida)  //subrutina que permite dar el valor correcto a las salidas
{                           //primero obtiene un valor binario del arreglo secuencia
                            //{B01000, B01100, B00100, B00110, B00010, B00011, B00001,
B01001}
                            //luego obtiene un bit y lo asigna a la salida
  digitalWrite(motorPin1, bitRead(secuencia[salida], 0));
  digitalWrite(motorPin2, bitRead(secuencia[salida], 1));
  digitalWrite(motorPin3, bitRead(secuencia[salida], 2));
  digitalWrite(motorPin4, bitRead(secuencia[salida], 3));
```

```
}

void detener()  //subrutina que para los pulsos despues de haber soltado un boton
{
  digitalWrite(motorPin1, LOW);  //asigna al min un nivel logico bajo
  digitalWrite(motorPin2, LOW);  //asigna al min un nivel logico bajo
  digitalWrite(motorPin3, LOW);  //asigna al min un nivel logico bajo
  digitalWrite(motorPin4, LOW);  //asigna al min un nivel logico bajo
}

void imprimir1() //subrutina imprime en el lcd los grados mientras un boton sea presionado
{
  lcd.setCursor(0,0);        // se coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
  lcd.print ("MOTOR STEPPER");   // Se imprime el mensaje en LCD
  lcd.setCursor(0,1);        // se coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
  lcd.print ("GIRANDO");   // Se imprime el mensaje en LCD
}

void imprimir2() ////subrutina imprime en el lcd los grados mientras el stepper esta detenido
```

```
{

  lcd.setCursor(0,0);        // se coloca el cursor en 0,0 (columna 1, fila 1 del LCD)

  lcd.print ("SERVOMOTOR   ");   // Se imprime el mensaje en LCD

  lcd.setCursor(0,1);        // se coloca el cursor en 0,0 (columna 1, fila 1 del LCD)

  lcd.print ("GIRANDO");   // Se imprime el mensaje en LCD

}
```

3.4. COMUNICACIÓN I2C

En esta sección se revisaran los conceptos básicos de la comunicación I2C y se realizara la práctica correspondiente con un LCD y un sensor de temperatura, ambos con I2C.

3.4.1. PREPARACIÓN PARA LA PRÁCTICA 4

Para poder realizar la práctica 4 es necesario primero saber de qué se trata la comunicación I2C, es por esto que se dará una breve descripción de que es y cómo funciona el protocolo de comunicación I2C.

I2C (Inter-Integrated Circuit) es uno de los protocolos más utilizados para comunicación serial. Fue creado por Phillips en los

80´s. Este protocolo usa solamente 2 cables para comunicarse entre dos o más circuitos integrados, de los cuales uno es el master y los demás son esclavos [17]. Estos dos cables de comunicación son bidireccionales, el primero se llama SDA (Serial Data) el cual se encarga de enviar y recibir datos y el segundo se llama SCL (Serial Clock) que es el reloj. El SDA Y SCL siempre van conectados con una resistencia de Pull-up a Vcc cada uno para poder realizar la comunicación. Estas resistencias por lo general solo van en master y tienen un valor que es por lo general de 10K. En la figura 3.18 se puede apreciar como es la conexión entre dispositivos con el protocolo I2C [18].

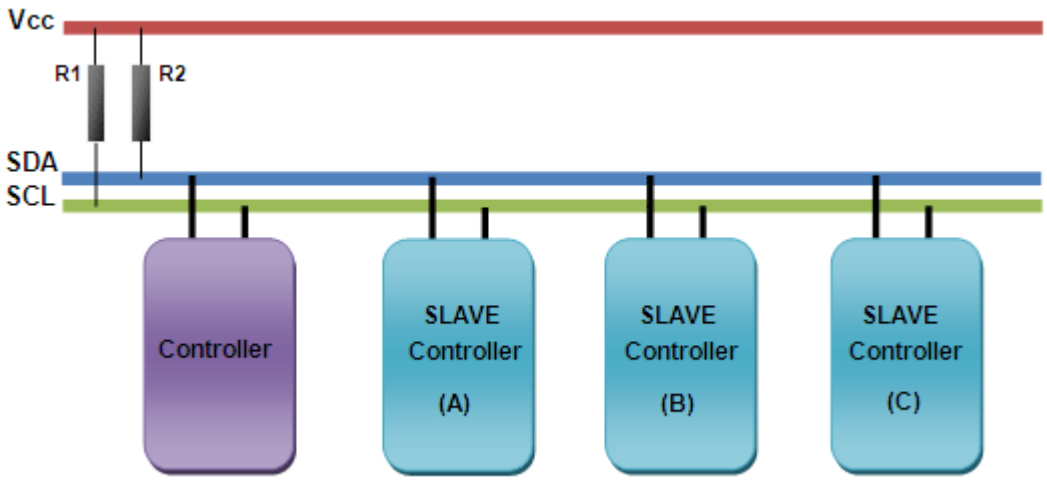


Figura 3.18 Conexión entre master y esclavos con protocolo I2C

[http://www.engineersgarage.com/tutorials/twi-i2c-interface]

I2C tiene un protocolo de maestro/esclavo donde el maestro es el que controla la comunicación entre los dispositivos. A continuación se muestra una descripción simple de lo que es el protocolo.

1. El maestro impone una condición de inicio. Esta condición alerta a todos los esclavos que escuchen cuando sean llamados con su dirección por medio del cable SDA.

2. El maestro envía la dirección del esclavo con el que se desea comunicar y además envía una bandera de leer/escribir.

3. El esclavo que fue llamado responde con una señal de reconocimiento.

4. La comunicación procede entre el maestro y el esclavo por medio del cable SDA. Tanto como el maestro y el esclavo pueden enviar y recibir datos dependiendo de que si la comunicación haya sido de lectura o escritura. El transmisor envía 8 bits de datos al receptor, el cual responde con un bit de reconocimiento.

5. Cuando la comunicación esta completa el maestro impone una condición de paro indicando así que la comunicación a finalizado.

La figura 3.19 muestra gráficamente como se realiza la comunicación I2C.

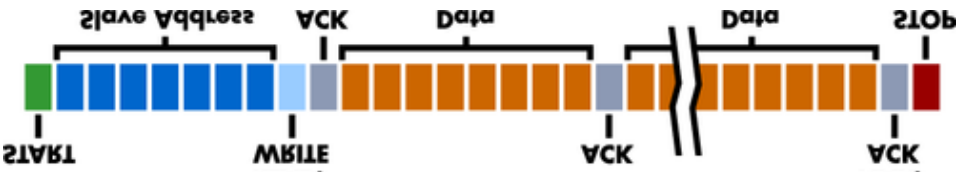


Figura 3.19 Representación grafica de comunicación I2C [http://www.totalphase.com/support/articles/200468316-Aardvark-Adapter-User-Manual]

Así es como básicamente funciona el protocolo I2C. Como se pudo apreciar saber la dirección de los esclavos que se van a usar es muy importante. A continuación se mostrará como hallar la dirección del sensor de Temperatura TMP102. La figura 3.20 Muestra como es el TMP102

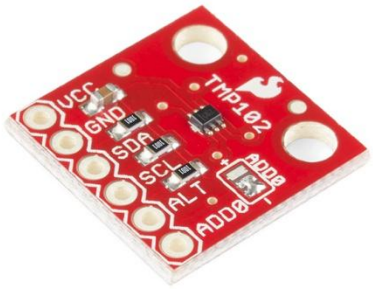


Figura 3.20 TMP102

El integrado TMP102 es un sensor de temperatura muy pequeño y que funciona con comunicación I2C. Por lo tanto para poder trabajar con él es necesario conocer su dirección. En la tabla 3.9 se muestra una parte del data sheet del TMP102 donde se muestra cual es la dirección. Como se puede apreciar hay 4 diferentes direcciones y esto es porque dejan un pin que está marcado como ADD0 libre para poder elegir qué dirección queremos. En nuestro caso conectaremos el pin ADD0 a tierra, por lo tanto se obtendrá la dirección 1001000 que en hexadecimal es 0x48 que es el formato con el que se va a trabajar.

Tabla 3.9 Direcciones del TMP102

DEVICE TWO-WIRE ADDRESS	A0 PIN CONNECTION
1001000	Ground
1001001	V+
1001010	SDA
1001011	SCL

Otro dato importante a saber es en que formato el TMP102 envía los datos y la respuesta a esto es que para poder leer los datos de temperatura tienen que tener un formato de 12 bits, pero el protocolo I2C solo permite mandar datos en 8 bits, por lo que es necesario preguntar por 2 datos de 8 bits cada uno para después combinarlos y sacar un dato de 12 bits que sería la temperatura real. En la Tabla 3.10 se puede apreciar unos ejemplos de temperatura del datasheet.

Tabla 3.10 Datos de temperatura con formato de 12 bits del TMP102

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	HEX
128	0111 1111 1111	7FF
127.9375	0111 1111 1111	7FF
100	0110 0100 0000	640
80	0101 0000 0000	500
75	0100 1011 0000	4B0
50	0011 0010 0000	320
25	0001 1001 0000	190
0.25	0000 0000 0100	004
0	0000 0000 0000	000
–0.25	1111 1111 1100	FFC
–25	1110 0111 0000	E70
–55	1100 1001 0000	C90

Para el caso del LCD 20x4 con I2C la dirección esta fija ya que este viene soldado y no hay como cambiarla. La dirección que se uso para este dispositivo es 0x20. La marca del dispositivo que se uso con el LCD es mjkdz (chino), esto es necesario saberlo ya que cada marca tiene diferente dispositivo I2C, por lo tanto cambian las direcciones.

A continuación se muestra en la figura 3.25 las conexiones hechas entre el Arduino Mega, el sensor de temperatura TMP102 y el LCD con I2C.

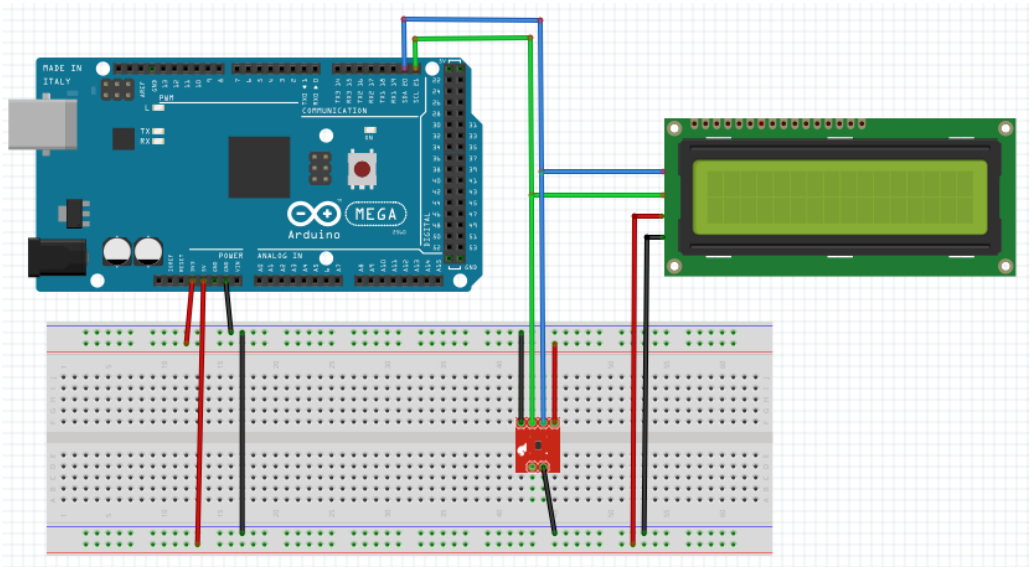


Figura 3.21 Conexión Arduino mega + TMP103 + LCD con I2c

A continuación se muestra en la tabla 3.9 los pines usados en la conexión entre el Arduino Mega, el sensor de temperatura TMP102 y el LCD con I2C.

Tabla 3.11 Tabla de pines Practica con I2C

Arduino	**TMP102**	**LCD I2C**
5V		VCC
3.3V	Vcc	
GND	GND	GND, ADD0
SDA	SDA	SDA
SCL	SCL	SCL

3.4.2. PRÁCTICA 4: IMPLEMENTACIÓN DE LA COMUNICACIÓN I2C

En esta práctica se hizo uso de la placa Arduino Mega, un sensor de temperatura TMP102, y un LCD de 20x4 con comunicación I2C (mjkdz) como los materiales principales de la práctica.

3.4.2.1. OBJETIVOS

- Familiarizarse con la librería Wire.h
- Aprender a utilizar dispositivos I2C
- Familiarizarse con la librería LiquidCrystal_I2C.h

3.4.2.2. SENSOR DE TEMPERATURA CON COMUNICACIÓN I2C

Esta práctica consiste en realizar lecturas de temperatura por medio del sensor de temperatura TMP102 y mostrar en pantalla (LCD 20x4) cual es la temperatura ambiente en grados Celsius y en grados Fahrenheit. Cabe recalcar que tanto el TMP102 y el LCD 20x4 usan

comunicación I2C y las direcciones de estos son 0x48 y 0x20 respectivamente.

A continuación se muestra el código de este proyecto.

```
#include <LiquidCrystal_I2C.h> // Libreria necesaria para LCD I2C
#include <Wire.h> //libreria necesari para I2C

//Se declaran los datos necesarios para que el lcd funcione.
//cada marca tiene una secuencia diferente
                    //dir, en,rw,rs,d4,d5,d6,d7,bl,blpol
LiquidCrystal_I2C lcd(0x20, 4, 5, 6, 0, 1, 2, 3, 7, NEGATIVE);

int dirTMP102= 0x48; //direccion en hexadeximal del TMP102

void setup()
{
 Serial.begin(9600); //Se inicializa la el puerto seial
 Wire.begin(); // Se inicializa la libreria wire
```

```
 lcd.begin(20,4); //Se inicializa el display de 20x4


 lcd.backlight(); //Se enciende la luz de fondo
 lcd.setCursor(0,0); // coloca el cursor en 0,0 (columna 1, fila 1 del LCD)
 lcd.print("TEMPERATURA AMBIENTE");  //Imprime en pantalla el mensaje


}


void loop()
{
 float celsius = tomarTemperatura(); //Toma la temperatura y la asigna a la variable Celsius
 lcd.setCursor(0,2);  // coloca el cursor en 0,2 (columna 1, fila 3 del LCD)
 lcd.print("Celsius: ");  //Imprime en pantalla el mensaje
 lcd.print(celsius);  //Imprime la temperatura en celsius


 float fahrenheit = (1.8 * celsius) + 32;   //Convierte la temperatura de celsius a Fahrenheit
 lcd.setCursor(0,3)  // coloca el cursor en 0,3 (columna 1, fila 4 del LCD)
 lcd.print("Fahrenheit "); //Imprime en pantalla el mensaje
 lcd.print(fahrenheit); //Imprime la temperatura en celsius
```

```
 delay(500); //Retardo de 500 ms
}

float tomarTemperatura() //Subrutine que obtiene la temperatura del TMP102
{
 Wire.requestFrom(dirTMP102,2); //Se pide al TMP102 2 bytes de temperatura

//Es neserario leer lo que se pidio
 byte MSB = Wire.read(); //Se obtiene el Byte mas significativo MSB
 byte LSB = Wire.read(); //El segundo es el Byte menos significativo LSB

 //Operaciones con 2 bytes para dejar la temperatura expresada en 12 bits
 //y luego es asignado a una variable tipo entero
 int sumaTemp = ((MSB << 8) | LSB) >> 4;

 float celsius = sumaTemp*0.0625; // convierte la temperatura en celsius
 return celsius; // Retorna la temperatura en celsius
}
```

3.5. *COMUNICACIÓN XBEE*

En esta sección se revisaran los conceptos básicos de la comunicación XBee y se realizara la práctica correspondiente con un Arduino Mega, 2 módulos Xbee (Serie 2), un XBee shield y un XBee explorer.

3.5.1. PREPARACIÓN PARA LA PRÁCTICA 5

Para poder realizar la práctica 5 es necesario primero saber de qué se trata el tema XBee, es por esto que a continuación se dará una breve descripción de que es y cómo funcionan los dispositivos XBee.

XBee es un modulo de comunicación inalámbrica, creado por la compañía Digi. Estos módulos trabajan bajo el protocolo de comunicación IEEE 802.15.4 conocido como Zigbee [19].

Existen 3 niveles en una red Zigbee, primero tenemos al coordinador, el cual es el encargado de configurar la red (se necesita uno por red), este dispositivo nunca puede dormir, ya que sin él la red no funcionaria. Como segundo nivel tenemos a los routers los cuales tienen la función de retransmitir señales ya sea entre routers o a los puntos finales y nunca pueden dormir. Por

último tenemos a los Puntos finales (End Points), los cuales no retransmiten señales, solo cumplen la función que se les ha dado (pueden dormir para ahorro de energía). En la figura 3.22 se muestra una representación de los 3 niveles que existen en las redes zigbee [20].

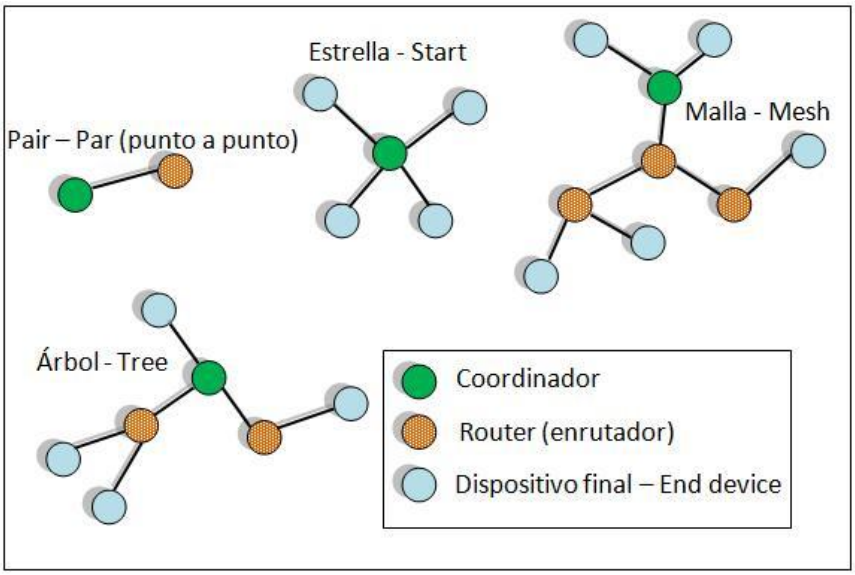


Figura 3.22 Niveles de jerarquía en comunicación Zigbee

Existen dos modos con los cuales un XBee puede ser configurado. El primero es el modo AT más conocido como modo transparente en el cual la comunicación se realiza a través de los XBee, es decir los mismos datos que entran por el puerto serial del dispositivo A son recibidos por el puerto serial del dispositivo B. El segundo se

llama modo API (Application Programming Interface) el cual es más complicado ya que la comunicación se realiza mediante tramas, pero a su vez tiene mayores ventajas ya que permite decidir con que dispositivo comunicarse, esto ayuda cuando en la red hay más de un dispositivo (múltiples direcciones) [21].

Para poder realizar la práctica es necesario saber que cuando se quiere establecer una comunicación a través de los módulos XBee (Transparente) ambos módulos deben ser configurados en modo AT y cuando se quiere realizar una comunicación al XBee (Modo comando) el que envía los datos debe estar en modo AT y el que recibe debe estar en modo API.

A continuación en la tabla 3.10 se muestra la trama que hay que tener en cuenta cuando queremos recibir valores analógicos o digitales de un router o un punto final con configuración AT en un coordinador con configuración API.

Tabla 3.12 Trama receptor API y Transmisor AT

Byte	Ejemplo	Descripción
0	0x7E	Byte de inicio- Indica el comienzo de la trama
1	0x00	Longitud - Numero de Bytes (ChecksumByte# - 1- 2)
2	0x14	
3	0x92	Tipo de trama - 0x92 indica que habrá una muestra de datos
4	0x00	Dirección de la fuente en 64 bits (Numero Serial), MSB es el Byte 4, LSB es el Byte 11
5	0x13	
6	0xA2	
7	0x00	
8	0x40	
9	0x77	
10	0x9C	
11	0x49	
12	0x36	Dirección de red de la fuente - 16 Bits
13	0x6A	
14	0x01	Opciones de recepción. 01=Paquete Reconocido, 02=Paquete de difusión
15	0x01	Numero de muestras. Poner siempre1 por limitaciones de XBee
16	0x00	Mascara digital del canal - Indica cueles pines fueron fijados a DI/O
17	0x20	

18	0x01	Mascara analógica del canal - Indica cuales pines fueron fijados a ADC
19 20	0x0 0x14	Muestras de datos digitales. Si hay alguno lee lo mismo que la máscara digital
21 22	0x04 0x25	Muestras de datos analógicos. Si hay algún dato cada pin fijado como ADC tendrá 2 Bytes
23	0xF5	Checksum (la suma de 8 bits desde el byte 3 hasta este)

Es importante recalcar que en los bytes que pertenecen a la máscara digital del canal (byte 16 y byte 17) se puede ver que pines son los que están habilitados como entradas y salida digitales, estando el primer Byte configurado de la siguiente forma (n/a n/a n/a D12 D11 D10 n/a n/a) y el segundo byte (D7 D6 D5 D4 D3 D2 D1 D0) esto quiere decir que si por ejemplo tenemos 0x00 0X13 en las lecturas de estos dos bites tendremos 0x00 0X13 = 0000 0000 0000 1101, lo que significa que Los pines D1, D2 y D1 están habilitados. De la misma forma se analiza los bytes 19 y 20, en los cuales si hay alguna lectura debería proporcionar la misma información que los bytes 16, 17.

De la misma forma sucede con la máscara analógica (Byte 18), la cual está configurada de la siguiente forma (volt n/a n/a n/a A3 A2

A1 A0) y si por ejemplo tenemos 0x05 = 0000 0101 significa que A2 y A0 están habilitados y tendríamos 2 bytes de muestras por cada pin que este habilitado para ADC.

Ahora para poder enviar comandos desde un coordinador API hacia un Router o punto final en modo AT es necesario saber cómo enviar una trama correctamente. En la tabla 3.11 se muestra la trama para enviar órdenes desde un coordinador API hacia router AT [22].

Tabla 3.13 Trama Transmisor API y Receptor AT

Byte	Ejemplo	Descripción
0	0x7E	Byte de inicio- Indica el comienzo de la trama
1 2	0x00 0x10	Longitud - Numero de Bytes (ChecksumByte# - 1- 2)
3	0x17	Tipo de trama - 0x17 indica que es un requerimiento de comando AT
4	0x52	ID de trama - numero de secuencia de comando
5 6 7 8	0x00 0x13 0xA2 0x00	Dirección de la fuente en 64 bits (Numero Serial), MSB es el Byte 5, LSB es el Byte 12 0x0000000000000000 = Coordinador 0x000000000000FFFF = Difusión

9	0x40	
10	0x77	
11	0x9C	
12	0x49	
13	0xFF	Dirección de red de destino
14	0xFE	(0xFFFE para difusión)
15	0x02	Opción de comando remoto (0x02 para aplicar cambios)
16	0x44 (D)	Nombre de comando AT (Dos caracteres ASCII)
17	0x02 (2)	
18	0x04	Parámetro de comando (Si es alto o bajo)
19	0xF5	Checksum

Para poder configurara los módulos XBee es necesario descargar el software X-CTU, este software se lo puede encontrar en la página www.digi.com. A continuación se muestran los pasos para configurar los módulos XBee :

1. Instalar el programa X-CTU
2. Colocar un xbee en el xbee explorer y conectar a la computadora por medio de cable USB a USB mini

3. Ejecutar el programa X-CTU. Se debe mostrar la conexión USB con los puertos disponibles (figura 3.23)

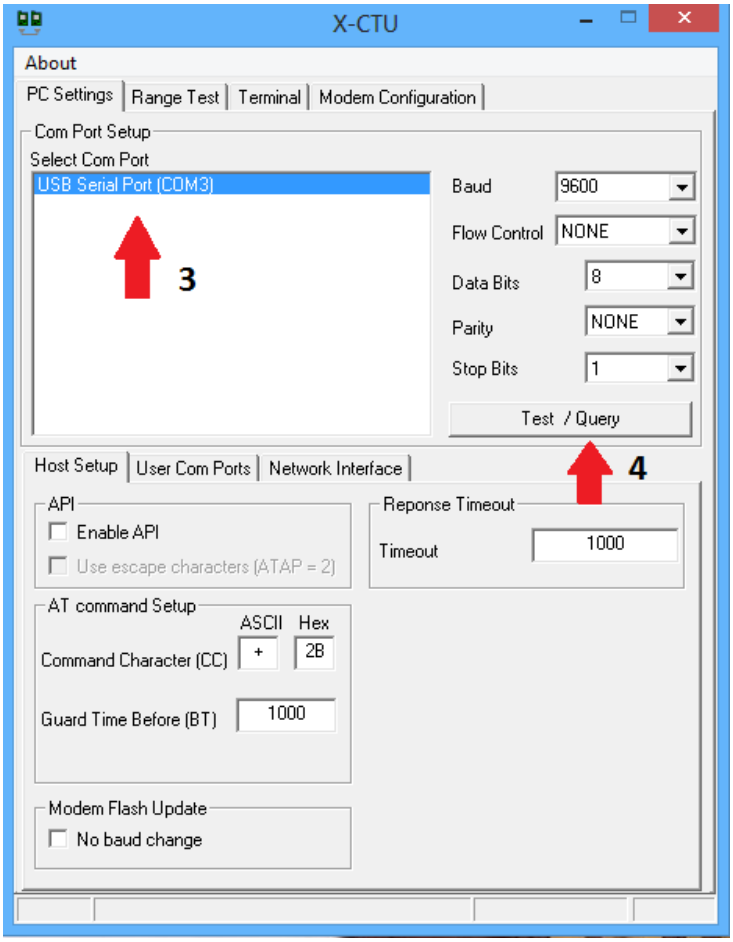


Figura 3.23

4. Seleccionar el puerto COM apropiado y seleccionar “Test / Query” (figura 3.23). Si la respuesta es OK (Figura 3.24), significa que la conexión está bien, de otra manera hay que

asegurarse de que se seleccionó el puerto COM apropiado o que no esté mal conectado el Xbee en el Xbee Explorer.

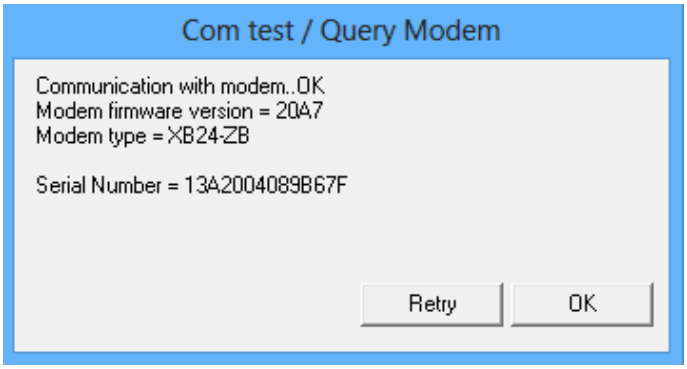


Figura 3.24

5. Seleccionar la pestaña “Modem Configuration”(figura 3.25)
6. Seleccionar la opción “Read” (figura 3.25)

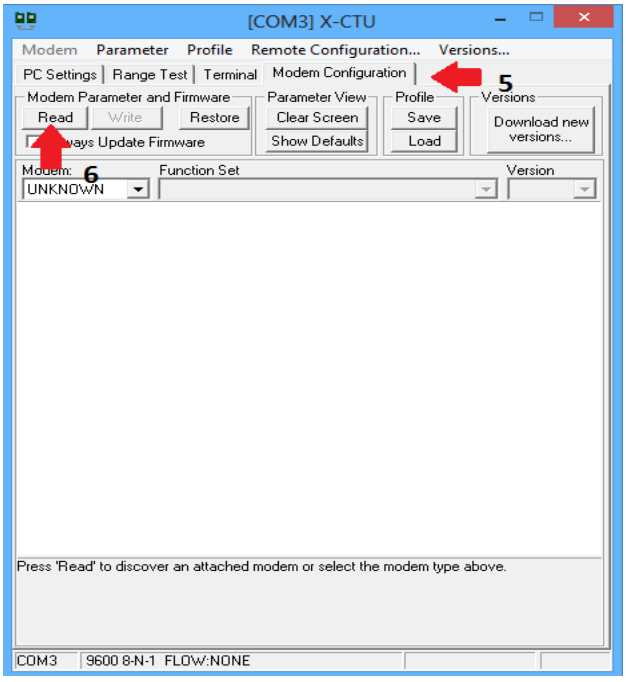


Figura 3.25

7. En la opción “Function Set”, seleccionar el modo en que se quiere que funcione el XBee, ya sea coordinador, router , punto final y en modo AT o API. En este caso se selecciono ZIGBEE COORDINADOR AT (figura 3.26)
8. En la carpeta “Networking” seleccionar ID – PAN ID y establecer un número entre los valores permitidos. El PAN ID es un número de identificación el cual debe ser el mismo para todos los Xbee de la red (figura 3.26)
9. En la carpeta “Addressing” En la opción “DH – Destination Address High” establecer el valor SH del otro Xbee. Regularmente su valor es 0013A200 (figura 3.26)
10. En la opción “DL – Destination Address Low” establecer el valor SL del otro Xbee (figura 3.26).Esto se hace para obtener una comunicación punto a punto, de lo contrario se puede poner DH (0) y DL (FFFF) para hacer una difusión a todos los xbee de la red, o DH (0) y DL (0) Para comunicarse solo con el coordinador.

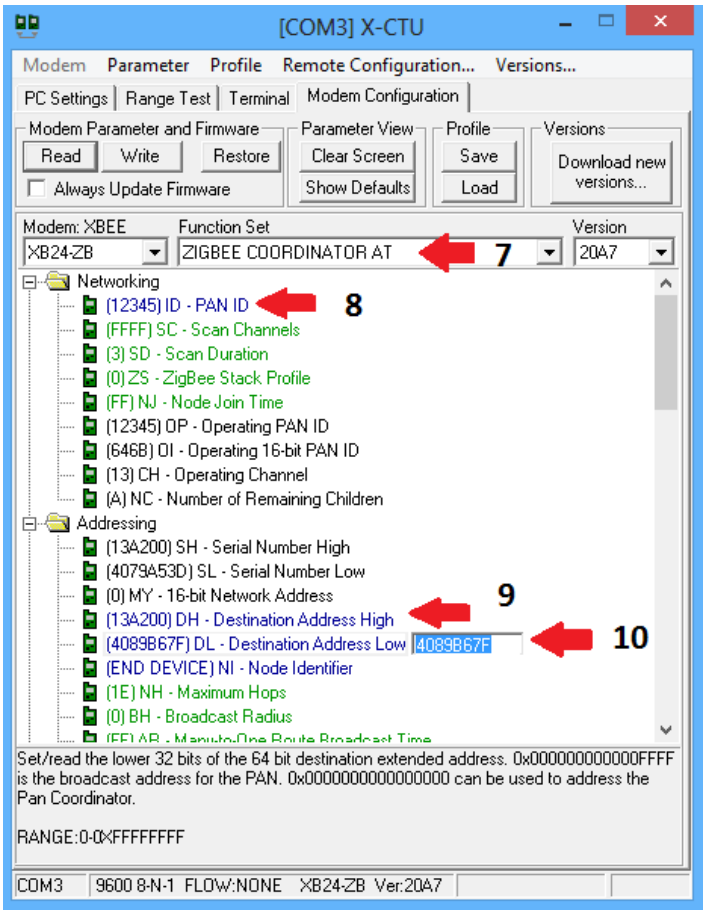


Figura 3.26

11. Seleccionar el botón “Write” y esperar a que se termine la configuración.
12. Cerrar el X-CTU, desconectar el XBee explorer y desconectar el XBee que se acaba de configurar [23].

En este proyecto se trata de obtener señales digitales de una botonera que está conectada a un router en modo AT y enviar esas señales a un coordinador que está en modo API y con esos datos encender un led (Figura 3.27). En donde el pin 30 del Arduino está conectado a un led y el pin 11 del del XBee router está conectado a una botonera. Es importante tener en cuenta que la alimentación del XBee es de 3.3V.

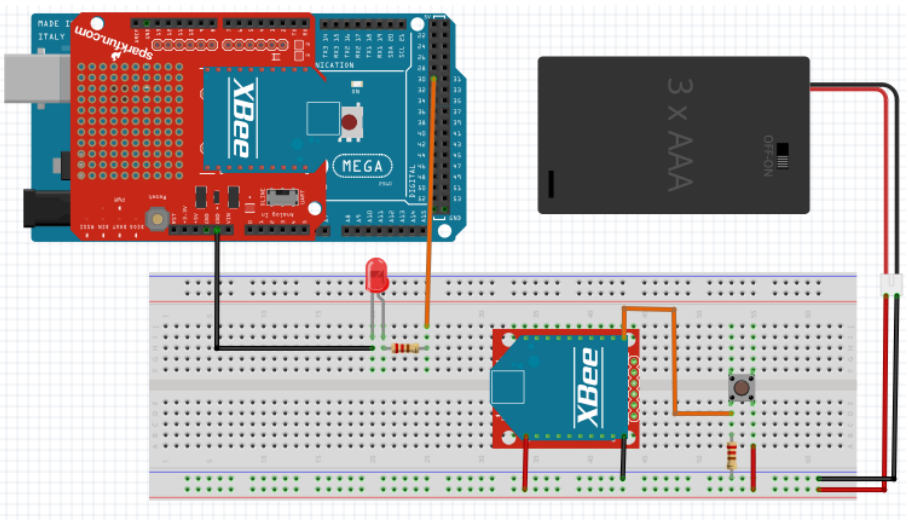


Figura 3.27 Proyecto XBee2

3.5.2. PRÁCTICA 5: USO DE LA COMUNICACIÓN XBEE

En esta práctica se hizo uso de la placa Arduino Mega, 2 XBee serie 2, un XBee explorer y un shield XBee como materiales principales de la práctica.

3.5.2.1. OBJETIVOS

- Aprender como programar un XBee
- Aprender a usar XBee con Arduino
- Aprender que es modo AT y como usarlo
- Aprender que es modo API y como usarlo

3.5.2.2. IMPLEMENTACION PRACTICA DE LA COMUNICACIÓN XBEE

En este proyecto se uso un XBee configurado como coordinador en modo API y otro XBee configurado como router en modo AT. Para este último además de configurar el PAN ID también se configuro JV = 1, el I/O 4 como entrada digital y con un tiempo de muestreo de 1s.

Este proyecto consiste en detectar una señal digital la cual es proporcionada por medio de un botón en el lado del router y enviar la información al coordinador donde por medio del Arduino se enciende un led y a la vez se muestra por el puerto serial el estado del botón.

A continuación se muestra el código de este proyecto:

```
int leervalor = 0; //Variable entera para leer valor

int const led = 30; //Constante del pin del led

void setup()

{

  pinMode (led, OUTPUT); //Se declara a led como salida

  Serial.begin(9600); // Se inicializa el puerto serial

}



void loop()

{

  if(Serial.available() > 21)//Asegura que haya 21 Bytes listos antes de leer

  {

    if(Serial.read() == 0x7E)//Asegura que empiece con el bit de inicio

    {
```

```
      for (int i = 0 ; i<19 ; i++)//Los siguientes 19 Bytes se descartan
      {
        byte descarte = Serial.read();//se descarta
      }
      leervalor = Serial.read(); // se lee el Byte que contiene la informacion importante
      Serial.print("El boton esta: "); // se imprime el mensaje

      if(leervalor == 0) //si no lee nada es que el boton no esta presionado
      {
        Serial.println("sin presionar"); //Se imprime el mensaje
        digitalWrite(led,LOW); //Se apaga el led
      }
      else if(leervalor == 16)//Si se lee 16 o en binario 10000 (corresponde a la entrada
4 del router)
                              // significa que el boton esta presionado
      {
        Serial.println("presionado");  //Se imprime el mensaje
        digitalWrite(led,HIGH);   //Se enciende el led
      }
    }
  }
}
```

3.6. Propuesta de futuros proyectos de fin de curso

La siguiente es una lista de los proyectos que se podrían realizar como proyectos de fin de curso. Algunos de estos proyectos se podrían realizar en grupo debido a su complejidad. A continuación la lista:

- **Cubo Led 3x3x3:** Este proyecto es de tipo decorativo y consiste en armar un cubo de leds de dimensión 3x3x3 con el que se comienzan a desplegar cierto tipo de patrones luminosos dentro del cubo.

- **Control de circuitos usando Arduino y control remoto:** Este proyecto consiste en usar un control remoto para controlar el encendido y apagado de una lámpara con Arduino.

- **Simón dice:** Es un juego en que hay que seguir patrones. Varios leds muestran cuales son los patrones que hay que seguir y el jugador tiene que poner sus manos en sensores de proximidad infrarrojos para seguir la secuencia. La dificultad va aumentando a medida que el jugador acierta.

- **Elevador:** Un elevador armado en una pequeña maqueta de mínimo 3 pisos el cual cuente con sensores y con indicadores de en qué piso se encuentra el elevador

- **Seleccionador de colores:** Un circuito que reconozca el color de muestra y mediante un motor y una flecha muestre cual fue el color que se usó.

- **Zumobot (2 personas):** Un pequeño robot que empuja objetos fuera de un círculo. Este proyecto puede ser elaborado entre dos o tres personas.

- **Ping pong:** Un juego para dos personas que consiste en una tira de leds que en sus extremos tienen leds de otro color. Los leds tienen una secuencia de encendido de un lado para el otro, a lo que el led de la esquina se enciende hay que presionar un botón para que vaya del otro lado, de lo contrario pierde.

- **Competencia de velocidad:** Este juego es parecido al de ping pong pero aquí la luz es situada en el medio, los jugadores tienen que aplastar las botoneras lo más rápido que puedan y dependiendo de esto la luz se comienza a mover y gana quien hace que la luz llegue hasta el extremo donde está su oponente.

- **Vúmetro:** Es un dispositivo indicador del nivel de la señal de sonido y consta de una tira de leds las cuales al llegar al nivel superior marca las notas más altas.

- **Lazo de control de temperatura:** Un pequeño lazo de control con un ventilador, un foco y un sensor de temperatura. Se debe poner un valor fijo de temperatura y esta se debería mantener con las perturbaciones.

- **Encendido con aplausos:** Un circuito que capte el sonido de un aplauso y encienda una luz y cuando suenen 2 aplausos se apague.

- **Sensor de temperatura inalámbrico:** Sensor de temperatura que funciona remotamente usando módulos XBee.

- **Maquina dispensadora de comida para perros:** Una pequeña máquina que mueve un motor que deja caer una porción de comida para perros cada cierto tiempo.

- **Balanza electrónica:** Como dice el titulo una balanza que muestre en un display cuál es el valor de lo que se está pesando.

- **Sensor de proximidad ultrasónico:** Es un sensor de proximidad que advere mediante sonido cuando un objeto está a punto de chocar.

- **Encuesta Wireless:** Una pequeña encuesta que por un lado muestra una interfaz con las preguntas y un teclado y por el otro lado usando los módulos XBee muestra los resultados de las encuestas.

CAPÍTULO 4
PRUEBAS UTILIZANDO ARDUINO

En este capítulo se revisaran los códigos con los cuales se hicieron las pruebas antes de realizar cada práctica. Algunos de estos códigos son ejemplos que vienen dentro de cada librería y muestran como es el funcionamiento de cada una. Estos ejemplos fueron muy importantes en el desarrollo de las prácticas ya que son la base de cada una.

4.1. PRUEBAS CON TECLADO MATRICIAL Y LCD

Aquí se revisaran los códigos que se usaron como prueba para el LCD 16x2 y para el teclado matricial

- **LCD 16x2**

El siguiente código imprime en pantalla "HOLA, MUNDO!" en la primera línea del LCD y en la segunda se imprimen los segundos que han pasado desde que inició el programa.

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {

  lcd.begin(16, 2);

  lcd.print("HOLA, MUNDO!");

}

void loop() {

  lcd.setCursor(0, 1);

  lcd.print(millis()/1000);

}
```

- **TECLADO MATRICIAL**

El siguiente código imprime en el monitor serial cada tecla que es presionada del teclado matricial, nótese que el teclado que se muestra en el código es de 4x3.

```
#include <Keypad.h>

const byte ROWS = 4;

const byte COLS = 3;

char keys[ROWS][COLS] = {

  {'1','2','3'},

  {'4','5','6'},

  {'7','8','9'},

  {'*','0','#'}

};

byte rowPins[ROWS] = {5, 4, 3, 2};

byte colPins[COLS] = {8, 7, 6};

Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

void setup(){

  Serial.begin(9600);

}

void loop(){
```

```
  char key = keypad.getKey();

  if (key){

    Serial.println(key);

  }

}
```

4.2. PRUEBAS CON MOTORES DC

Aquí se revisara un código básico con el que se trabajo para poder manejar los motores DC.

Los motores DC son sencillos de manejar, tienen dos cables uno de los cuales es el positivo y el otro el negativo, El positivo se lo conecta a VCC y el negativo a tierra y para darle el sentido contrario se invierten las polaridades. Con el código que se muestra a continuación se controla un terminal del motor dc , el otro va conectado a tierra. El motor gira y se detiene con intervalos de un segundo.

```
int motorPin = 9;
void setup() {

 pinMode(motorPin, OUTPUT);
}
void loop() {
 digitalWrite(motorPin, HIGH);
 delay(1000);
```

```
 digitalWrite(motorPin, LOW);
 delay(1000);
}
```

4.3. PRUEBAS CON MOTORES DE PASO (STEPPER) Y SERVOMOTORES

Aquí se revisaran los códigos que se usaron como prueba para el servomotor y para el motor de paso.

- **SERVOMOTOR**

El siguiente código muestra cómo usar la librería Servo.h para controlar fácilmente un servomotor. El servomotor gira 180 grados y luego vuelve a su posición inicial.

```
#include <Servo.h>
Servo myservo;
int pos = 0;
void setup()
{
  myservo.attach(9);
}
void loop()
```

```
{
  for(pos = 0; pos < 180; pos += 1)
  {
    myservo.write(pos);
    delay(15);
  }
  for(pos = 180; pos>=1; pos-=1)
  {
    myservo.write(pos);
    delay(15);
  }
}
```

- **MOTOR DE PASO**

El código que se muestra a continuación hace que el servomotor dé una revolución y luego vuelva a su posición original. La librería Stepper.h es muy útil para controlar muchos tipos de motores de paso, sin embargo dado que el motor de paso que se uso para las prácticas no es del todo compatible con la librería, en la práctica se decidió no usar la librería Stepper.h para controlar el servomotor.

```
#include <Stepper.h>

const int stepsPerRevolution = 200;

Stepper myStepper(stepsPerRevolution, 8,9,10,11);

void setup() {

  myStepper.setSpeed(60);

  Serial.begin(9600);

}

void loop() {

  Serial.println("clockwise");

  myStepper.step(stepsPerRevolution);

  delay(500);



  Serial.println("counterclockwise");

  myStepper.step(-stepsPerRevolution);

  delay(500);

}
```

4.4. PRUEBAS UTILIZANDO LA COMUNICACIÓN I2C

Para esta prueba se uso un LCD de 20x4 con comunicación I2C. En este caso gracias a la utilización del componente con I2C solo se usaron cuatro cables para la conexión, esto es bueno ya que a la hora de hacer un proyecto entre menos cables y terminales usados mejor. Para hacer uso del LCD con I2C se uso una librería especial que permite este tipo de comunicación.

El siguiente código muestra un simple “HOLA MUNDO” en la pantalla del LCD 20x4.

```
#include <LiquidCrystal_I2C.h>

#include <Wire.h>

LiquidCrystal_I2C lcd(0x20, 4, 5, 6, 0, 1, 2, 3, 7, NEGATIVE);



void setup()

{

  Wire.begin();

  lcd.begin(20,4);

  lcd.backlight();

}
```

```
void loop()

{

   lcd.setCursor(0,0);

   lcd.print("HOLA MUNDO");

}
```

4.5. PRUEBAS UTILIZANDO LA COMUNICACIÓN XBEE

El tema relacionado con XBee es un poco complicado y amplio como para explicarlo todo en una sola práctica, es por esto que en esta sección de pruebas se incluirán otros tres códigos que servirán para el curso de Microcontroladores.

- **COMUNICACIÓN TRANSPARENTE**

En este ejercicio tanto el coordinador como el router están configurados en modo AT. Mediante comunicación serial se envía a través del router la letra H o L y llega por medio del coordinador al Arduino el cual enciende el led si la letra fue H y lo apaga si la letra fue L.

```
const int Led = 30; // Pin del led

int byteEntrante; // Variable donde se almacenta el valor de la tecla
```

```
void setup()
{
  Serial.begin(9600); // Se inicia el puerto serial
  pinMode(Led, OUTPUT); // Se declara al led como salida
}

void loop()
{
    if (Serial.available() > 0) // si hay algo en el puerto serial entonces
    {
      byteEntrante = Serial.read(); // Se lee el puerto y se asigna a byte Entrante
      if (byteEntrante == 'H') // Si el caracter es H entonces
      {
        digitalWrite(Led, HIGH); // el led se enciende
      }

      if (byteEntrante == 'L') // Si el caracter es L entonces
      {
        digitalWrite(Led, LOW);  // el led se apaga
```

```
      }
    }
}
```

- **RECEPCION DE DATOS ANALOGICOS**

En este ejercicio se trata de obtener señales analógicas de un potenciómetro que está conectado a un router en modo AT y enviar esas señales a un coordinador que está en modo API y con esos datos dimerizar un led . En donde el pin 30 del Arduino está conectado a un led y el pin 17 del XBee router está conectado a un potenciómetro. Es importante tener en cuenta que la alimentación del XBee es de 3.3V.

```
int valor = 0; //Variable entera para leer valor
int const led = 8; //Variable del led

void setup()
{
  Serial.begin(9600); // Se inicializa el puerto serial
  pinMode(led,OUTPUT); //Se inicializa el led como salida
}
```

```
void loop()
{
  if(Serial.available() >= 21)//Asegura que hayan 21 Bytes para ser leidos
  {
    if(Serial.read() == 0x7E)//Asegura que empiece por el Byte de inicio
    {
      for (int i = 1 ; i<19 ; i++)//los siguientes 19 Bytes son descartados
      {
        byte descarte = Serial.read();//se descarta el Byte
      }
      byte MSB = Serial.read(); //se lee el bit mas significativo
      byte LSB = Serial.read(); //se lee el bit menos significativo

      int val = ((MSB << 8) | LSB); //queda un valor de 16 bits

      Serial.println(val); //imprime en el monitor serial el valor leido

      int valor = map (val,0,1023,0,255); //convierte el valor de a un rango
                                          //de 0 a 255
```

```
      analogWrite(led,valor); //dimeriza el led
    }
  }
}
```

- **ENVIO DE SEÑALES DIGITALES A UN ROUTER**

Este ejercicio se trata de enviar una secuencia de encendido y apagado desde un coordinador en modo API hacia un router en modo AT y con esos datos encender y apagar un led. En donde el pin 17 del del XBee router está conectado a un led. Es importante tener en cuenta que la alimentación del XBee es de 3.3V.

```
void setup()
{
 Serial.begin(9600); // se inicializa el puerto serial
}

void loop()
{
```

```
 remoto (0x5); // enciende
 delay (5000); // retardo
 remoto (0x4); // apaga
 delay (5000); // retardo
}

void remoto(char valor) //Subrutina para enviar informacion
{
  Serial.write(0x7E); // Byte de inicio
  Serial.write((byte)0x0); //Byte mas significativo de la longitud
  Serial.write(0x10); //Byte mas significativo de la longitud
  Serial.write(0x17); //0x17 significa que es un comando AT remoto
  Serial.write((byte)0x0);// no se solicita reconocimiento

  Serial.write((byte)0x0);//direccion de destino
  Serial.write((byte)0x0);
  Serial.write((byte)0x0);
  Serial.write((byte)0x0);
  Serial.write((byte)0x0);
  Serial.write((byte)0x0);
```

```
  Serial.write(0xFF); // FFFF para difusion
  Serial.write(0xFF);

  Serial.write(0xFF); // FFFE para difusion
  Serial.write(0xFE);

  Serial.write(0x02); // Aplica los cambios inmediatamente

  Serial.write('D'); // Opcion D
  Serial.write('3'); // Opcion 3

  Serial.write(valor); // selecciona como HIGH o LOW

  long suma = 0x17 + 0xFF + 0xFF + 0xFF + 0xFE + 0x02 + 'D' + '3' + valor;
  Serial.write(0xFF - (suma & 0xFF)); //Checksum
}
```

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. Se elaboraron exitosamente 5 prácticas para el laboratorio de Microcontroladores
2. La tarjeta Arduino Mega resultó ser una herramienta fácil de programar y usar. Por esto es ideal para el aprendizaje.
3. Fue conveniente usar la tarjeta Arduino Mega 2560 por la cantidad de entradas y salidas
4. Es importante la selección de un LCD y un teclado matricial ya que dependiendo de las marcas tienen diferente configuración.
5. Los pines de salida del Arduino son mayormente para control y no deben conectarse directamente con dispositivos que demanden mucha corriente como motores. Es por esto que se uso el integrado L293 para motores DC y ULN2003 para motores de paso.

6. No todos los motores de paso son iguales, como en el caso del motor de paso 28BYJ-48 que tiene un sistema de engranajes de reducción y que necesita una secuencia específica para su correcto funcionamiento. Es por esto que es necesario revisar los datos de los dispositivos a usar antes de ir a la práctica.
7. Es beneficioso usar la comunicación I2C porque solo se usan dos pines para realizar la comunicación de un sinnúmero de dispositivos, lo cual permite ahorrar espacio en cuanto a cableado se refiere. Sin embargo existe un problema al querer usar muchos dispositivos iguales y es que las direcciones para poder realizar la comunicación se repiten.
8. Hacer uso de los dispositivos XBee es un poco complicado, pero vale la pena aprender cómo funcionan, sobre todo para las personas interesadas en la domótica.
9. Los dispositivos XBee se pueden usar por si solos sin la necesidad de usar Arduino u otro microcontrolador.

RECOMENDACIONES

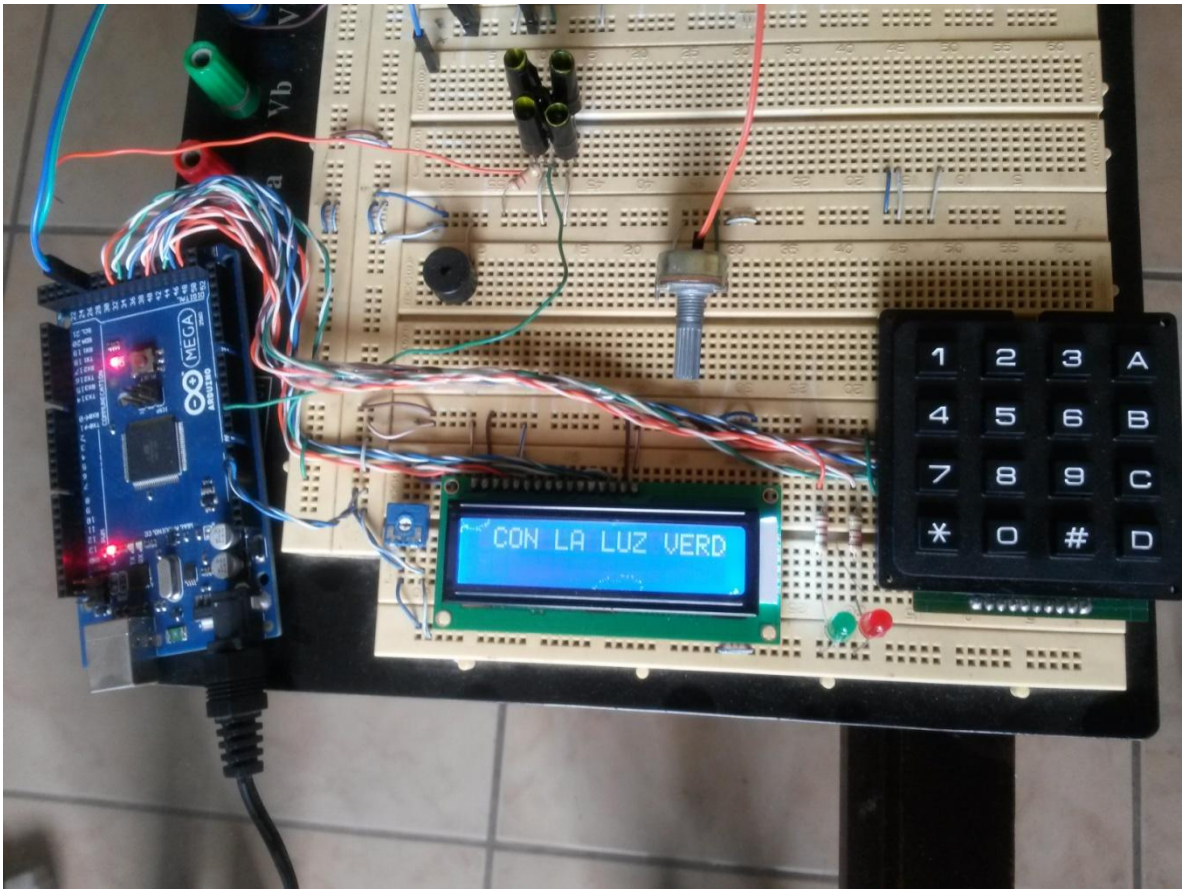
1. Se recomienda hacer uso de Arduino en los laboratorios de otras materias, como Control Automático, Instrumentación, Domótica, Robótica para que los

estudiantes tengan conocimiento de otras alternativas de dispositivos de control, sensores y actuadores.

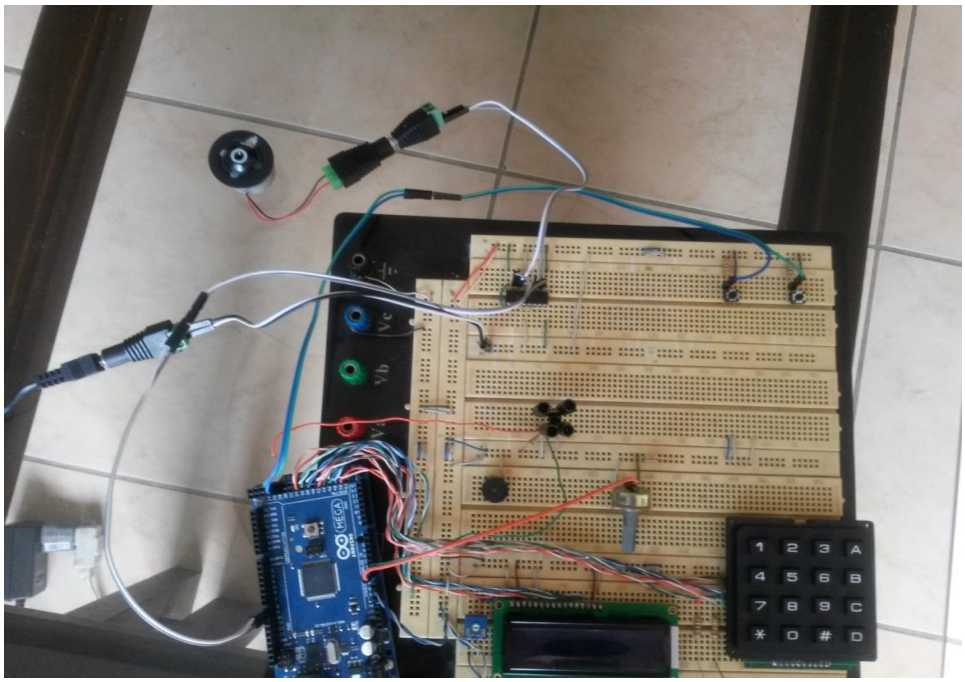
2. Se recomienda a los estudiantes investigar más sobre las librerías usadas en las prácticas ya que existen un sinnúmero de comandos que no fueron usados en estas y que pueden ayudar en sus proyectos.
3. Se recomienda adquirir más tipos de sensores y actuadores para que los estudiantes tengan un mayor conocimiento y puedan realizar diferentes tipos de proyectos.
4. Se recomienda incentivar a los a estudiantes a usar Arduino realizando concursos de proyectos.
5. Se recomienda incluir la comunicación entre Arduino y Labview.

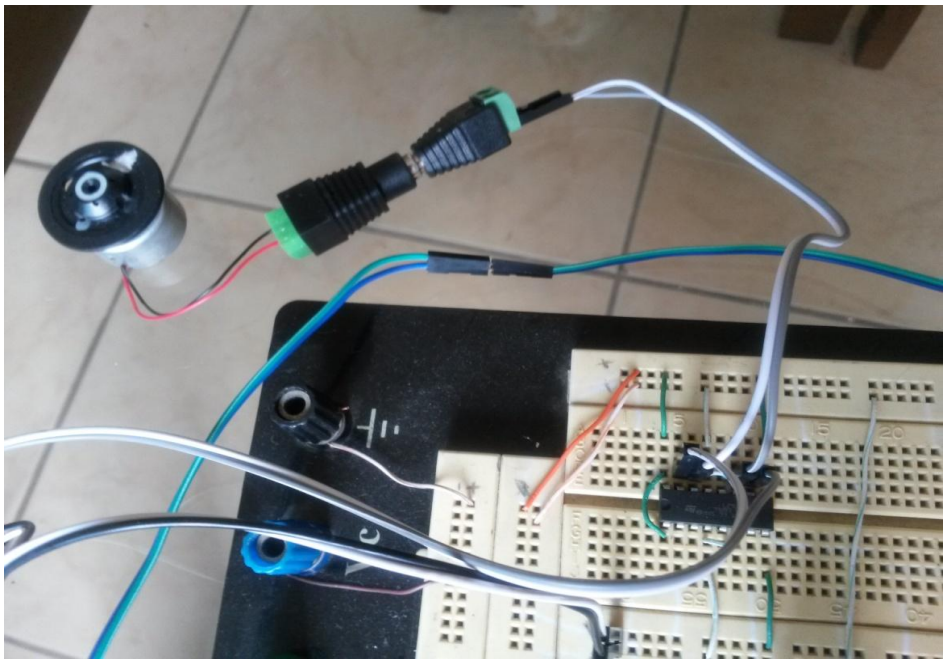
ANEXOS

Uso del teclado matricial 4x4 y LCD 16x2

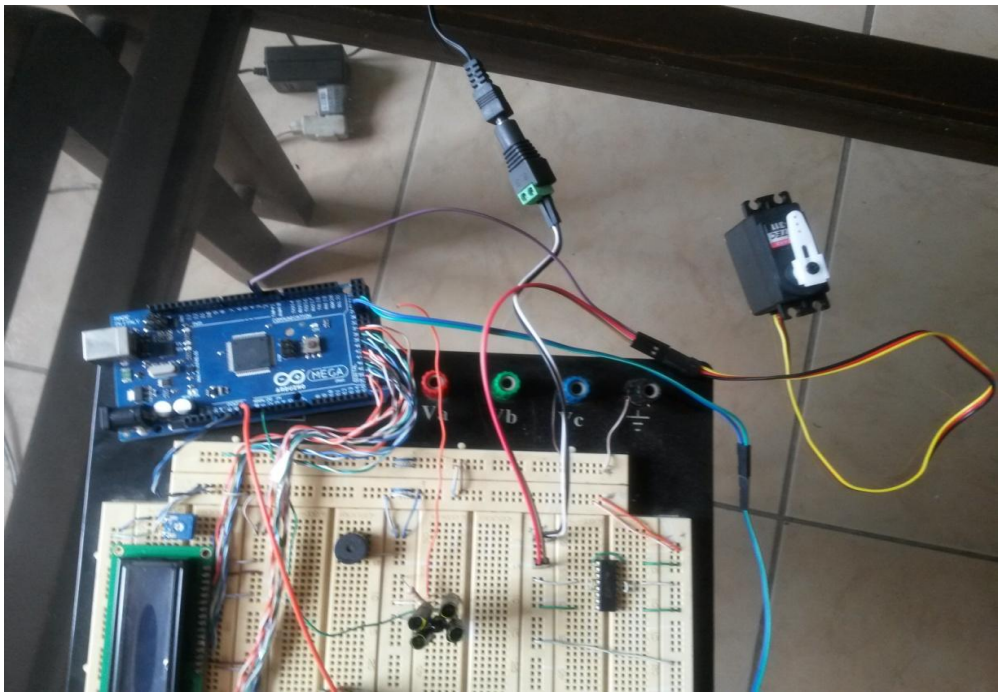


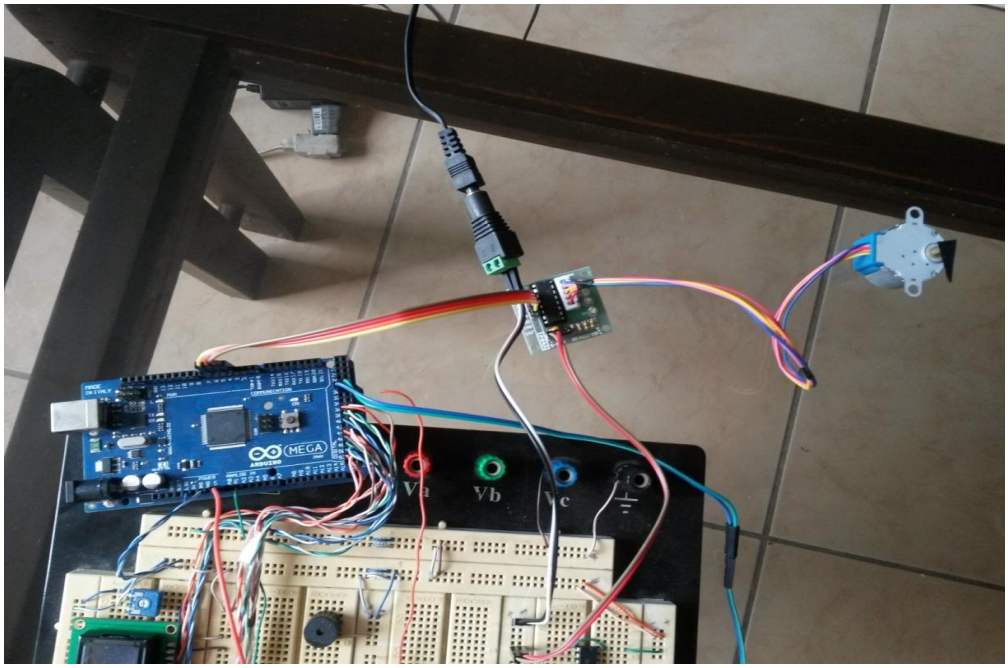
Uso de motores DC con Arduino



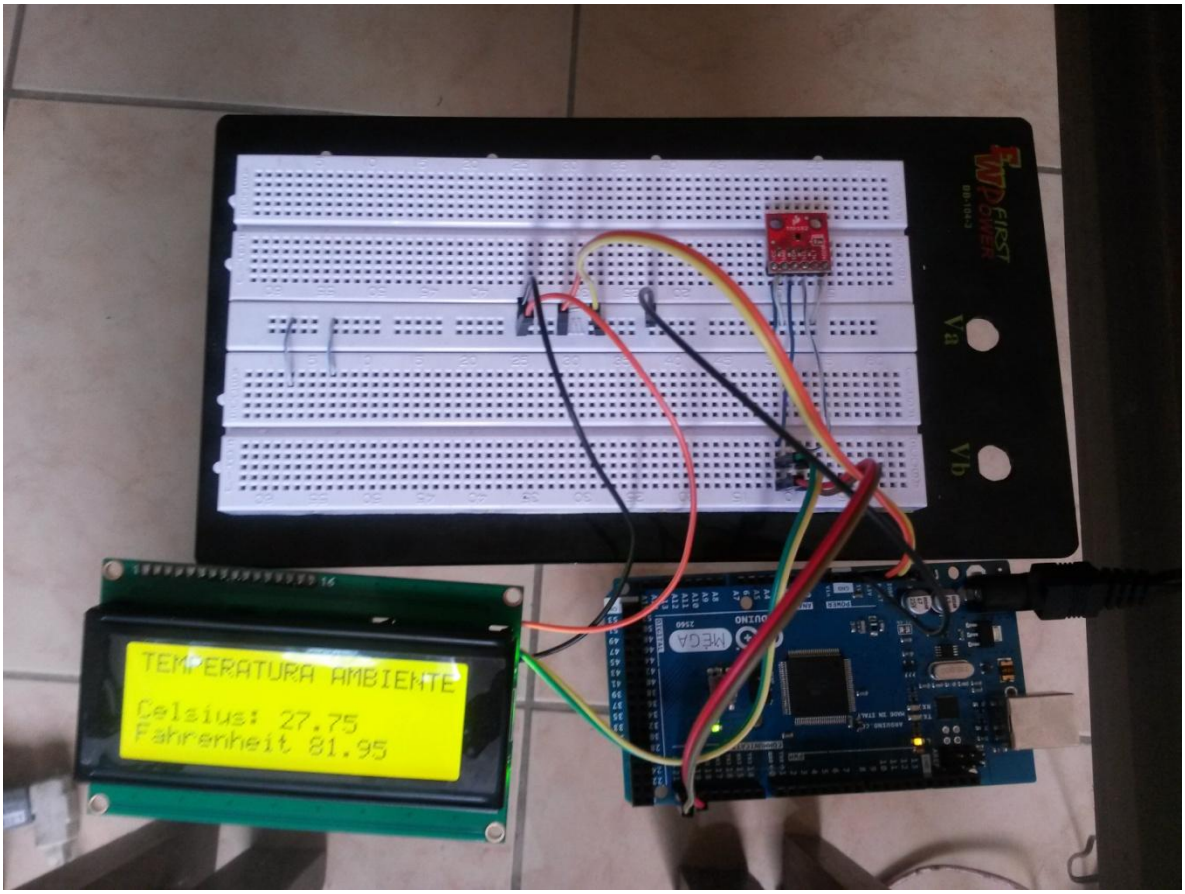


Uso de motores de paso y servomotores

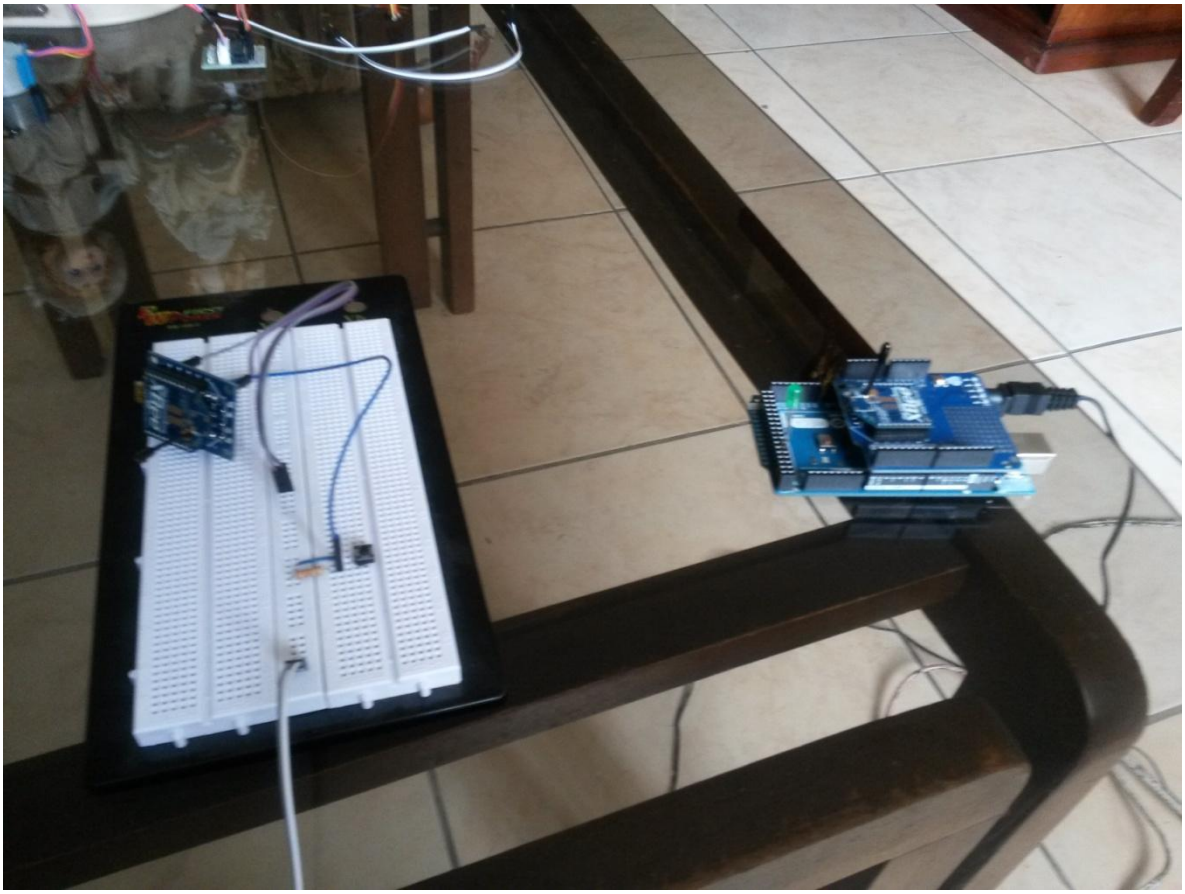




Uso de la comunicación I2C



Uso de dispositivos Xbee



BIBLIOGRAFÍA

[1] **Arduino**, Que es Arduino, http://arduino.cc/en/pmwiki.php?n=Guide/Introduction, 12/05/2014

[2] **Arduino**, Historia de Arduino, http://arduinodhtics.weebly.com/historia.html, 12/05/2014

[3] **Mouser**, Microcontrolador Atmega2560, http://www.mouser.mx/ProductDetail /Atmel/ ATmega2560-16AU/?qs=aqrrBurbvGciYmlAokFS0w==, 15/05/2014

[4] **Arduino**, Microcontrolador Atmega2560 http://arduino.cc/en/Main/arduino BoardMega2560, 15/05/2014

[5] **Wiring.org**, Programación de Atmega2560 http://wiring.org.co/, 22/05/2014

[6] **Arduino**, Estructura básica para programas Atmega2560 http://playground.arduino.cc/ArduinoNotebookTraduccion/Structure, 24/05/201

[7] **Arduino**, Introducción a Implementación con Arduino, http://arduino.cc/es/Guide/Introduction, 24/05/2014

[8] **Scribd**, Lcd 16x2, http://es.scribd.com/doc/44252680/LCD-16X2, 03/06/2014

[9] **Todoelectrodo**, Pines de Lcd 16x2, http://todoelectrodo.blogspot.com/2013/02/lcd-16x2.html, 03/06/2014

[10] **Programarpicenc**, Taclado Matricial, http://www.programarpicenc.com/libro/cap08-teclado-matricial-4x4-microcontroladores-pic.html, 04/06/2014

[11] **eHow**, Definicion de Motor, http://www.ehowenespanol.com/definicion-motor-corriente-directa-sobre_55810/, 20/06/2014

[12] **Robots**, Motor DC, http://robots-argentina.com.ar/MotorCC_L293D.htm, 03/06/2014

[13] Tienda Robotica, Servo Motor, http://tienda.tdrobotica.co/categoria/81, 07/07/2014

[14] **Docentes.unal.edu.co**, Motores de Paso, http://www.docentes.unal.edu.co/hfvelascop/docs/CLASES/DIGITALES2/LABORATORIO/Motor%20Paso%20a%20Paso.pdf,12/07/2014

[15] **Robocraf**, Funcionamiento de Servo Motor http://robocraft.ru/files/datasheet/28BYJ-48.pdf, 12/07/2014

[16] **Datasheetcatalog**, Integrado ULN2003, http://pdf.datasheetcatalog.com/datasheet2/f/0c6x6a46ig46qlxf3j2qsaii8o3y.pdf, 15/07/2014

[17] **Riverstone embedded**, I2C, http://riverstoneembedded.zxq.net/html/i2c_basics.html, 02/08/2014

[18] **Components**, I2C, http://components.about.com/od/Theory/a/Overview-Of-I2c.htm, 03/08/2014

[19] **Open Circuits**, XBee, http://www.opencircuits.com/Xbee_wireless_module#XBee_ZB_ZigBee, 05/09/2014

[20] **Plataformas XBee, XBee**, http://plataformaszigbee.blogspot.com/2012/05/practica-1-configuracion-y-conceptos.html, 09/09/2014

[21] **Blog Electrónica**, Configuracion de XBee, http://www.blogelectronica.com/zigbee-maxstream-sdk/,13/09/2014

[22] **Tunnelsup**, API a Router en XBee, http://www.tunnelsup.com/xbee-s2-quick-reference-guide-cheat-sheet/, 19/09/2014

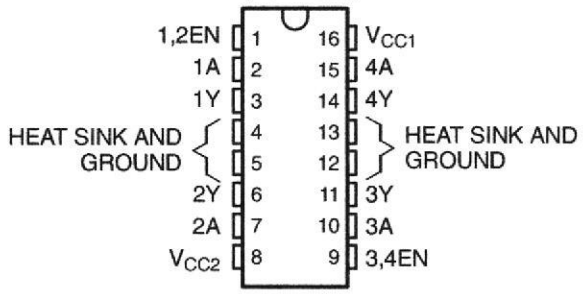
[23] **Mecatronica**, Pasos para configuración de XBee, http://mecatronicauaslp.wordpress.com/2013/07/04/tutorial-xbee-parte-2-configuracion-xbee-serie-2/, 23/09/2014

L293, L293D
QUADRUPLE HALF-H DRIVERS

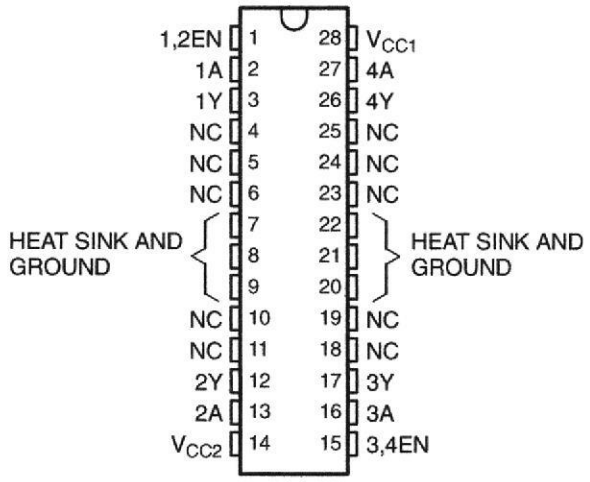
SLRS008C – SEPTEMBER 1986 – REVISED NOVEMBER 2004

- **Featuring Unitrode L293 and L293D Products Now From Texas Instruments**
- **Wide Supply-Voltage Range: 4.5 V to 36 V**
- **Separate Input-Logic Supply**
- **Internal ESD Protection**
- **Thermal Shutdown**
- **High-Noise-Immunity Inputs**
- **Functionally Similar to SGS L293 and SGS L293D**
- **Output Current 1 A Per Channel (600 mA for L293D)**
- **Peak Output Current 2 A Per Channel (1.2 A for L293D)**
- **Output Clamp Diodes for Inductive Transient Suppression (L293D)**

L293 . . . N OR NE PACKAGE
L293D . . . NE PACKAGE
(TOP VIEW)



L293 . . . DWP PACKAGE
(TOP VIEW)



description/ordering information

The L293 and L293D are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, dc and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

All inputs are TTL compatible. Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled, and their outputs are active and in phase with their inputs. When the enable input is low, those drivers are disabled, and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.

ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
0°C to 70°C	HSOP (DWP)	Tube of 20	L293DWP	L293DWP
	PDIP (N)	Tube of 25	L293N	L293N
	PDIP (NE)	Tube of 25	L293NE	L293NE
		Tube of 25	L293DNE	L293DNE

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

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

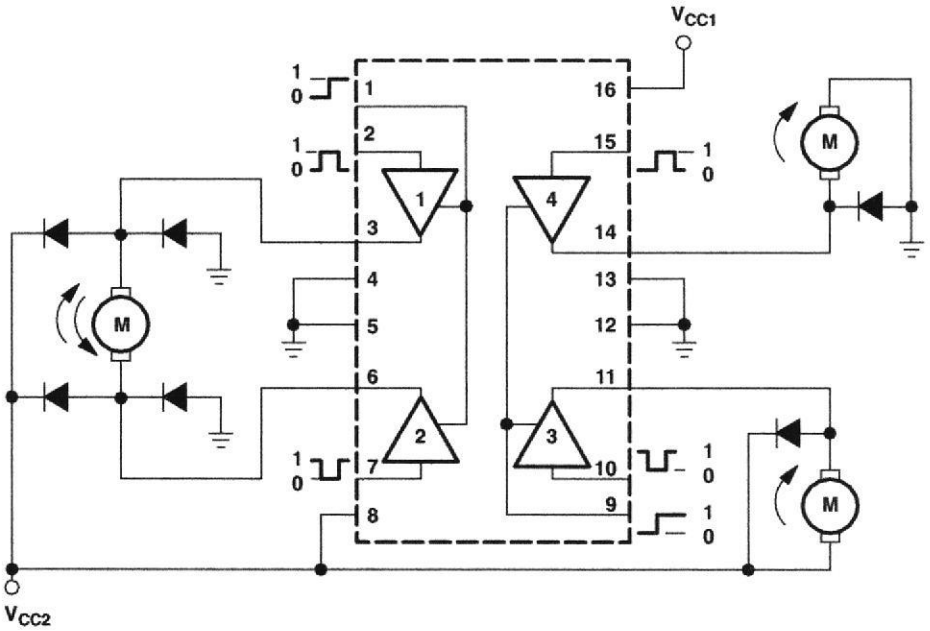
description/ordering information (continued)

On the L293, external high-speed output clamp diodes should be used for inductive transient suppression.

A V_{CC1} terminal, separate from V_{CC2}, is provided for the logic inputs to minimize device power dissipation.

The L293and L293D are characterized for operation from 0°C to 70°C.

block diagram



NOTE: Output diodes are internal in L293D.

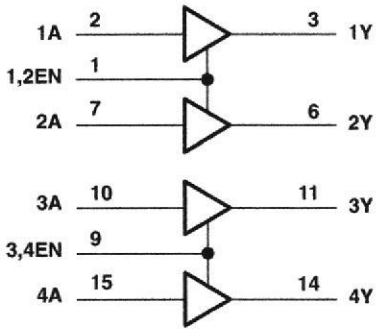
FUNCTION TABLE
(each driver)

INPUTS†		OUTPUT
A	EN	Y
H	H	H
L	H	L
X	L	Z

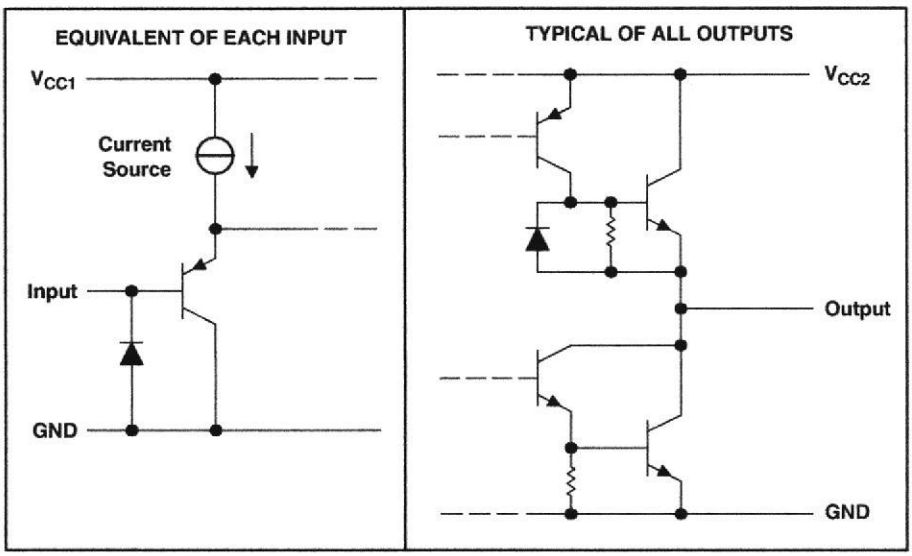
H = high level, L = low level, X = irrelevant,
Z = high impedance (off)

† In the thermal shutdown mode, the output is in the high-impedance state, regardless of the input levels.

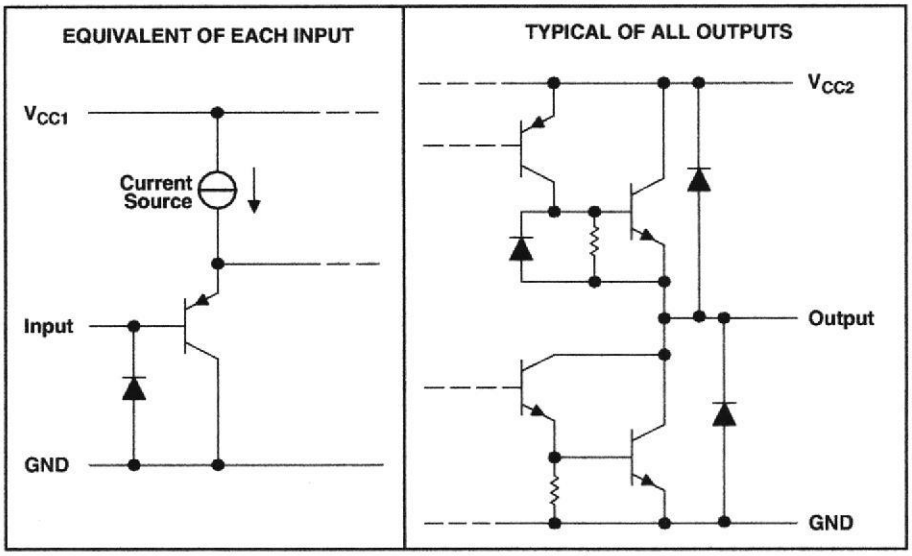
logic diagram



schematics of inputs and outputs (L293)



schematics of inputs and outputs (L293D)



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

Supply voltage, V_{CC1} (see Note 1)	36 V
Output supply voltage, V_{CC2}	36 V
Input voltage, V_I	7 V
Output voltage range, V_O	−3 V to V_{CC2} + 3 V
Peak output current, I_O (nonrepetitive, $t \leq 5$ ms): L293	±2 A
Peak output current, I_O (nonrepetitive, $t \leq 100$ µs): L293D	±1.2 A
Continuous output current, I_O: L293	±1 A
Continuous output current, I_O: L293D	±600 mA
Package thermal impedance, θ_{JA} (see Notes 2 and 3): DWP package	TBD°C/W
N package	67°C/W
NE package	TBD°C/W
Maximum 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 voltage values are with respect to the network ground terminal.
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	MAX	UNIT
	Supply voltage	V_{CC1}	4.5	7	V
		V_{CC2}	V_{CC1}	36	
V_{IH}	High-level input voltage	$V_{CC1} \le 7$ V	2.3	V_{CC1}	V
		$V_{CC1} \ge 7$ V	2.3	7	V
V_{IL}	Low-level output voltage		−0.3†	1.5	V
T_A	Operating free-air temperature		0	70	°C

† The algebraic convention, in which the least positive (most negative) designated minimum, is used in this data sheet for logic voltage levels.

electrical characteristics, V_{CC1} = 5 V, V_{CC2} = 24 V, T_A = 25°C

PARAMETER			TEST CONDITIONS		MIN	TYP	MAX	UNIT
V_{OH}	High-level output voltage		L293: I_{OH} = −1 A L293D: I_{OH} = −0.6 A		V_{CC2} − 1.8	V_{CC2} − 1.4		V
V_{OL}	Low-level output voltage		L293: I_{OL} = 1 A L293D: I_{OL} = 0.6 A			1.2	1.8	V
V_{OKH}	High-level output clamp voltage		L293D: I_{OK} = −0.6 A			V_{CC2} + 1.3		V
V_{OKL}	Low-level output clamp voltage		L293D: I_{OK} = 0.6 A			1.3		V
I_{IH}	High-level input current	A	V_I = 7 V			0.2	100	µA
		EN				0.2	10	
I_{IL}	Low-level input current	A	V_I = 0			−3	−10	µA
		EN				−2	−100	
I_{CC1}	Logic supply current		I_O = 0	All outputs at high level		13	22	mA
				All outputs at low level		35	60	
				All outputs at high impedance		8	24	
I_{CC2}	Output supply current		I_O = 0	All outputs at high level		14	24	mA
				All outputs at low level		2	6	
				All outputs at high impedance		2	4	

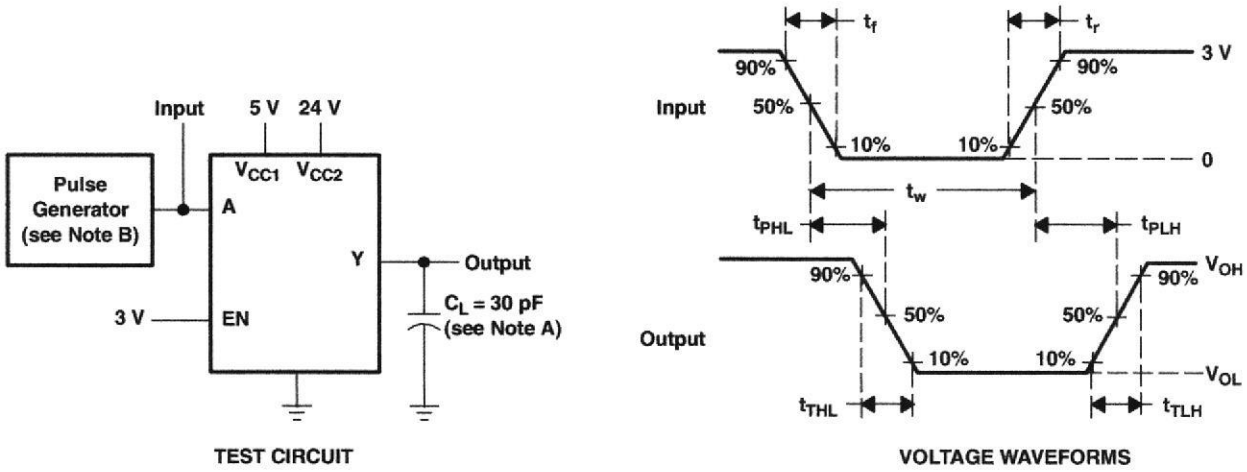
switching characteristics, V_{CC1} = 5 V, V_{CC2} = 24 V, T_A = 25°C

PARAMETER		TEST CONDITIONS	L293NE, L293DNE			UNIT
			MIN	TYP	MAX	
t_{PLH}	Propagation delay time, low-to-high-level output from A input	C_L = 30 pF, See Figure 1		800		ns
t_{PHL}	Propagation delay time, high-to-low-level output from A input			400		ns
t_{TLH}	Transition time, low-to-high-level output			300		ns
t_{THL}	Transition time, high-to-low-level output			300		ns

switching characteristics, V_{CC1} = 5 V, V_{CC2} = 24 V, T_A = 25°C

PARAMETER		TEST CONDITIONS	L293DWP, L293N L293DN			UNIT
			MIN	TYP	MAX	
t_{PLH}	Propagation delay time, low-to-high-level output from A input	C_L = 30 pF, See Figure 1		750		ns
t_{PHL}	Propagation delay time, high-to-low-level output from A input			200		ns
t_{TLH}	Transition time, low-to-high-level output			100		ns
t_{THL}	Transition time, high-to-low-level output			350		ns

PARAMETER MEASUREMENT INFORMATION



TEST CIRCUIT

VOLTAGE WAVEFORMS

NOTES: A. C_L includes probe and jig capacitance.
B. The pulse generator has the following characteristics: $t_r \leq 10$ ns, $t_f \leq 10$ ns, $t_w = 10$ µs, PRR = 5 kHz, $Z_O = 50$ Ω.

Figure 1. Test Circuit and Voltage Waveforms

APPLICATION INFORMATION

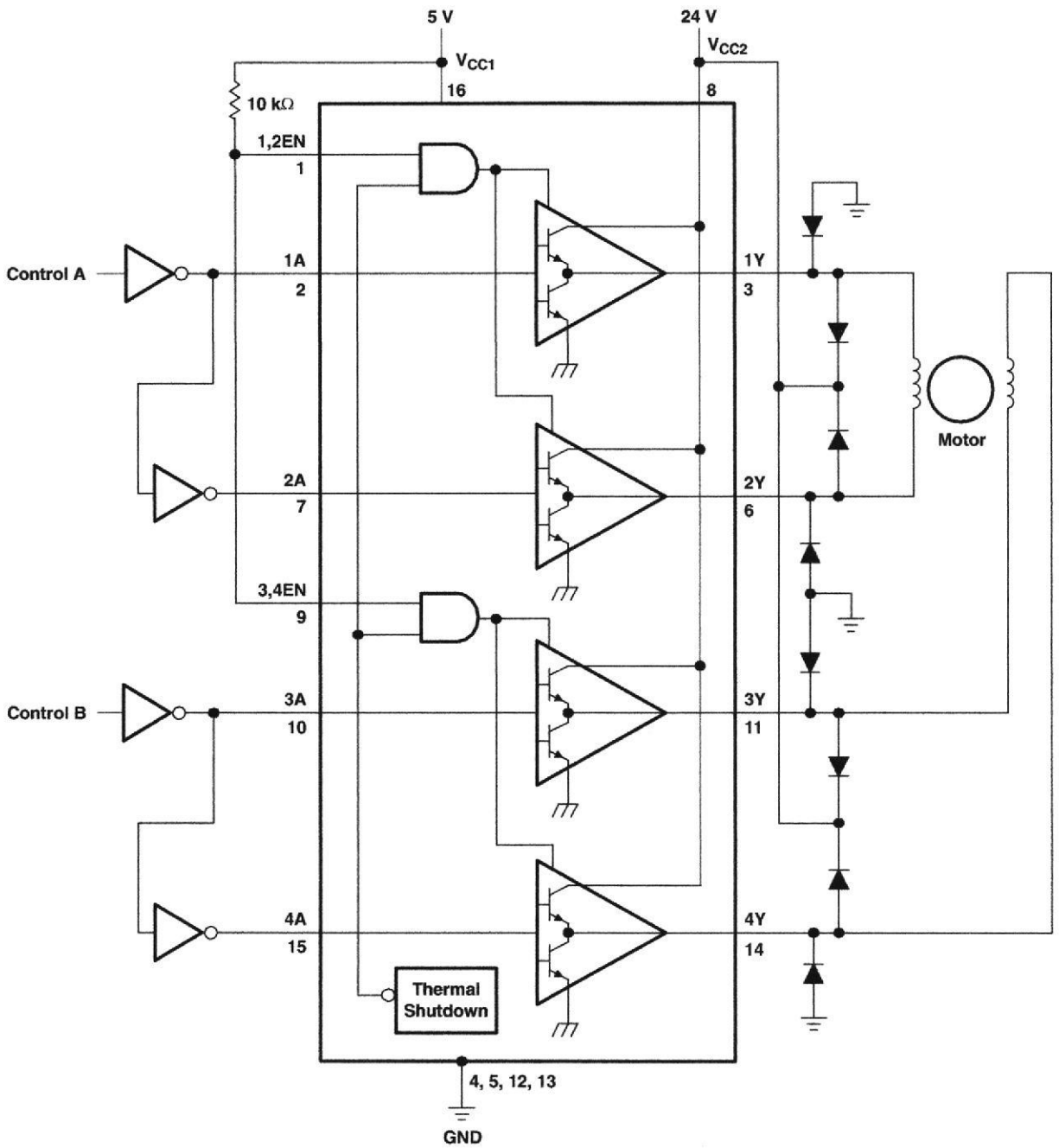


Figure 2. Two-Phase Motor Driver (L293)

APPLICATION INFORMATION

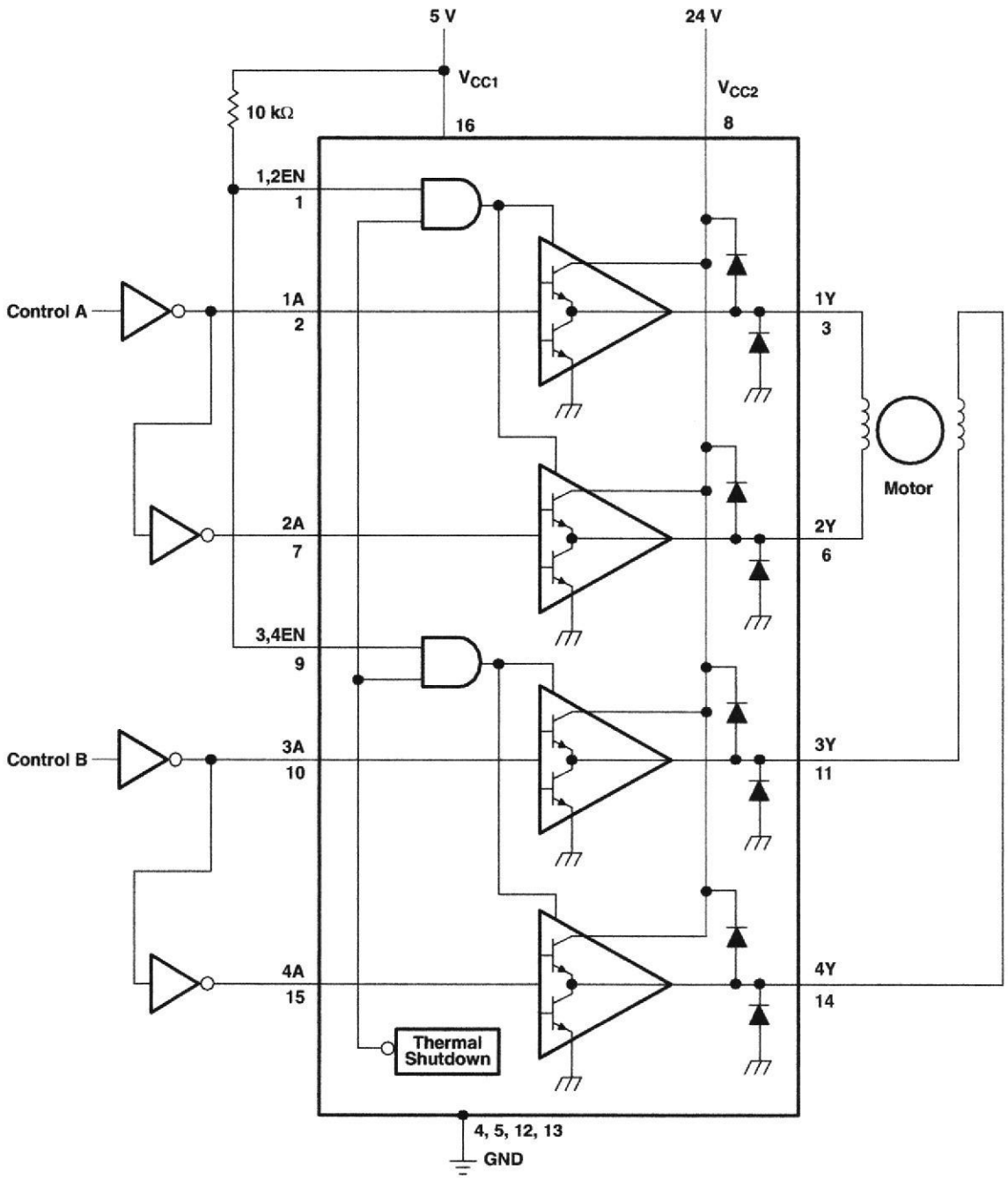


Figure 3. Two-Phase Motor Driver (L293D)

APPLICATION INFORMATION

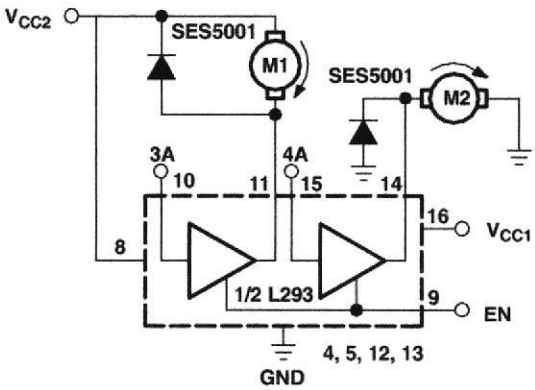


Figure 4. DC Motor Controls (connections to ground and to supply voltage)

EN	3A	M1	4A	M2
H	H	Fast motor stop	H	Run
H	L	Run	L	Fast motor stop
L	X	Free-running motor stop	X	Free-running motor stop

L = low, H = high, X = don't care

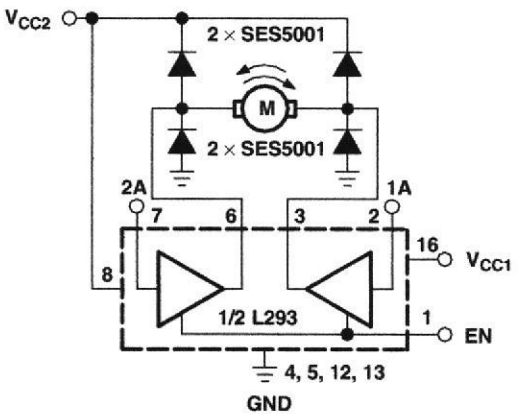
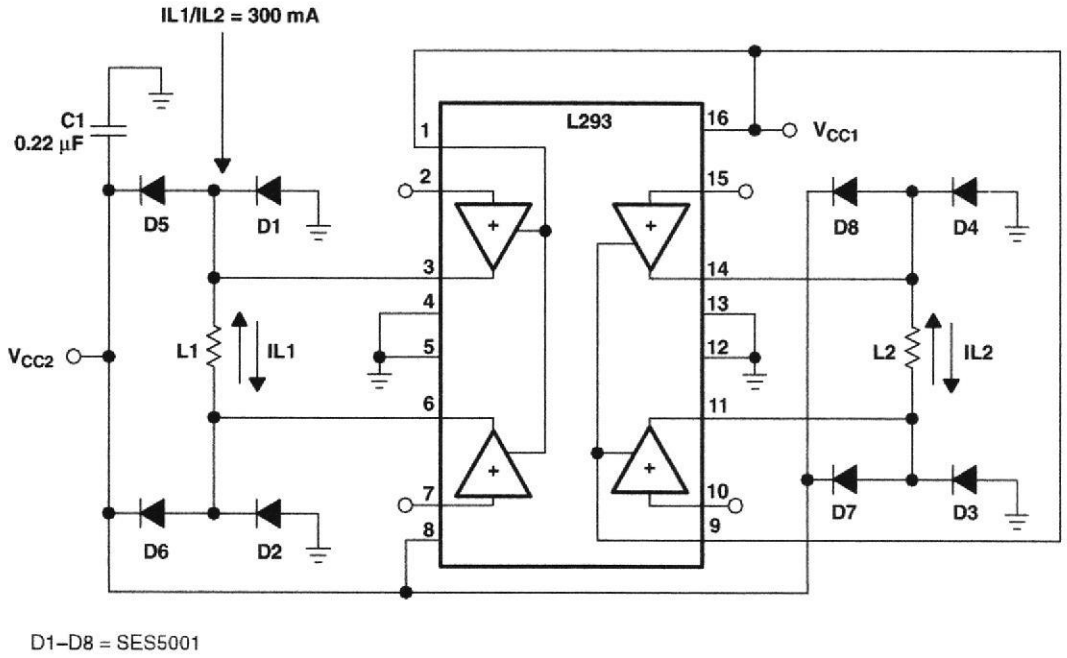


Figure 5. Bidirectional DC Motor Control

EN	1A	2A	FUNCTION
H	L	H	Turn right
H	H	L	Turn left
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Fast motor stop

L = low, H = high, X = don't care

APPLICATION INFORMATION



D1–D8 = SES5001

Figure 6. Bipolar Stepping-Motor Control

mounting instructions

The Rthj-amp of the L293 can be reduced by soldering the GND pins to a suitable copper area of the printed circuit board or to an external heat sink.

Figure 9 shows the maximum package power P_{TOT} and the θ_{JA} as a function of the side ℓ of two equal square copper areas having a thickness of 35 µm (see Figure 7). In addition, an external heat sink can be used (see Figure 8).

During soldering, the pin temperature must not exceed 260°C, and the soldering time must not exceed 12 seconds.

The external heatsink or printed circuit copper area must be connected to electrical ground.

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

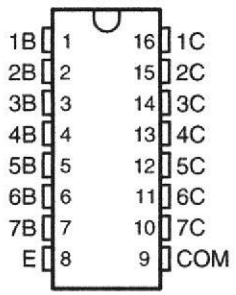
HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON TRANSISTOR ARRAYS

Check for Samples: ULN2002A, ULN2003A, ULN2003AI, ULN2004A, ULQ2003A, ULQ2004A

FEATURES

- **500-mA-Rated Collector Current (Single Output)**
- **High-Voltage Outputs: 50 V**
- **Output Clamp Diodes**
- **Inputs Compatible With Various Types of Logic**
- **Relay-Driver Applications**

ULN2002A . . . N PACKAGE
ULN2003A . . . D, N, NS, OR PW PACKAGE
ULN2004A . . . D, N, OR NS PACKAGE
ULQ2003A, ULQ2004A . . . D OR N PACKAGE
(TOP VIEW)



DESCRIPTION

The ULN2002A, ULN2003A, ULN2003AI, ULN2004A, ULQ2003A, and ULQ2004A are high-voltage high-current Darlington transistor arrays. Each consists of seven npn Darlington pairs that feature high-voltage outputs with common-cathode clamp diodes for switching inductive loads. The collector-current rating of a single Darlington pair is 500 mA. The Darlington pairs can be paralleled for higher current capability. Applications include relay drivers, hammer drivers, lamp drivers, display drivers (LED and gas discharge), line drivers, and logic buffers. For 100-V (otherwise interchangeable) versions of the ULN2003A and ULN2004A, see the SN75468 and SN75469, respectively.

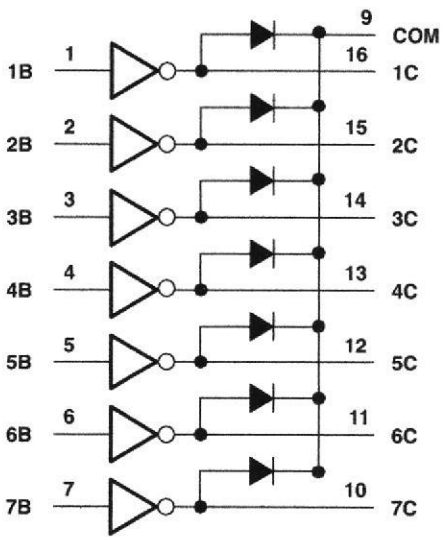
The ULN2002A is designed specifically for use with 14-V to 25-V PMOS devices. Each input of this device has a Zener diode and resistor in series to control the input current to a safe limit. The ULN2003A and ULQ2003A have a 2.7-kΩ series base resistor for each Darlington pair for operation directly with TTL or 5-V CMOS devices. The ULN2004A and ULQ2004A have a 10.5-kΩ series base resistor to allow operation directly from CMOS devices that use supply voltages of 6 V to 15 V. The required input current of the ULN/ULQ2004A is below that of the ULN/ULQ2003A, and the required voltage is less than that required by the ULN2002A.

ORDERING INFORMATION[1]

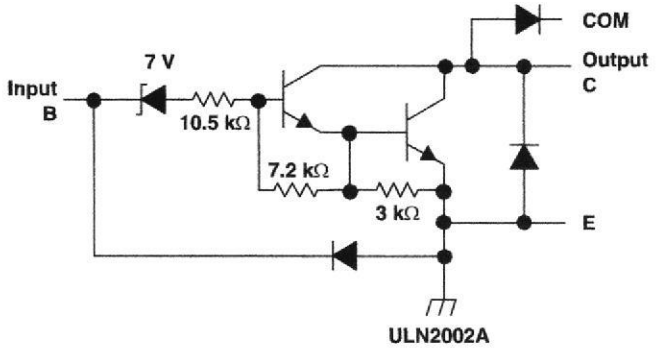
T_A	PACKAGE[2]		ORDERABLE PART NUMBER	TOP-SIDE MARKING
–20°C to 70°C	PDIP – N	Tube of 25	ULN2002AN	ULN2002AN
			ULN2003AN	ULN2003AN
			ULN2004AN	ULN2004AN
	SOIC – D	Tube of 40	ULN2003AD	ULN2003A
		Reel of 2500	ULN2003ADR	
		Reel of 2500	ULN2003ADRG3	
		Tube of 40	ULN2004AD	ULN2004A
		Reel of 2500	ULN2004ADRG3	
	SOP – NS	Reel of 2000	ULN2003ANSR	ULN2003A
			ULN2004ANSR	ULN2004A
	TSSOP – PW	Tube of 90	ULN2003APW	UN2003A
		Reel of 2000	ULN2003APWR	
–40°C to 85°C	PDIP – N	Tube of 25	ULQ2003AN	ULQ2003A
			ULQ2004AN	ULQ2004AN
	SOIC – D	Tube of 40	ULQ2003AD	ULQ2003A
		Reel of 2500	ULQ2003ADR	
		Tube of 40	ULQ2004AD	ULQ2004A
		Reel of 2500	ULQ2004ADR	
–40°C to 105°C	SOP – NS	Reel of 2000	ULN2003AINSR	ULN2003AI
	PDIP – N	Tube of 425	ULN2003AIN	ULN2003AIN
	SOIC – D	Tube of 40	ULN2003AID	ULN2003AI
		Reel of 2500	ULN2003AIDR	
	TSSOP – PW	Reel of 2500	ULN2003AIPWR	UN2003AI

(1) For the most current package and ordering information, see the Package Option Addendum at the end of this document, or see the TI web site at www.ti.com.
(2) Package drawings, thermal data, and symbolization are available at www.ti.com/packaging.

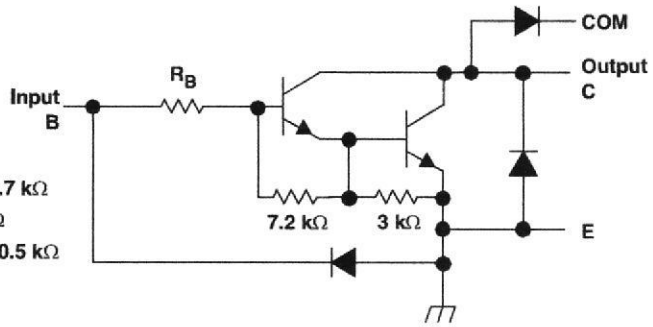
LOGIC DIAGRAM



SCHEMATICS (EACH DARLINGTON PAIR)



ULN2002A



ULN/ULQ2003A: R_B = 2.7 kΩ
ULN2003AI: R_B = 2.7 kΩ
ULN/ULQ2004A: R_B = 10.5 kΩ

ULN2003A, ULN2003AI, ULN2004A, ULQ2003A, ULQ2004A

All resistor values shown are nominal.

The collector-emitter diode is a parasitic structure and should not be used to conduct current. If the collector(s) go below ground an external Schottky diode should be added to clamp negative undershoots.

ABSOLUTE MAXIMUM RATINGS(1)

at 25°C free-air temperature (unless otherwise noted)

			MIN	MAX	UNIT
V_{CC}	Collector-emitter voltage			50	V
	Clamp diode reverse voltage(2)			50	V
V_I	Input voltage(2)			30	V
	Peak collector current	See Figure 14 and Figure 15		500	mA
I_{OK}	Output clamp current			500	mA
	Total emitter-terminal current			–2.5	A
T_A	Operating free-air temperature range	ULN200xA	–20	70	°C
		ULN200xAI	–40	105	
		ULQ200xA	–40	85	
		ULQ200xAT	–40	105	
θ_{JA}	Package thermal impedance(3) (4)	D package		73	°C/W
		N package		67	
		NS package		64	
		PW package		108	
θ_{JC}	Package thermal impedance(5) (6)	D package		36	
		N package		54	
T_J	Operating virtual junction temperature			150	°C
	Lead temperature for 1.6 mm (1/16 inch) from case for 10 seconds			260	°C
T_{stg}	Storage temperature range		–65	150	°C

(1) 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.
(2) All voltage values are with respect to the emitter/substrate terminal E, unless otherwise noted.
(3) 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.
(4) The package thermal impedance is calculated in accordance with JESD 51-7.
(5) Maximum power dissipation is a function of $T_J(max)$, θ_{JC}, and T_A. The maximum allowable power dissipation at any allowable ambient temperature is $P_D = (T_J(max) - T_A)/\theta_{JC}$. Operating at the absolute maximum T_J of 150°C can affect reliability.
(6) The package thermal impedance is calculated in accordance with MIL-STD-883.

ELECTRICAL CHARACTERISTICS

T_A = 25°C

PARAMETER		TEST FIGURE	TEST CONDITIONS		ULN2002A			UNIT
					MIN	TYP	MAX	
$V_{I(on)}$	On-state input voltage	Figure 6	V_{CE} = 2 V,	I_C = 300 mA			13	V
$V_{CE(sat)}$	Collector-emitter saturation voltage	Figure 4	I_I = 250 µA,	I_C = 100 mA		0.9	1.1	V
			I_I = 350 µA,	I_C = 200 mA		1	1.3	
			I_I = 500 µA,	I_C = 350 mA		1.2	1.6	
V_F	Clamp forward voltage	Figure 7	I_F = 350 mA			1.7	2	V
I_{CEX}	Collector cutoff current	Figure 1	V_{CE} = 50 V,	I_I = 0			50	µA
		Figure 2	V_{CE} = 50 V, T_A = 70°C	I_I = 0			100	
				V_I = 6 V			500	
$I_{I(off)}$	Off-state input current	Figure 2	V_{CE} = 50 V,	I_C = 500 µA	50	65		µA
I_I	Input current	Figure 3	V_I = 17 V			0.82	1.25	mA
I_R	Clamp reverse current	Figure 6	V_R = 50 V	T_A = 70°C			100	µA
							50	
C_I	Input capacitance		V_I = 0,	f = 1 MHz			25	pF

ELECTRICAL CHARACTERISTICS

T_A = 25°C

PARAMETER		TEST FIGURE	TEST CONDITIONS		ULN2003A			ULN2004A			UNIT
					MIN	TYP	MAX	MIN	TYP	MAX	
$V_{I(on)}$	On-state input voltage	Figure 6	V_{CE} = 2 V	I_C = 125 mA						5	V
				I_C = 200 mA			2.4			6	
				I_C = 250 mA			2.7				
				I_C = 275 mA						7	
				I_C = 300 mA			3				
				I_C = 350 mA						8	
$V_{CE(sat)}$	Collector-emitter saturation voltage	Figure 5	I_I = 250 µA,	I_C = 100 mA		0.9	1.1		0.9	1.1	V
			I_I = 350 µA,	I_C = 200 mA		1	1.3		1	1.3	
			I_I = 500 µA,	I_C = 350 mA		1.2	1.6		1.2	1.6	
I_{CEX}	Collector cutoff current	Figure 1	V_{CE} = 50 V,	I_I = 0			50			50	µA
		Figure 2	V_{CE} = 50 V, T_A = 70°C	I_I = 0			100			100	
				V_I = 6 V						500	
V_F	Clamp forward voltage	Figure 8	I_F = 350 mA			1.7	2		1.7	2	V
$I_{I(off)}$	Off-state input current	Figure 3	V_{CE} = 50 V, T_A = 70°C,	I_C = 500 µA	50	65		50	65		µA
I_I	Input current	Figure 4	V_I = 3.85 V			0.93	1.35				mA
			V_I = 5 V						0.35	0.5	
			V_I = 12 V						1	1.45	
I_R	Clamp reverse current	Figure 7	V_R = 50 V				50			50	µA
				T_A = 70°C			100			100	
C_i	Input capacitance		V_I = 0,	f = 1 MHz		15	25		15	25	pF

ELECTRICAL CHARACTERISTICS

T_A = 25°C

PARAMETER		TEST FIGURE	TEST CONDITIONS		ULN2003AI			UNIT
					MIN	TYP	MAX	
$V_{I(on)}$	On-state input voltage	Figure 6	V_{CE} = 2 V	I_C = 200 mA			2.4	V
				I_C = 250 mA			2.7	
				I_C = 300 mA			3	
$V_{CE(sat)}$	Collector-emitter saturation voltage	Figure 5	I_I = 250 µA,	I_C = 100 mA		0.9	1.1	V
			I_I = 350 µA,	I_C = 200 mA		1	1.3	
			I_I = 500 µA,	I_C = 350 mA		1.2	1.6	
I_{CEX}	Collector cutoff current	Figure 1	V_{CE} = 50 V,	I_I = 0			50	µA
V_F	Clamp forward voltage	Figure 8	I_F = 350 mA			1.7	2	V
$I_{I(off)}$	Off-state input current	Figure 3	V_{CE} = 50 V,	I_C = 500 µA	50	65		µA
I_I	Input current	Figure 4	V_I = 3.85 V			0.93	1.35	mA
I_R	Clamp reverse current	Figure 7	V_R = 50 V				50	µA
C_i	Input capacitance		V_I = 0,	f = 1 MHz		15	25	pF

ELECTRICAL CHARACTERISTICS

T_A = –40°C to 105°C

PARAMETER		TEST FIGURE	TEST CONDITIONS		ULN2003AI			UNIT
					MIN	TYP	MAX	
$V_{I(on)}$	On-state input voltage	Figure 6	V_{CE} = 2 V	I_C = 200 mA			2.7	V
				I_C = 250 mA			2.9	
				I_C = 300 mA			3	
$V_{CE(sat)}$	Collector-emitter saturation voltage	Figure 5	I_I = 250 µA,	I_C = 100 mA		0.9	1.2	V
			I_I = 350 µA,	I_C = 200 mA		1	1.4	
			I_I = 500 µA,	I_C = 350 mA		1.2	1.7	
I_{CEX}	Collector cutoff current	Figure 1	V_{CE} = 50 V,	I_I = 0			100	µA
V_F	Clamp forward voltage	Figure 8	I_F = 350 mA			1.7	2.2	V
$I_{I(off)}$	Off-state input current	Figure 3	V_{CE} = 50 V,	I_C = 500 µA	30	65		µA
I_I	Input current	Figure 4	V_I = 3.85 V			0.93	1.35	mA
I_R	Clamp reverse current	Figure 7	V_R = 50 V				100	µA
C_I	Input capacitance		V_I = 0,	f = 1 MHz		15	25	pF

ELECTRICAL CHARACTERISTICS

over recommended operating conditions (unless otherwise noted)

PARAMETER		TEST FIGURE	TEST CONDITIONS		ULQ2003A			ULQ2004A			UNIT
					MIN	TYP	MAX	MIN	TYP	MAX	
$V_{I(on)}$	On-state input voltage	Figure 6	V_{CE} = 2 V	I_C = 125 mA						5	V
				I_C = 200 mA			2.7			6	
				I_C = 250 mA			2.9				
				I_C = 275 mA						7	
				I_C = 300 mA			3				
				I_C = 350 mA						8	
$V_{CE(sat)}$	Collector-emitter saturation voltage	Figure 5	I_I = 250 µA,	I_C = 100 mA		0.9	1.2		0.9	1.1	V
			I_I = 350 µA,	I_C = 200 mA		1	1.4		1	1.3	
			I_I = 500 µA,	I_C = 350 mA		1.2	1.7		1.2	1.6	
I_{CEX}	Collector cutoff current	Figure 1	V_{CE} = 50 V,	I_I = 0			100			50	µA
		Figure 2	V_{CE} = 50 V, T_A = 70°C	I_I = 0						100	
				V_I = 6 V						500	
V_F	Clamp forward voltage	Figure 8	I_F = 350 mA			1.7	2.3		1.7	2	V
$I_{I(off)}$	Off-state input current	Figure 3	V_{CE} = 50 V, T_A = 70°C,	I_C = 500 µA		65		50	65		µA
I_I	Input current	Figure 4	V_I = 3.85 V			0.93	1.35				mA
			V_I = 5 V						0.35	0.5	
			V_I = 12 V						1	1.45	
I_R	Clamp reverse current	Figure 7	V_R = 50 V	T_A = 25°C			100			50	µA
							100			100	
C_I	Input capacitance		V_I = 0,	f = 1 MHz		15	25		15	25	pF

SWITCHING CHARACTERISTICS

T_A = 25°C

PARAMETER		TEST CONDITIONS	ULN2002A, ULN2003A, ULN2004A			UNIT
			MIN	TYP	MAX	
t_{PLH}	Propagation delay time, low- to high-level output	See Figure 9		0.25	1	µs
t_{PHL}	Propagation delay time, high- to low-level output	See Figure 9		0.25	1	µs
V_{OH}	High-level output voltage after switching	V_S = 50 V, I_O = 300 mA, See Figure 10	V_S – 20			mV

SWITCHING CHARACTERISTICS

T_A = 25°C

PARAMETER		TEST CONDITIONS	ULN2003AI			UNIT
			MIN	TYP	MAX	
t_{PLH}	Propagation delay time, low- to high-level output	See Figure 9		0.25	1	µs
t_{PHL}	Propagation delay time, high- to low-level output	See Figure 9		0.25	1	µs
V_{OH}	High-level output voltage after switching	V_S = 50 V, I_O ≈ 300 mA, See Figure 10	V_S – 20			mV

SWITCHING CHARACTERISTICS

T_A = –40°C to 105°C

PARAMETER		TEST CONDITIONS	ULN2003AI			UNIT
			MIN	TYP	MAX	
t_{PLH}	Propagation delay time, low- to high-level output	See Figure 9		1	10	µs
t_{PHL}	Propagation delay time, high- to low-level output	See Figure 9		1	10	µs
V_{OH}	High-level output voltage after switching	V_S = 50 V, I_O ≈ 300 mA, See Figure 10	V_S – 50			mV

SWITCHING CHARACTERISTICS

over recommended operating conditions (unless otherwise noted)

PARAMETER		TEST CONDITIONS	ULQ2003A, ULQ2004A			UNIT
			MIN	TYP	MAX	
t_{PLH}	Propagation delay time, low- to high-level output	See Figure 9		1	10	µs
t_{PHL}	Propagation delay time, high- to low-level output	See Figure 9		1	10	µs
V_{OH}	High-level output voltage after switching	V_S = 50 V, I_O = 300 mA, See Figure 10	V_S – 20			mV

PARAMETER MEASUREMENT INFORMATION

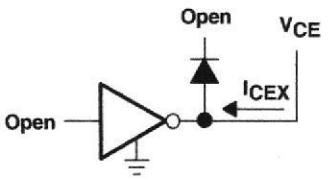


Figure 1. I_{CEX} Test Circuit

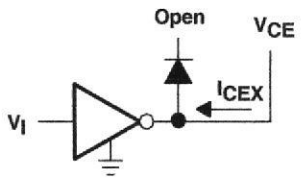


Figure 2. I_{CEX} Test Circuit

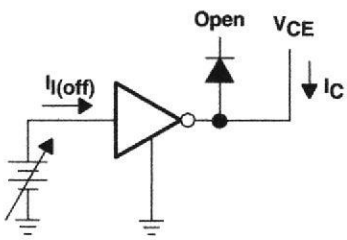


Figure 3. $I_{I(off)}$ Test Circuit

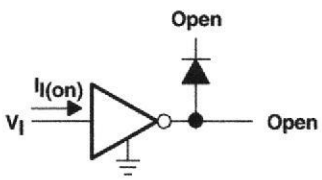


Figure 4. I_I Test Circuit

A. I_I is fixed for measuring $V_{CE(sat)}$, variable for measuring h_{FE}.

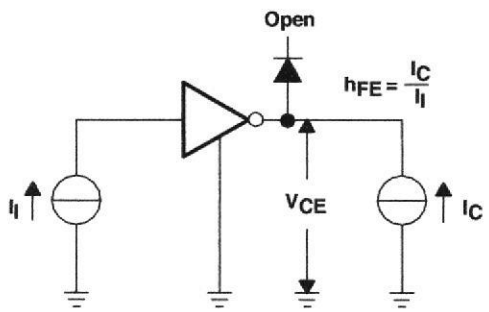


Figure 5. h_{FE}, $V_{CE(sat)}$ Test Circuit

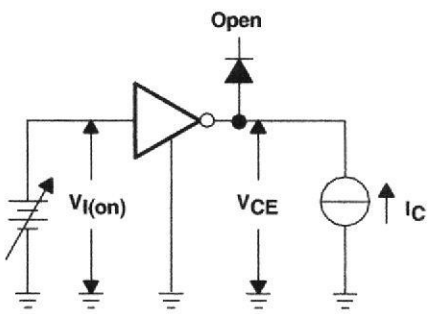


Figure 6. $V_{I(on)}$ Test Circuit

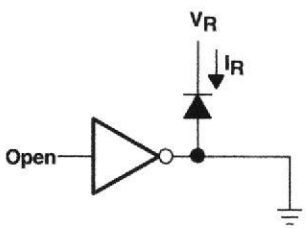


Figure 7. I_R Test Circuit

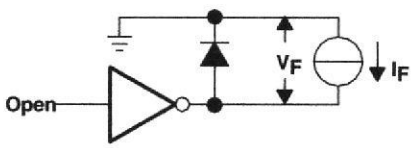
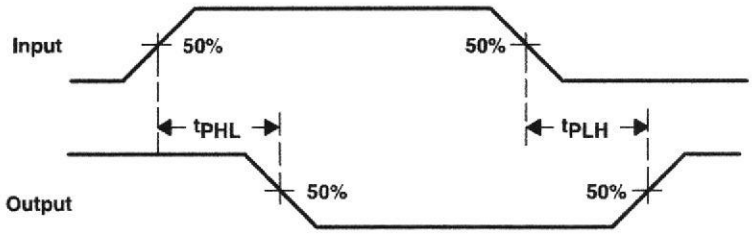


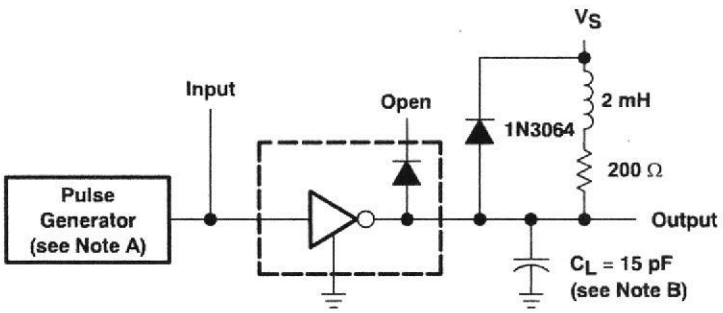
Figure 8. V_F Test Circuit

PARAMETER MEASUREMENT INFORMATION (continued)

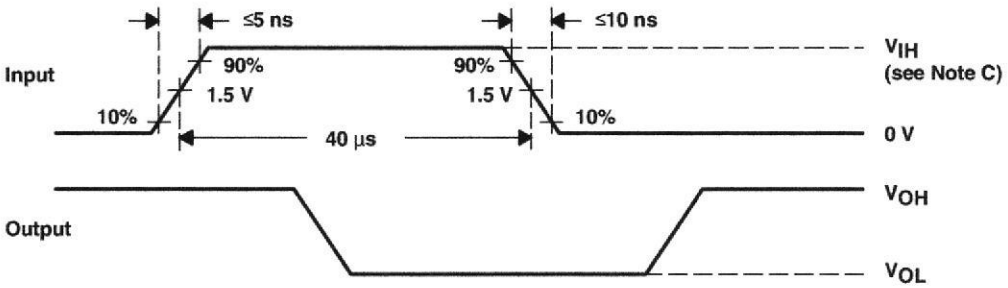


VOLTAGE WAVEFORMS

Figure 9. Propagation Delay-Time Waveforms



TEST CIRCUIT



VOLTAGE WAVEFORMS

Figure 10. Latch-Up Test Circuit and Voltage Waveforms

A. The pulse generator has the following characteristics: PRR = 12.5 kHz, Z_O = 50 Ω.

B. C_L includes probe and jig capacitance.

C. For testing the ULN2003A, ULN2003AI, and ULQ2003A, V_{IH} = 3 V; for the ULN2002A, V_{IH} = 13 V; for the ULN2004A and the ULQ2004A, V_{IH} = 8 V.

TYPICAL CHARACTERISTICS

COLLECTOR-EMITTER SATURATION VOLTAGE
vs
COLLECTOR CURRENT (ONE DARLINGTON)

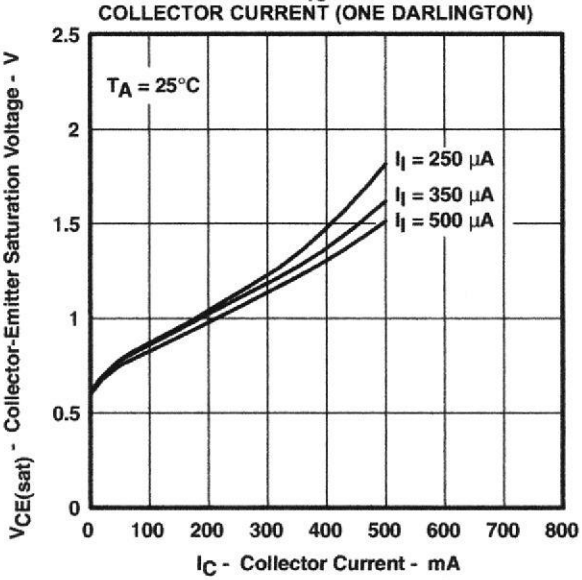


Figure 11.

COLLECTOR-EMITTER SATURATION VOLTAGE
vs
TOTAL COLLECTOR CURRENT (TWO DARLINGTONS IN PARALLEL)

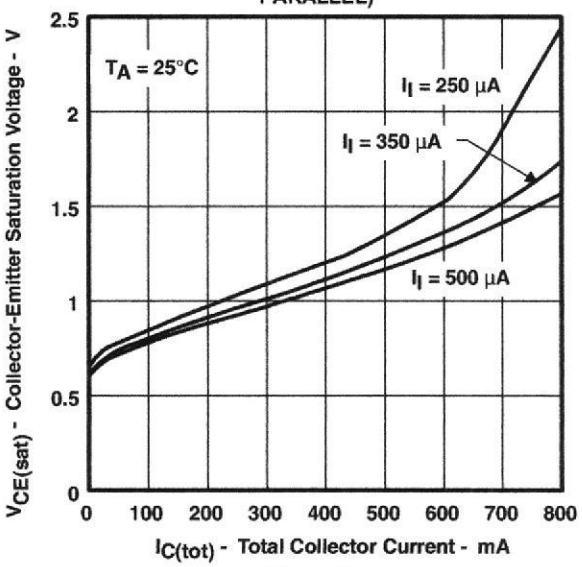


Figure 12.

COLLECTOR CURRENT
vs
INPUT CURRENT

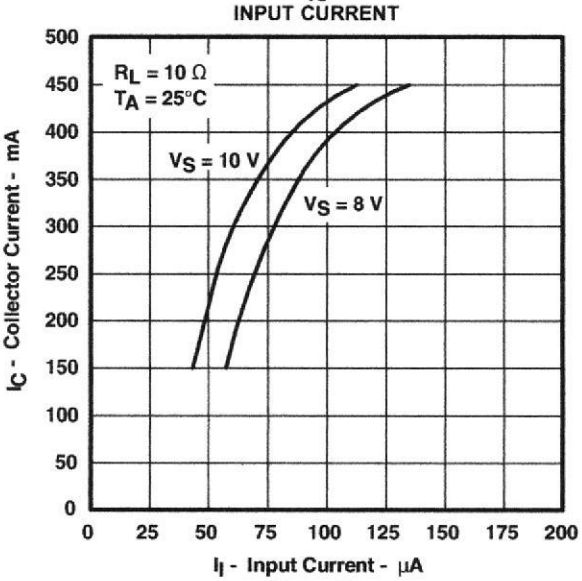


Figure 13.

D PACKAGE
MAXIMUM COLLECTOR CURRENT
vs
DUTY CYCLE

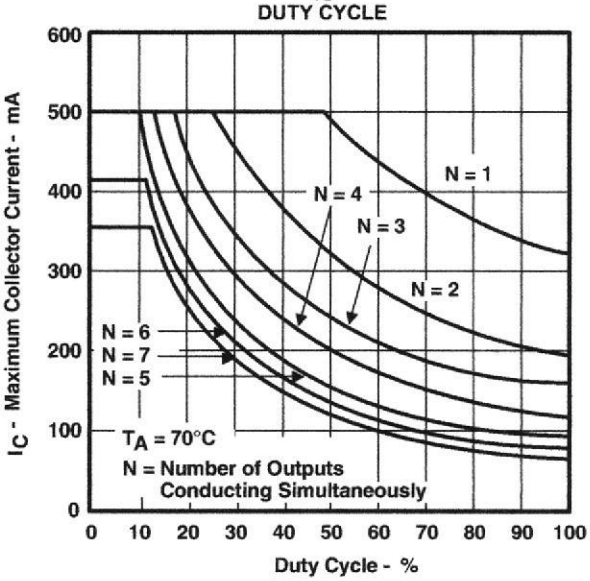


Figure 14.

TYPICAL CHARACTERISTICS (continued)

N PACKAGE
MAXIMUM COLLECTOR CURRENT
vs
DUTY CYCLE

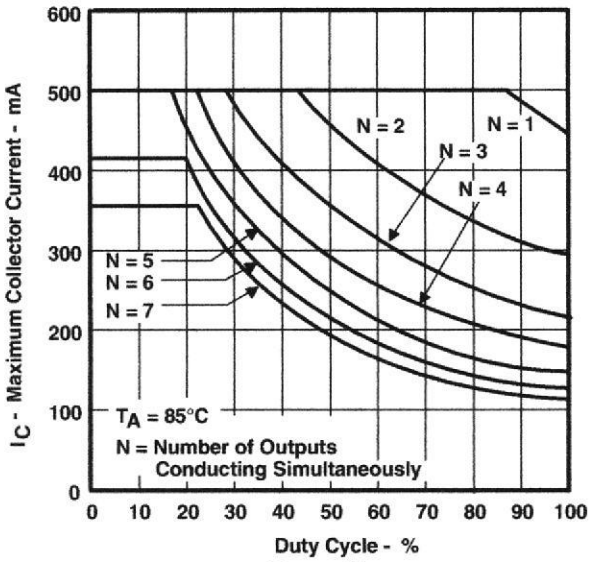


Figure 15.

MAXIMUM AND TYPICAL INPUT CURRENT
vs
INPUT VOLTAGE

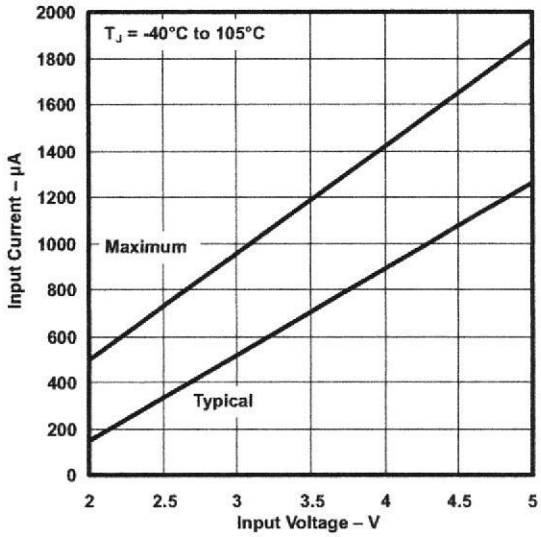


Figure 16.

MAXIMUM AND TYPICAL SATURATED V_{CE}
vs
OUTPUT CURRENT

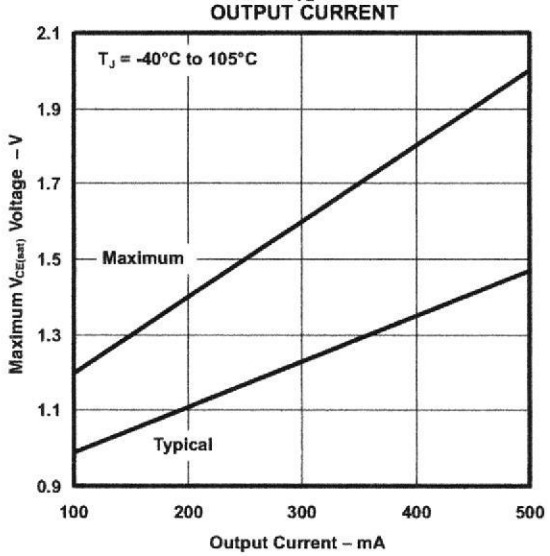


Figure 17.

MINIMUM OUTPUT CURRENT
vs
INPUT CURRENT

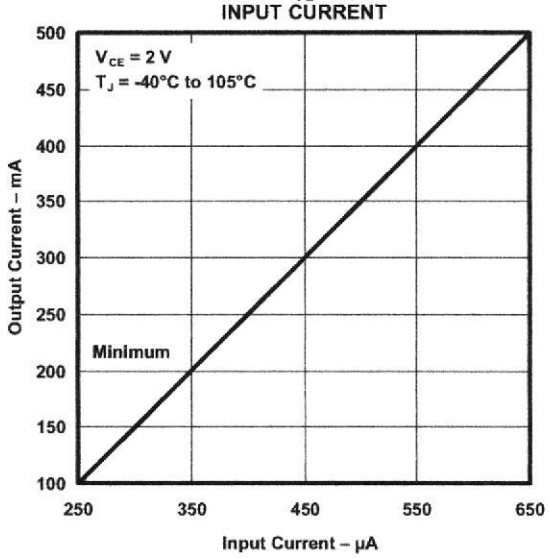


Figure 18.

APPLICATION INFORMATION

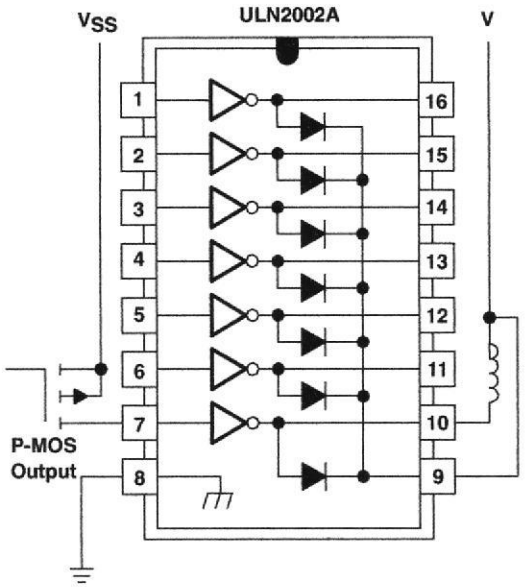


Figure 19. P-MOS to Load

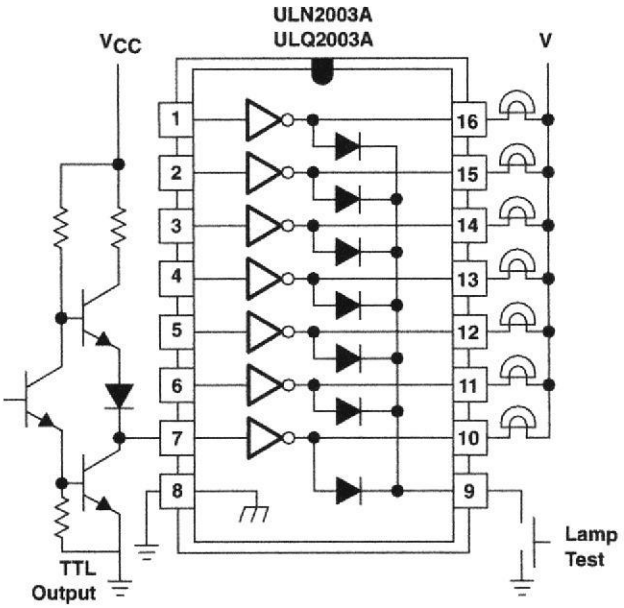


Figure 20. TTL to Load

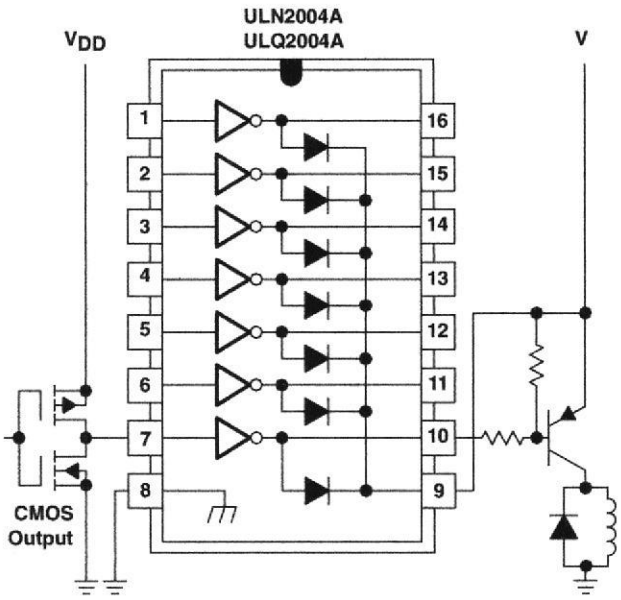


Figure 21. Buffer for Higher Current Loads

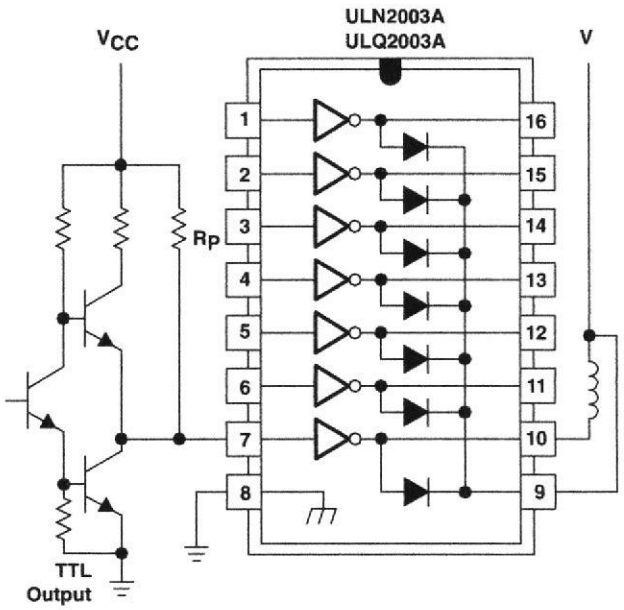


Figure 22. Use of Pullup Resistors to Increase Drive Current

Low Power Digital Temperature Sensor With SMBus™/Two-Wire Serial Interface in SOT563

Check for Samples: TMP102

FEATURES

- **TINY SOT563 PACKAGE**
- **ACCURACY: 0.5°C (–25°C to +85°C)**
- **LOW QUIESCENT CURRENT:**
  10µA Active (max)
  1µA Shutdown (max)
- **SUPPLY RANGE: 1.4V to 3.6V**
- **RESOLUTION: 12 Bits**
- **DIGITAL OUTPUT: Two-Wire Serial Interface**

APPLICATIONS

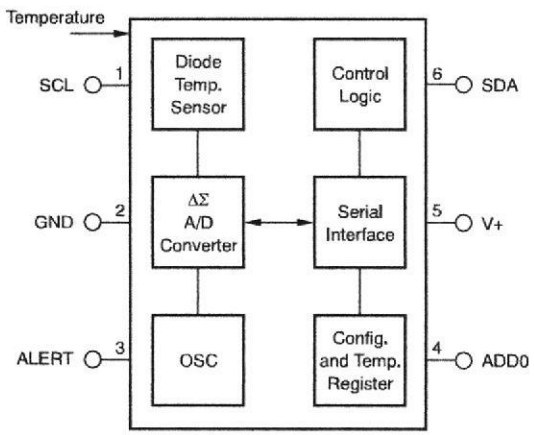
- **PORTABLE AND BATTERY-POWERED APPLICATIONS**
- **POWER-SUPPLY TEMPERATURE MONITORING**
- **COMPUTER PERIPHERAL THERMAL PROTECTION**
- **NOTEBOOK COMPUTERS**
- **BATTERY MANAGEMENT**
- **OFFICE MACHINES**
- **THERMOSTAT CONTROLS**
- **ELECTROMECHANICAL DEVICE TEMPERATURES**
- **GENERAL TEMPERATURE MEASUREMENTS:**
  Industrial Controls
  Test Equipment
  Medical Instrumentations

DESCRIPTION

The TMP102 is a two-wire, serial output temperature sensor available in a tiny SOT563 package. Requiring no external components, the TMP102 is capable of reading temperatures to a resolution of 0.0625°C.

The TMP102 features SMBus and two-wire interface compatibility, and allows up to four devices on one bus. It also features an SMB alert function.

The TMP102 is ideal for extended temperature measurement in a variety of communication, computer, consumer, environmental, industrial, and instrumentation applications. The device is specified for operation over a temperature range of –40°C to +125°C.



This integrated circuit can be damaged by ESD. Texas Instruments recommends that all integrated circuits be handled with appropriate precautions. Failure to observe proper handling and installation procedures can cause damage.

ESD damage can range from subtle performance degradation to complete device failure. Precision integrated circuits may be more susceptible to damage because very small parametric changes could cause the device not to meet its published specifications.

ORDERING INFORMATION(1)

PRODUCT	PACKAGE-LEAD	PACKAGE DESIGNATOR	PACKAGE MARKING
TMP102	SOT563	DRL	CBZ

(1) For the most current package and ordering information, see the Package Option Addendum at the end of this document, or see the TI web site at www.ti.com.

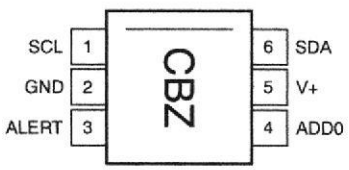
ABSOLUTE MAXIMUM RATINGS(1)

PARAMETER		TMP102	UNIT
Supply Voltage		3.6	V
Input Voltage(2)		–0.5 to +3.6	V
Operating Temperature		–55 to +150	°C
Storage Temperature		–60 to +150	°C
Junction Temperature		+150	°C
ESD Rating	Human Body Model (HBM)	2000	V
	Charged Device Model (CDM)	1000	V
	Machine Model (MM)	200	V

(1) Stresses above these ratings may cause permanent damage. Exposure to absolute maximum conditions for extended periods may degrade device reliability. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those specified is not supported.
(2) Input voltage rating applies to all TMP102 input voltages.

PIN CONFIGURATION

DRL Package
SOT563
Top View



ELECTRICAL CHARACTERISTICS

At T_A = +25°C and V_S = +1.4V to +3.6V, unless otherwise noted.

PARAMETER		CONDITIONS	TMP102 MIN	TMP102 TYP	TMP102 MAX	UNIT
TEMPERATURE INPUT						
Range			–40		+125	°C
Accuracy (Temperature Error)		–25°C to +85°C		0.5	2	°C
		–40°C to +125°C		1	3	°C
vs Supply				0.2	0.5	°C/V
Resolution				0.0625		°C
DIGITAL INPUT/OUTPUT						
Input Logic Levels:						
V_{IH}			0.7 (V+)		3.6	V
V_{IL}			–0.5		0.3 (V+)	V
Input Current	I_{IN}	$0 < V_{IN} < 3.6V$			1	µA
Output Logic Levels:						
V_{OL} SDA		V+ > 2V, I_{OL} = 3mA	0		0.4	V
		V+ < 2V, I_{OL} = 3mA	0		0.2 (V+)	V
V_{OL} ALERT		V+ > 2V, I_{OL} = 3mA	0		0.4	V
		V+ < 2V, I_{OL} = 3mA	0		0.2 (V+)	V
Resolution				12		Bit
Conversion Time				26	35	ms
Conversion Modes		CR1 = 0, CR0 = 0		0.25		Conv/s
		CR1 = 0, CR0 = 1		1		Conv/s
		CR1 = 1, CR0 = 0 (default)		4		Conv/s
		CR1 = 1, CR0 = 1		8		Conv/s
Timeout Time				30	40	ms
POWER SUPPLY						
Operating Supply Range			+1.4		+3.6	V
Quiescent Current	I_Q	Serial Bus Inactive, CR1 = 1, CR0 = 0 (default)		7	10	µA
		Serial Bus Active, SCL Frequency = 400kHz		15		µA
		Serial Bus Active, SCL Frequency = 3.4MHz		85		µA
Shutdown Current	I_{SD}	Serial Bus Inactive		0.5	1	µA
		Serial Bus Active, SCL Frequency = 400kHz		10		µA
		Serial Bus Active, SCL Frequency = 3.4MHz		80		µA
TEMPERATURE RANGE						
Specified Range			–40		+125	°C
Operating Range			–55		+150	°C
Thermal Resistance, SOT563	θ_{JA}			260		°C/W

TYPICAL CHARACTERISTICS

At T_A = +25°C and V+ = 3.3V, unless otherwise noted.

QUIESCENT CURRENT vs TEMPERATURE
(4 Conversions per Second)

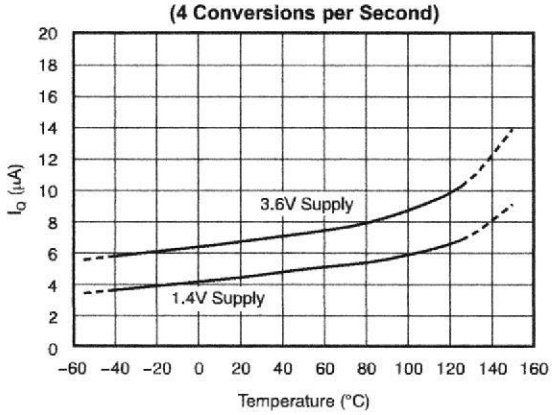


Figure 1.

SHUTDOWN CURRENT vs TEMPERATURE

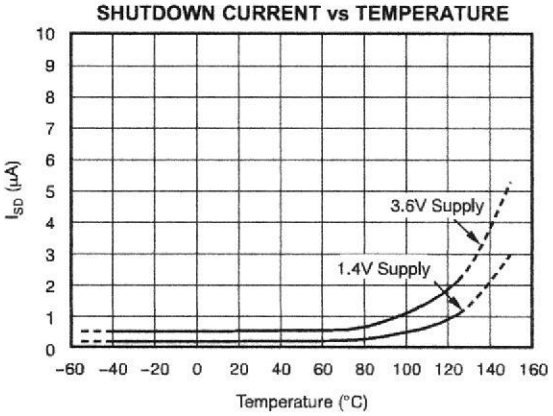


Figure 2.

CONVERSION TIME vs TEMPERATURE

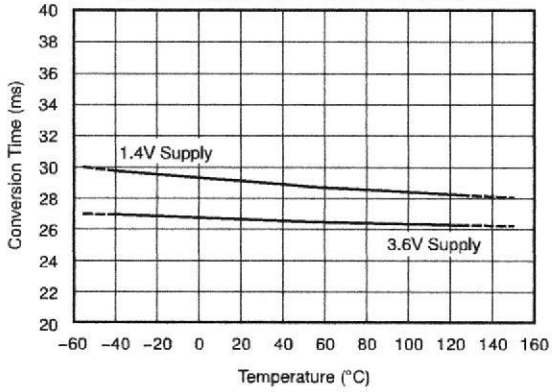


Figure 3.

QUIESCENT CURRENT vs BUS FREQUENCY
(Temperature at 3.3V Supply)

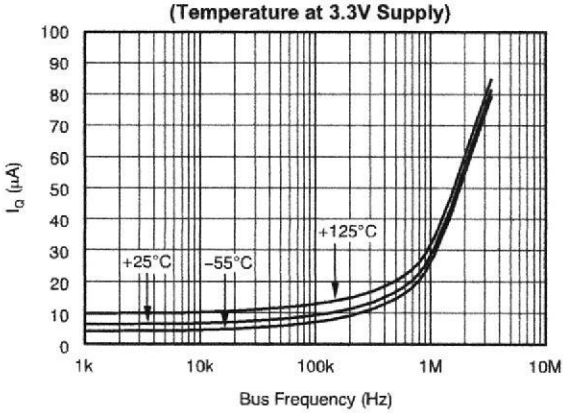


Figure 4.

TEMPERATURE ERROR vs TEMPERATURE

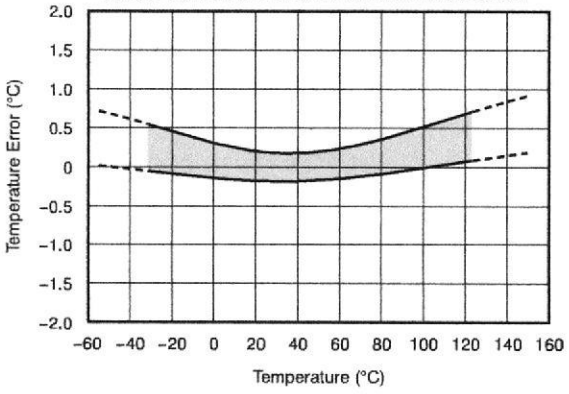


Figure 5.

TEMPERATURE ERROR AT +25°C

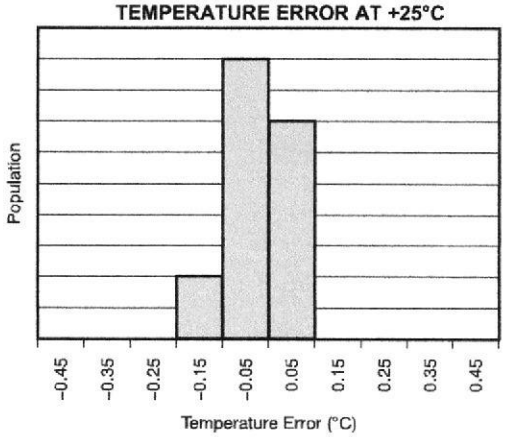


Figure 6.

APPLICATION INFORMATION

The TMP102 is a digital temperature sensor that is optimal for thermal-management and thermal-protection applications. The TMP102 is two-wire- and SMBus interface-compatible, and is specified over a temperature range of –40°C to +125°C.

Pull-up resistors are required on SCL, SDA, and ALERT. A 0.01µF bypass capacitor is recommended, as shown in Figure 7.

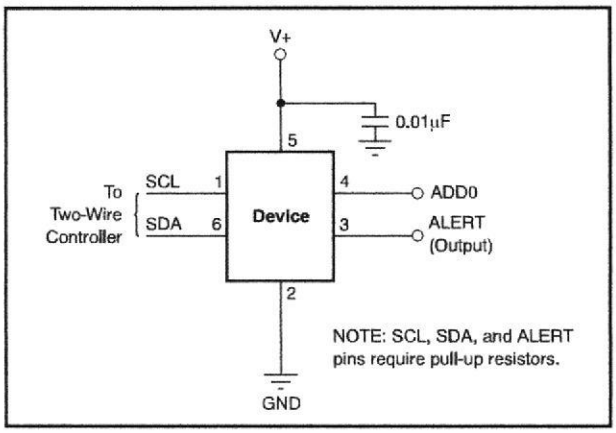


Figure 7. Typical Connections

The temperature sensor in the TMP102 is the chip itself. Thermal paths run through the package leads, as well as the plastic package. The lower thermal resistance of metal causes the leads to provide the primary thermal path.

To maintain accuracy in applications requiring air or surface temperature measurement, care should be taken to isolate the package and leads from ambient air temperature. A thermally-conductive adhesive is helpful in achieving accurate surface temperature measurement.

POINTER REGISTER

Figure 8 shows the internal register structure of the TMP102. The 8-bit Pointer Register of the device is used to address a given data register. The Pointer Register uses the two LSBs (see Table 11) to identify which of the data registers should respond to a read or write command. Table 1 identifies the bits of the Pointer Register byte. During a write command, P2 through P7 must always be '0'. Table 2 describes the pointer address of the registers available in the TMP102. Power-up reset value of P1/P0 is '00'. By default, the TMP102 reads the temperature on power-up.

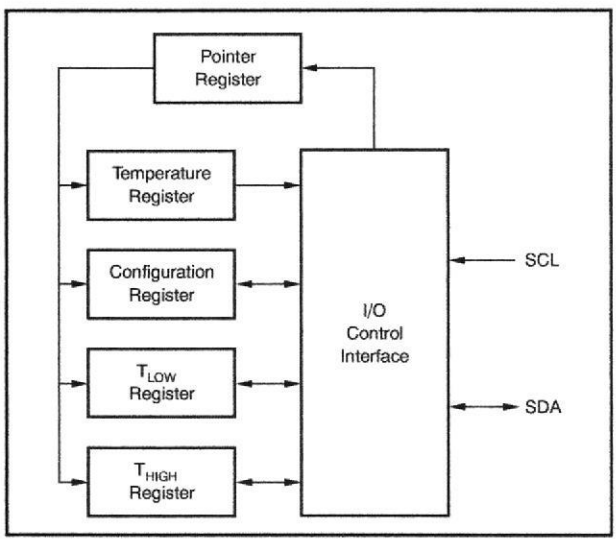


Figure 8. Internal Register Structure

Table 1. Pointer Register Byte

P7	P6	P5	P4	P3	P2	P1	P0
0	0	0	0	0	0	Register Bits	

Table 2. Pointer Addresses

P1	P0	REGISTER
0	0	Temperature Register (Read Only)
0	1	Configuration Register (Read/Write)
1	0	T_{LOW} Register (Read/Write)
1	1	T_{HIGH} Register (Read/Write)

TEMPERATURE REGISTER

The Temperature Register of the TMP102 is configured as a 12-bit, read-only register (Configuration Register EM bit = '0', see the *Extended Mode* section), or as a 13-bit, read-only register (Configuration Register EM bit = '1') that stores the output of the most recent conversion. Two bytes must be read to obtain data, and are described in Table 3 and Table 4. Note that byte 1 is the most significant byte, followed by byte 2, the least significant byte. The first 12 bits (13 bits in Extended mode) are used to indicate temperature. The least significant byte does not have to be read if that information is not needed. The data format for temperature is summarized in Table 5 and Table 6. One LSB equals 0.0625°C. Negative numbers are represented in binary twos complement format. Following power-up or reset, the Temperature Register will read 0°C until the first conversion is complete. Bit D0 of byte 2 indicates Normal mode (EM bit = '0') or Extended mode (EM bit = '1') and can be used to distinguish between the two temperature register data formats. The unused bits in the Temperature Register always read '0'.

Table 3. Byte 1 of Temperature Register(1)

D7	D6	D5	D4	D3	D2	D1	D0
T11	T10	T9	T8	T7	T6	T5	T4
(T12)	(T11)	(T10)	(T9)	(T8)	(T7)	(T6)	(T5)

(1) Extended mode 13-bit configuration shown in parenthesis.

Table 4. Byte 2 of Temperature Register(1)

D7	D6	D5	D4	D3	D2	D1	D0
T3	T2	T1	T0	0	0	0	0
(T4)	(T3)	(T2)	(T1)	(T0)	(0)	(0)	(1)

(1) Extended mode 13-bit configuration shown in parenthesis.

Table 5. 12-Bit Temperature Data Format(1)

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	HEX
128	0111 1111 1111	7FF
127.9375	0111 1111 1111	7FF
100	0110 0100 0000	640
80	0101 0000 0000	500
75	0100 1011 0000	4B0
50	0011 0010 0000	320
25	0001 1001 0000	190
0.25	0000 0000 0100	004
0	0000 0000 0000	000
–0.25	1111 1111 1100	FFC
–25	1110 0111 0000	E70
–55	1100 1001 0000	C90

(1) The resolution for the Temp ADC in Internal Temperature mode is 0.0625°C/count.

For positive temperatures (for example, +50°C):

Twos complement is not performed on positive numbers. Therefore, simply convert the number to binary code with the 12-bit, left-justified format, and MSB = 0 to denote a positive sign.

Example: (+50°C)/(0.0625°C/count) = 800 = 320h = 0011 0010 0000

For negative temperatures (for example, –25°C):

Generate the twos complement of a negative number by complementing the absolute value binary number and adding 1. Denote a negative number with MSB = 1.

Example: (|–25°C|)/(0.0625°C/count) = 400 = 190h = 0001 1001 0000

Twos complement format: 1110 0110 1111 + 1 = 1110 0111 0000

Table 6. 13-Bit Temperature Data Format

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	HEX
150	0 1001 0110 0000	0960
128	0 1000 0000 0000	0800
127.9375	0 0111 1111 1111	07FF
100	0 0110 0100 0000	0640
80	0 0101 0000 0000	0500
75	0 0100 1011 0000	04B0
50	0 0011 0010 0000	0320
25	0 0001 1001 0000	0190
0.25	0 0000 0000 0100	0004
0	0 0000 0000 0000	0000
–0.25	1 1111 1111 1100	1FFC
–25	1 1110 0111 0000	1E70
–55	1 1100 1001 0000	1C90

CONFIGURATION REGISTER

The Configuration Register is a 16-bit read/write register used to store bits that control the operational modes of the temperature sensor. Read/write operations are performed MSB first. The format and power-up/reset value of the Configuration Register is shown in Table 7. For compatibility, the first byte corresponds to the Configuration Register in the TMP75 and TMP275. All registers are updated byte by byte.

Table 7. Configuration and Power-Up/Reset Format

BYTE	D7	D6	D5	D4	D3	D2	D1	D0
1	OS	R1	R0	F1	F0	POL	TM	SD
	0	1	1	0	0	0	0	0
2	CR1	CR0	AL	EM	0	0	0	0
	1	0	1	0	0	0	0	0

EXTENDED MODE (EM)

The Extended mode bit configures the device for Normal mode operation (EM = 0) or Extended mode operation (EM = 1). In Normal mode, the Temperature Register and high- and low-limit registers use a 12-bit data format. Normal mode is used to make the TMP102 compatible with the TMP75.

Extended mode (EM = 1) allows measurement of temperatures above +128°C by configuring the Temperature Register, and high- and low-limit registers, for 13-bit data format.

ALERT (AL Bit)

The AL bit is a read-only function. Reading the AL bit will provide information about the comparator mode status. The state of the POL bit inverts the polarity of data returned from the AL bit. For POL = 0, the AL bit will read as '1' until the temperature equals or exceeds T_{HIGH} for the programmed number of consecutive faults, causing the AL bit to read as '0'. The AL bit will continue to read as '0' until the temperature falls below T_{LOW} for the programmed number of consecutive faults, when it will again read as '1'. The status of the TM bit does not affect the status of the AL bit.

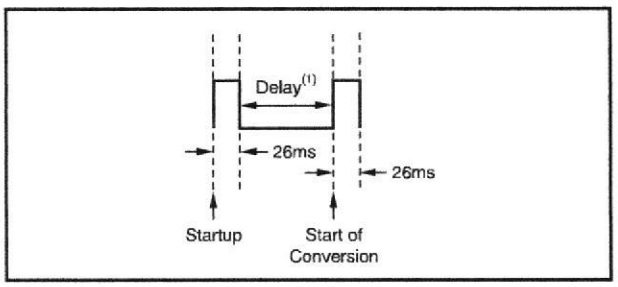
CONVERSION RATE

The conversion rate bits, CR1 and CR0, configure the TMP102 for conversion rates of 8Hz, 4Hz, 1Hz, or 0.25Hz. The default rate is 4Hz. The TMP102 has a typical conversion time of 26ms. To achieve different conversion rates, the TMP102 makes a conversion and after that powers down and waits for the appropriate delay set by CR1 and CR0. Table 8 shows the settings for CR1 and CR0.

Table 8. Conversion Rate Settings

CR1	CR0	CONVERSION RATE
0	0	0.25Hz
0	1	1Hz
1	0	4Hz (default)
1	1	8Hz

After power-up or general-call reset, the TMP102 immediately starts a conversion, as shown in Figure 9. The first result is available after 26ms (typical). The active quiescent current during conversion is 40µA (typical at +27°C). The quiescent current during delay is 2.2µA (typical at +27°C).



(1) Delay is set by CR1 and CR0.

Figure 9. Conversion Start

SHUTDOWN MODE (SD)

The Shutdown mode bit saves maximum power by shutting down all device circuitry other than the serial interface, reducing current consumption to typically less than 0.5µA. Shutdown mode is enabled when the SD bit is '1'; the device shuts down when current conversion is completed. When SD is equal to '0', the device maintains a continuous conversion state.

THERMOSTAT MODE (TM)

The Thermostat mode bit indicates to the device whether to operate in Comparator mode (TM = 0) or Interrupt mode (TM = 1). For more information on comparator and interrupt modes, see the *High- and Low-Limit Registers* section.

POLARITY (POL)

The Polarity bit allows the user to adjust the polarity of the ALERT pin output. If POL = 0, the ALERT pin will be active low, as shown in Figure 10. For POL = 1, the ALERT pin will be active high, and the state of the ALERT pin is inverted.

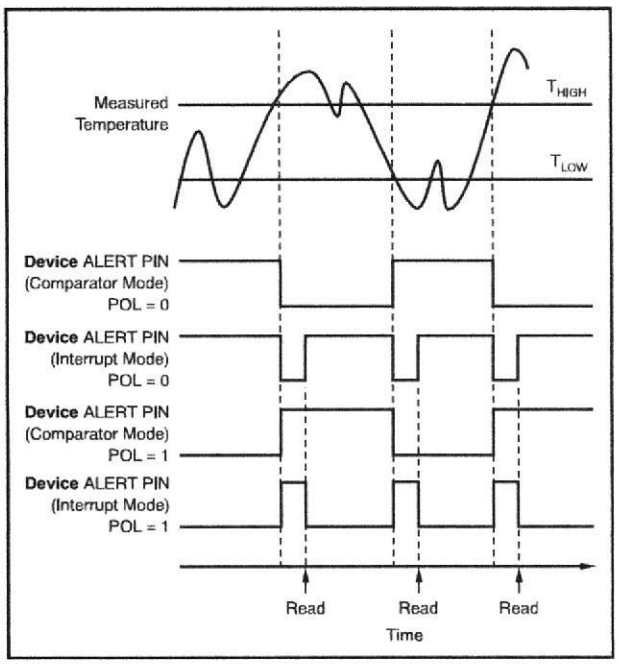


Figure 10. Output Transfer Function Diagrams

FAULT QUEUE (F1/F0)

A fault condition exists when the measured temperature exceeds the user-defined limits set in the T_{HIGH} and T_{LOW} registers. Additionally, the number of fault conditions required to generate an alert may be programmed using the fault queue. The fault queue is provided to prevent a false alert as a result of environmental noise. The fault queue requires consecutive fault measurements in order to trigger the alert function. Table 9 defines the number of measured faults that may be programmed to trigger an alert condition in the device. For T_{HIGH} and T_{LOW} register format and byte order, see the *High- and Low-Limit Registers* section.

Table 9. TMP102 Fault Settings

F1	F0	CONSECUTIVE FAULTS
0	0	1
0	1	2
1	0	4
1	1	6

CONVERTER RESOLUTION (R1/R0)

R1/R0 are read-only bits. The TMP102 converter resolution is set on start up to '11'. This sets the temperature register to a 12 bit-resolution.

ONE-SHOT/CONVERSION READY (OS)

The TMP102 features a One-Shot Temperature Measurement mode. When the device is in Shutdown mode, writing a '1' to the OS bit starts a single temperature conversion. During the conversion, the OS bit reads '0'. The device returns to the shutdown state at the completion of the single conversion. After the conversion, the OS bit reads '1'. This feature is useful for reducing power consumption in the TMP102 when continuous temperature monitoring is not required.

As a result of the short conversion time, the TMP102 can achieve a higher conversion rate. A single conversion typically takes 26ms and a read can take place in less than 20µs. When using One-Shot mode, 30 or more conversions per second are possible.

HIGH- AND LOW-LIMIT REGISTERS

In Comparator mode (TM = 0), the ALERT pin becomes active when the temperature equals or exceeds the value in T_{HIGH} and generates a consecutive number of faults according to fault bits F1 and F0. The ALERT pin remains active until the temperature falls below the indicated T_{LOW} value for the same number of faults.

In Interrupt mode (TM = 1), the ALERT pin becomes active when the temperature equals or exceeds the value in T_{HIGH} for a consecutive number of fault conditions (as shown in Table 9). The ALERT pin remains active until a read operation of any register occurs, or the device successfully responds to the SMBus Alert Response address. The ALERT pin will also be cleared if the device is placed in Shutdown mode. Once the ALERT pin is cleared, it becomes active again only when temperature falls below T_{LOW}, and remains active until cleared by a read operation of any register or a successful response to the SMBus Alert Response address. Once the ALERT pin is cleared, the above cycle repeats, with the ALERT pin becoming active when the temperature equals or exceeds T_{HIGH}. The ALERT pin can also be cleared by resetting the device with the General Call Reset command. This action also clears the state of the internal registers in the device, returning the device to Comparator mode (TM = 0).

Both operational modes are represented in Figure 10. Table 10 and Table 11 describe the format for the T_{HIGH} and T_{LOW} registers. Note that the most significant byte is sent first, followed by the least significant byte. Power-up reset values for T_{HIGH} and T_{LOW} are: T_{HIGH} = +80°C and T_{LOW} = +75°C. The format of the data for T_{HIGH} and T_{LOW} is the same as for the Temperature Register.

Table 10. Bytes 1 and 2 of T_{HIGH} Register(1)

BYTE	D7	D6	D5	D4	D3	D2	D1	D0
1	H11	H10	H9	H8	H7	H6	H5	H4
	(H12)	(H11)	(H10)	(H9)	(H8)	(H7)	(H6)	(H5)
BYTE	D7	D6	D5	D4	D3	D2	D1	D0
2	H3	H2	H1	H0	0	0	0	0
	(H4)	(H3)	(H2)	(H1)	(H0)	(0)	(0)	(0)

(1) Extended mode 13-bit configuration shown in parenthesis.

Table 11. Bytes 1 and 2 of T_{LOW} Register(1)

BYTE	D7	D6	D5	D4	D3	D2	D1	D0
1	L11	L10	L9	L8	L7	L6	L5	L4
	(L12)	(L11)	(L10)	(L9)	(L8)	(L7)	(L6)	(L5)
BYTE	D7	D6	D5	D4	D3	D2	D1	D0
2	L3	L2	L1	L0	0	0	0	0
	(L4)	(L3)	(L2)	(L1)	(L0)	(0)	(0)	(0)

(1) Extended mode 13-bit configuration shown in parenthesis.

BUS OVERVIEW

The device that initiates the transfer is called a *master*, and the devices controlled by the master are *slaves*. The bus must be controlled by a master device that generates the serial clock (SCL), controls the bus access, and generates the START and STOP conditions.

To address a specific device, a START condition is initiated, indicated by pulling the data-line (SDA) from a high to low logic level while SCL is high. All slaves on the bus shift in the slave address byte on the rising edge of the clock, with the last bit indicating whether a read or write operation is intended. During the ninth clock pulse, the slave being addressed responds to the master by generating an Acknowledge and pulling SDA low.

Data transfer is then initiated and sent over eight clock pulses followed by an Acknowledge Bit. During data transfer SDA must remain stable while SCL is high, because any change in SDA while SCL is high will be interpreted as a START or STOP signal.

Once all data have been transferred, the master generates a STOP condition indicated by pulling SDA from low to high, while SCL is high.

SERIAL INTERFACE

The TMP102 operates as a slave device only on the two-wire bus and SMBus. Connections to the bus are made via the open-drain I/O lines SDA and SCL. The SDA and SCL pins feature integrated spike suppression filters and Schmitt triggers to minimize the effects of input spikes and bus noise. The TMP102 supports the transmission protocol for both fast (1kHz to 400kHz) and high-speed (1kHz to 3.4MHz) modes. All data bytes are transmitted MSB first.

SERIAL BUS ADDRESS

To communicate with the TMP102, the master must first address slave devices via a slave address byte. The slave address byte consists of seven address bits, and a direction bit indicating the intent of executing a read or write operation.

The TMP102 features an address pin to allow up to four devices to be addressed on a single bus. Table 12 describes the pin logic levels used to properly connect up to four devices.

Table 12. Address Pin and Slave Addresses

DEVICE TWO-WIRE ADDRESS	A0 PIN CONNECTION
1001000	Ground
1001001	V+
1001010	SDA
1001011	SCL

WRITING/READING OPERATION

Accessing a particular register on the TMP102 is accomplished by writing the appropriate value to the Pointer Register. The value for the Pointer Register is the first byte transferred after the slave address byte with the R/$\overline{W}$ bit low. Every write operation to the TMP102 requires a value for the Pointer Register (see Figure 13).

When reading from the TMP102, the last value stored in the Pointer Register by a write operation is used to determine which register is read by a read operation. To change the register pointer for a read operation, a new value must be written to the Pointer Register. This action is accomplished by issuing a slave address byte with the R/$\overline{W}$ bit low, followed by the Pointer Register byte. No additional data are required. The master can then generate a START condition and send the slave address byte with the R/$\overline{W}$ bit high to initiate the read command. See Figure 14 for details of this sequence. If repeated reads from the same register are desired, it is not necessary to continually send the Pointer Register bytes, because the TMP102 remembers the Pointer Register value until it is changed by the next write operation.

Note that register bytes are sent with the most significant byte first, followed by the least significant byte.

SLAVE MODE OPERATIONS

The TMP102 can operate as a slave receiver or slave transmitter. As a slave device, the TMP102 never drives the SCL line.

Slave Receiver Mode:

The first byte transmitted by the master is the slave address, with the R/$\overline{W}$ bit low. The TMP102 then acknowledges reception of a valid address. The next byte transmitted by the master is the Pointer Register. The TMP102 then acknowledges reception of the Pointer Register byte. The next byte or bytes are written to the register addressed by the Pointer Register. The TMP102 acknowledges reception of each data byte. The master can terminate data transfer by generating a START or STOP condition.

Slave Transmitter Mode:

The first byte transmitted by the master is the slave address, with the R/$\overline{W}$ bit high. The slave acknowledges reception of a valid slave address. The next byte is transmitted by the slave and is the most significant byte of the register indicated by the Pointer Register. The master acknowledges reception of the data byte. The next byte transmitted by the slave is the least significant byte. The master acknowledges reception of the data byte. The master can terminate data transfer by generating a *Not-Acknowledge* on reception of any data byte, or generating a START or STOP condition.

SMBus ALERT FUNCTION

The TMP102 supports the SMBus Alert function. When the TMP102 operates in Interrupt mode (TM = '1'), the ALERT pin may be connected as an SMBus Alert signal. When a master senses that an ALERT condition is present on the ALERT line, the master sends an SMBus Alert command (00011001) to the bus. If the ALERT pin is active, the device acknowledges the SMBus Alert command and responds by returning its slave address on the SDA line. The eighth bit (LSB) of the slave address byte indicates if the ALERT condition was caused by the temperature exceeding T_{HIGH} or falling below T_{LOW}. For POL = '0', this bit is low if the temperature is greater than or equal to T_{HIGH}; this bit is high if the temperature is less than T_{LOW}. The polarity of this bit is inverted if POL = '1'. Refer to Figure 15 for details of this sequence.

If multiple devices on the bus respond to the SMBus Alert command, arbitration during the slave address portion of the SMBus Alert command determines which device will clear its ALERT status. The device with the lowest two-wire address wins the arbitration. If the TMP102 wins the arbitration, its ALERT pin becomes inactive at the completion of the SMBus Alert command. If the TMP102 loses the arbitration, its ALERT pin remains active.

GENERAL CALL

The TMP102 responds to a two-wire General Call address (0000000) if the eighth bit is '0'. The device acknowledges the General Call address and responds to commands in the second byte. If the second byte is 00000110, the TMP102 internal registers are reset to power-up values. The TMP102 does not support the General Address acquire command.

HIGH-SPEED (Hs) MODE

In order for the two-wire bus to operate at frequencies above 400kHz, the master device must issue an Hs-mode master code (00001xxx) as the first byte after a START condition to switch the bus to high-speed operation. The TMP102 does not acknowledge this byte, but switches its input filters on SDA and SCL and its output filters on SDA to operate in Hs-mode, allowing transfers at up to 3.4MHz. After the Hs-mode master code has been issued, the master transmits a two-wire slave address to initiate a data transfer operation. The bus continues to operate in Hs-mode until a STOP condition occurs on the bus. Upon receiving the STOP condition, the TMP102 switches the input and output filters back to fast-mode operation.

TIMEOUT FUNCTION

The TMP102 resets the serial interface if SCL is held low for 30ms (typ). The TMP102 releases the bus if it is pulled low and waits for a START condition. To avoid activating the timeout function, it is necessary to maintain a communication speed of at least 1kHz for SCL operating frequency.

NOISE

The TMP102 is a very low-power device and generates very low noise on the supply bus. Applying an RC filter to the V+ pin of the TMP102 can further reduce any noise the TMP102 might propagate to other components. R_F in Figure 11 should be less than 5kΩ and C_F should be greater than 10nF.

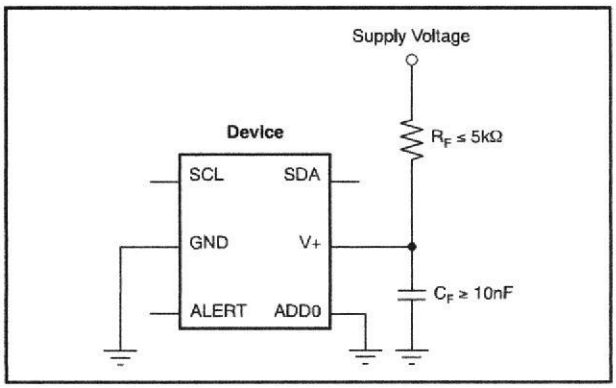


Figure 11. Noise Reduction

TIMING DIAGRAMS

The TMP102 is two-wire and SMBus compatible. Figure 12 to Figure 15 describe the various operations on the TMP102. Parameters for Figure 12 are defined in Table 13. Bus definitions are:

Bus Idle: Both SDA and SCL lines remain high.

Start Data Transfer: A change in the state of the SDA line, from high to low, while the SCL line is high, defines a START condition. Each data transfer is initiated with a START condition.

Stop Data Transfer: A change in the state of the SDA line from low to high while the SCL line is high defines a STOP condition. Each data transfer is terminated with a repeated START or STOP condition.

Data Transfer: The number of data bytes transferred between a START and a STOP condition is not limited and is determined by the master device. It is also possible to use the TMP102 for single byte updates. To update only the MS byte, terminate the communication by issuing a START or STOP communication on the bus.

Acknowledge: Each receiving device, when addressed, is obliged to generate an Acknowledge bit. A device that acknowledges must pull down the SDA line during the Acknowledge clock pulse in such a way that the SDA line is stable low during the high period of the Acknowledge clock pulse. Setup and hold times must be taken into account. On a master receive, the termination of the data transfer can be signaled by the master generating a *Not-Acknowledge* ('1') on the last byte that has been transmitted by the slave.

Table 13. Timing Diagram Definitions

PARAMETER	TEST CONDITIONS	FAST MODE		HIGH-SPEED MODE		UNIT
		MIN	MAX	MIN	MAX	
$f_{(SCL)}$	SCL Operating Frequency, V_S > 1.7V	0.001	0.4	0.001	3.4	MHz
$f_{(SCL)}$	SCL Operating Frequency, V_S < 1.7V	0.001	0.4	0.001	2.75	MHz
$t_{(BUF)}$	Bus Free Time Between STOP and START Condition	600		160		ns
$t_{(HDSTA)}$	Hold time after repeated START condition. After this period, the first clock is generated.	100		100		ns
$t_{(SUSTA)}$	Repeated START Condition Setup Time	100		100		ns
$t_{(SUSTO)}$	STOP Condition Setup Time	100		100		ns
$t_{(HDDAT)}$	Data Hold Time	100		10		ns
$t_{(SUDAT)}$	Data Setup Time	100		10		ns
$t_{(LOW)}$	SCL Clock Low Period, V_S > 1.7V	1300		160		ns
$t_{(LOW)}$	SCL Clock Low Period, V_S < 1.7V	1300		200		ns
$t_{(HIGH)}$	SCL Clock High Period	600		60		ns
t_F	Clock/Data Fall Time		300			ns
t_R	Clock/Data Rise Time		300		160	ns
t_R	Clock/Data Rise Time for SCLK ≤ 100kHz		1000			ns

TWO-WIRE TIMING DIAGRAMS

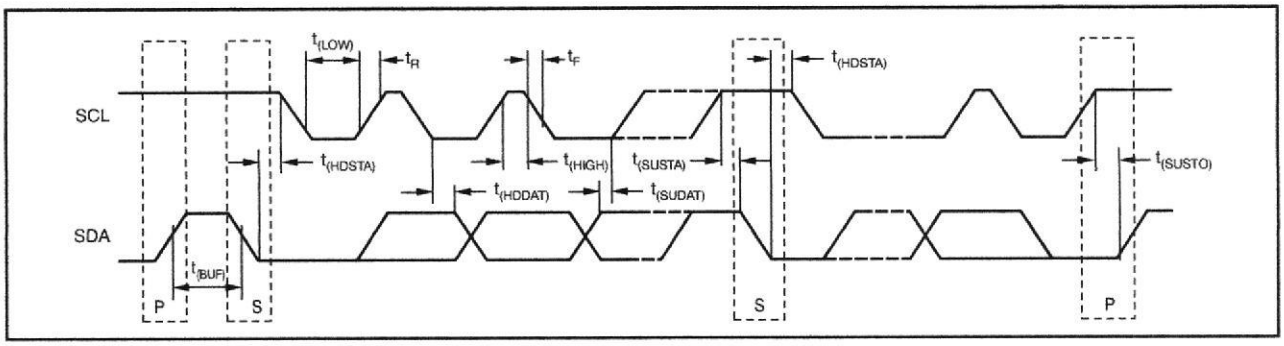


Figure 12. Two-Wire Timing Diagram

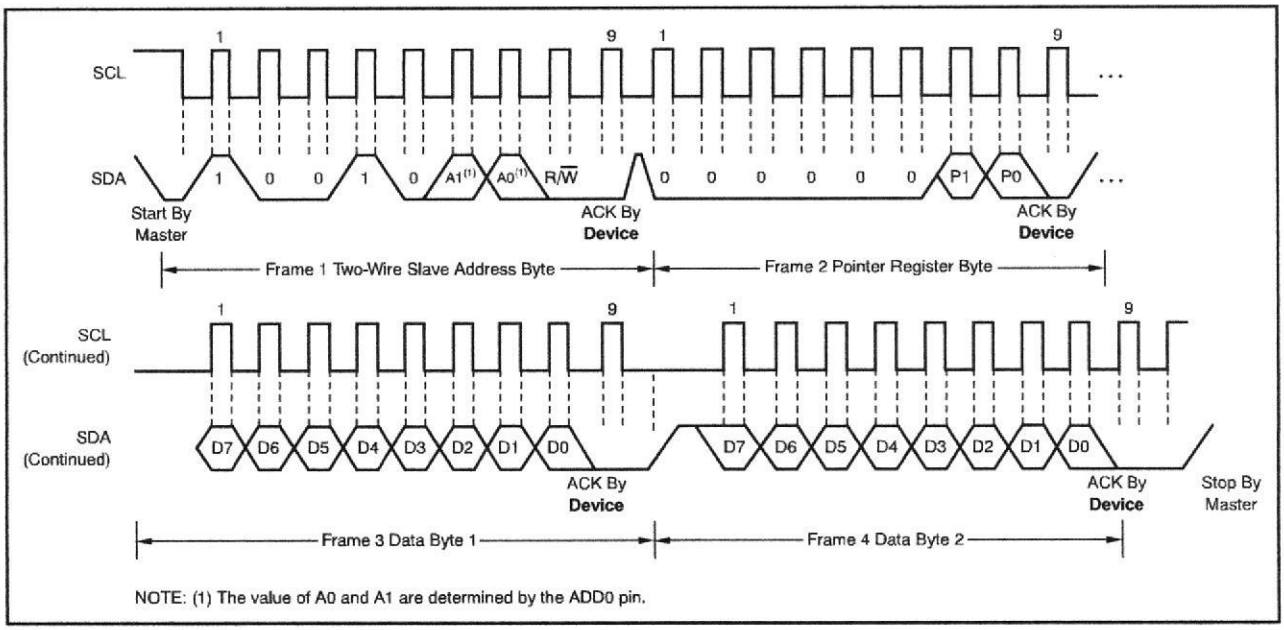


Figure 13. Two-Wire Timing Diagram for Write Word Format

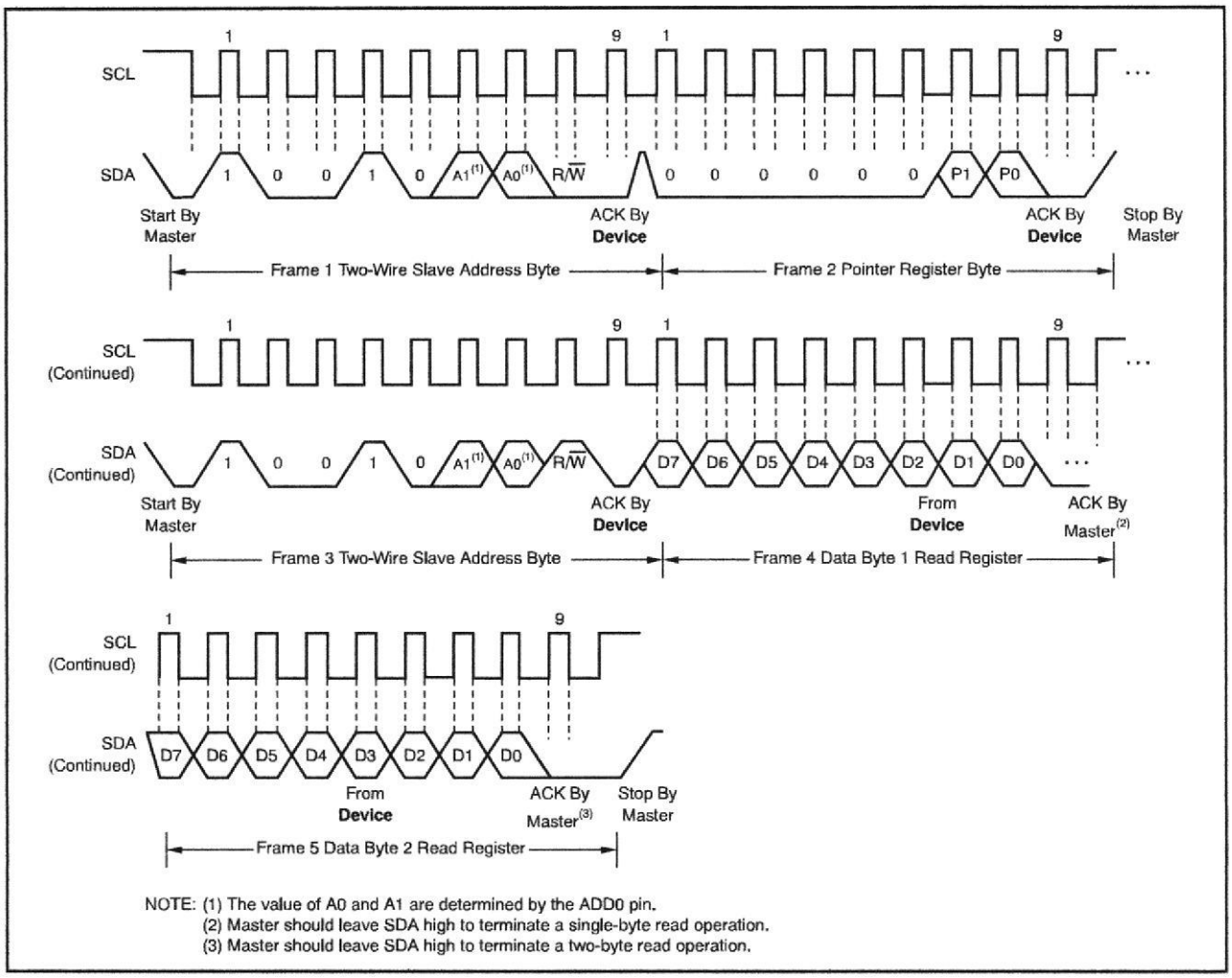


Figure 14. Two-Wire Timing Diagram for Read Word Format

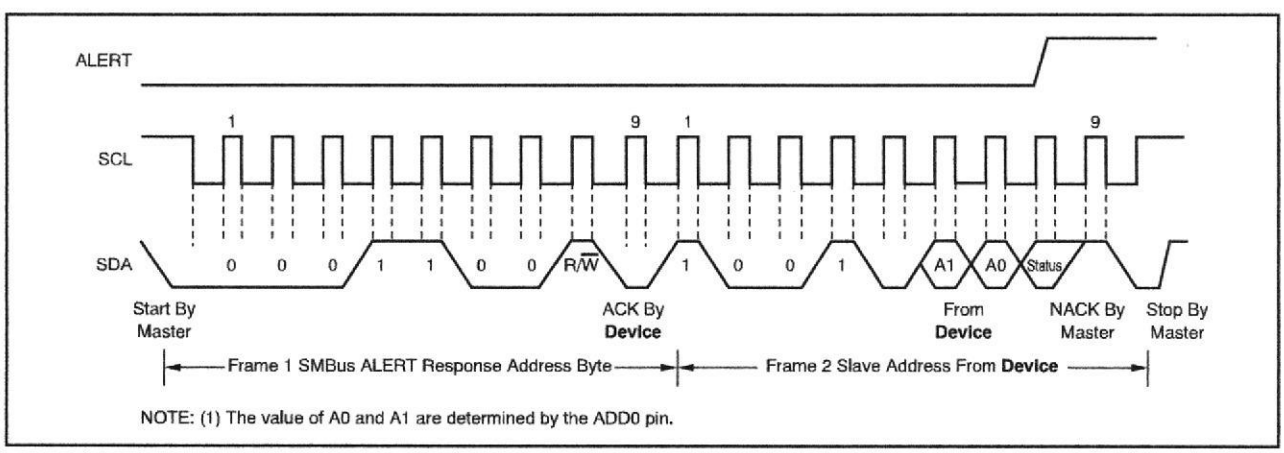
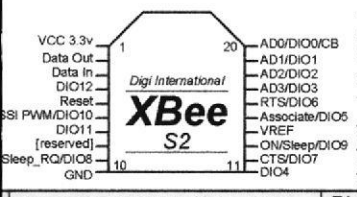


Figure 15. Timing Diagram for SMBus ALERT

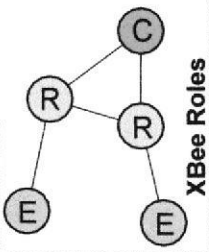


XBee S2 Quick Reference Guide

IEEE 802.15.4 = Zigbee Protocol. XBee is a microcontroller made by digi which uses the Zigbee protocol.
The XBee uses 3.3V and has a smaller pin spacing than most breadboards/proto boards. Because of this, it is often useful to purchase a kit to interface the XBee with a breadboard.

Sept/2012 http://tunnelsup.com

Operating Voltage: 2.1 – 3.6V	Digital I/O pins: 11	RF Data Rate: 250kbps
Operating Current: 40mA@3.3V	Analog input pins: 4	Throughput speed: 35kbps
Indoor range: 40 Meters	Mesh routable	Frequency: ISM 2.4GHz
Line of sight range: 120 Meters	Self Healing network	OK Temp: -40 to 85C
Max Analog Pin Reading: 1.2V	Firmware: ZB ZigBee	



XBee Roles

- Coordinator – 1 required in every network
  In charge of setting up the network
  Can never sleep
- Router – multiple may exist
  Can relay signals from other routers/EPs
  Can never sleep
- End Point – multiple may exist
  Cannot relay signals
  Can sleep to save power

Modes

Transparent – Communication through the XBee. If data is not generated from the XBee itself then both XBees should be set to AT.
Command – Communication to the XBee. If one XBee is sensing data, that XBee should be in AT mode while the receiving one should be in API mode.

Setup

Connect the XBee to a TTL Serial FTDI adapter – OR – Arduino hack: Connect RX to RX, TX to TX, RESET to ground to bypass the Arduino entirely and get serial to XBee.
Use the free X-CTU software to configure the XBee.
Baud: 9600 – FC: Hardware – Data Bits: 8 – Parity: None – Stop Bits: 1

Settings

PAN ID – The network to communicate over. If 0, the XBee will join any.
DH/DL – Destination Serial number. Used to send to a specific XBee's Serial. Set to 0 to send to just the Coorindator. Set to 0x0000000000FFFF to broadcast.
JV – Router/EP should be set to 1 so it rejoins the network on startup

Settings

For pin settings to work, receiver XBee must be in API mode
D0 – Set pin 0 to start sensing
IR – Collect data on sensing pins every XX millisecs

Arduino Connectivity:
Arduino TX connects to XBee RX (Data in)
Arduino RX connects to XBee TX (Data out)

Arduino Integration:
Data sent to Serial.print() will go out TX port of Arduino which is then connected to the RX port of XBee. If XBee is in AT mode it will transmit it wirelessly. Data received from XBee will be sent to the Serial.

Arduino Example: Read an analog value using API

```
// Remote XBee: AT, Base XBee: API
if (Serial.available() >= 21) {   // Make sure the frame is all there
 if (Serial.read() == 0x7E) {     // 7E is the start byte
  for (int i = 1; i<19; i++) {    // Skip ahead to the analog data
    byte discardByte = Serial.read();
  }
  int analogMSB = Serial.read(); // Read the first analog byte data
  int analogLSB = Serial.read();  // Read the second byte
  int analogReading = analogLSB + (analogMSB * 256);
 }
}
```

Byte	Example	Description
0	0x7e	Start byte – Indicates beginning of data frame
1	0x00	Length – Number of bytes (ChecksumByte# – 1 – 2)
2	0x10	
3	0x17	Frame type - 0x17 means this is a AT command Request
4	0x52	Frame ID – Command sequence number
5	0x00	64-bit Destination Address (Serial Number)
6	0x13	MSB is byte 5, LSB is byte 12
7	0xA2	
8	0x00	0x0000000000000000 = Coordinator
9	0x40	0x000000000000FFFF = Broadcast
10	0x77	
11	0x9C	
12	0x49	
13	0xFF	Destination Network Address
14	0xFE	(Set to 0xFFFE to send a broadcast)
15	0x02	Remote command options (set to 0x02 to apply changes)
16	0x44 (D)	AT Command Name (Two ASCII characters)
17	0x02 (2)	
18	0x04	Command Parameter (queries if not present)
19	0XF5	Checksum

Arduino Example: Change the pin setting on a remote Xbee

```
// Remote XBee: AT, Base XBee: API
Serial.write(0x7E);  // Sync up the start byte
Serial.write((byte)0x0);   // Length MSB (always 0)
Serial.write(0x10);  // Length LSB
Serial.write(0x17);  // 0x17 is the frame ID for sending an AT command
Serial.write((byte)0x0);   // Frame ID (no reply needed)
Serial.write((byte)00);    // Send the 64 bit destination address
Serial.write((byte)00);    // (Sending 0x000000000000FFFF (broadcast))
Serial.write((byte)00);
Serial.write((byte)00);
Serial.write((byte)00);
Serial.write((byte)00);
Serial.write(0xFF);
Serial.write(0xFF);
Serial.write(0xFF); // Destination Network
Serial.write(0xFE); // (Set to 0xFFFE if unknown)
Serial.write(0x02); // Set to 0x02 to apply these changes
Serial.write('D');   // AT Command: D1
Serial.write('1');
Serial.write(0x05); // Set D1 to be 5 (Digital Out HIGH)
long chexsum = 0x17 + 0xFF + 0xFF +0xFF + 0xFE + 0x02 + 'D' + '1' + 0x05;
Serial.write( 0xFF – (chexsum & 0xFF)); // Checksum
```

Byte	Example	Description
0	0x7e	Start byte – Indicates beginning of data frame
1	0x00	Length – Number of bytes (ChecksumByte# – 1 – 2)
2	0x14	
3	0x92	Frame type - 0x92 indicates this will be a data sample
4	0x00	64-bit Source Address (Serial Number)
5	0x13	MSB is byte 4, LSB is byte 11
6	0xA2	
7	0x00	
8	0x40	
9	0x77	
10	0x9C	
11	0x49	
12	0x36	Source Network Address – 16 Bit
13	0x6A	
14	0x01	Receive Opts. 01=Packet Acknowledged. 02=Broadcast packet
15	0x01	Number of sample sets. Always set to 1 due to XBEE limitations
16	0x00	Digital Channel Mask – Indicates which pins are set to DIO
17	0x20	
18	0x01	Analog Channel Mask – Indicates which pins are set to ADC
19	0x00	Digital Sample Data (if any) – Reads the same as Digital Mask
20	0x14	
21	0x04	Analog Sample data (if any)
22	0x25	There will be two bytes here for every pin set for ADC
23	0xF5	Checksum(0xFF - the 8 bit sum of the bytes from byte 3 to this byte)

Sleep Mode

Endpoints can sleep to save power. An endpoint that only wakes up every 5 minutes to send data may only be awake for 6 seconds a day.
SM – 4 = Cyclic sleep
SP – Sleep time (up to 28 secs)
SN – Number of sleep cycles
ST – Time awake

Pin I/O Options

0 – Disabled
1 – N/A
2 – ADC
3 – Digital IN
4 – Digital OUT, LOW
5 – Digital OUT, HIGH

Digital Ch Mask

First Byte
n/a n/a n/a D12 D11 D10 n/a n/a
Second Byte
D7 D6 D5 D4 D3 D2 D1 D0
Example:
0x00 0x13 = 0000 0000 0000 1101
Pins D3, D2 and D0

Analog Ch Mask

(volt) n/a n/a n/a A3 A2 A1 A0
Example:
0x05 = 0000 0101 = Pin A2 and A0

BIBLIOGRAFÍA

[1] **Arduino**, Que es Arduino,

http://arduino.cc/en/pmwiki.php?n=Guide/Introduction, 12/05/2014

[2] **Arduino**, Historia de Arduino, http://arduinodhtics.weebly.com/historia.html, 12/05/2014

[3] **Mouser**, Microcontrolador Atmega2560, http://www.mouser.mx/ProductDetail /Atmel/ ATmega2560-16AU/?qs=aqrrBurbvGciYmlAokFS0w==, 15/05/2014

[4] **Arduino**, Microcontrolador Atmega2560 http://arduino.cc/en/Main/arduino BoardMega2560, 15/05/2014

[5] **Wiring.org**, Programación de Atmega2560 http://wiring.org.co/, 22/05/2014

[6] **Arduino**, Estructura básica para programas Atmega2560 http://playground.arduino.cc/ArduinoNotebookTraduccion/Structure, 24/05/2014

[7] **Arduino**, Introducción a Implementación con Arduino, http://arduino.cc/es/Guide/Introduction, 24/05/2014

[8] **Scribd**, Lcd 16x2, http://es.scribd.com/doc/44252680/LCD-16X2, 03/06/2014

[9] **Todoelectrodo**, Pines de Lcd 16x2, http://todoelectrodo.blogspot.com/2013/02/lcd-16x2.html, 03/06/2014

[10] **Programarpicenc**, Taclado Matricial, http://www.programarpicenc.com/libro/cap08-teclado-matricial-4x4-microcontroladores-pic.html, 04/06/2014

[11] **eHow**, Definicion de Motor, http://www.ehowenespanol.com/definicion-motor-corriente-directa-sobre_55810/, 20/06/2014

[12] **Robots**, Motor DC, http://robots-argentina.com.ar/MotorCC_L293D.htm, 03/06/2014

[13] Tienda Robotica, Servo Motor, http://tienda.tdrobotica.co/categoria/81, 07/07/2014

[14] **Docentes.unal.edu.co**, Motores de Paso, http://www.docentes.unal.edu.co/hfvelascop/docs/CLASES/DIGITALES2/LABORATORIO/Motor%20Paso%20a%20Paso.pdf, 12/07/2014

[15] **Robocraf**, Funcionamiento de Servo Motor http://robocraft.ru/files/datasheet/28BYJ-48.pdf, 12/07/2014

[16] **Datasheetcatalog**, Integrado ULN2003, http://pdf.datasheetcatalog.com/datasheet2/f/0c6x6a46ig46qlxf3j2qsaii8o3y.pdf, 15/07/2014

[17] **Riverstone embedded**, I2C, http://riverstoneembedded.zxq.net/html/i2c_basics.html, 02/08/2014

[18] **Components**, I2C, http://components.about.com/od/Theory/a/Overview-Of-I2c.htm, 03/08/2014

[19] **Open Circuits**, XBee, http://www.opencircuits.com/Xbee_wireless_module#XBee_ZB_ZigBee, 05/09/2014

[20] **Plataformas XBee, XBee**, http://plataformaszigbee.blogspot.com/2012/05/practica-1-configuracion-y-conceptos.html, 09/09/2014

[21] **Blog Electronica**, Configuracion de XBee, http://www.blogelectronica.com/zigbee-maxstream-sdk/, 13/09/2014

[22] **Tunnelsup**, API a Router en XBee, http://www.tunnelsup.com/xbee-s2-quick-reference-guide-cheat-sheet/, 19/09/2014

[23] **Mecatronica**, Pasos para configuración de XBee, http://mecatronicauaslp.wordpress.com/2013/07/04/tutorial-xbee-parte-2-configuracion-xbee-serie-2/, 23/09/2014