Universidad de La Laguna

Escuela Superior de Ingeniería y Tecnología

Grado en Ingeniería Electrónica Industrial y Automática

— Trabajo de Fin de Grado —

Implementación de la electrónica para el control dinámico de un manipulador Scorbot-ER Vplus

Intefaz electrónica, lectura de codificadores digitales de posición y cálculos cinemáticos.

Autora: Ana Estévez Pérez

Tutorizado por: Santiago Torres Álvarez

Julio, 2019

Índice

1. Agradecimientos

Quería dedicar un apartado para agradecer a todos aquellos que nos han ayudado de una forma u otra a afrontrar estos meses de trabajo.

A mi familia, por apoyarme siempre.

A nuestro tutor D. Santiago Torres Álvarez, por guiarnos y atender nuestras dudas incluso fuera del horario de tutorías.

A D. Manuel Fernández Vera por el apoyo y ayuda con los materiales que necesitabamos para cumplimentar el trabajo.

Al Servicio de Electrónica de la Universidad de La Laguna, por prestarnos su ayuda con la elaboración de una placa para los componentes I2C y por soldarlos.

A mi compañero de Trabajo de Fin de Grado, por apoyarme y ayudarme durante la realización del mismo.

2. Abstract

This project consists of the development of an electronic circuit that allows the dynamic control of the handling robot Scorbot-ER V. The electronics designed has to handle the information provided by the encoders of the handling robot.

To handle the data received from the counters, to move the motors and to carry out the control of the Scorbot, an Arduino Mega 2560 which works with the microcontroller ATmega 2560 is used.

- The counters LS7166 are in charge of the quadrature count of the signals of the encoders. Due to the limitations regarding the number of pins available on the board, it has been decided to use the PCA9535 integrated circuit, which work with I2C communication. This integrated circuit will be connected to a counter, one I2C chip for each counter, so it dumps its account value in the I2C to be transmitted to the Arduino later by using only two pins for the six counters (those of the I2C protocol, the data line SDA and the SCL watch line).
- The L298N modules are the drivers in charge of the movement of the motors, prepared to work with the PWM signal generated from the Arduino.

The alarm management has been done via software, using the counters account for obstacle detection. During a movement, if it is detected that the handling robot can not advance, the controller stops all movements to prevent the engine from being damaged.

Regarding the control of the Scorbot-ER V, a PID control has been developed using the Arduino libraries. To avoid saturations in the input, the integral action is not used until reaching the middle of the setpoint.

We choose the software Phyton to carry out the communication between the Arduino and the equipment. Phyton has a library that allows the the communication with the Arduino. A program has been designed to carry out several commands using our own coding.

The main objective is the implementation of an electronics circuit that allows the Scorbot-ER V to be controlled with a dynamic control so that it is not necessary to use the controller provided by the manufacturer. In addition, with this electronics the delays of the controller's protocols will be eliminated and the hardware currently occupied by the current controller of the manipulator will be reduced.

Some of the Specific objectives are the following:

- Move the motors of each axis of the Scorbot using a PWM signal generated with Arduino for the L298N module.
- Quadrature counting, through the integrated LS7166, of the signals that come out of the encoders associated with each motor.

- Reading of the account carried out by integrated LS7166 through I2C communication (integrated PCA9535) in Arduino.
- Establish communication of the Arduino and the scorbot (hardware-I2C- and software -python).
- Carry out the instruction 'home' on all axes of the Scorbot-ER V Plus.
- Set alarm against over-intensities.
- Implementation of a descentraliced PID control loop.

3. Introducción

3.1. Precedentes

El presente Trabajo de Fin de Grado tiene como precedente el Proyecto de Fin de Carrera '*Programación y puesta en marcha de la electrónica de control de un robot manipulador*' realizado por D. Julio Victor Méndez Bethencourt. Se pretende llevar a cabo una actualización de la electrónica utilizada por D. Julio en su Proyecto de Fin de Carrera, así como reducir el hardware del controlador del Scorbot.

En el proyecto en el que nos basamos, se utilizan tres tarjetas PIC, cada una con una función en concreto: un PIC para generar la señal PWM de una parte de los motores, una segunda para generar la PWM de los motores restantes y tratar con la información aportada por los contadores y un último PIC para la gestión de alarmas. En nuestro proyecto, en cambio, se emplea un único microcontrolador (ATMEGA2560) para la realización de todas estas tareas ya que dispone de la suficiente capacidad y flexibilidad para llevarlas a cabo. [5]

3.2. Descripción general del proyecto

Cabe destacar que este Trabajo de Fin de Grado se divide en dos partes, aunque se dispone de partes comunes tales como la introducción, las pruebas y las conclusiones. Una parte es el presente trabajo y la otra es el Trabajo de Fin de Grado de Carlos Javier Siverio con título: "Implementación de la electrónica para el control dinámico de un manipulador Scorbot-ER Vplus, Desarrollo de la Comunicación".

El proyecto consiste en el desarrollo de una electrónica que permita llevar a cabo un control dinámico del manipulador Scorbot-ER Vplus. La electrónica diseñada ha de ser capaz de manejar la información que proviene de los encoders del mismo.

Para el manejo de los datos que recibimos de los contadores, mover los motores y realizar las acciones de control vamos a utilizar una placa de Arduino Mega 2560, que trabaja con un microcontrolador ATmega 2560.

- La electrónica encargada de la cuenta en cuadratura de las señales de los encoder consiste en una serie de contadores LS7166. Debido a las limitaciones en cuanto al número de pines de los que dispone la placa se ha decidido utilizar los integrados PCA9535, que trabajan mediante comunicación I2C. Éstos integrados irán conectados cada uno a un contador para que este vuelque su valor de cuenta en el I2C y este útilimo se lo transmita al Arduino utilizando únicamente dos pines para los seis contadores (los propios del protocolo I2C, la línea de datos SDA y la línea de reloj SCL).
- Se ha recurrido a los módulos (drivers) L298N, para controlar y mover los motores del Scorbot-ER V. Éstos módulos trabajan con la señal de PWM que se genera desde el dispositivo Arduino.

En cuanto al diseño del control del manipulador se ha implementado un controlador PID descentralizado (un esquema PID para cada una de las articulaciones) utilizando las librerías de Arduino. Para evitar las saturaciones a la entrada se trabaja inicialmente sin utilizar la acción integral y la activamos cuando la salida llegue a la mitad de la consigna.

Para realizar la comunicación entre el Arduino y el PC de control se ha optado por utilizar Python, ya que este cuenta con una librería que permite la comunicación con el Arduino. Se ha diseñado un programa que permite realizar una serie de comandos y órdenes utilizando una codificación propia.

3.3. Objetivos

3.3.1. Objetivo general

El objetivo principal es la implementación de una electrónica que permita controlar el Scorbot-ER Vplus mediante un control dinámico de forma que no se tenga que recurrir al controlador proporcionado por el fabricante.

Además, con esta electrónica se eliminarán los retardos propios de los protocolos del controlador y se reducirá el hardware que actualmente ocupa el controlador de dicho manipulador.

3.3.2. Objetivos específicos

- Mover los motores de cada eje del Scorbot utilizando una señal PWM generada con Arduino que transmite al módulo L298N.
- Conteo en modo cuadratura, mediante el integrado LS7166, de las señales que salen de los encoders asociados a cada motor.
- Lectura de la cuenta llevada a cabo por los integrados LS7166 mediante la comunicación I2C, con el integrado PCA9535, en Arduino.
- Establecer comunicación del arduino y el scorbot (hardware -I2C- y software -python).
- Llevar a cabo la instrucción **home** en todos los ejes del Scorbot-ER V Plus.
- Establecer alarma contra sobre-intensidades.
- Implementación de un lazo de control, en principio se recurre a un PID.

3.4. Planteamiento inicial

Para el desarrollo del proyecto se propuso inicialmente la configuración que se muestra a continuación:

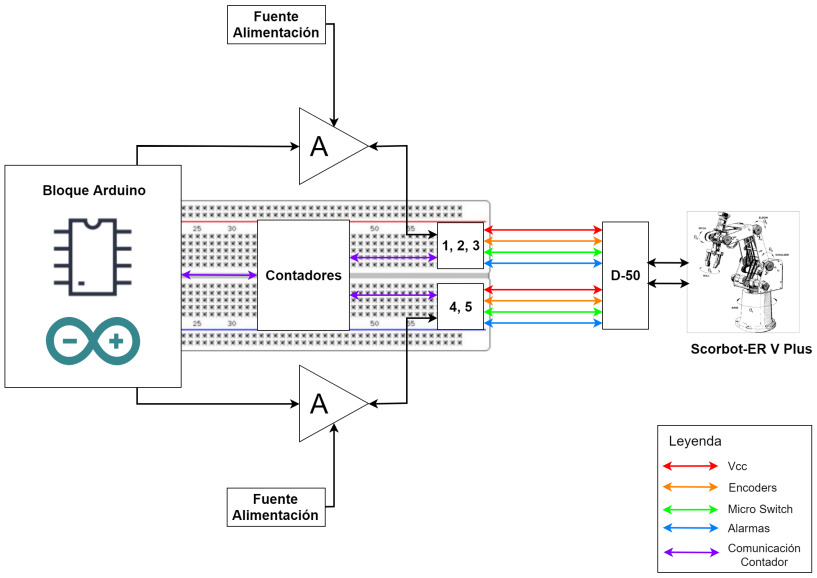


Figura 1: Configuración inicial

Concretamente, se planteó el uso del "Bloque Arduino"para realizar el control dinámico del brazo robótico, enviando las señales de potencia a unos amplificadores que muevan los motores del Scorbot y midiendo las señales que provienen de los sensores de los que se dispone. Además, se planteó el uso de unos contadores para que el Arduino pudiese tomar de ellos la posición de las articulaciones del brazo cuando lo necesite y no cargar así demasiado al microcontrolador.

A continuación se muestran las diferentes configuraciones que se plantearon para el "Bloque Arduino":

A) Dos Arduinos que trabajan con una programación al mismo nivel y que se encuentran comunicados entre sí. Quizás sea la configuración más compleja pues requiere que ambos Arduino estén sincronizados y en constante comunicación.

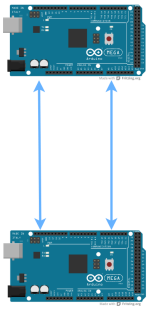


Figura 2: Comunicación entre dos controladores Arduino

B) Dos Arduinos que trabajan con una programación al mismo nivel y que se encuentran comunicados gracias a otro Arduino que trabaja con una programación a un nivel superior. Esta opción plantea que exista un Arduino con una jerarquía superior que controle a los otros dos.

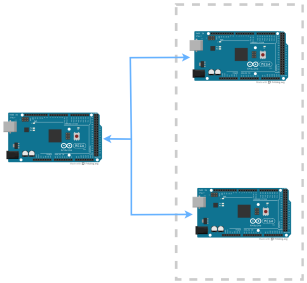


Figura 3: Comunicación entre dos Arduinos esclavos y un maestro

C) Un único Arduino que se encargue de todo el control. En caso de que el Arduino reúna los requisitos necesarios de memoria para realizar todo el control, sería posible la utilización de un único microcontrolador para el desarrollo del control dinámico.

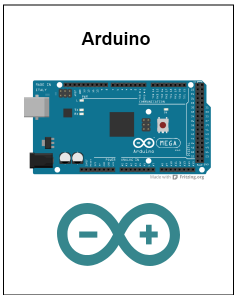


Figura 4: Control con un único Arduino

Se han estudiado algunos modelos de los controladores Arduino y se han recogido en la siguiente tabla las especificaciones de los mismos:[6]

Especificaciones	UNO Rev3	Mega 2560 Rev3	PIC16F877
Microcontrolador	ATmega328P	ATmega2560	PIC16F877
Voltaje Funcionamiento	5V	5V	2-5.5V
Voltaje Entrada (recomendado)	7-12V	7-12V	
Voltaje Entrada (límite)	6-20V	6-20V	
Pines I/O Digitales	14	54	
Pines I/O PWM Digitales	6	15	2
Pines de Entrada Analógicos	6	16	
Corriente DC por Pin I/O	20 mA	20 mA	
Corriente DC por Pin de 3,3V	50 mA	50 mA	
Memoria Flash	32 KB	256 KB	8 KB
SRAM	2 KB	8 KB	368 B
EEPROM	1 KB	4 KB	256 B
Velocidad Reloj	16 MHz	16 MHz	20 MHz
LEDs Incorporados	13	13	
Largo	68,6 mm	101,52 mm	
Ancho	53,4 mm	53,3 mm	
Peso	25 g	37 g	

Tal y como se muestra en la tabla se han tomado los dos modelos más representativos de las placas Arduino; siendo estos el Arduino UNO R3 y el Arduino Mega 2560. Para determinar la posibilidad de que un único Arduino controlase seis motores al mismo tiempo, ya que nos interesa mover los 6 motores del Scorbot, se realizó un experimento con el Arduino UNO R3 en el que se controlaban seis servomotores al mismo tiempo con la ayuda de dos joystick.

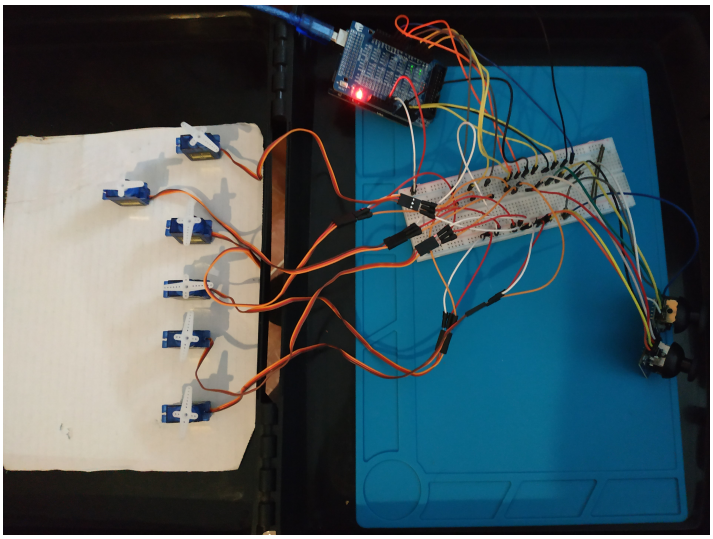


Figura 5: Prueba-mover seis motores con arduino

Tras la realización de la prueba se concluyó que era posible controlar los seis servomotores sin ningún tipo de problema, ocupando tan solo un 11 % de las capacidades del microcontrolador.

Dado que con el Arduino UNO R3 no se cubre el número suficiente de pines, se recurre al Arduino MEGA 2560. Para no quedarnos justos con el Arduino UNO R3 para realizar más tareas (lectura de los contadores, control de las alarmas...) en términos de memoria. Por el contrario el Arduino MEGA 2560 cuenta con mayores capacidades de memoria y una mayor cantidad de pines.

En conclusión, se ha decidido elegir la configuración en la que se utiliza un único Arduino para realizar el control del Scorbot-ER, siendo el modelo elegido el Arduino MEGA 2560.

Para conocer la posición de cada motor del Scorbot, se realiza la cuenta de los flancos de las señales aportadas por los encoders (situados en los ejes de los diferentes motores); cuentas que serán realizadas por los contadores LS7166.

Para poder leer las salidas de los contadores barajamos varias opciones:

- **DAC (Conversor Digital-Analógico):** Para convertir la salida digital de los contadores en una señal analógica y poder llevar esta información al Arduino utilizando solo una única entrada analógica (porque de no ser así, se necesitarían muchas entradas, más de las que dispone el arduino mega 2560).
- **DAC + I2C:** Con respecto a la idea anterior, había un problema, pues el DAC no dispone de entradas para transmitir directamente la salida de los contadores en el mismo, por lo que la solución que se plantea es la de

utilizar un I2C + DAC.

- **I2C:** Finalmente, nos decantamos por usar únicamente un I2C, pues con éste podemos meter directamente la salida digital del contador en el mismo. Así mismo, del I2C saldrán 2 señales SDA Y SCL que se podrán transmitir al Arduino, que dispone de 2 entradas de SDA y otras 2 de SCL.

El esquema planteado inicialmente queda de la siguiente manera:

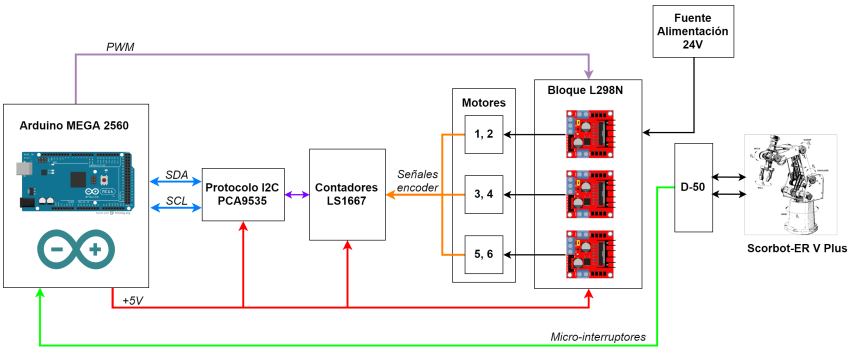


Figura 6: Configuración final

4. Cronograma

En la siguiente página se encuentra el cronograma, en el que se realiza la descripción de las diferentes tareas que se han llevado a cabo durante el desarrollo del trabajo.

Diciembre 2018

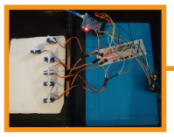
Documentación

Scorbot-ER V
Proyecto de Fin de Carrera en el que se basa nuestro Trabajo de Fin de Grado

Enero 2019

Pruebas/Documentación

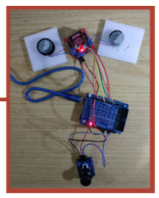
Estudio de las capacidades de Arduino
Prueba mover 6 motores con Arduino (Comprobación cantidad de dispositivos Arduinos necesarios)
Investigación y documentación para comenzar con la implementación de la electrónica; inicialmente se busca poder mover los motores (Se elige módulo L298N)

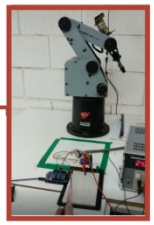


Febrero 2019

Pruebas mover motores

Comprobación de la intensidad que requiere cada motor para asegurar que el módulo escogido es viable
Pruebas con módulo L298N:
- Prueba moviendo motores de 5v con un módulo del que se disponía (funciona)
- Se compran más módulos para mover los demás motores y se prueba de nuevo con motores de 5v pero no funciona
- Se cambian los módulos y sigue sin funcionar
- Se prueban los módulos con motores de mayor potencia (12v) y sin el jumper de regulación y funcionan los módulos
- Se prueban los módulos con el motor de la base del Scorbot-ER V
- Se prueban los módulos con los demás motores del Scorbot-ER V





Marzo 2019

Lectura encoders

Elección de los componentes necesarios para trabajar con las señales de los encoders de los motores del Scorbot
Se elige contador LS7166 para realizar cuenta en cuadratura de las señales
Se plantea el uso de un DAC para llevar la salida de los contadores al dispositivo Arduino, como no es posible conectar directamente los contadores al DAC, se plantea usar DAC+I2C y finalmente sólo un I2C
Se escoge el controlador I2C PCA9535

Abril 2019

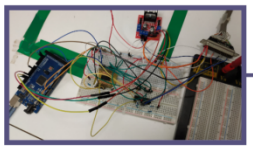
Componentes

Se plantea esquemático de conexiones inicial
Se compran los componentes escogidos (LS7166 y PCA9535) que como se mandan a pedir tardan en llegar. Los controladores I2C tardan aproximadamente 2 semanas y los contadores un mes.
Los I2C son de tecnología de montaje superficial (SMD) por lo que el Servicio de electrónica elabora un circuito impreso para los mismos y los suelda
Documentación con ayuda de los datasheets sobre contador LS7166 y el controlador I2C PCA9535

Mayo 2019

Pruebas lectura con I2C y contadores

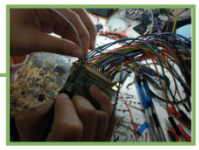
Documentación con ayuda de los datasheets sobre contador LS7166 y el controlador I2C PCA9535
Programación de la lectura de los contadores
Se realizan pruebas con el contador disponible y un controlador I2C para comprobar que la lectura de los contadores se haga correctamente, hay fallos y se van corrigiendo

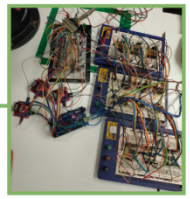


Junio 2019

Prueba Lectura contadores, Home y PID

Cuando se realiza correctamente la lectura, se sueldan los pines de las demás placas con los controladores I2C para poder colocarlos en la protoboard y poder probar el programa de lectura de contadores con los demás motores
Diseño del prototipo final
Soldadura de los cables en un adaptador para el conector D50
Se prueba la instrucción Home en la base del Scorbot, cuando se logra que funcione se replica para los demás motores
Implementación del prototipo real (se colocan todos los componentes y conexiones)
Se comprueba funcionamiento del prototipo
Se realiza la instrucción Home para todos los motores del Scorbot-ER V
Se prueba programa PID





5. Descripción de la interfaz electrónica

5.1. Driver L298N

Para controlar y mover los motores se ha recurrido al módulo L298N, que contiene los siguientes componentes y elementos:

- Circuito integrado L298N formado por un doble puente en H.
- Regulador de tensión LM7805 para establecer el voltaje lógico a 5v de corriente continua si está activo.
- Diodos para la protección del circuito contra corrientes inducidas.
- Disipador de calor para el circuito integrado L298N.
- Jumpers para habilitar los pines de control de las salidas para los motores y para activar el regulador. Si el regulador está activo (jumper del regulador está colocado) entonces la alimentación puede ser de 12 v corriente continua, en caso contrario la alimentación puede ser de 35 v de corriente continua (con dos amperios como máximo, corriente más que suficiente para mover los motores del Scorbot-ER Vplus).
- Clemas de conexión (entrada de alimentación, salida para los motores).

Todos estos elementos se pueden apreciar en la figura 7. En la figura 8, se incluye el diagrama de bloques del integrado L298N, en el que se pueden identificar los dos puentes en H de los que dispone.[7] [8]

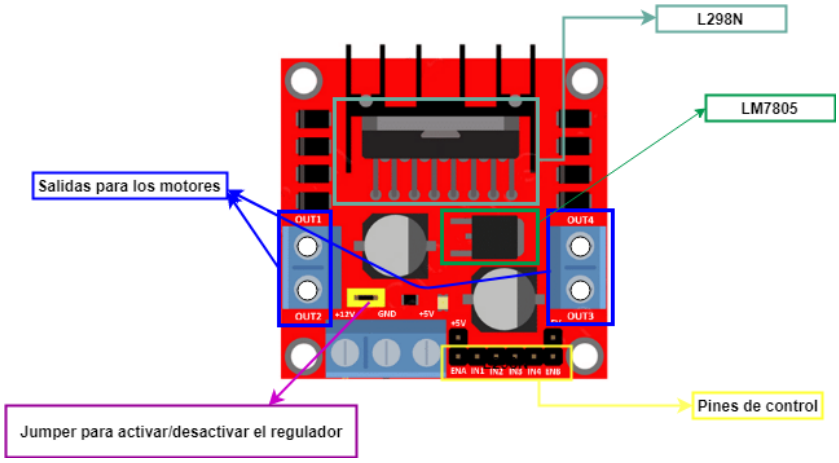


Figura 7: Módulo L298N

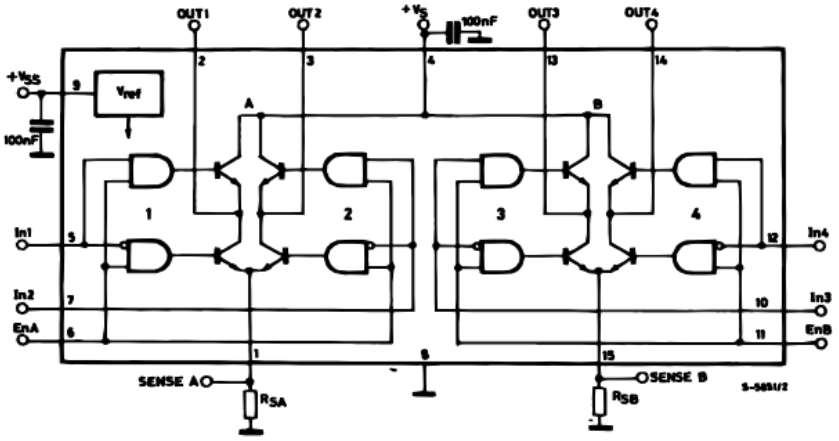


Figura 8: Diagrama de bloques L298N

Se utilizan tres pines de control por motor, dos que controlan la dirección de giro y uno para transmitir la PWM generada por Arduino. Para su utilización en el Scorbot-ER V se usará un módulo por cada dos motores, además, es necesario utilizar una fuente externa que los alimente con 24V.

Para evitar problemas (picos de tensión ...) en la parte digital, que trabaja con menor voltaje que la de potencia, sería necesario incluir una etapa de aislamiento. Aunque gracias a la protección de la que dispone el módulo L298N, no se precisa de un aislamiento adicional entre la parte de potencia y la digital.

5.2. Encoders

Los encoders generan impulsos eléctricos en función de la cantidad de movimiento de los ejes, es decir, el codificador traduce el movimiento giratorio de los ejes de los motores, de forma que los pueda entender el controlador.

El diseño de nuestra electrónica lo haremos en base al "peor caso" (entre Scorbot-ER V y Scorbot ER IX la resolución del SCORBOT ER IX es más restrictiva), dado que en principio se contemplaba la opción de disponer de una electrónica que fuese aplicable en ambos manipuladores.

Por lo tanto, los encoders que usamos como referencia serán los del Scorbot IX (codificadores ópticos incrementales y con pulso de referencia).

Estos codificadores cuentan con 512 slots (ranuras) y además, de una ranura adicional que sirve para generar el pulso de referencia (salida que indica que se ha dado una vuelta completa).

Teniendo en cuenta tanto los flancos de subida y bajada de los impulsos generados por los encoders (para cada ranura 4 conteos), la resolución de los mismos es la siguiente:

$$512\,x\,4 = 2048\,cuentas/revolución$$

$$ResoluciónEncoder = (360°)/2048 = ,176°$$

Del encoder salen 2 señales en cuadratura, que podemos ver en la figura 9, que permiten determinar el sentido de giro y la posición de los motores. Estas señales son enviadas a los contadores.

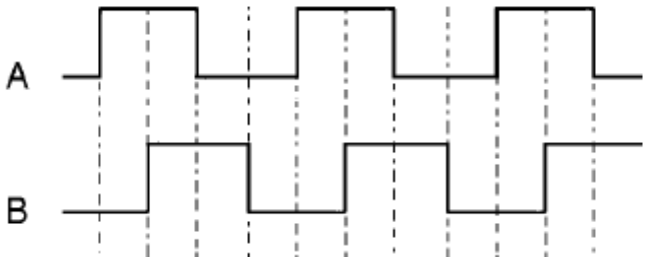


Figura 9: Señales de salida del encoder

5.3. Integrado LS7166

El contador escogido es el LS7166 y lo usaremos en el modo de conteo de cuadratura, debido a los impulsos generados por los encoders. El objetivo del contador es realizar la cuenta de forma que permita conocer al controlador la posición y sentido de giro de los motores.
Usaremos los pines 6 y 7 de los contadores como entrada de las señales que provienen de los encoders.

Para conocer más en detalle el integrado LS7166, estudiaremos la función de cada uno de sus pines:

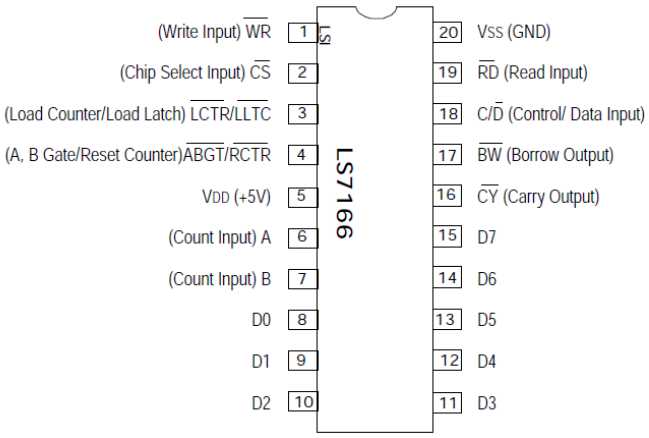


Figura 10: Pines LS7166

- **PIN 1 (Write Input - $\overline{WR}$):** Esta entrada se activa a nivel bajo ('0 lógico') y habilita la escritura en los registros de datos o de control (de los datos contenidos en el bus).
- **PIN 2 (Chip Select Input):** La entrada $\overline{CS}$ se activa a nivel bajo ('0 lógico') y habilita el contador para escritura y para lectura.
- **PIN 3 (LOAD CTR/LOAD LATCH - $\overline{LCTR/LLTC}$):** Este pin que se activa a nivel bajo, se puede programar para funcionar como control externo de carga tanto del contador interno (CNTR) como del registro de salida (OL). No se utiliza, por lo que estará siempre a alta.
- **PIN 4 (A, B GATE/LOAD LATCH - $\overline{(ABGT)/(RCTR)}$):** Esta entrada, que se activa a nivel bajo, se puede programar de forma que habilite las entradas A y B o como entrada de reset del contador. No se utiliza, por lo que estará siempre a alta.
- **PIN 5 (V_{DD}):** Terminal positivo de la tensión de alimentación. (5v)
- **PIN 6 (Count input):** Entrada de conteo programable que puede funcionar de tres modos: conteo en sentido ascendente, conteo en sentido descendente o conteo en modo de cuadratura. A esta entrada le llegará una de las señales de cuadratura de fase del encoder.
- **PIN 7 (Count input):** Entrada de conteo programable que puede funcionar como entrada de conteo descendente, como entrada de selección de conteo de la entrada A o como entrada de conteo en modo cuadratura. A esta entrada le llegará la segunda señal transmitida del encoder.
- **PIN 8 – 15 (Data Bus (D0...D7)):** Las ocho líneas de E/S triestado (D0...D7) componen el bus de datos d del integrado. Por este bus de 8 bits se transmiten los valores del contador. Dado que el contador es de 24 bits, se hará de tres veces por lectura. Estos pines irán conectados al bus $IOx_0 \ldots IOx_1$ del controlador I2C.

- **PIN 16 (Carry Output):** La salida $\overline{CY}$ puede ser programada para actuar en diferentes modos de acarreo o como salida del comparador. No se utiliza.
- **PIN 17 (Borrow Output):** La salida $\overline{BW}$ puede ser programada para actuar de diferentes modos e indicar que se ha alcanzado el último estado de la cuenta en cuenta descendente. No se utiliza.
- **PIN 18 (Control/Data Input):** La entrada C/$\overline{D}$ se utiliza para seleccionar:
  C/$\overline{D}$ = 0 - Registros de control
  C/$\overline{D}$ = 1 - Registros de datos
  (Durante una operación de lectura o de escritura)
- **PIN 19 (Read Input):** Un nivel bajo en la entrada $\overline{RD}$ habilita la lectura de los registros OSR y OL.
- **PIN 20 (V_{SS}):** Terminal negativo de la tensión de alimentación. (GND)

En la siguiente figura se pueden observar los diferentes modos de configuración del modo de conteo de cuadratura, nosotros utilizaremos el modo X4, correspondiente a las señales UPCLK(X4), incrementa en uno la cuenta y DNCLK(X4), disminuye en uno la cuenta.

La cuenta se realiza de tal forma que se incrementa o decrementa una cuenta con cada flanco de las señales A y B, en función de si los motores giran hacia un lado u a otro.

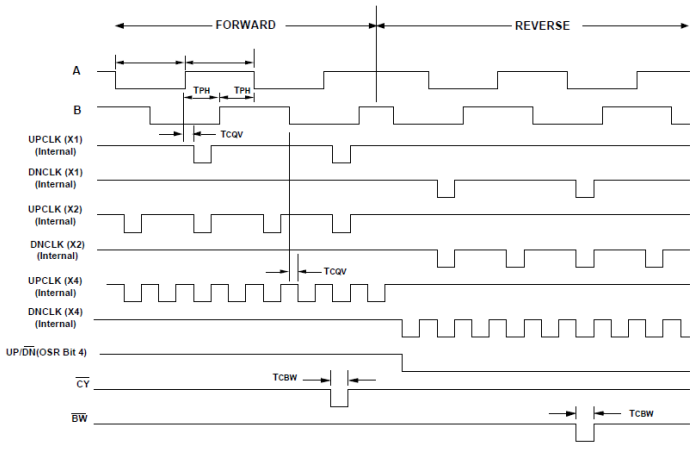


Figura 11: Modos de configuración de conteo en cuadratura

En el Datasheet del contador encontramos el diagrama interno del mismo:

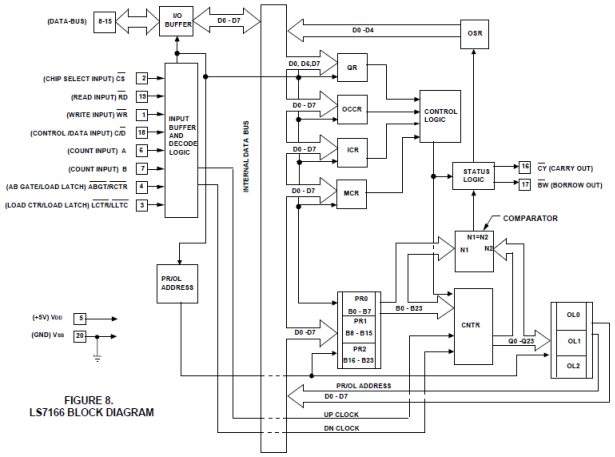


Figura 12: Diagrama interno del contador LS7166

Descripción de los registros:

- **Registros de datos:**
  - **PR (Preset Register – Registro pre-carga):** Este registro es la entrada al contador en sí (CNTR), por lo que el CNTR se carga con datos de 24 bits a través de este registro. Los datos se escriben en una secuencia de 3 ciclos de escritura (Byte 0 a PR0, Byte 1 a PR1 y Byte 2 a PR2).
  - **OL (Output Lacth):** Salida del CNTR. Se puede acceder al valor de CNTR de 24 bits en cualquier momento realizando una transferencia de CNTR a OL y leyendo el registro OL en una secuencia de 3 ciclos de lectura (Byte 0 a OL0, Byte 1 a OL1 y Byte 2 a OL2).
- **Registros de control:**
  - **OSR (Registro de Estado de salida):** Indica el estado de CNTR.
  - **MCR (Registro de Control Maestro):** Resetea tanto los registros de datos como los de control.
  - **ICR (Registro de Control de Entradas):** Inicializa los modos de operación de entrada del contador. Con este registro configuramos A como cuenta de entrada, así como las operaciones de carga del contador. Como utilizamos el modo cuadratura no hace falta definir B (pues A y B están desfasados 90°).
  - **OCCR (Registro de Control de Salidas):** Inicializa el CNTR y los modos de operación de salida. Podemos escoger entre los siguientes modos de conteo:

    - Binario o BCD.
    - Normal o no cíclico.

- Normal o de división por N.
- Binario o BCD, o modo 24 horas.

En este caso usaremos el modo de conteo binario.

– **QR (Registro de cuadratura):** Registro para seleccionar el modo de conteo de cuadratura. En este caso, escogemos el modo X4 (una cuenta cada flanco de subida y bajada de las señales que entran en A y B).

En la figura 13, se pueden observar los ciclos de lectura y escritura a tener en cuenta para la correcta programación de los contadores.

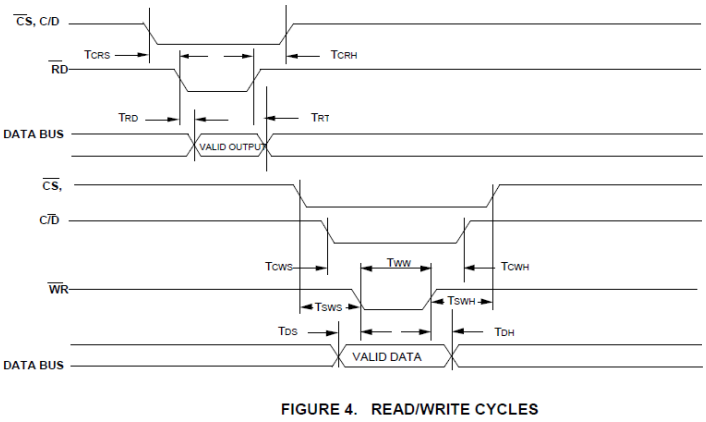


FIGURE 4. READ/WRITE CYCLES

Figura 13: Ciclos de lectura y escritura

Vemos como se lleva a cabo el ciclo de lectura cuando se produce un pulso de bajada en la línea $\overline{RD}$ y el de escritura cuando se produce en la línea $\overline{WR}$, estando además, las líneas C/$\overline{D}$ y $\overline{CS}$ a baja.

5.4. Integrado PCA9535

Pines del integrado PCA9535:

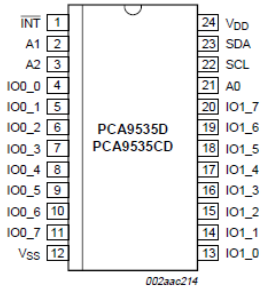


Figura 14: Pines del integrado PCA9535

El bus I2C, formado por las líneas SDA (System Data) y SCL (System Clock), es el enlace entre el maestro del bus (Arduino MEGA 2560) y los esclavos con los que establece comunicación (controladores I2C).

Podemos ver una representación del bus en la siguiente figura:

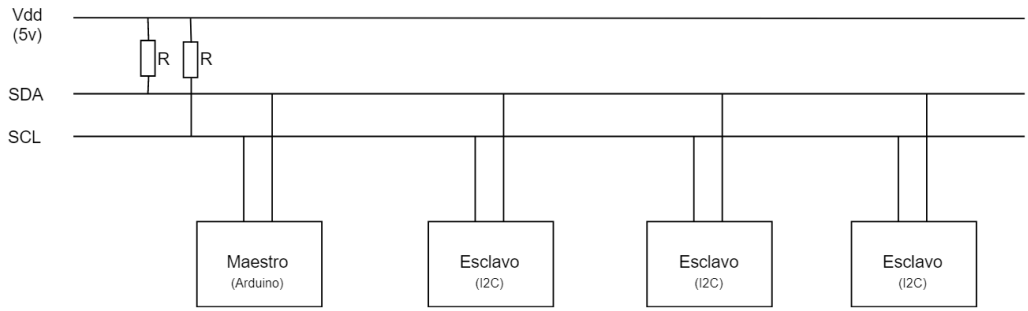


Figura 15: Bus I2C - Maestro y esclavos

La línea SDA se utiliza para el envío de datos.

La línea SCL es la señal de reloj que, en este caso, la establece Arduino. Cuando los datos son enviados por SDA, los pulsos de reloj son enviados por SCL para sincronizar maestro y esclavo.

Las resistencias que conectan las líneas SDA y SCL con V_{DD} (alimentación 5v) son resistencias pull-up necesarias debido a que son líneas de tipo drenador abierto.

Proceso de comunicación:

El maestro del bus es el encargado de establecer la comunicación. Para iniciarla, envía una condición de inicio (start condition), que radica en un cambio de ALTA a BAJA en la línea de comunicación SDA mientras SCL está en ALTA. Por el contrario, cuando el maestro quiera finalizar la comunicación, manda una condición de parada (stop condition). Esto es, cuando SDA cambia de BAJA a

ALTA mientras la línea de comunicación SCL está BAJA; liberando así el bus. Este proceso lo se puede observar de forma más clara en la siguiente imagen:

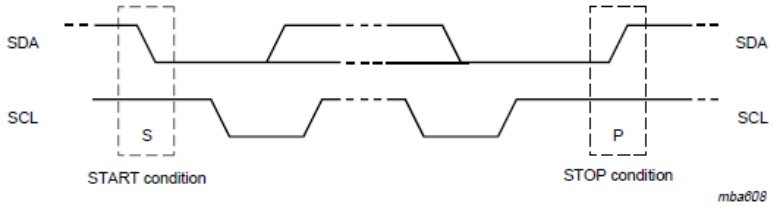


Figura 16: Línea de datos -SDA- y línea de reloj -SCL-

Para conocer con qué dispositivo se establece la comunicación, el maestro ha de enviar un byte con los bits que componen la dirección del dispositivo esclavo (controlador I2C) con el que se quiera establecer la comunicación. Este byte se compone por 7 bits que contienen la dirección y un bit con el que se indica si se desea leer (recibir información del esclavo) o escribir (enviar información al esclavo). Si el bit R/$\overline{W}$ está a 1 indica que se va a realizar lectura y si está a 0 una escritura.

Todos los esclavos del bus comparan la dirección enviada por el maestro con su propia dirección. Si coincide, el esclavo es direccionado como esclavo-transmisor (si R/$\overline{W}$ = 0) o esclavo-receptor (si R/$\overline{W}$ = 1). Además, el dispositivo esclavo envía un bit de reconocimiento (ACK) si la dirección coincide.

Así mismo, el dispositivo esclavo usa el bit ACK/NACK para indicar si ha recibido correctamente la secuencia de bits anterior.

En el caso de una lectura, tras el último byte se envía un bit NACK indicando el final de la transmisión, seguido del bit de parada que finaliza la comunicación.

Dirección de los esclavos:

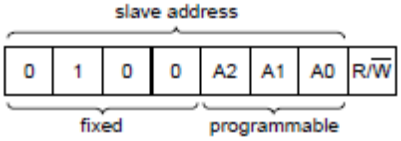


Figura 17: Dirección de los esclavos del bus I2C

La dirección de cada dispositivo esclavo se establece con los pines A0, A1 y A2 del controlador I2C, de tal forma que para definir la dirección del primer esclavo, por ejemplo, conectaremos A0, A1 y A2 a GND (masa) mientras que para la dirección del segundo esclavo A0 y A1 irán conectados a GND y A2 a V_{DD}.

Conectar estos pines a GND implica asignarles un '0 lógico' y conectarlos a V_{DD} implica asignarles un '1 lógico'. [10] [11]

5.5. Resistencias PULL-UP

Las resistencias PULL-UP son necesarias para asegurar el 1 lógico (5v) y el 0 lógico (0v) en los integrados digitales, pues la lógica que entienden es la TTL(lógica transistor a transistor).

Utilizamos diferencias resistencia pull-up en la electrónica implementada:

- En las líneas SDA y SCL se usan resistencias 2,2KΩ conectadas a V_{DD}.
- En cuanto a las entradas $\overline{ABGT/RCTR}$ y $\overline{LCTR/LLTC}$ del contador para fijarlas a 1 lógico , que es como están desactivadas,se usan resistencias de 47KΩ V_{DD}.
- Se utiliza una resistencia de 10KΩ para el pin $\overline{INT}$ del PCA9535 conectada a V_{DD}.
- Resistencias de 100KΩ para las entradas del contador en las que se introducen las salidas de los encoders.

5.6. Prototipo

En la figura 18 se muestra el prototipo diseñado para la electrónica implementada y en la figura 19 se muestra el diseño real implementado. A la hora de realizarlo físicamente se tuvo que adaptar este diseño a los materiales que teníamos disponibles (básicamente el cambio está en que no disponíamos de protoboard simples para los contadores y aprovechamos una doble para los mismos).

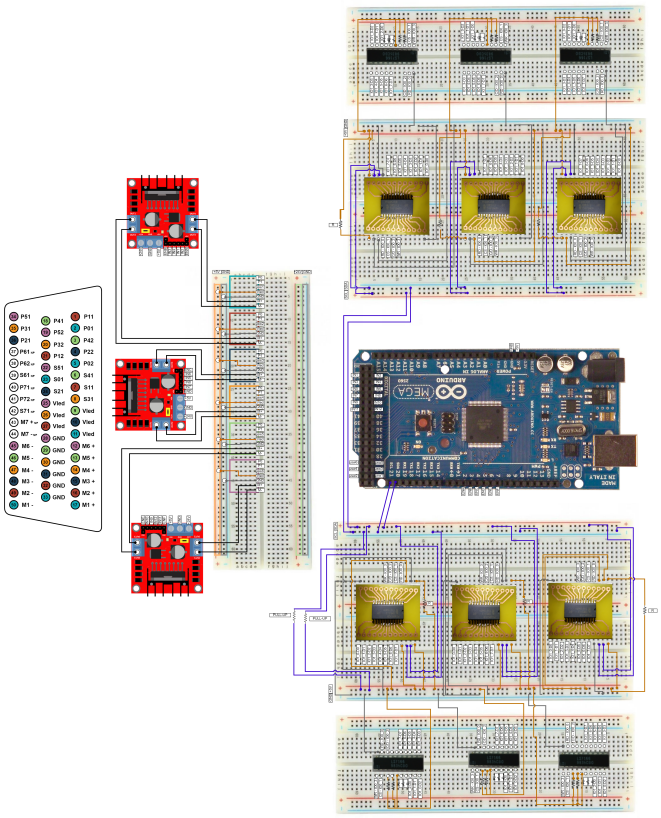


Figura 18: Prototipo de la implementación electrónica

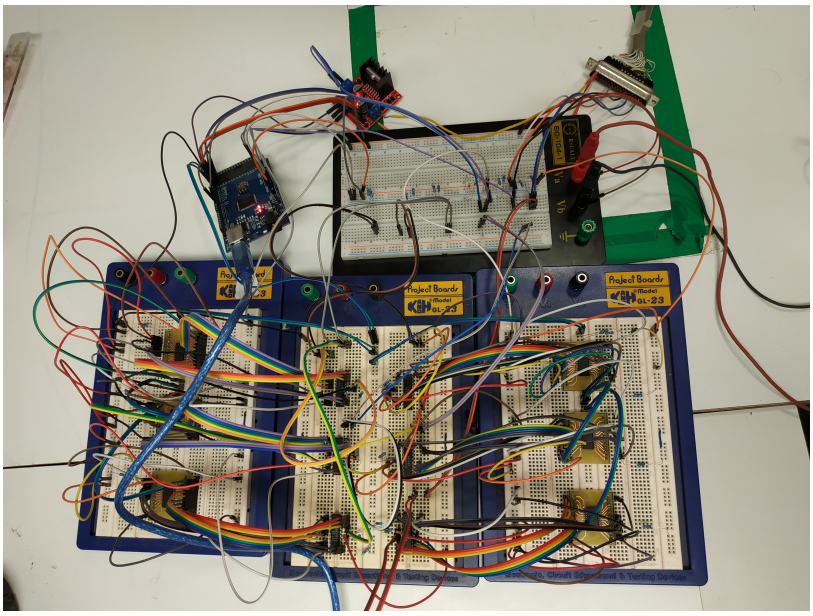


Figura 19: Implementación electrónica real

6. Programación

6.1. Librería L298N

Esta librería se utiliza para poder controlar la velocidad y movimiento de los motores. [13]

Algunas de las funciones que contiene son:

- **setSpeed()**: se encarga de establecer la velocidad de giro de los motores.
- **stop()**: se usa para parar los motores.
- **forward()**: se utiliza para hacer que el motor gire en un sentido
- **backward()**: se utiliza para hacer que el motor gire en el otro sentido

Esta librería nos permite optimizar el código y evitar errores, ya que al hacerlo manualmente tendríamos que poner, por ejemplo:

```
1 DigitalWrite(IN1, HIGH);
2 DigitalWrite(IN2, LOW);
3 DigitalWrite(ENA, PWM);
```

De esta forma es más fácil equivocarse y poner tanto IN1 como IN2 a ALTA y estropear el módulo y el motor. Sin embargo, con la librería sólo tendríamos que poner:

```
1 Motor.forward(); //Determina giro del motor
2 Motor.setSpeed() //Establece la velocidad del motor
```

6.2. Lectura Contadores

6.2.1. Diagramas de flujos

Lectura contadores:

Figura 20: Diagrama de flujo general - Lectura Contadores

Función inicializeCounter():

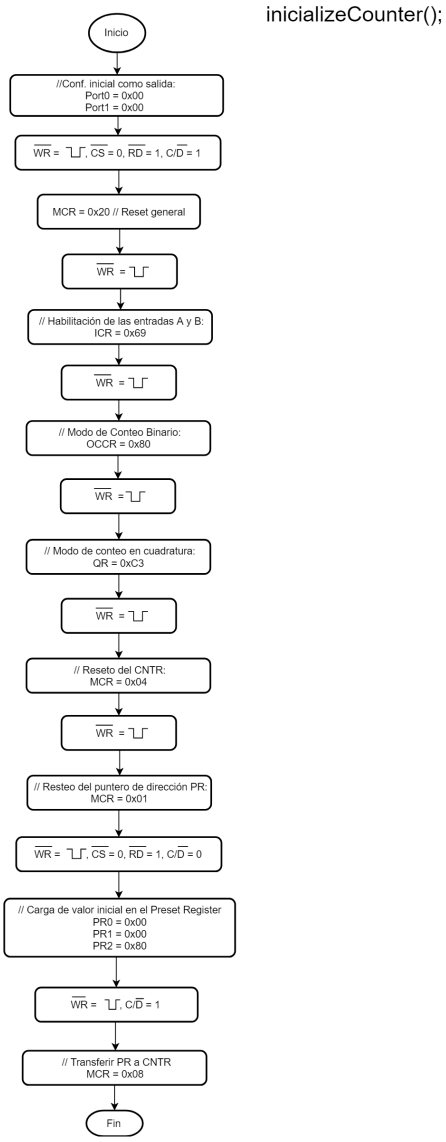


Figura 21: Diagrama de flujo Inicialización Contadores

Función readCounter():

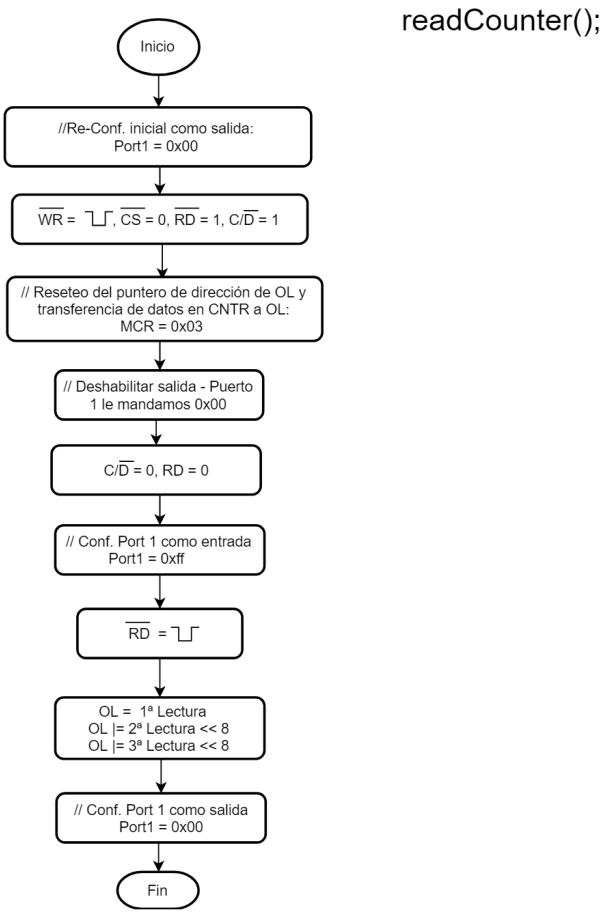


Figura 22: Diagrama de flujo Lectura de los Contadores

6.2.2. Librería Wire

Esta librería se utiliza para establecer la comunicación entre maestro y esclavos del bus I2C mediante las líneas SDA y SCL del controlador Arduino (maestro) y los controladores I2C (esclavos).

Las funciones a las que hemos recurrido para programar la comunicación maestro-esclavos son las siguientes:

- **begin():** Esta función solo se llama una vez, al inicio, y se encarga de inicializar la librería Wire, así como de especificar si es maestro o esclavo.
- **beginTransmission():** Inicia la comunicación con los esclavos, se le ha de pasar la dirección del esclavo con el que se quiere establecer la comunicación. Esta función incluye el bit de start (condición de arranque).

- **endTransmission():** Finaliza la comunicación con el esclavo, liberando así el bus I2C.
- **requestFrom():** Se emplea por parte del maestro del bus para solicitar una serie de bytes al esclavo con el que se quiera comunicar. Hay que pasarle la dirección de dicho esclavo y el número de bytes que desea que éste le transmita.
- **write():** Escribe los datos transmitidos por los esclavos a petición del maestro. Esta función se utiliza en primer lugar para enviar el byte de comando, para que el PC9535 sepa en qué registro se quiere guardar los datos que salen del mismo. Una vez transmitido el byte de comando, se vuelve a utilizar esta función pero esta vez para escribir un byte de datos enviados por los esclavos y que han de ser llevados al maestro por las líneas SDA y SCL.
- **read():** Lee un byte de datos transmitido por los esclavos. (Se utiliza esta funcion por cada byte que se reciba/ que se desee leer)

Es la propia librería la que se encarga de enviar los bits de reconocimiento. [9] [12]

6.2.3. Lectura contadores

La programación de la lectura de los contadores implica establecer la comunicación entre el contador, el I2C y el Arduino. [10] [11] [12]

Para escribir los bytes de los resgistros de control (MCR, ICR, OCCR, QR) que han de ser transmitidos para configurar el contador, se realiza la función **writeFunction(int art, byte outRegister, byte dataSend)**, dentro de la cual se llevan a cabo las siguientes acciones:

En primer lugar se utiliza la función **beginTransmission()** que transmite el bit de start, seguido de la dirección del esclavo con el que se quiere establecer la comunicación.

En segundo lugar se envía mediante la función **write()** el byte de comando que indica al I2C el registro en el que se guardará el byte del registro de control que se desee transmitir, en este caso será en el registro de salida del puerto 1. Posteriormente, utilizando la misma función **write()**, transmitimos el byte que se desea que llegue al contador y, por último, se finaliza la comunicación con el bit de stop (condición de parada) con la función **endTransmission()**.

En la figura 23 podemos observar la secuencia de bytes para la escritura en el registro de salida del I2C. Donde S es el bit de star, A el bit de reconocimiento y P el bit de stop.

S	0	1	0	0	A2	A1	A0	0	A	Command Byte	A	DATA to port X	A	P

Figura 23: Secuencia de bytes para la escritura

```
1 //Funcion de escritura para el I2C
```

```
void writeFunction(int art, byte outRegister, byte dataSend)
    {
  Wire.beginTransmission(art);
  Wire.write(outRegister);
  Wire.write(dataSend);
  Wire.endTransmission();
}
```

Para leer los bytes de datos que salen del contador y que el I2C los transmita posteriormente al arduino mediante las líneas SDA y SCL, se utiliza la función **readFunction(int art, byte inRegister, int i)**.

En primer lugar con la función **beginTransmission()** se transmite el bit de start seguido de la dirección del esclavo con el que se quiere establecer la comunicación. En segundo lugar se envía mediante la función **write()** el registro de entrada del puerto 1, en el que se quiere guardar los byte de datos que salen del contador. Posteriormente, se utiliza la función **endTransmission(false)** que permite reiniciar la comunicación sin liberar el bus para llevar a cabo la lectura.

A continuación, se ha de mandar un bit de re-start para realizar la lectura en sí. Se usa la función **requestFrom(art, 1)** indicando de nuevo el esclavo con el que se está comunicando el maestro y el número de bytes que se requieren del mismo.

La lectura se realiza de tres veces con un pulso de la línea $\overline{RD}$ cada vez que permite la lectura cada 8 bits, que se irán desplazando hasta formar los 24 bits de datos. Por último, se finaliza la comunicación con el bit de stop mediante la función **endTransmission()**.

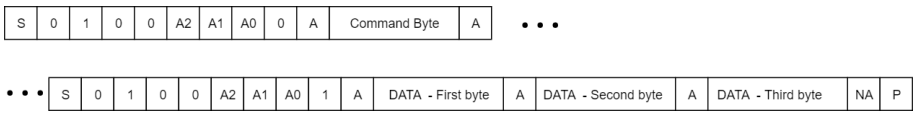


Figura 24: Secuencia de bytes para la lectura

```
//Funcion de lectura para el I2C
long int readFunction(int art, byte inRegister, int i){
  Wire.beginTransmission(art);
  Wire.write(inRegister);
  Wire.endTransmission(false);

  Wire.beginTransmission(art);
  Wire.requestFrom(art, 1);
  while(Wire.available()){
    _buff[i] = Wire.read();
  }
  Wire.endTransmission();

  return _buff[i];
}
```

Para la configuración de los puertos del I2C como entrada recurrimos a la función **inPortConfig(int art, byte configPort)**, a la que hay que pasarle la dirección del esclavo con el que se va a establecer la comunicación, así como el byte de comando necesario para acceder al registro de configuración del puerto. Además, hay que pasarle un byte de unos para que el puerto sea de entrada.

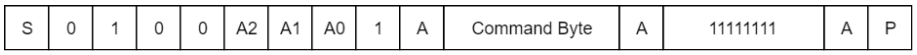


Figura 25: Secuencia de bytes para configuración del puerto 1 como entrada

```
//Funcion de configuracion del puerto como entrada
void inPortConfig(int art, byte configPort){
  writeFunction(art, configPort, input);
  delayMicroseconds(20);
}
```

Para la configuración de los puertos del I2C como salida recurrimos a la función **outPortConfig(int art, byte configPort)**, a la que hay que paserle la dirección del esclavo con el que se va a establecer la comunicación, así como el byte de comando necesario para acceder al registro de configuración del puerto. Además, hay que pasarle un byte de ceros para que el puerto sea de salida.

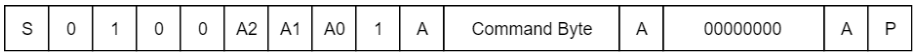


Figura 26: Secuencia de bytes para configuración de los puertos como salida

```
//Funcion de configuracion del puerto como salida
void outPortConfig(int art, byte configPort){
  writeFunction(art, configPort, output);
  delayMicroseconds(20);
}
```

En cada función que implique la transmisión de bytes añadimos un pequeño tiempo de espera **(delayMicroseconds())** para asegurar que el byte se manda correctamente.

La función **controlCounter(int art, bool WR, bool CD, bool CS, bool RD)** se encarga de formar el byte con las líneas de control que se transmitirán por el puerto 0 del I2C. Este byte estaría formado en los cuatro bits menos significativos por las líneas de control, siendo el valor de RD el que ocupa el bit menos significativo. Los bits más significativos son rellenados con 0, pues no los vamos a utilizar.

```
//Funcion CONTROL del contador
void controlCounter(int art, bool WR, bool CD, bool CS, bool
    RD){
  dataSend = 0;
```

```
4     dataSend = WR;
5     dataSend = (dataSend << 1) + CD;
6     dataSend = (dataSend << 1) + CS;
7     dataSend = (dataSend << 1) + RD;
8     writeFunction(art, outControlRegister, dataSend);
9     delayMicroseconds(20);
10  }
```

La siguiente función, **flankWR(int art)**, se utiliza para generar los flancos de bajada de la línea de control WR, necesario para poder escribir en los registros de control del contador para la configuración del mismo.

```
1   //Funcion para enviar valor de los registros de control
2   void flankWR(int art){
3     RD = HIGH;
4     CD = HIGH;
5
6     delayMicroseconds(20);
7     WR = LOW;
8     controlCounter(art, WR, CD, CS, RD);
9     delayMicroseconds(20);
10    WR = HIGH;
11    controlCounter(art, WR, CD, CS, RD);
12    delayMicroseconds(20);
13  }
```

La siguiente función, **flankPR(int art)**, se utiliza para generar los flancos de bajada de la línea de control WR, necesario para poder escribir en el registro de datos PR del contador.

```
1   //Funcion para acceder a los registros de datos
2   void flankPR(int art){
3     RD = HIGH;
4     CD = LOW;
5
6     delayMicroseconds(20);
7     WR = LOW;
8     controlCounter(art, WR, CD, CS, RD);
9     delayMicroseconds(20);
10    WR = HIGH;
11    controlCounter(art, WR, CD, CS, RD);
12    delayMicroseconds(20);
13  }
```

Para la lectura de los contadores se ha creado la función **readCounter(int art)**, en la que configuramos los registros de control del contador y los puertos del I2C tal y como se establece en los datasheets para llevar a cabo dicha lectura, es decir, para obtener los valores que salen del CNTR (contador en sí del LS7166) al registro de datos OL.

Al final de esta función volvemos a configurar el puerto 1 como salida ya que para las demás operaciones ha de estar configurado como salida y no como entrada.

```
1  //Lectura de los contadores
2  long int readCounter(int art){
3    //Re-configuramos el puerto como salida
4    outPortConfig(art, configPort1);
5
6    //Reseteamos puntero OL y transferimos el CNTR a OL
       (00000011)
7    dataSend = 0x03;
8    writeFunction(art, outRegister, dataSend);
9    flankWR(art);
10
11   //Deshabilitamos la salida para poder usar puerto 1 como
       entrada
12   dataSend = 0;
13   writeFunction(art, outRegister, dataSend);
14
15   CD = LOW;
16   RD = LOW;
17   controlCounter(art, WR, CD, CS, RD);
18
19   //Configuramos el puerto 1 como entrada
20   inPortConfig(art, configPort1);
21
22   //Primera lectura
23   _buff[0] = readFunction(art, inRegister, 0);
24
25   RD = HIGH;
26   controlCounter(art, WR, CD, CS, RD);
27   delayMicroseconds(20);
28   RD = LOW;
29   controlCounter(art, WR, CD, CS, RD);
30
31   //Segunda lectura
32   _buff[1] = readFunction(art, inRegister, 1);
33
34   RD = HIGH;
35   controlCounter(art, WR, CD, CS, RD);
36   delayMicroseconds(20);
37   RD = LOW;
38   controlCounter(art, WR, CD, CS, RD);
39
40   //Tercera lectura
41   _buff[2] = readFunction(art, inRegister, 2);
42
43   RD = HIGH;
44   controlCounter(art, WR, CD, CS, RD);
45
46   OL = _buff[0];
47   OL |= (_buff[1] << 8);
48   OL |= (_buff[2] << 16);
49
50   //Mensaje de depuracion - VALOR CONTADORES -
51   //Serial.println(OL);
```

```
52
53    //Re-configuramos el puerto como salida
54    outPortConfig(art, configPort1);
55
56    return 0L;
57  }
```

Con la función **resetCounter(int art)** se realiza por un lado el reseteo del CNTR de los contadores, y por otro lado, se encarga de cargar un valor inicial en el Preset Register, lo que permite inicializar el CNTR. El valor inicial cargado en el registro PR es tal para que la cuenta se inicialice en un valor intermedio de forma que el Scorbot pueda girar a un lado y a otro a partir de este valor (pensando en el home del Scorbot). Este valor es el 0x800000 en he esto se debe a que el contador es de 24 bits y el número máximo de cuentas es 16777215 y cogemos la mitad de este valor que es 8388608.

```
1   //Reseteo de los contadores
2   void resetCounter(int art){
3     byte PR0 = 0x00;
4     byte PR1 = 0x00;
5     byte PR2 = 0x80;
6
7     //Reseteamos el contador (00000100)
8     dataSend = 0x04;
9     writeFunction(art, outRegister, dataSend);
10    flankWR(art);
11    delayMicroseconds(20);
12
13    //---------------------------------------
14    //Reseteamos puntero direccion PR (00000001)
15    dataSend = 0x01;
16    writeFunction(art, outRegister, dataSend);
17    flankWR(art);
18    delayMicroseconds(20);
19
20    //Envio de los valores de inicio del PR
21    writeFunction(art, outRegister, PR0);
22    flankPR(art);
23    delayMicroseconds(20);
24
25    writeFunction(art, outRegister, PR1);
26    flankPR(art);
27    delayMicroseconds(20);
28
29    writeFunction(art, outRegister, PR2);
30    flankPR(art);
31    delayMicroseconds(20);
32    //---------------------------------------
33
34    //Transferencia de datos del PR al CNTR
35    dataSend = B0001000;
36    writeFunction(art, outRegister, dataSend);
37    flankWR(art);
```

```
38    delayMicroseconds(20);
39  }
```

Para la inicialización de los contadores se ha creado la función **inicializeCounter(int art)** con la que se realiza la configuración del contador mediante los registros OCCR, QR, ICR y MCR; además, se carga un valor inicializa el CNTR mediante la carga de un valor inicial en el Preset Register.

```
1   //Inicializacion de los contadores
2   void inicializeCounter(int art){
3     byte PR0 = 0x00;
4     byte PR1 = 0x00;
5     byte PR2 = 0x80;
6
7     WR = HIGH;
8     RD = HIGH;
9     controlCounter(art, WR, CD, CS, RD);
10
11    //Reseteo GENERAL usando MCR (00100000)
12    dataSend = 0x20;
13    writeFunction(art, outRegister, dataSend);
14    flankWR(art);
15
16    //Habilitar entradas A y B usando ICR (01101001)
17    dataSend = 0x69;
18    writeFunction(art, outRegister, dataSend);
19    flankWR(art);
20    delayMicroseconds(60);
21
22    //Modo binario usando OCCR (10000000)
23    dataSend = 0x80;
24    writeFunction(art, outRegister, dataSend);
25    flankWR(art);
26    delayMicroseconds(60);
27
28    //Modo cuadratura mediante QR (11000011)
29    dataSend = 0xC3;
30    writeFunction(art, outRegister, dataSend);
31    flankWR(art);
32    delayMicroseconds(60);
33
34    //Reseteo CNTR - MCR(00000100)
35    dataSend = 0x04;
36    writeFunction(art, outRegister, dataSend);
37    flankWR(art);
38    delayMicroseconds(60);
39
40    //Reseteamos puntero direccion PR (00000001)
41    //--------------------------------------
42    dataSend = 0x01;
43    writeFunction(art, outRegister, dataSend);
44    flankWR(art);
45    delayMicroseconds(60);
```

```
46
47    //Envio de los valores de inicio del PR
48    writeFunction(art, outRegister, PR0);
49    flankPR(art);
50    delayMicroseconds(60);
51
52    writeFunction(art, outRegister, PR1);
53    flankPR(art);
54    delayMicroseconds(60);
55
56    writeFunction(art, outRegister, PR2);;
57    flankPR(art);
58    delayMicroseconds(60);
59    //--------------------------------------
60
61    //Transferencia de datos del PR al CNTR - MCR (0001000)
62    dataSend = 0x08;
63    writeFunction(art, outRegister, dataSend);
64    flankWR(art);
65    delayMicroseconds(60);
66  }
```

La función **counterSetup(int art)** se encarga de configurar los puertos 0 y 1 como salida mediante la función **outPortConfig()** de la articulación que se pasa por parámetro. Así mismo, inicializa las líneas de control y el contador.

```
1   void counterSetup(int art){
2     //Configuramos los puertos como salida inicialmente
3     outPortConfig(art, configPort0);
4     outPortConfig(art, configPort1);
5
6     //Inicializamos variables de control del contador
7     controlCounter(art, WR, CD, CS, RD);
8
9     //Inicializamos el contador
10    inicializeCounter(art);
11
12  }
```

La función **completeCounterSetup(int art1, int art2, int art3, int art4, int art5, int art6)** realiza la función **counterSetup()** de todas las articulaciones a la vez, evitando así la repetición de esta función en el código principal.

```
1   void completeCounterSetup(int art1, int art2, int art3, int
        art4, int art5, int art6){
2     counterSetup(art1);
3     counterSetup(art2);
4     counterSetup(art3);
5     counterSetup(art4);
6     counterSetup(art5);
7     counterSetup(art6);
8   }
```

7. Controlador del Scorbot-ER V

El controlador actual del que dispone el Scorbot-ER Vplus se basa en el control PID mono-articular, es decir, realiza un control PID para cada articulación del Scorbot sin tener en cuenta lo que sucede en las demás articulaciones.

El proceso de control que lleva a cabo se divide en varias etapas que van desde el cálculo de la posición y la velocidad requeridas cada 10 ms, pasando por la conversión a analógico mediante un DAC, del valor digital previamente emitido por el procesador. Este valor analógico se multiplica por dos de forma que se ajusta a lógica TTL (lógica transistor a transistor). Posteriormente la unidad analógica crea una señal continua a 20Khz.

Se lleva a cabo un control PWM (Modulación de Anchura de Impulsos). Con cada movimiento de los motores, los encoders producen una señal de impulsos que el procesador procesador lee cada 10 ms para calcular la posición y velocidad del motor. Una vez disponga de los valores reales de posición y velocidad, el procesador los compara con los deseados, determinando los errores y ejecutando una serie de acciones para eliminarlos. [3]

8. PID

8.1. Cálculo ángulos mediante Cinemática Inversa (PID).

Para llevar a cabo un control PID del Scorbot-ER Vplus es necesario la transformación de las cuentas de encoders a ángulos; operación que realiza el software desarrollado en el Trabajo de Fin de Grado de Carlos Javier Siverio.

Así mismo, para conocer los ángulos extremos de cada articulación que se requieren para dicha transformación se ha recurrido a la cinemática inversa. Los ángulos obtenidos son θ_1, θ_2 y θ_3, es decir, los que corresponden a la base y a las articulaciones del hombro y codo del Scorbot, en las que se aplicarán las acciones de control pertinentes.
Se recurre al controlador propio del Scorbot-ER V para obtener el valor del punto final (al que se ha de restar la longitud de la pinza) de cada articulación con la que se va a trabajar, para poder conocer mediante la cinemática inversa los ángulos extremos de cada una.

Las longitudes de los links son las siguientes:

- Longitud base - articulación hombro: $L_1 = 35$ cm
- Longitud articulación hombro - articulación codo: $L_2 = 22$ cm
- Longitud articulación codo - articulación muñeca: $L_3 = 22$ cm
- Longitud de la muñeca - pinza: Lm = 15 cm

Cálculo de los ángulos extremos de las articulaciones del Scorbot-ER V:

- **Cálculo ángulos extremos de la base del Scorbot (θ_1).**

  Extremo 0: Punto de referencia (X = -30,89 cm; Y = -33,9 cm; Z = 21,43 cm).

$$\theta_1 = arc\ tg\frac{y}{x} = arc\ tg\frac{-33,9}{-30,89} = 47,66^\circ$$

  Extremo 1: Punto de referencia (X = -45,31 cm; Y = 7,10 cm; Z = 21,43 cm).

$$\theta_1 = arc\ tg\frac{y}{x} = arc\ tg\frac{7,10}{-45,31} = -8,91^\circ$$

  Observamos tras varias pruebas, que para θ_1 los ángulos obtenidos varían en un rango de 0° a -90° ó de 0° a 90° por cuadrante:

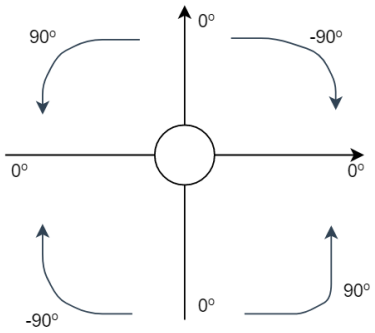


Figura 27: Ángulos θ_1

Adaptamos los ángulos extremos de θ_1 para trabajar con un rango de valores que vayan de 0° a 360°:

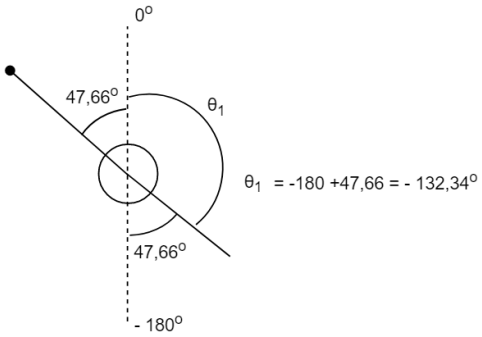


Figura 28: Ángulos θ_1 rango de 0° a 360°

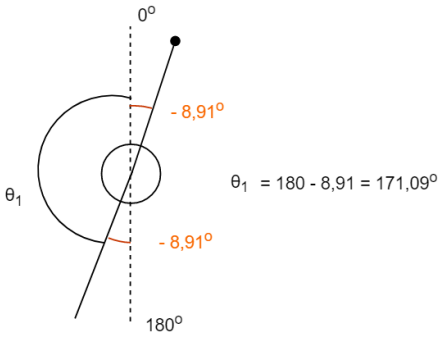


Figura 29: Ángulos θ_1 rango de 0° a 360°

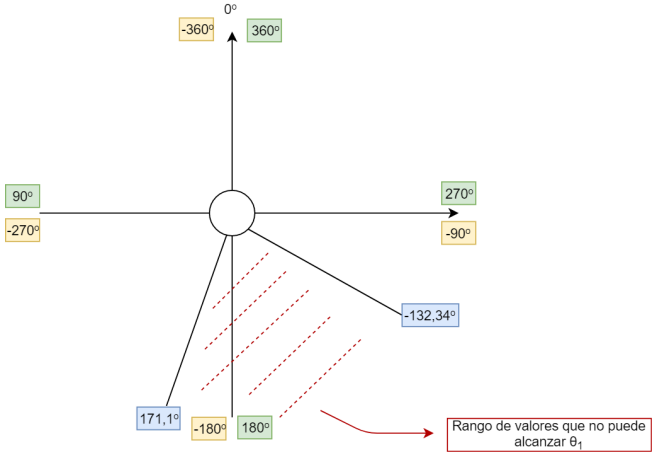


Figura 30: Ángulos θ_1 para rango de 0° a 360°

Como se puede observar en la figura 31, los ángulos límite de θ_1 son 171,1° y -132,34°.

- **Cálculo ángulos extremos del hombro del Scorbot (θ_2).**

  Extremo 0: Punto de referencia (X = 2,28 cm; Y = 57,16 cm; Z = 25,11 cm).

$$\theta_3 = arc\ cos\left(\frac{x^2 + y^2 + (z - l_1)^2 - l_2^2 - l_3^2}{2\ l_2 l_3}\right)$$

$$= arc\ cos\left(\frac{2,28^2 + (57,16 - 15)^2 + (25,11 - 35)^2 - 22^2 - 22^2}{2\ * 22 * 22}\right) = 19,5°.$$

$$\phi = arc\ tg\ \frac{z - l_1}{\sqrt{x^2 + y^2}} = arc\ tg\ \frac{25,11 - 35}{\sqrt{2,28^2 + 42,16^2}} = -13,18°.$$

$$\alpha \ = arc\ tg\ \frac{l_3\ sen\ \theta_3}{l_2 + l_3\ cos\ \theta_3}$$
$$= arc\ tg\ \frac{22\ sen\ \theta_3}{22 + 22\ cos\ \theta_3} = 9,75^\circ$$

$$\theta_2 = \phi - \alpha = -13,18 - 9,75 = -22,93^\circ.$$

Extremo 1: Punto de referencia (X = 26,03 cm; Y = 0 cm; Z = 54,31 cm).

$$\theta_3 = arc\ cos\ \left(\frac{(26,03-15)^2 + (54,31-35)^2 - 22^2 - 22^2}{2\ *22*22}\right) = 119,28^\circ.$$

$$\phi = arc\ tg\ \frac{54,31-35}{\sqrt{(26,03-15)^2}} = 60,26^\circ.$$
$$\alpha \ = arc\ tg\ \frac{l_3\ sen\ \theta_3}{l_2 + l_3\ cos\ \theta_3} = 59,64^\circ.$$

$$\theta_2 = \phi + \alpha = 60,26 + 59,64 = 119,9^\circ.$$

- **Cálculo ángulos extremos del codo del Scorbot (θ_3).**

  Extremo 0: Punto de referencia (X = 0,06 cm; Y = 36,23 cm; Z = 56,50 cm).

$$\theta_3 = arc\ cos\ \left(\frac{0,06^2 + 36,23^2 + (56,50-35)^2 - 22^2 - 22^2}{2\ *22*22}\right) = 93,26^\circ.$$

  Extremo 1: Punto de referencia (X = 0,06 cm; Y = 34,97 cm; Z = 12,62 cm).

$$\theta_3 = arc\ cos\ \left(\frac{0,06^2 + (34,97-15)^2 + (12,62-35)^2 - 22^2 - 22^2}{2\ *22*22}\right) = -94,05^\circ.$$

  (Este ángulo se obtuvo con una configuración çodo arriba", por lo que es negativo).

Estos cálculos de cinemática inversa se basan en el desarrollo teórico explicado en la otra parte en la que se divide este trabajo. [1] [4]

9. Pruebas

- Comprobación de la capacidad del controlador Arduino para mover y controlar seis motores, que es la cantidad de articulaciones que hemos de mover en el manipulador.

  Esta prueba consiste en la implementación de un progama con el que se pueda controlar simultaneamente seis servomotores con un joystick mediante un microcontrolador Arduino. Los resultados de la prueba fueron satisfactorios, pues fue posible su control y movimiento, lo nos llevo a la conlcusion de que era posible la implementación de un único Arduino.

  En el siguiente enlace se puede observar la ejecución de la prueba:
  Control 6 Servos

- Comprobación de la intensidad necesaria para que cada motor se pueda mover. La prueba consiste en la conexion directa del motor a una fuente de voltaje de 24v con la posibilidad de limitar la intensiad (para evitar se queme motor). Se mueven todos los motores del Scorbot-ER V elevando la intensidad hasta que se alcanza la velocidad adecuada, momento en el que se anota su intensidad para tenerla en cuenta a la hora de diseñar la electrónica necesaria para mover los motores.

- Control y movimiento de los motores de 5v con el dispositivo Arduino, un joystick y el módulo L298N.

  La prueba consiste en el movimiento de dos motores de pequeño consumo utilizando el módulo L298N y su librería, para comprobar verificar su funcionamiento. Dado que cumple con las prestaciones necesarias se llega a la conclusión de que es posible su implementación.

  La realización e la prueba se puede observar en el siguiente enlace:
  Control motores 5v con L298N

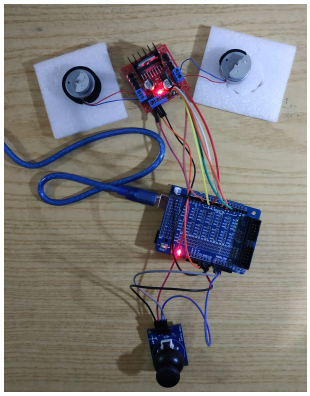


Figura 31: Movimiento de dos motores de 5v con un joystick, Arduino y módulo L298N

- Control y movimiento de los motores del Scorbot-ER V con el módulo L298N, Arduino y un joystick.

  Una vez comprobado el funcionamiento de los módulos L298N con motores de menor consumo, se lleva a cabo la prueba con los motores del Scorbot. Dado que la prueba consiste en mover motores de 24v, se ha de quitar el jumper del regulador y alimentar el módulo L298N con 5v. La ejecución de la prueba del motor de la base del Scorbot se puede visualizar en el siguiente enlace:

  Control motores Scorbot con L298N

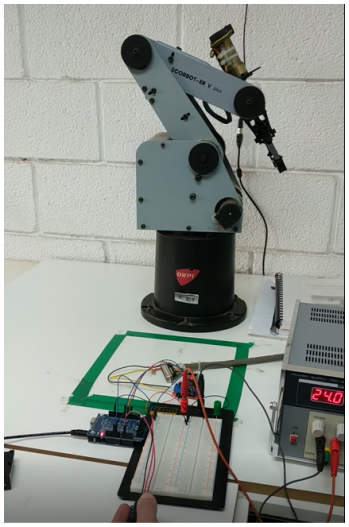


Figura 32: Movimiento de los motores del Scorbot con Arduino y módulo L298N

- Control y movimiento de los motores del Scorbot-ER V sin joystick.

  Una vez comprobado que es posible controlarlos con el joystick, probamos a controlarlos vía software, ya que lo que se quiere es controlar los motores directamente desde el microcontrolador Arduino.

- Lectura de las señales de los encoders de los motores del Scorbot mediante controladores I2C y contadores.

  Inicialmente se realiza la prueba con el motor de la base del Scorbot. Ésta consiste en la ejecución del programa de Lectura de contadores que se explica con antelación, en el apartado de "Descripción de la interfaz informática", y la comprobación de la correcta lectura de los contadores cuando se mueven los motores.

  Recurrimos a un osciloscopio digital para la visualización simultánea de las señales del encoder cuando la prueba no daba los resultados esperados, para verificar si dichas señales llegaban correctamente al contador. Comprobación señales encoder

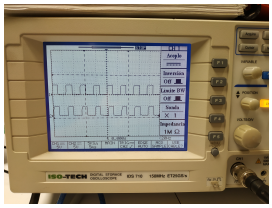


Figura 33: Comprobación de la lectura de los contadores

- Lectura de los contadores del resto de motores del Scorbot-ER V.

  De forma previa a la realización de esta prueba se han de colocar todos los componentes y sus conexiones, lo que implica soldar los pines de las placas con los controladores I2C y los cables que van en el adaptador del conector D50.

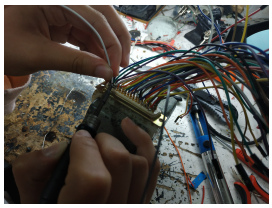


Figura 34: Soldadura de los cables en adaptador del conector D50

- Prueba de la utilización del amplificador TL08x para la gestión de alarmas mediante un circuito electrónico externo. Se ha utilizado el siguiente circuito:

  En este caso se conecta en la patilla inversora un voltaje de 3,23V, mientras que en la patilla no inversora tendremos unos 5V conectados a un

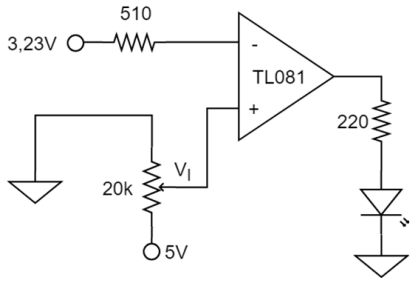


Figura 35: Circuito para la prueba de la gestión alarmas

potenciómetro. Entonces, dependiendo del valor que le demos a la resistencia tendremos $V_I \geq 3,23V$ y se encenderá el LED. Si por el contrario $V_I 3,23V$ el LED permanecerá apagado.

Una vez que funciona correctamente la lectura de los contadores, se procede a la ejecución de la instrucción home en los motores Scorbot-ER V.

Estas pruebas consisten en la puesta del Scorbot en su posición de reposo mediante la instrucción home.

Cabe destacar que dado que hasta el momento no se ha implementado un control sobre el manipulador, el brazo del mismo cae por inercia (no es capaz de vencer la fuerza de la gravedad), teniendo que ser sujetado para la correcta finalización del home en un instante determinado.

- Prueba de la instrucción home en el motor de la base del Scorbot. En el siguiente enlace se puede visualizar esta prueba:

  Home motor base Scorbot

- Prueba de la instrucción home en el motor de la base y en el la articulación hombro del Scorbot de manera secuencial.

  Home hombro Scorbot

- Prueba de la instrucción home en la articulación codo del Scorbot.

  - Home codo Scorbot
  - Home codo Scorbot (1)

- Prueba de la instrucción home en la articulación muñeca (movimiento de cabeceo y de alabeo) y en la pinza del Scorbot:
  Home muñeca y pinza Scorbot

- Prueba de la instrucción home en la pinza del Scorbot. Esta prueba se puede observar en el siguiente enlace: Home pinza Scorbot

- Prueba de la instrucción home de todos los motores del Scorbot de manera secuencial. Esta prueba se ha realizado colocando el brazo manipulador en diferentes posiciones:

  - Home completo primera configuración:
    Home completo primera configuración
  - Home completo segunda configuración:
    Home completo segunda configuración
  - Home completo tercera configuración:
    Home completo tercera configuración

- Prueba de la instrucción home de la articulación 3 (codo del Scorbot) con la aplicación de un control PID en la misma para mantenerlo en su posición de reposo, es decir, para lograr vencer la fuerza de la gravedad que lo hacía caer dificultando así la correcta consecución del home. En el video que se incluye a continuación, se puede apreciar como va aumentando la acción integral hasta hacer llegar a la articulación el voltaje que necesita para llegar y mantenerse en su posición de reposo. Además, al final del video se puede observar como el link de la articulación del codo se cae una vez se deja de aplicar la acción de control.

  Así mismo, se prueba la instrucción "move" con la misma articulación.

  Home articulación - codo + PID

- Prueba de la instrucción home en todas las articulaciones incluyendo la acción de control en la articulación 3 del Scorbot.

  Home de todas las articulaciones + PID en la articulación 3

- Prueba de la instrucción home en todas las articulaciones incluyendo acción de control en las articulaciones correspondientes al hombro y codo del Scorbot.

  Estas acciones de control se pueden apreciar en el siguiente enlace:

  Home de todas las articulaciones + PID en la articulación 2 y 3

- Prueba de la instrucción "move" en la base del Scorbot-ER Vplus. Previamente, y mediante las intrucciones "defp" y "here" definimos las posiciones a las que queremos que se mueva la base del Scorbot.

  Move para mover base del Scorbot a una posición determinada

10. Dificultades encontradas

- Una de las dificultades la encontramos a la hora de utilizar los módulos L298N.

  - En primer lugar recurrimos a motores de continua de 5v para verificar el funcionamiento de los mismos utilizando uno del que ya disponíamos. Este primer módulo funcionaba y movía los motores sin problemas; sin embargo, cuando compramos y probamos los demás módulos L298N no funcioban para los mismos motores de 5v. Entonces, se cambiaron los módulos y se volvieron a probar sin éxito.
  - Posteriormente, se probaron de nuevo los mismos módulos pero con un motor de 12v (quitando el jumper de regulación y alimentando el módulo con 5v) y se logró mover el motor.

  - Parece que los módulos obtenidos posteriormente estaban defectuosos y no les llegaba la tensión mínima de la que precisan para funcionar sin alimentación externa (mirando el datasheet ésta es Vih +2,5 = 4.8v).
  - Dado que estos módulos sí funcionan con alimentación externa y quitando el jumper del regulador, que es la manera en la que debíamos usarlos con los motores del Scorbot ya que son de 24v, decidimos verificar su funcionamiento probando a moverlos. La prueba tuvo un resultado positivo, por lo que decidimos utilizarlos.

- Los componentes que necesitabamos para la implementación de la electrónica no estaban dispobles en las diferentes tiendas de electrónica de la isla, por lo que tuvimos que pedirlos por internet, lo que suponía esperar hasta que llegasen para poder realizar pruebas con todos los componentes.

  Durante el tiempo que tardaron en llegar los contadores (tardaron un mes) estuvimos probando con un contador LS7166 del que disponíamos y con uno de los controladores I2C el código de lectura de los contadores.

- Otro de los inconvenientes era que los I2C que necesitabamos son de tecnología de montaje superficial (SMD) por lo que no podíamos colocarlos directamente el las protoboards.

  Como solución, el Servicio de Electrónica de la Universidad realizó unas placas de circuito impreso para poder conectar los controladores I2C en la protoboard y los soldaron a las mismas.

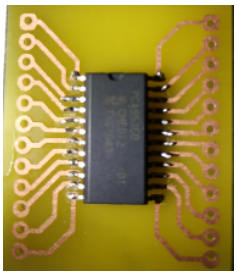


Figura 36: Placa I2C

- La programación del código de lectura de los contadores no lograbamos que funcionase.

  - Por un lado, había incluido en el byte de control MCR dos acciones que se tenían que hacer por separado (el reseteo del CNTR no se hace solo en con el reseteo general) pues había entendido que había que realizar el general a la misma vez que el reseteo del CNTR, pero no, posteriormente comprobamos que era necesario resetearlos por separado (aunque el reseteo general incluya el reseteo del CNTR).
  - Además, se planteó usar un puerto para lectura y otro para escritura del controlador I2C, lo que generaba problemas al realizar la lectura, pues el puerto de escritura mantenía sus valores activos (se guardan

en los registros y se mantienen a menos que se deshabiliten). Razón por la cual se leía el último valor que se enviaba. Este problema se solventó utilizando un mismo puerto escritura y lectura, teniendo en cuenta que la salida ha de deshabilitarse una vez se haya realizado la escirtura. Esto permitió, además, la utilización del puerto 0 para la líneas de control.

- Otro problema surgió con el adaptador del conector D50 del Scorbot-ER V, pues algunos de los cables previamente soldados, al colocarlos en la protoboard se caían y había que volver a soldarlos.

- Primero se realizaron las pruebas de lectura de los contadores sin resistencias pull-up en las entradas del contador donde de conectan las salidas del encoder puesto que se pensó que las resistencias eran internas (que estaban situadas dentro del scorbot). Al no funcionar colocamos resistencias de 4,7KΩ, pero tampoco daba resultado.

  Comprobamos de nuevo el diseño del proyecto en el que se basa nuestro trabajo y probamos las resistencias que se usan en el mismo (de 100Kω), que previamente habíamos interpretado como internas, verificando que con éstas sí se realizaba de manera correcta la lectura.

- Otro problema con el que nos encontrabamos radica en el link de la articulación 3 (codo) del Scorbot, que se cae por su propia inercia; lo que dificulta la correcta ejecución de la instrucción home de la muñeca del Scorbot. Finalmente, mediante la implementación de un control PID en el mismo, se ha logrado mantener para poder realizar la instrucción home de todas las articulaciones de forma correcta.

- A la hora de trabajar con los PID de la librería PID_v1 de Brett Beauregard nos percatamos que al introducir los valores de cuenta como consigna el sistema no reaccionaba como debía, esto era porque las variables con las que trabaja la librería son de tipo 'double', las cuales tienen una mayor precisión con los decimales, pero no nos permite almacenar la cantidad de bytes que se almacenan en un 'int long'.

  Debido a esto, se propuso trabajar con ángulos en vez de con cuentas, por lo que se desarrollo una función que interpolase entre los valores límites de cuenta y los valores límites de ángulos de cada articulación. Para ello fue necesaria la utilización del otro Scorbot-ER V Plus del laboratorio, que utilizamos junto con su controlador para buscar esas posiciones límite y calcular el ángulo mediante cinemática inversa. Para obtener los valores límite de cuenta se llevaron las articulaciones al extremo y se midieron con Arduino, que en ese punto era capaz de utilizar la instrucción 'listpv' para devolvernos el valor de cuenta.

11. Mejoras futuras

- Una mejora importante sería la realización de un circuito impreso que contenga todos los componentes y conexiones de nuestra implementación electrónica, de forma que quede más compacto el circuito y sea más cómodo trabajar con el mismo. Eliminando, además, la posibilidad de errores debidos a que se desconecte alguno de los cables.
- Respecto al adaptador del conector D50, habría que soldarlos y posteriormente aplicarle una capa de silicona para que las conexiones queden bien colocadas de tal forma que no se puedan caer al colocarlas en la protoboard.
- Se podría hacer una interfaz de usuario (GUI).
- Respecto al código se podría optimizar mediante el uso de clases e incluso se podrían crear librerías propias.
- Se podrían añadir más comandos al programa realizado.
- Desarrollo de un controlador PID para las articulaciones 4 y 5, que requieren más tiempo, ya que se trabaja simultaneamente con dos motores, lo que se traduce a trabajar con dos valores de cuenta.
- Desarrollo de un control PID que permita el movimiento simultaneo de todas las articulaciones, ya que el que se encuentra implementado actualmente solo permite mover los motores individualmente y de forma secuencial.
- Se podría desarrollar un control más eficiente que el PID monoarticular, tal como PID con compensación de gravedad.
- Otra posible mejora es la optimización de la instrucción "home", llevando todas las articulaciones a su posición de reposo tal y como hace el propio controlador del Scorbot-ER V.

12. Conclusiones

Mediante la ejecución de este Trabajo de Fin de Grado hemos aprendido a trabajar con el bus I2C, a programar en Arduino y Phyton e incluso a soldar otro tipo de componentes.Así mismo, Nos hemos familiarizado con los datasheets de los componentes y con los brazos manipuladores.

Además, hemos aprendido lo que es implica planificar un proyecto y que no todo sale como uno tiene previsto, pues surgen inconvenientes hemos ido solventando.

Se ha logrado alcanzar gran parte del objetivo general, pues aunque no se ha desarrollado un control dinámico para el Scorbot, sí que se ha llevado a cabo una electrónica que permitirá realizarlo a futuros alumnos que continúen con este TFG.

Para alcanzar este objetivo, se han tenido que realizar previamente los específicos, tales como mover los motores con ayuda de los módulos L298N, realizar la cuenta de las señales que salen de los encoders de cada motor y la lectura de la misma, establecer la comunicación entre Arduino y Scorbot, llevar a cabo la instrucción home.

Aunque no se hayan conseguido finalizar dos de los objetivos específicos, se ha trabajado sobre estos, lo que nos ha llevado a un proceso de aprendizaje sobre los mismos.

El desarollo de este trabajo nos ha permitido acercarnos un poco más a la realidad del trabajo de un ingeniero.

13. Conclusions

By producting this project we have learned to work with the I2C bus, to program in Arduino and Phyton, even to weld other types of components. We have become familiar with the datasheets of the components and we also learned about the Scorbot-ER V.

Besides, we have learned what it is to plan a project in which not everything goes as planned, as problems have arised that we then have had to solve.

A great part of the general objective has been achieved because, although a dynamic control has not been developed for the Scorbot an electronics circuit has been carried out that will allow it to be completed by future students who continue with this TFG.

In order to achieve this objective, we have had to previously carry out the specifics ones, such as moving the motors with the help of the L298N modules, making the count of the signals coming out of the encoders of each motor and reading it, establishing the communication between Arduino and Scorbot, carry

out the home instruction.

Even although two of the specific objectives have not been finalized, we have worked on them, which has led us to a process of learning about them.

This work have allowed us to approach the reality of the work of an engineer.

14. Presupuesto

Material/Componente	Cantidad	Precio unidad	Precio total/material
Arduino MEGA 2560	1	15,98 €	15,98 €
Módulos L298N	3	4,95 €	14,85 €
LS7166	6	7,15 €	42,90 €
PCA9535	6	1,55 €	9,30 €
Cables Macho-Macho	2	3,25 €	6,50 €
Cables Macho-Hembra	1	3,25 €	3,25 €
Protoboards dobles	4	10 €	40 €
Precio total			132,78 €

	Horas	Precio/hora	Precio total
Mano de obra	170	15€/h	2550€

Concepto		Importe
Proyecto		2682,78 €
Beneficio Industrial	(+6 %)	7,97 €
Total proyecto		2823,53 €
IGIC	(+0.65 %)	18,35 €
Total, IGIC incluido		2841,88 €

15. Bibliografía

Referencias

[1] R. Miranda Colorado, *Cinemática y Dinámica de Robots Manipuladores*, primera edición. Marcombo, 2016.

[2] SCORBOT-ER Vplus - Manual de Usuario

[3] Controlador-A - Manual de Usuario

[4] Apuntes de la asignatura Sistemas Robotizados

[5] J. V. Méndez Bethencourt, *Programación y puesta en marcha de la electrónica de control de un robot manipulador*, proyecto de fin de carrera, Enero 2006.

[6] http://arduino.cl/arduino-2/

[7] https://www.youtube.com/watch?v=c0L4gNKwjRw

[8] http://proyectosconarduino.com/modulos/motor-driver-l298n/

[9] https://www.luisllamas.es/arduino-i2c/

[10] http://robots-argentina.com.ar/didactica/descripcion-y-funcionamiento-del-bus-i2c/

[11] http://www.uco.es/~el1mofer/Docs/IntPerif/Bus%20I2C.pdf

[12] https://www.arduino.cc/en/Reference/Wire

[13] https://github.com/AndreaLombardo/L298N

16. Anexos

16.1. Programación - Códigos

16.1.1. artDirection.h

```
1
2  /*----------------------------------------
3   * Ana Estevez Perez
4   * 01/05/2019
5   * Declaracion direcciones I2C
6  ----------------------------------------*/
7
8  int art1 = 0x20;
9  int art2 = 0x21;
10 int art3 = 0x22;
11 int art4 = 0x23;
12 int art5 = 0x24;
13 int art6 = 0x25;
```

16.1.2. counter.h

```
1
2  /*----------------------------------------
3   * Ana Estevez Perez
4   * 01/05/2019
5   * Definicion funciones I2C + Contador
6  ----------------------------------------*/
7
8  void completeCounterSetup(int art1, int art2, int art3, int
       art4, int art5, int art6);
9
10 void inicializeCounter(int art);
11 void resetCounter(int art);
12 long int readCounter(int art);
```

16.1.3. counter.cpp

```
1
2
3    /*----------------------------------------
4   * Ana Estevez Perez
5   * 01/05/2019
6   * Declaracion funciones I2C + Contador
7  ----------------------------------------*/
8
9  #include"Arduino.h"
10 #include "counter.h"
11 #include<Wire.h>
```

```
12
13  //Direcciones de los REGISTROS DE CONTROL
14  byte outControlRegister = 0x02; //Registro de salida Puerto
        0
15  byte outRegister = 0x03; //Registro de salida Puerto 1
16  byte inRegister = 0x01;  //Registro de entrada Puerto 1
17  byte configPort0 = 0x06;  //Registro de configuracion Puerto
        0
18  byte configPort1 = 0x07;  //Registro de configuracion Puerto
        1
19
20  //Variables globales
21  int long OL = 0;
22  int long _buff[2];
23
24  //Variables de configuracion de puertos
25  byte output = 0x00;
26  byte input = 0xff;
27
28  //Datos enviados al I2C
29  byte dataSend = 0x00;
30
31  //Declaracion de variables auxiliares de CONTROL
32  bool WR = LOW;
33  bool CD = HIGH;
34  bool CS = LOW;
35  bool RD = LOW;
36
37  //Funcion de escritura para el I2C
38  void writeFunction(int art, byte outRegister, byte dataSend)
        {
39    Wire.beginTransmission(art);
40    Wire.write(outRegister);
41    Wire.write(dataSend);
42    Wire.endTransmission();
43  }
44
45  //Funcion de lectura para el I2C
46  long int readFunction(int art, byte inRegister, int i){
47    Wire.beginTransmission(art);
48    Wire.write(inRegister);
49    Wire.endTransmission(false);
50
51    Wire.beginTransmission(art);
52    Wire.requestFrom(art, 1);
53    while(Wire.available()){
54      _buff[i] = Wire.read();
55    }
56    Wire.endTransmission();
57
58    return _buff[i];
59  }
60
61  //Funcion de configuracion del puerto como entrada
```

```
62  void inPortConfig(int art, byte configPort){
63    writeFunction(art, configPort, input);
64    delayMicroseconds(20);
65  }
66
67  //Funcion de configuracion del puerto como salida
68  void outPortConfig(int art, byte configPort){
69    writeFunction(art, configPort, output);
70    delayMicroseconds(20);
71  }
72
73  //Funcion CONTROL del contador
74  void controlCounter(int art, bool WR, bool CD, bool CS, bool
        RD){
75    dataSend = 0;
76    dataSend = WR;
77    dataSend = (dataSend << 1) + CD;
78    dataSend = (dataSend << 1) + CS;
79    dataSend = (dataSend << 1) + RD;
80    writeFunction(art, outControlRegister, dataSend);
81    delayMicroseconds(20);
82  }
83
84  //Funcion para enviar los registros de control
85  void flankWR(int art){
86    RD = HIGH;
87    CD = HIGH;
88
89    delayMicroseconds(20);
90    WR = LOW;
91    controlCounter(art, WR, CD, CS, RD);
92    delayMicroseconds(20);
93    WR = HIGH;
94    controlCounter(art, WR, CD, CS, RD);
95    delayMicroseconds(20);
96  }
97
98  //Funcion para acceder a los registros de datos
99  void flankPR(int art){
100   RD = HIGH;
101   CD = LOW;
102
103   delayMicroseconds(20);
104   WR = LOW;
105   controlCounter(art, WR, CD, CS, RD);
106   delayMicroseconds(20);
107   WR = HIGH;
108   controlCounter(art, WR, CD, CS, RD);
109   delayMicroseconds(20);
110 }
111
112 //Lectura de los contadores
113 long int readCounter(int art){
114   //Re-configuramos el puerto como salida
```

```
115     outPortConfig(art, configPort1);
116
117     //Reseteamos puntero OL y transferimos el CNTR a OL
            (00000011)
118     dataSend = 0x03;
119     writeFunction(art, outRegister, dataSend);
120     flankWR(art);
121
122     //Deshabilitamos la salida
123     dataSend = 0;
124     writeFunction(art, outRegister, dataSend);
125
126     CD = LOW;
127     RD = LOW;
128     controlCounter(art, WR, CD, CS, RD);
129
130     //Re-configuramos el puerto como entrada
131     inPortConfig(art, configPort1);
132
133     //Primera lectura
134     _buff[0] = readFunction(art, inRegister, 0);
135
136     RD = HIGH;
137     controlCounter(art, WR, CD, CS, RD);
138     delayMicroseconds(20);
139     RD = LOW;
140     controlCounter(art, WR, CD, CS, RD);
141
142     //Segunda lectura
143     _buff[1] = readFunction(art, inRegister, 1);
144
145     RD = HIGH;
146     controlCounter(art, WR, CD, CS, RD);
147     delayMicroseconds(20);
148     RD = LOW;
149     controlCounter(art, WR, CD, CS, RD);
150
151     //Tercera lectura
152     _buff[2] = readFunction(art, inRegister, 2);
153
154     RD = HIGH;
155     controlCounter(art, WR, CD, CS, RD);
156
157     OL = _buff[0];
158     OL |= (_buff[1] << 8);
159     OL |= (_buff[2] << 16);
160
161     //Mensaje de depuracion - VALOR CONTADORES -
162     //Serial.println(OL);
163
164     //Re-configuramos el puerto como salida
165     outPortConfig(art, configPort1);
166
167     return OL;
```

```
168   }
169
170   //Reseteo de los contadores
171   void resetCounter(int art){
172     byte PR0 = 0x00;
173     byte PR1 = 0x00;
174     byte PR2 = 0x80;
175
176     //Reseteamos el contador (00000100)
177     dataSend = 0x04;
178     writeFunction(art, outRegister, dataSend);
179     flankWR(art);
180     delayMicroseconds(20);
181
182     //--------------------------------------
183     //Reseteamos puntero direccion PR (00000001)
184     dataSend = 0x01;
185     writeFunction(art, outRegister, dataSend);
186     flankWR(art);
187     delayMicroseconds(20);
188
189     //Envio de los valores de inicio del PR
190     writeFunction(art, outRegister, PR0);
191     flankPR(art);
192     delayMicroseconds(20);
193
194     writeFunction(art, outRegister, PR1);
195     flankPR(art);
196     delayMicroseconds(20);
197
198     writeFunction(art, outRegister, PR2);;
199     flankPR(art);
200     delayMicroseconds(20);
201     //--------------------------------------
202
203     //Transferencia de datos del PR al CNTR
204     dataSend = B0001000;
205     writeFunction(art, outRegister, dataSend);
206     flankWR(art);
207     delayMicroseconds(20);
208   }
209
210   //Inicializacion de los contadores
211   void inicializeCounter(int art){
212     byte PR0 = 0x00;
213     byte PR1 = 0x00;
214     byte PR2 = 0x80;
215
216     WR = HIGH;
217     RD = HIGH;
218     controlCounter(art, WR, CD, CS, RD);
219
220     //Reseteo GENERAL usando MCR (00100000)
221     dataSend = 0x20;
```

```
222     writeFunction(art, outRegister, dataSend);
223     flankWR(art);
224 
225     //Habilitar entradas A y B usando ICR (01101001)
226     dataSend = 0x69;
227     writeFunction(art, outRegister, dataSend);
228     flankWR(art);
229     delayMicroseconds(60);
230 
231     //Modo binario usando OCCR (10000000)
232     dataSend = 0x80;
233     writeFunction(art, outRegister, dataSend);
234     flankWR(art);
235     delayMicroseconds(60);
236 
237     //Modo cuadratura mediante QR (11000011)
238     dataSend = 0xC3;
239     writeFunction(art, outRegister, dataSend);
240     flankWR(art);
241     delayMicroseconds(60);
242 
243     //Reseteo CNTR - MCR(00000100)
244     dataSend = 0x04;
245     writeFunction(art, outRegister, dataSend);
246     flankWR(art);
247     delayMicroseconds(60);
248 
249     //Reseteamos puntero direccion PR (00000001)
250     //--------------------------------------
251     dataSend = 0x01;
252     writeFunction(art, outRegister, dataSend);
253     flankWR(art);
254     delayMicroseconds(60);
255 
256     //Envio de los valores de inicio del PR
257     writeFunction(art, outRegister, PR0);
258     flankPR(art);
259     delayMicroseconds(60);
260 
261     writeFunction(art, outRegister, PR1);
262     flankPR(art);
263     delayMicroseconds(60);
264 
265     writeFunction(art, outRegister, PR2);;
266     flankPR(art);
267     delayMicroseconds(60);
268     //--------------------------------------
269 
270     //Transferencia de datos del PR al CNTR - MCR (0001000)
271     dataSend = 0x08;
272     writeFunction(art, outRegister, dataSend);
273     flankWR(art);
274     delayMicroseconds(60);
275   }
```

```
276
277 void counterSetup(int art){
278   //Configuramos los puertos como salida inicialmente
279   outPortConfig(art, configPort0);
280   outPortConfig(art, configPort1);
281
282   //Inicializamos variables de control del contador
283   controlCounter(art, WR, CD, CS, RD);
284
285   //Inicializamos el contador
286   inicializeCounter(art);
287 }
288
289 void completeCounterSetup(int art1, int art2, int art3, int
      art4, int art5, int art6){
290   counterSetup(art1);
291   counterSetup(art2);
292   counterSetup(art3);
293   counterSetup(art4);
294   counterSetup(art5);
295   counterSetup(art6);
296 }
```

17. Datasheets

LSI/CSI

LSI Computer Systems, Inc. 1235 Walt Whitman Road, Melville, NY 11747 (631) 271-0400 FAX (631) 271-0405

LS7166

December 2002

24-BIT QUADRATURE COUNTER

FEATURES:

- Programmable modes are: Up/Down, Binary, BCD, 24 Hour Clock, Divide-by-N, x1 or x2 or x4 Quadrature and Single Cycle.
- DC to 20 MHz Count Frequency.
- 8-Bit I/O Bus for Microprocessor Communication and Control.
- 24-Bit comparator for pre-set count comparison.
- Readable status register.
- Input/Output TTL and CMOS compatible.
- 5 Volt operation (Vdd - Vss).
- LS7166 (DIP); LS7166-S (SOIC); LS7166-TS24 (24-Pin TSSOP)* - See Fig. 1

GENERAL DESCRIPTION:

The LS7166 is a CMOS, 24-bit counter that can be programmed to operate in several different modes. The operating mode is set up by writing control words into internal control registers (see Figure 8). There are three 6-bit and one 2-bit control registers for setting up the circuit functional characteristics. In addition to the control registers, there is a 5-bit output status register (OSR) that indicates the current counter status. The IC communicates with external circuits through an 8-bit three state I/O bus. Control and data words are written into the LS7166 through the bus. In addition to the I/O bus, there are a number of discrete inputs and outputs to facilitate instantaneous hardware based control functions and instantaneous status indication.

REGISTER DESCRIPTION:

Internal hardware registers are accessible through the I/O bus (D0 - D7) for READ or WRITE when CS = 0. The C/D input selects between the control registers (C/D = 1) and the data registers (C/D = 0) during a READ or WRITE operation. (See Table 1)

20-Pin Package PIN ASSIGNMENT - Top View

*(Contact factory for 24-Pin TSSOP Package Pinout)

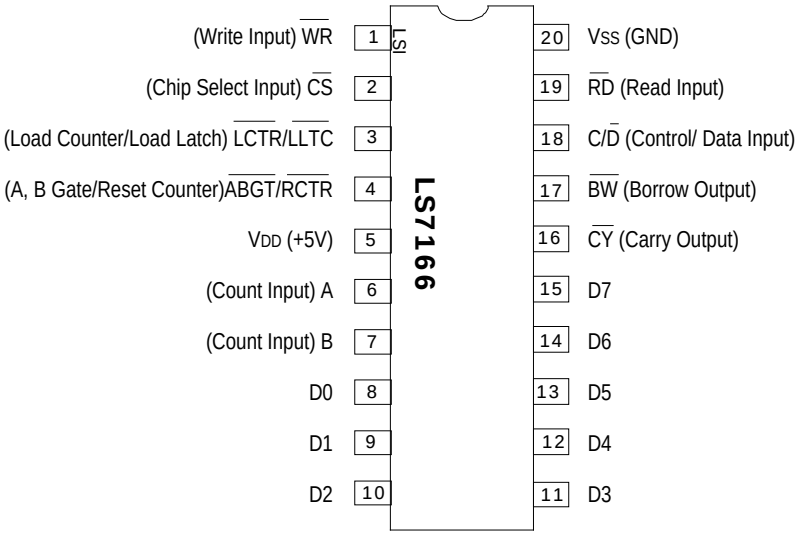


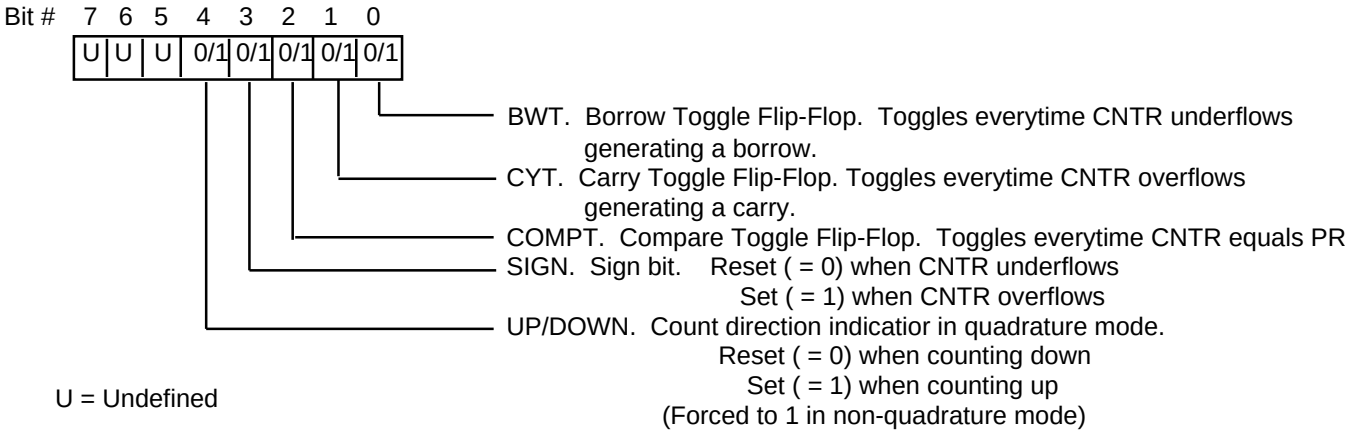
FIGURE 1

TABLE 1 - Register Addressing Modes

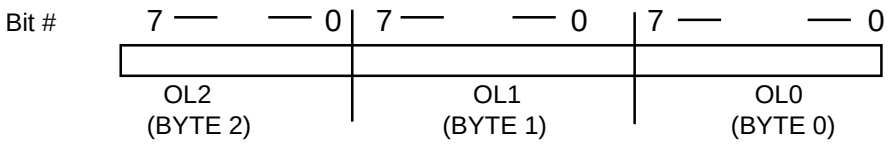
D7	D6	C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	COMMENT
X	X	X	X	X	1	Disable Chip for READ/WRITE
0	0	1	1	⊓⊔⊓	0	Write to Master Control Register (MCR)
0	1	1	1	⊓⊔	0	Write to input control register (ICR)
1	0	1	1	⊓⊔	0	Write to output/counter control register (OCCR)
1	1	1	1	⊓⊔⊓	0	Write to quadrature register (QR)
X	X	0	1	⊓⊔	0	Write to preset register (PR) and increment register address counter.
X	X	0	⊓⊔	1	0	Read output latch (OL) and increment register address counter
X	X	1	⊓⊔⊓	1	0	Read output status register (OSR).

X = Don't Care

OSR (Output Status Register). Indicates CNTR status: Accessed by**:** READ when C/$\overline{D}$ = 1, $\overline{CS}$ = 0.



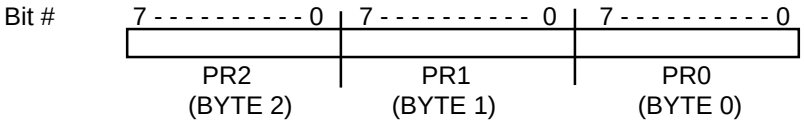
OL(Output latch). The OL is the output port for the CNTR. The 24 bit CNTR Value at any instant can be accessed by performing a CNTR to OL transfer and then reading the OL in 3 READ cycle sequence of Byte 0 (OL0), Byte 1 (OL1) and Byte 2 (OL2). The address pointer for OL0/OL1/OL2 is automatically incremented with each READ cycle. Accessed by**:** READ when C/$\overline{D}$ = 0, $\overline{CS}$ = 0.



Standard Sequence for Loading and Reading OL**:**

```
3 ⟶ MCR     ; Reset OL address pointer and Transfer CNTR to OL
READ OL     ; Read Byte 0 and increment address
READ OL     ; Read Byte 1 and increment address
READ OL     ; Read Byte 2 and increment address
```

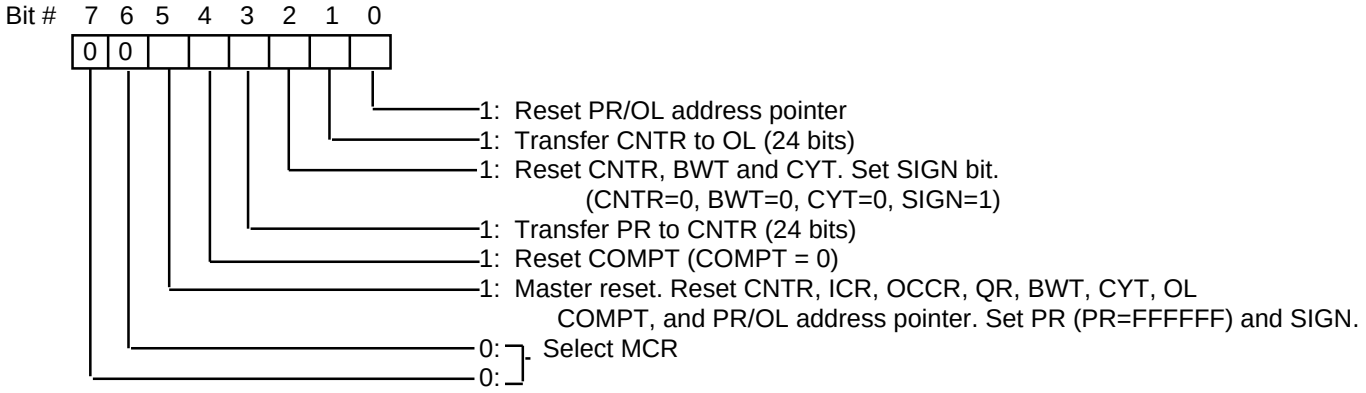
PR (Preset register). The PR is the input port for the CNTR. The CNTR is loaded with a 24 bit data via the PR. The data is first written into the PR in 3 WRITE cycle sequence of Byte 0 (PR0), Byte 1 (PR1) and Byte 2 (PR2).
The address pointer for PR0/PR1/PR2 is automatically incremented with each write cycle.
Accessed by**:** WRITE when C/$\overline{D}$ = 0, $\overline{CS}$ = 0.



Standard Sequence for Loading PR and Reading CNTR**:**

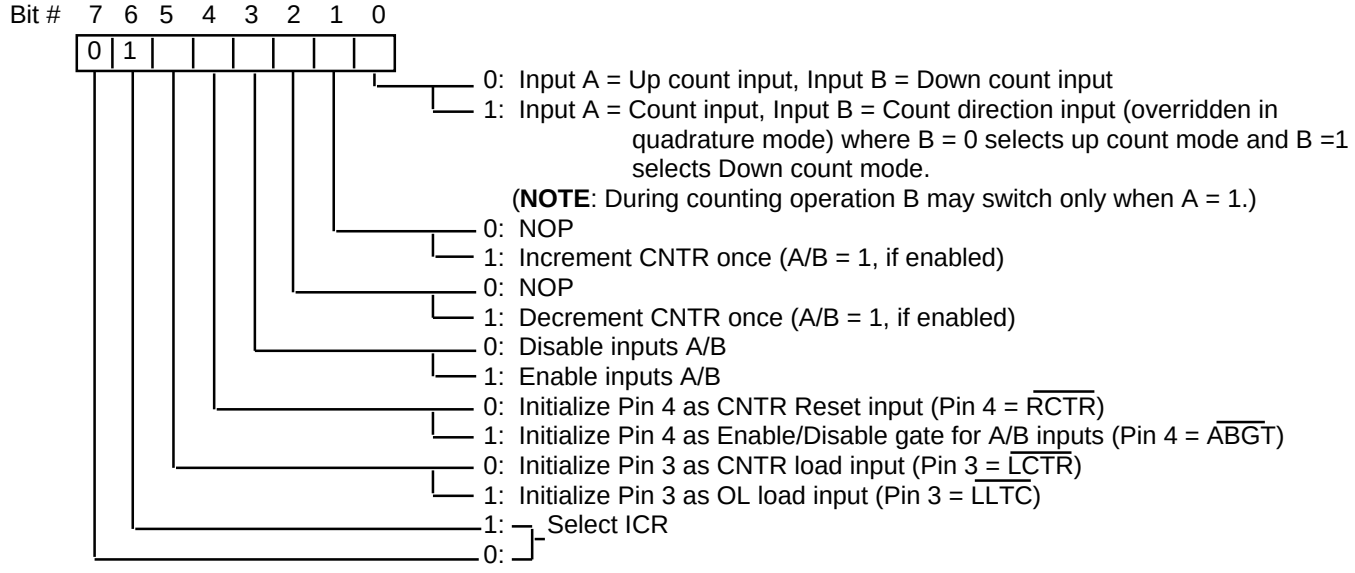
```
1 ⟶MCR      ;  Reset PR address pointer
WRITE PR    ;  Load Byte 0 and into PR0 increment address
WRITE PR    ;  Load Byte 1 and into PR1 increment address
WRITE PR    ;  Load Byte 2 and into PR3 increment address
8 ⟶MCR      ;  Transfer PR to CNTR
```

MCR (Master Control Register). Performs register reset and load operations. Writing a "non-zero" word to MCR does not require a follow-up write of an "all-zero" word to terminate a designated operation.
Accessed by**:** WRITE when C/$\overline{D}$ = 1, $\overline{CS}$ = 0.



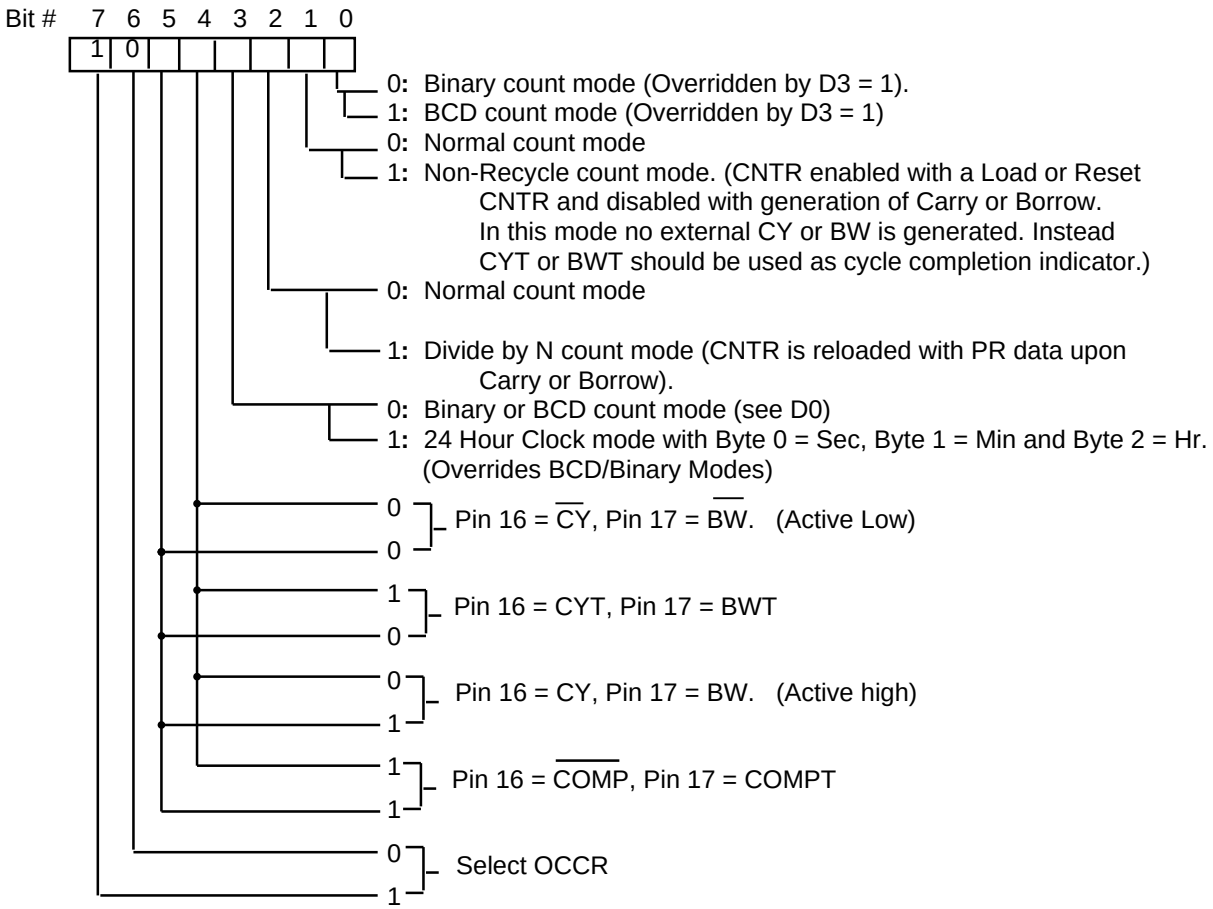
NOTE: Control functions may be combined.

ICR (Input Control Register). Initializes counter input operating modes.
Accessed by**:** WRITE when C/$\overline{D}$ = 1, $\overline{CS}$ = 0.

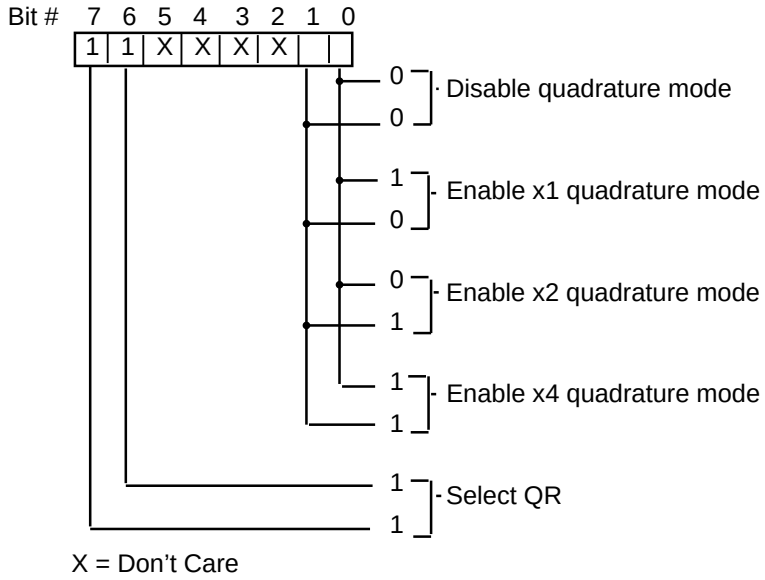


NOTE: Control functions may be combined.

OCCR (Output Control Register) Initializes CNTR and output operating modes.
Accessed by **:** WRITE when C/$\overline{D}$ = 1, $\overline{CS}$ = 0.



QR (Quadrature Register). Selects quadrature count mode (See Fig. 7)
Accessed by**:** WRITE when C/$\overline{D}$ = 1, $\overline{CS}$ = 0.



X = Don't Care

I/O DESCRIPTION:
(See REGISTER DESCRIPTION for I/O Prgramming.)

Data-Bus (D0-D7) (Pin 8-Pin 15). The 8-line data bus is a three-state I/O bus for interfacing with the system bus.
CS (Chip Select Input) (Pin 2). A logical "0" at this input enables the chip for Read and Write.
RD (Read Input) (Pin 19). A logical "0" at this input enables the OSR and the OL to be read on the data bus.
WR (Write Input) (Pin 1) A logical "0" at this input enables the data bus to be written into the control and data registers.
C/D (Control/Data Input) (Pin 18). A logical "1" at this input enables a control word to be written into one of the four control registers or the OSR to be read on the I/O bus. A logical "0" enables a data word to be written into the PR, or the OL to be read on the I/O bus.
A (Pin 6). Input A is a programmable count input capable of functioning in three different modes, such as up count input, down count input and quadrature input.
B (Pin 7). Input B is also a programmable count input that can be programmed to function either as down count input, or count direction control gate for input A, or quadrature input. When B is programmed as count direction control gate, B = 0 enables A as the Up Count input and B = 1 enables A as the Down Count input. When programmed as the direction input, B can switch state only when A is high.

$\overline{ABGT}$/$\overline{RCTR}$ (PIN 4). This input can be programmed to function as either inputs A and B enable gate or as external counter reset input. A logical "0" is the active level on this input.

In non-quadrature mode, if Pin 4 is programmed as A and B enable gate input, it may switch state only when A is high (if A is clock and B is direction) or when both A and B are high (if A and B are clocks. In quadrature mode, if Pin 4 is programmed as A and B enable gate, it may switch state only when either A or B switches.

$\overline{LCTR}$/$\overline{LLTC}$ (PIN 3). This input can be programmed to function as the external load command input for either the CNTR or the OL. When programmed as counter load input, the counter is loaded with the data contained in the PR. When programmed as the OL load input, the OL is loaded with data contained in the CNTR. A logical "0" is the active level on this input.
CY (Pin 16). This output can be programmed to serve as one of the following:
A. $\overline{CY}$. Complemented Carry out (active "0").
B. CY. True Carry out (active "1").
C. CYT. Carry Toggle flip-flop out.
D. $\overline{COMP}$. Comparator out (active "0")
$\overline{BW}$ (Pin 17). This output can be programmed to serve as one of the following:
A. $\overline{BW}$. Complemented Borrow out (active "0").
B. BW. True Borrow out (active "1").
C. BWT. Borrow Toggle flip-flop out.
D. COMPT. Comparator Toggle output.
V_{DD} (Pin 5). Supply voltage positive terminal.
V_{SS} (Pin 20). Supply voltage negative terminal.

Absolute Maximum Ratings:

Parameter	Symbol	Values	Unit
Voltage at any input	V_{IN}	V_{SS} - 0.3 to V_{DD} + 0 .3	Volts
Operating Temperature	T_A	0 to +70	^oC
Storage Temperature	T_{STG}	-65 to +150	^oC
Supply Voltage	V_{DD} - V_{SS}	+7.0	Volts

DC Electrical Characteristics. (All voltages referenced to V_{SS}.
T_A = 0˚ to 70˚C, V_{DD} = 4.5V to 5.5V, fc = 0, unless otherwise specified)

Parameter	Symbol	Min. Value	Max.Value	Unit	Remarks
Supply Voltage	V_{DD}	4.5	5.5	Volts	-
Supply Current	I_{DD}	-	350	µA	Outputs open
Input Low Voltage	V_{IL}	0	0.8	Volts	-
Input High Voltage	V_{IH}	2.0	V_{DD}	Volts	-
Output Low Voltage	V_{OL}	-	0.4	Volts	4mA Sink
Output High Voltage	V_{OH}	2.5	-	Volts	200µA Source
Input Current	-	-	15	nA	Leakage Current
Output Source Current	I_{SRC}	200	-	µA	V_{OH} = 2.5V
Output Sink Current	I_{SINK}	4	-	mA	V_{OL} = 0.4V
Data Bus Off-State Leakage Current	-	-	15	nA	-

TRANSIENT CHARACTERISTICS (See Timing Diagrams in Fig. 2 thru Fig. 7, V_{DD} = 4.5V to 5.5V, T_A = 0° to 70°C, unless otherwise specified)

Parameter	Symbol	Min.Value	Max.Value	Unit
Clock A/B "Low"	T_{CL}	20	No Limit	ns
Clock A/B "High"	T_{CH}	30	No Limit	ns
Clock A/B Frequency (See NOTE 1)	fc	0	20	MHz
Clock UP/DN Reversal Delay	T_{UDD}	100	-	ns
LCTR Positive edge to the next A/B positive or negative edge delay	T_{LC}	100	-	ns
Clock A/B to CY/BW/COMP "low" propagation delay	T_{CBL}	-	65	ns
Clock A/B to CY/BW/COMP "high" propagation delay	T_{CBH}	-	85	ns
LCTR and LLTC pulse width	T_{LCW}	60	-	ns
Clock A/B to CYT, BWT and COMPT "high" propagation delay	T_{TFH}	-	100	ns
Clock A/B to CYT, BWT and COMPT "low" progagation delay	T_{TFL}	-	100	ns
WR pulse width	T_{WW}	60	-	ns
RD to data out delay (C_L = 20pF)	T_R	-	110	ns
CS, RD Terminate to Data-Bus Tri-State	T_{RT}	-	30	ns
Data-Bus set-up time for WR	T_{DS}	15	-	ns **(see Note 3)**
Data-Bus hold time for WR	T_{DH}	30	-	ns **(see Note 3)**
C/D, CS set-up time for RD	T_{CRS}	0	-	ns
C/D, CS hold time for RD	T_{CRH}	0	-	ns
C/D set-up time for WR	T_{CWS}	15	-	ns **(see Note 3)**
C/D hold time for WR	T_{CWH}	30	-	ns **(see Note 3)**
CS set-up time for WR	T_{SWS}	15	-	ns **(see Note 3)**
CS holdtime for WR	T_{SWH}	0	-	ns **(see Note 3)**
Quadrature Mode:				
Clock A/B Validation delay (See NOTE 2)	T_{CQV}	-	160	ns
A and B phase delay	T_{PH}	208	-	ns
Clock A/B frequency	f_{CQ}	-	1.2	MHz
CY, BW, COMP pulse width	T_{CBW}	75	180	ns

NOTE 1: A) In Divide by N mode, the maximum clock frequency is 10 MHz.
B) The maximum frequency for valid CY, BW, CYT, BWT, COMP, COMPT is 10 MHz.

NOTE 2: In quadrature mode A/B inputs are filtered and required to be stable for at least T_{CQV} length to be valid.

NOTE 3: All WR specifications are critical for proper operation of LS7166

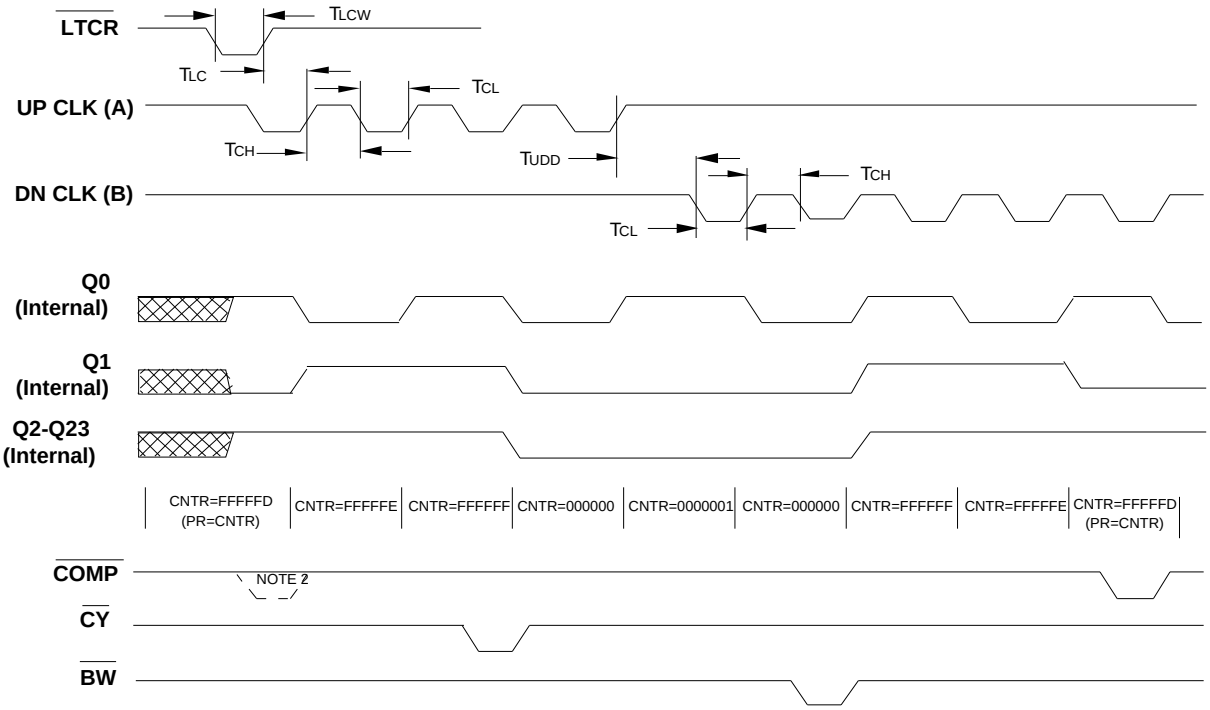


FIGURE 2 . LOAD COUNTER, UP CLOCK, DOWN CLOCK, COMPARE OUT, CARRY, BORROW

NOTE 1: The counter in this example is assumed to be operating in the binary mode.
NOTE 2: No COMP output is generated here, although PR = CNTR. COMP output is disabled with a counter load command and enabled with the rising edge of the next clock, thus eliminating invalid COMP outputs whenever the CNTR is loaded from the PR.
NOTE 3: When UP Clock is active, the DN Clock should be held "HIGH" and vice versa.

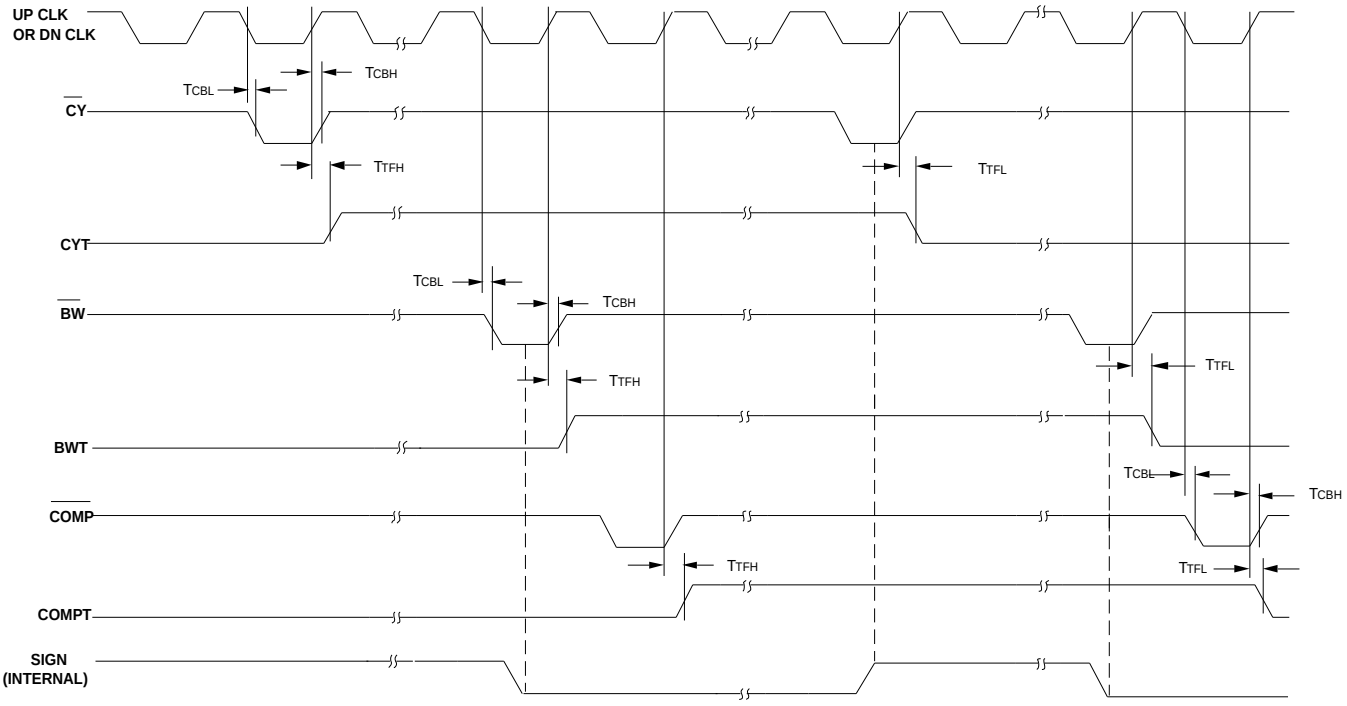


FIGURE 3. CLOCK TO $\overline{CY}$/$\overline{BW}$ OUTPUT PROPAGATION DELAYS

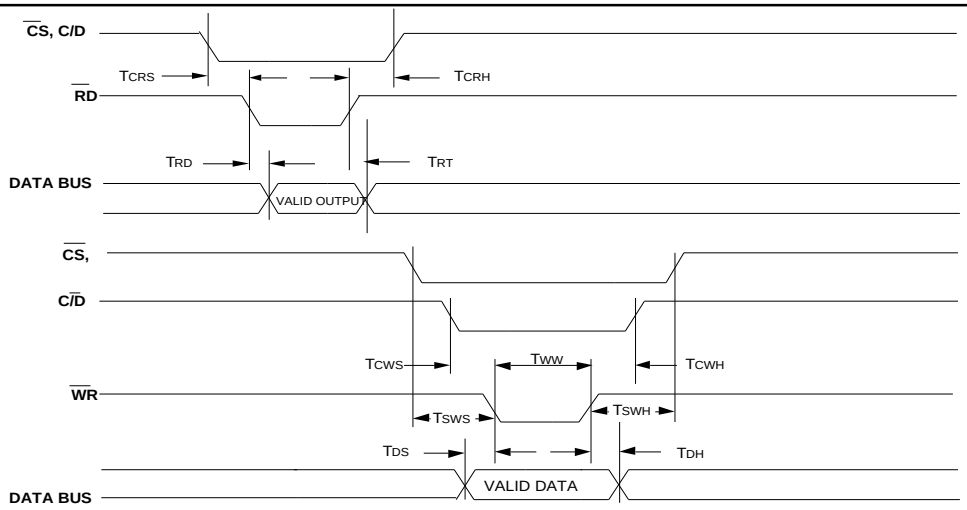
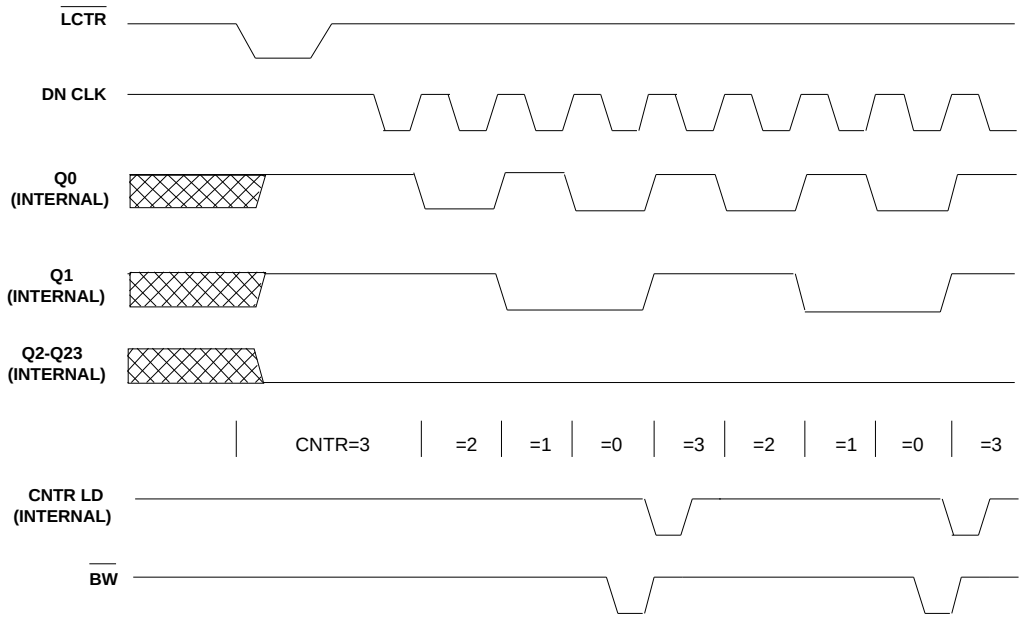


FIGURE 4. READ/WRITE CYCLES



NOTE: EXAMPLE OF DIVIDE BY 4 IN DOWN COUNT MODE

FIGURE 5. DIVIDE BY N MODE

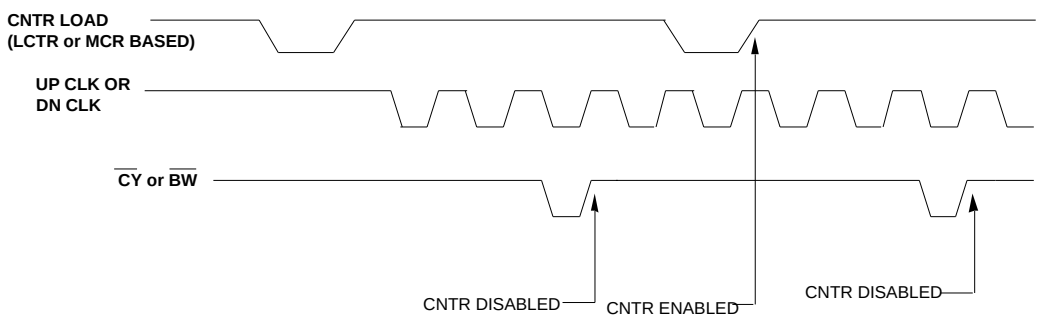


FIGURE 6 . CYCLE ONCE MODE

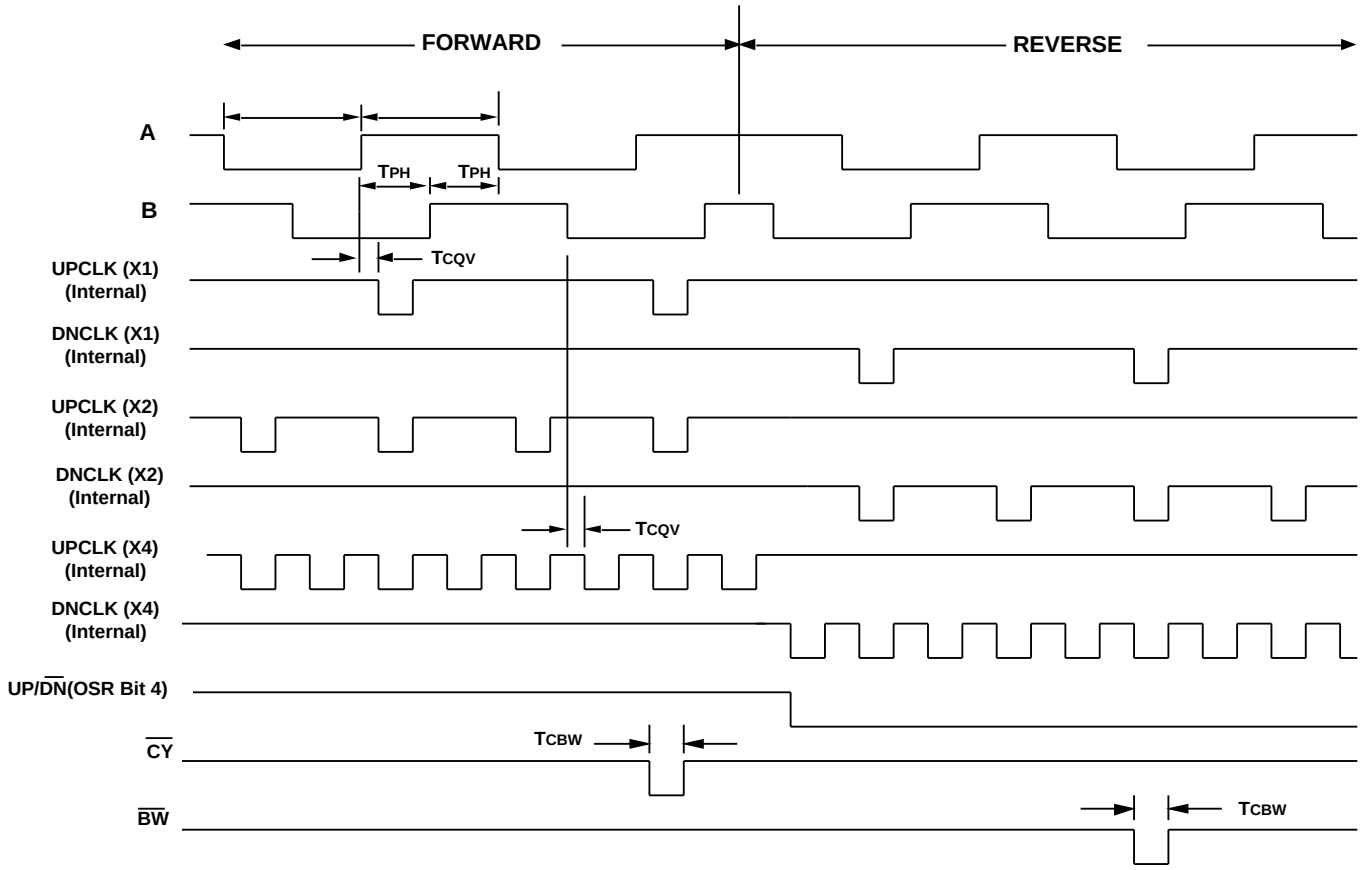


FIGURE 7.

QUADRATURE MODE INTERNAL CLOCKS

7166-030392-9

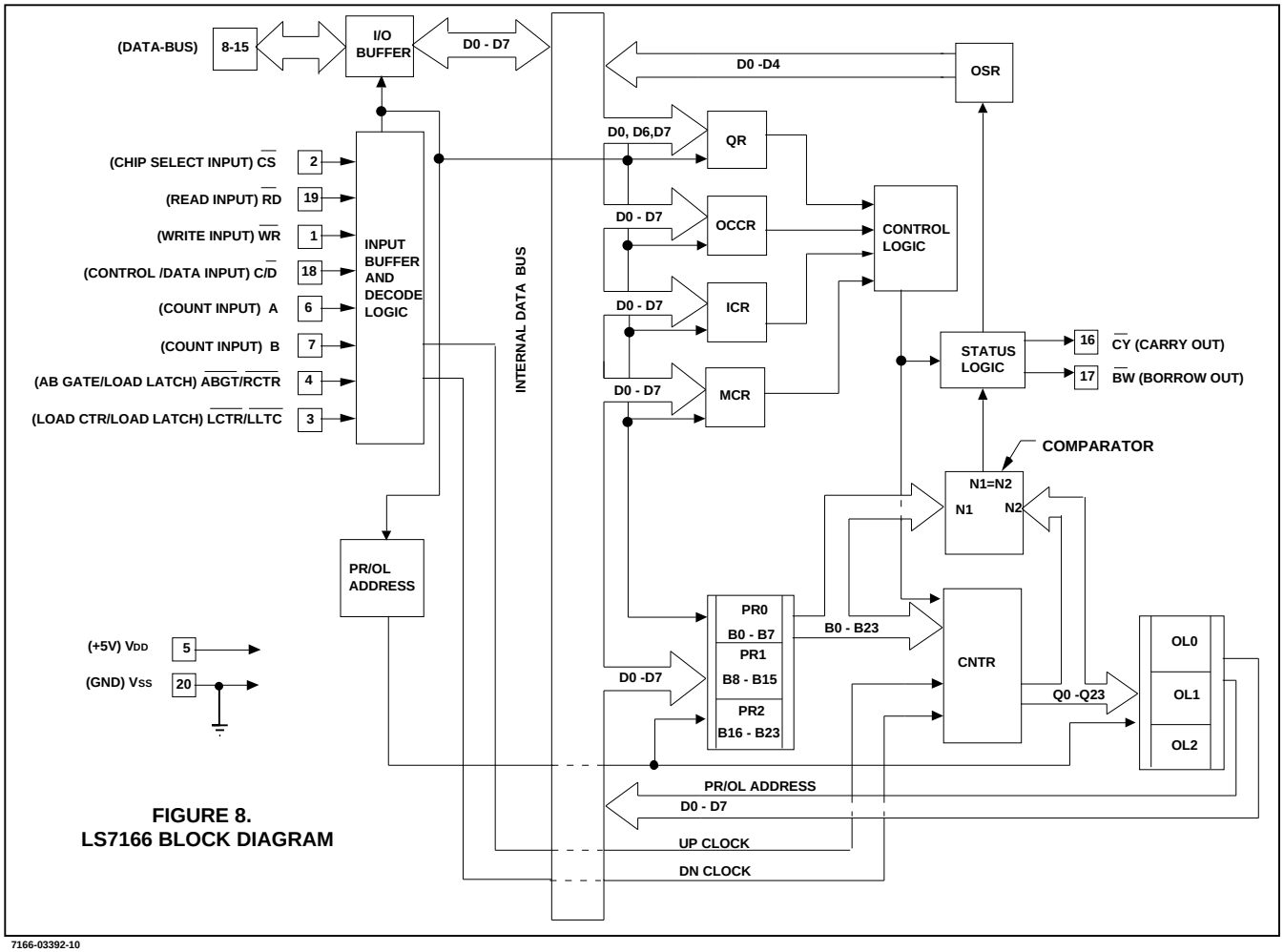
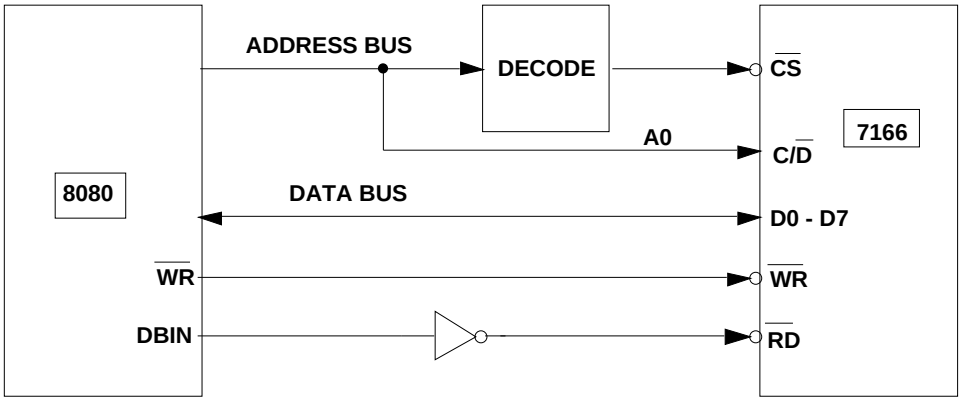
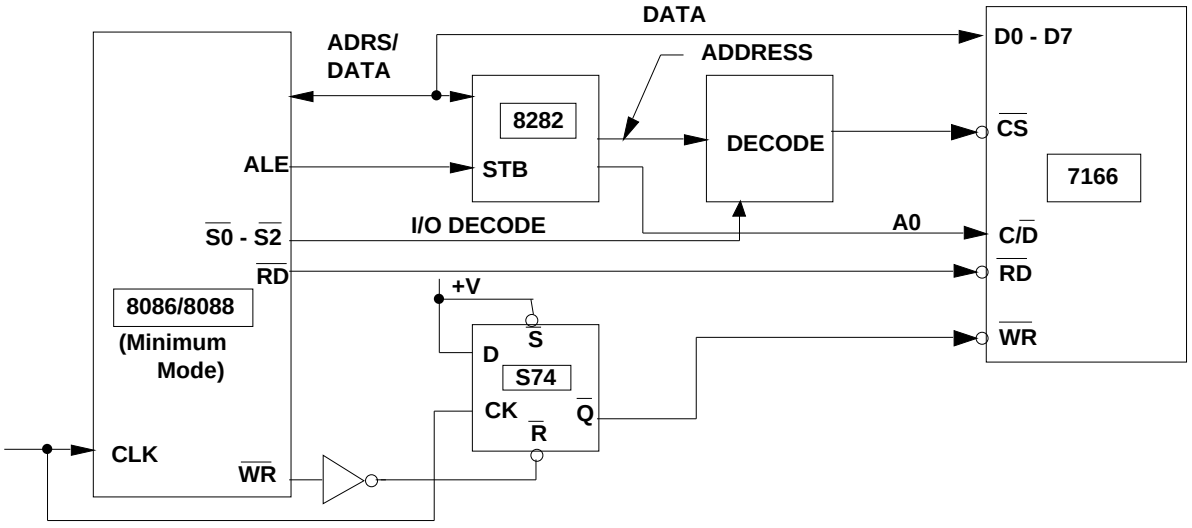


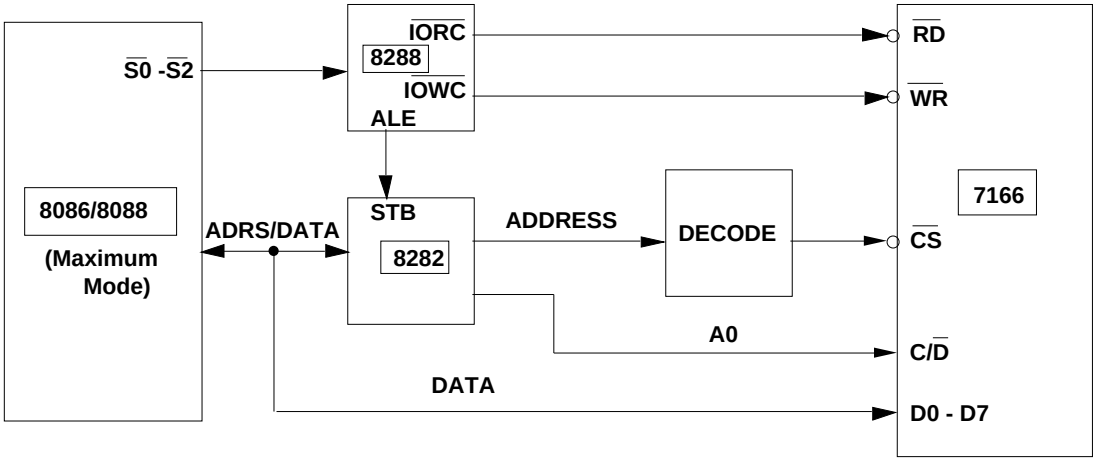
FIGURE 8.
LS7166 BLOCK DIAGRAM

7166-03392-10

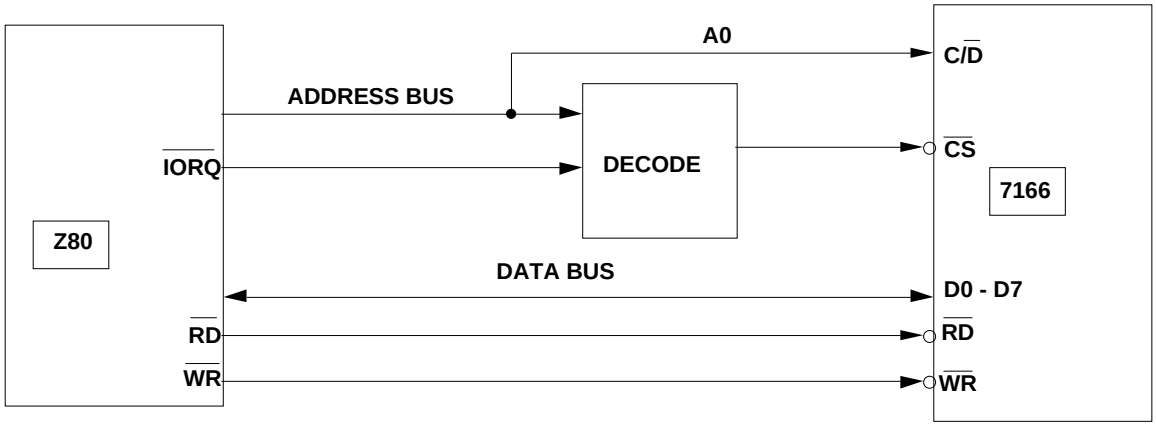
LS7166 INTERFACE EXAMPLES

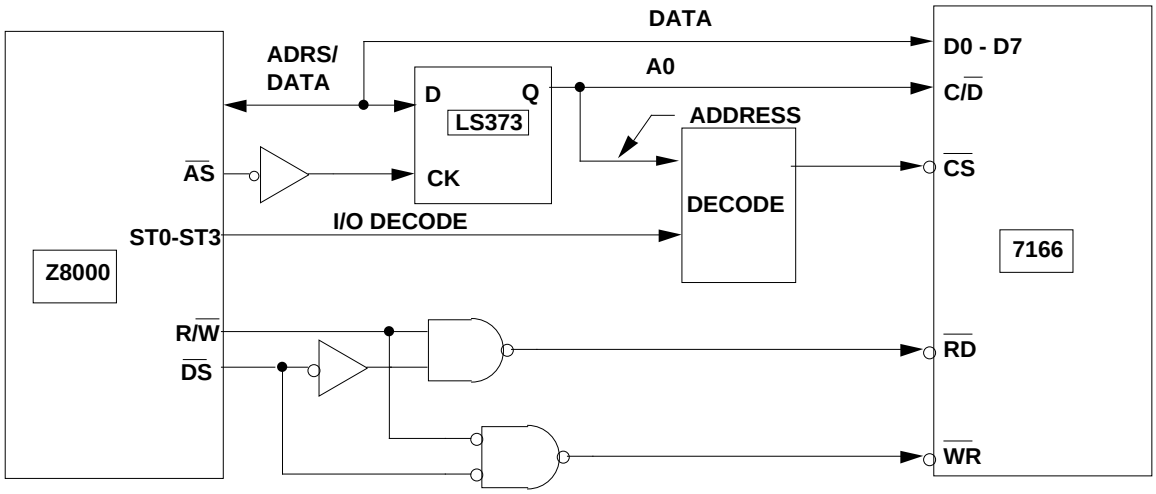


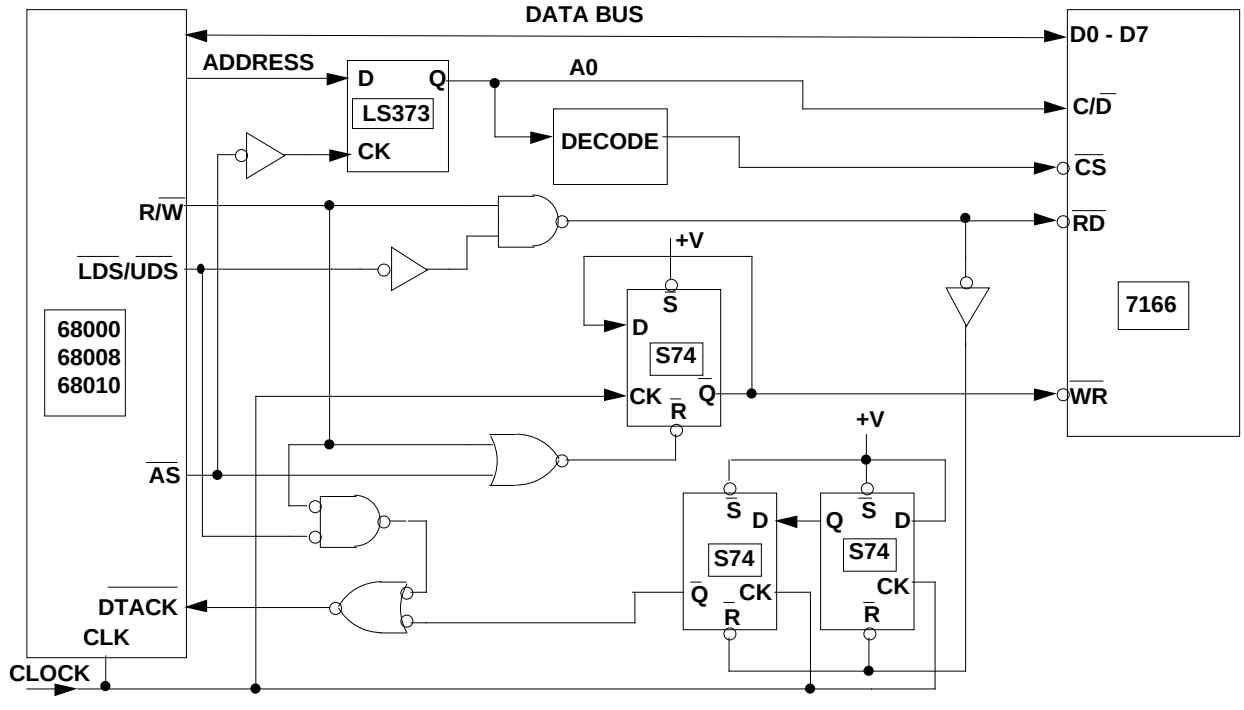




LS7166 INTERFACE EXAMPLES







7166-03392-12

PCA9535; PCA9535C

16-bit I²C-bus and SMBus, low power I/O port with interrupt

Rev. 05 — 15 September 2008 Product data sheet

1. General description

The PCA9535 and PCA9535C are 24-pin CMOS devices that provide 16 bits of General Purpose parallel Input/Output (GPIO) expansion for I²C-bus/SMBus applications and was developed to enhance the NXP Semiconductors family of I²C-bus I/O expanders. The improvements include higher drive capability, 5 V I/O tolerance, lower supply current, individual I/O configuration, and smaller packaging. I/O expanders provide a simple solution when additional I/O is needed for ACPI power switches, sensors, push buttons, LEDs, fans, etc.

The PCA9535 and PCA9535C consist of two 8-bit Configuration (Input or Output selection), Input, Output and Polarity Inversion (active HIGH or active LOW operation) registers. The system master can enable the I/Os as either inputs or outputs by writing to the I/O configuration bits. The data for each input or output is kept in the corresponding Input or Output register. The polarity of the read register can be inverted with the Polarity Inversion register. All registers can be read by the system master. Although pin-to-pin and I²C-bus address compatible with the PCF8575, software changes are required due to the enhancements and are discussed in *Application Note AN469*.

The PCA9535 is identical to the PCA9555 except for the removal of the internal I/O pull-up resistor which greatly reduces power consumption when the I/Os are held LOW.

The PCA9535C is identical to the PCA9535 except that all the I/O pins are high-impedance open-drain outputs.

The PCA9535 and PCA9535C open-drain interrupt output is activated when any input state differs from its corresponding Input Port register state and is used to indicate to the system master that an input state has changed. The power-on reset sets the registers to their default values and initializes the device state machine.

Three hardware pins (A0, A1, A2) vary the fixed I²C-bus address and allow up to eight devices to share the same I²C-bus/SMBus. The fixed I²C-bus address of the PCA9535 and PCA9535C are the same as the PCA9555 allowing up to eight of these devices in any combination to share the same I²C-bus/SMBus.

2. Features

- Operating power supply voltage range of 2.3 V to 5.5 V
- 5 V tolerant I/Os
- Polarity Inversion register
- Active LOW interrupt output
- Low standby current
- Noise filter on SCL/SDA inputs

- No glitch on power-up
- Internal power-on reset
- 16 I/O pins which default to 16 inputs
- 0 Hz to 400 kHz clock frequency
- ESD protection exceeds 2000 V HBM per JESD22-A114, 200 V MM per JESD22-A115, and 1000 V CDM per JESD22-C101
- Latch-up testing is done to JEDEC Standard JESD78 which exceeds 100 mA
- Offered in four different packages: SO24, TSSOP24, HVQFN24 and HWQFN24

3. Ordering information

Table 1. Ordering information

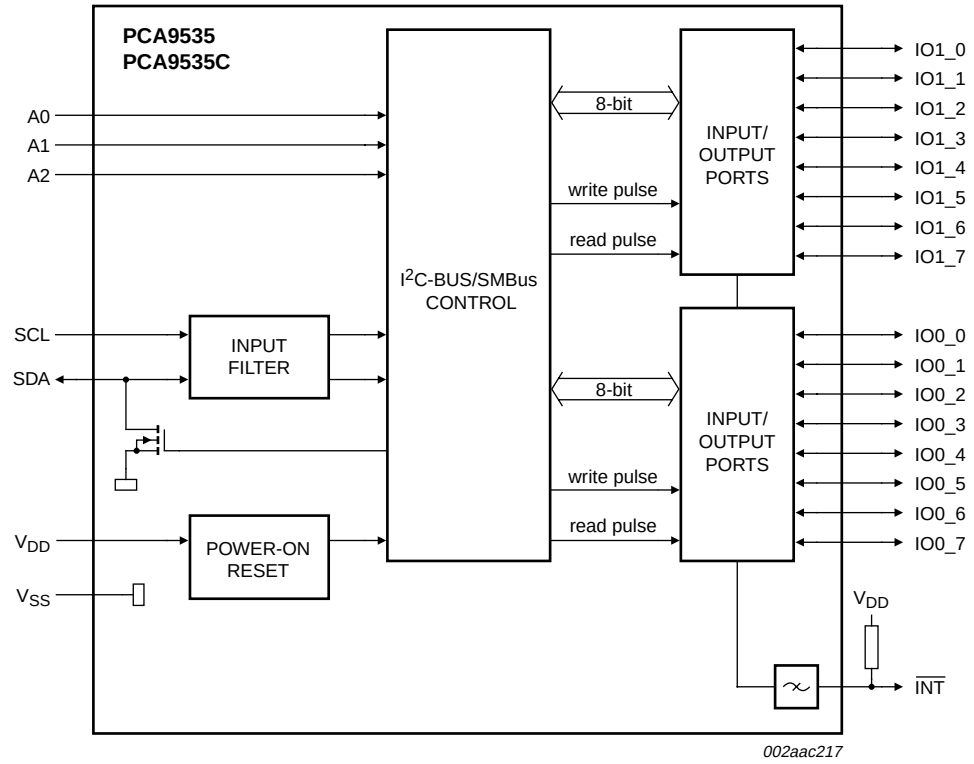
Type number	Package		
	Name	Description	Version
PCA9535D	SO24	plastic small outline package; 24 leads; body width 7.5 mm	SOT137-1
PCA9535PW	TSSOP24	plastic thin shrink small outline package; 24 leads; body width 4.4 mm	SOT355-1
PCA9535BS	HVQFN24	plastic thermal enhanced very thin quad flat package; no leads; 24 terminals; body 4 × 4 × 0.85 mm	SOT616-1
PCA9535HF	HWQFN24	plastic thermal enhanced very very thin quad flat package; no leads; 24 terminals; body 4 × 4 × 0.75 mm	SOT994-1
PCA9535CD	SO24	plastic small outline package; 24 leads; body width 7.5 mm	SOT137-1
PCA9535CPW	TSSOP24	plastic thin shrink small outline package; 24 leads; body width 4.4 mm	SOT355-1
PCA9535CHF	HWQFN24	plastic thermal enhanced very very thin quad flat package; no leads; 24 terminals; body 4 × 4 × 0.75 mm	SOT994-1

3.1 Ordering options

Table 2. Ordering options

Type number	Topside mark	Temperature range
PCA9535D	PCA9535D	T_{amb} = –40 °C to +85 °C
PCA9535PW	PCA9535PW	T_{amb} = –40 °C to +85 °C
PCA9535BS	9535	T_{amb} = –40 °C to +85 °C
PCA9535HF	P35H	T_{amb} = –40 °C to +85 °C
PCA9535CD	PCA9535CD	T_{amb} = –40 °C to +85 °C
PCA9535CPW	PCA9535C	T_{amb} = –40 °C to +85 °C
PCA9535CHF	P35C	T_{amb} = –40 °C to +85 °C

4. Block diagram



Remark: All I/Os are set to inputs at reset.

Fig 1. Block diagram of PCA9535; PCA9535C

5. Pinning information

5.1 Pinning

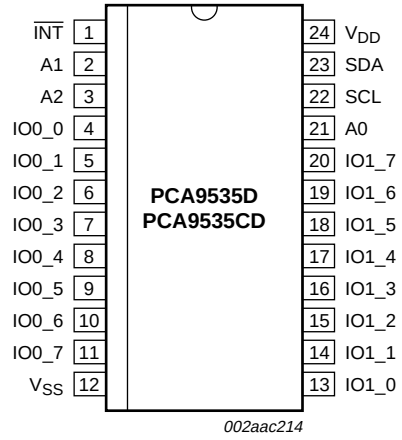


Fig 2. Pin configuration for SO24

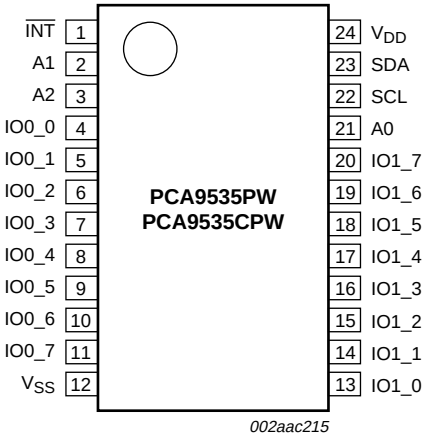


Fig 3. Pin configuration for TSSOP24

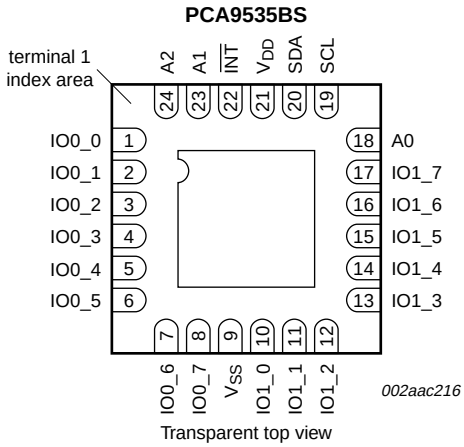


Fig 4. Pin configuration for HVQFN24

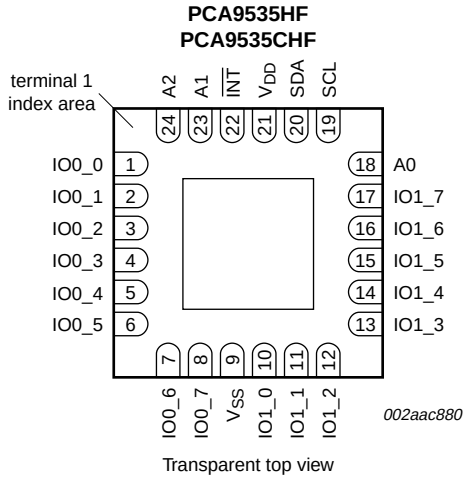


Fig 5. Pin configuration for HWQFN24

5.2 Pin description

Table 3. Pin description

Symbol	Pin		Description
	SO24, TSSOP24	HVQFN24, HWQFN24	
$\overline{INT}$	1	22	interrupt output (open-drain)
A1	2	23	address input 1
A2	3	24	address input 2
IO0_0	4	1	port 0 input/output[2]
IO0_1	5	2	
IO0_2	6	3	
IO0_3	7	4	
IO0_4	8	5	
IO0_5	9	6	
IO0_6	10	7	
IO0_7	11	8	
V_{SS}	12	9[1]	supply ground
IO1_0	13	10	port 1 input/output[2]
IO1_1	14	11	
IO1_2	15	12	
IO1_3	16	13	
IO1_4	17	14	
IO1_5	18	15	
IO1_6	19	16	
IO1_7	20	17	
A0	21	18	address input 0
SCL	22	19	serial clock line
SDA	23	20	serial data line
V_{DD}	24	21	supply voltage

[1] HVQFN24 and HWQFN24 package die supply ground is connected to both the V_{SS} pin and the exposed center pad. The V_{SS} pin must be connected to supply ground for proper device operation. For enhanced thermal, electrical, and board-level performance, the exposed pad needs to be soldered to the board using a corresponding thermal pad on the board, and for proper heat conduction through the board thermal vias need to be incorporated in the PCB in the thermal pad region.

[2] PCA9535 I/Os are totem pole, whereas the I/Os on PCA9535C are open-drain.

6. Functional description

Refer to Figure 1 “Block diagram of PCA9535; PCA9535C”.

6.1 Device address

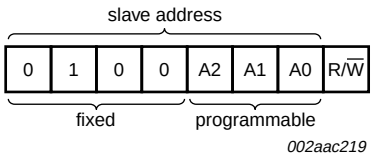


Fig 6. PCA9535; PCA9535C device address

6.2 Registers

6.2.1 Command byte

The command byte is the first byte to follow the address byte during a write transmission. It is used as a pointer to determine which of the following registers will be written or read.

Table 4. Command byte

Command	Register
0	Input port 0
1	Input port 1
2	Output port 0
3	Output port 1
4	Polarity Inversion port 0
5	Polarity Inversion port 1
6	Configuration port 0
7	Configuration port 1

6.2.2 Registers 0 and 1: Input port registers

This register is an input-only port. It reflects the incoming logic levels of the pins, regardless of whether the pin is defined as an input or an output by Register 3. Writes to this register have no effect.

The default value 'X' is determined by the externally applied logic level.

Table 5. Input port 0 Register

Bit	7	6	5	4	3	2	1	0
Symbol	I0.7	I0.6	I0.5	I0.4	I0.3	I0.2	I0.1	I0.0
Default	X	X	X	X	X	X	X	X

Table 6. Input port 1 register

Bit	7	6	5	4	3	2	1	0
Symbol	I1.7	I1.6	I1.5	I1.4	I1.3	I1.2	I1.1	I1.0
Default	X	X	X	X	X	X	X	X

6.2.3 Registers 2 and 3: Output port registers

This register is an output-only port. It reflects the outgoing logic levels of the pins defined as outputs by Registers 6 and 7. Bit values in this register have no effect on pins defined as inputs. In turn, reads from this register reflect the value that is in the flip-flop controlling the output selection, **not** the actual pin value.

Table 7. Output port 0 register

Bit	7	6	5	4	3	2	1	0
Symbol	O0.7	O0.6	O0.5	O0.4	O0.3	O0.2	O0.1	O0.0
Default	1	1	1	1	1	1	1	1

Table 8. Output port 1 register

Bit	7	6	5	4	3	2	1	0
Symbol	O1.7	O1.6	O1.5	O1.4	O1.3	O1.2	O1.1	O1.0
Default	1	1	1	1	1	1	1	1

6.2.4 Registers 4 and 5: Polarity Inversion registers

This register allows the user to invert the polarity of the Input port register data. If a bit in this register is set (written with '1'), the Input port data polarity is inverted. If a bit in this register is cleared (written with a '0'), the Input port data polarity is retained.

Table 9. Polarity Inversion port 0 register

Bit	7	6	5	4	3	2	1	0
Symbol	N0.7	N0.6	N0.5	N0.4	N0.3	N0.2	N0.1	N0.0
Default	0	0	0	0	0	0	0	0

Table 10. Polarity Inversion port 1 register

Bit	7	6	5	4	3	2	1	0
Symbol	N1.7	N1.6	N1.5	N1.4	N1.3	N1.2	N1.1	N1.0
Default	0	0	0	0	0	0	0	0

6.2.5 Registers 6 and 7: Configuration registers

This register configures the directions of the I/O pins. If a bit in this register is set (written with '1'), the corresponding port pin is enabled as an input with high-impedance output driver. If a bit in this register is cleared (written with '0'), the corresponding port pin is enabled as an output. At reset, the device's ports are inputs.

Table 11. Configuration port 0 register

Bit	7	6	5	4	3	2	1	0
Symbol	C0.7	C0.6	C0.5	C0.4	C0.3	C0.2	C0.1	C0.0
Default	1	1	1	1	1	1	1	1

Table 12. Configuration port 1 register

Bit	7	6	5	4	3	2	1	0
Symbol	C1.7	C1.6	C1.5	C1.4	C1.3	C1.2	C1.1	C1.0
Default	1	1	1	1	1	1	1	1

6.3 Power-on reset

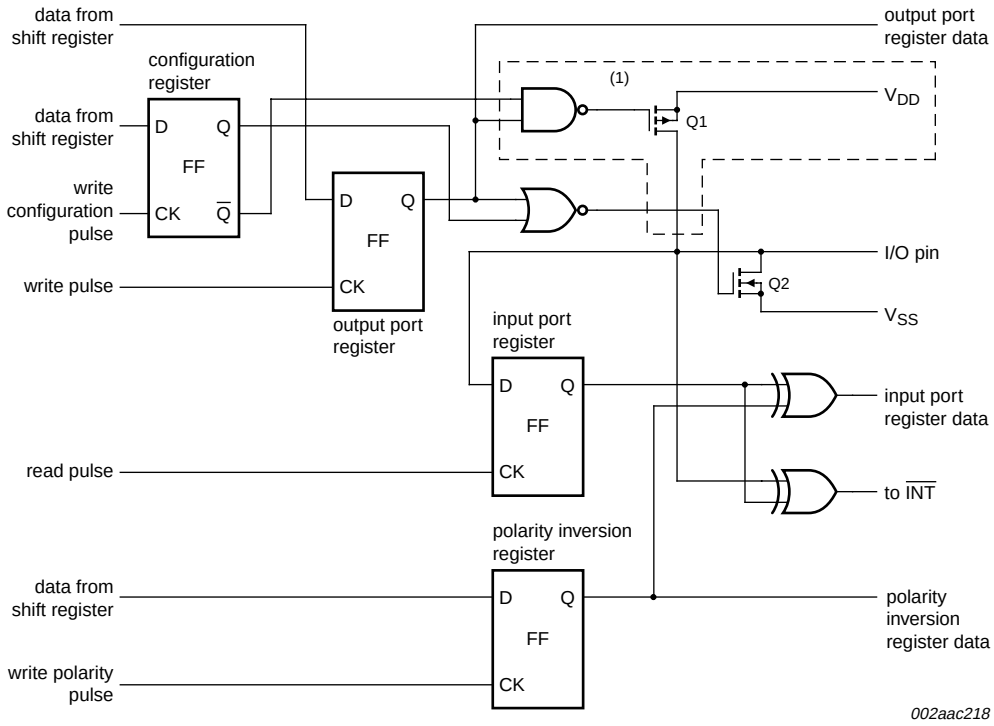
When power is applied to V_{DD}, an internal power-on reset holds the PCA9535/PCA9535C in a reset condition until V_{DD} has reached V_{POR}. At that point, the reset condition is released and the PCA9535/PCA9535C registers and SMBus state machine will initialize to their default states. Thereafter, V_{DD} must be lowered below 0.2 V to reset the device.

For a power reset cycle, V_{DD} must be lowered below 0.2 V and then restored to the operating voltage.

6.4 I/O port

When an I/O is configured as an input on PCA9535, FETs Q1 and Q2 are off, creating a high impedance input. The input voltage may be raised above V_{DD} to a maximum of 5.5 V. In the case of PCA9535C, FET Q1 has been removed and the open-drain FET Q2 will function the same as PCA9535.

If the I/O is configured as an output, then on PCA9535 either Q1 or Q2 is on, depending on the state of the Output Port register. Care should be exercised if an external voltage is applied to an I/O configured as an output because of the low-impedance path that exists between the pin and either V_{DD} or V_{SS}.



At power-on reset, all registers return to default values.

(1) PCA9535C I/Os are open-drain only. The portion of the PCA9535 schematic marked inside the dotted line box is not in PCA9535C.

Fig 7. Simplified schematic of I/Os

6.5 Bus transactions

6.5.1 Writing to the port registers

Data is transmitted to the PCA9535/PCA9535C by sending the device address and setting the least significant bit to a logic 0 (see Figure 6 “PCA9535; PCA9535C device address”). The command byte is sent after the address and determines which register will receive the data following the command byte.

The eight registers within the PCA9535/PCA9535C are configured to operate as four register pairs. The four pairs are Input Ports, Output Ports, Polarity Inversion Ports, and Configuration Ports. After sending data to one register, the next data byte will be sent to the other register in the pair (see Figure 8 and Figure 9). For example, if the first byte is sent to Output Port 1 (register 3), then the next byte will be stored in Output Port 0 (register 2). There is no limitation on the number of data bytes sent in one write transmission. In this way, each 8-bit register may be updated independently of the other registers.

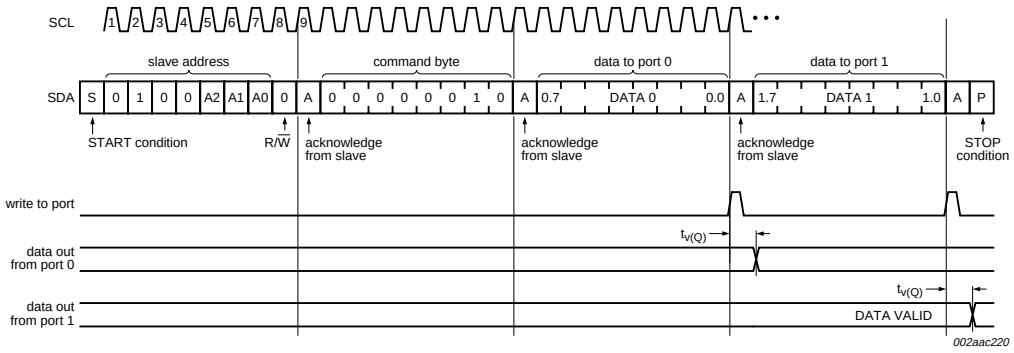


Fig 8. Write to Output Port registers

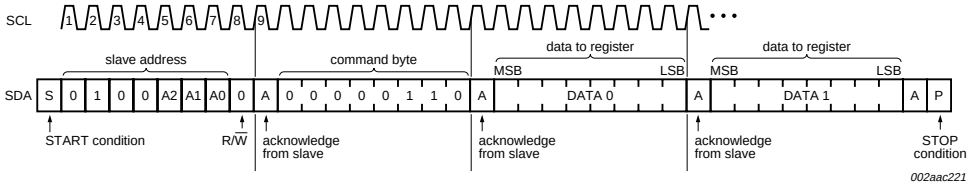
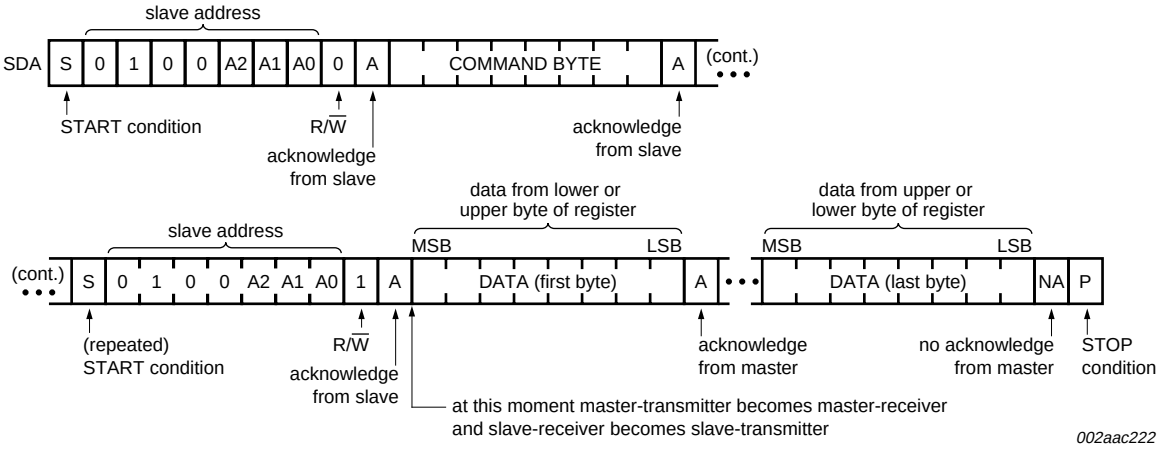


Fig 9. Write to Configuration registers

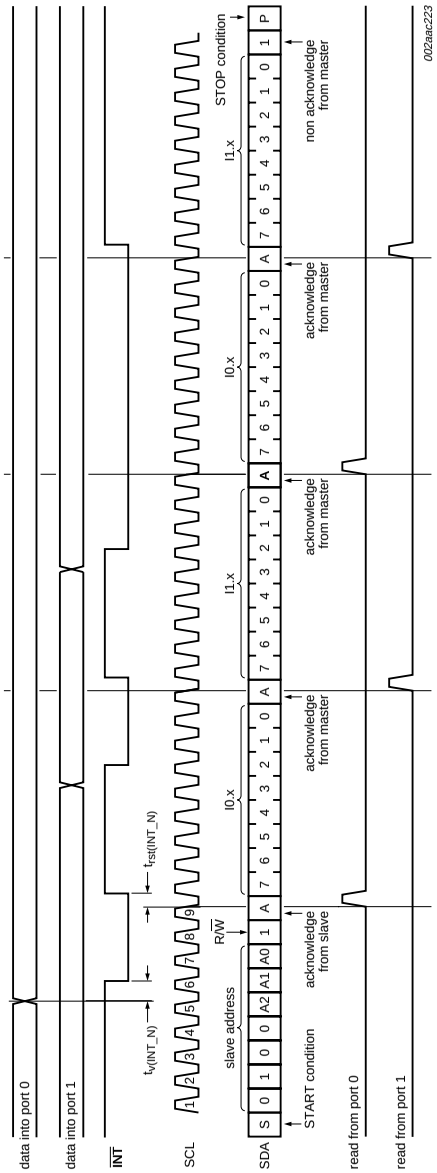
6.5.2 Reading the port registers

In order to read data from the PCA9535/PCA9535C, the bus master must first send the PCA9535/PCA9535C address with the least significant bit set to a logic 0 (see Figure 6 "PCA9535; PCA9535C device address"). The command byte is sent after the address and determines which register will be accessed. After a restart, the device address is sent again, but this time the least significant bit is set to a logic 1. Data from the register defined by the command byte will then be sent by the PCA9535/PCA9535C (see Figure 10, Figure 11 and Figure 12). Data is clocked into the register on the falling edge of the acknowledge clock pulse. After the first byte is read, additional bytes may be read but the data will now reflect the information in the other register in the pair. For example, if you read Input Port 1, then the next byte read would be Input Port 0. There is no limitation on the number of data bytes received in one read transmission but the final byte received, the bus master must not acknowledge the data.



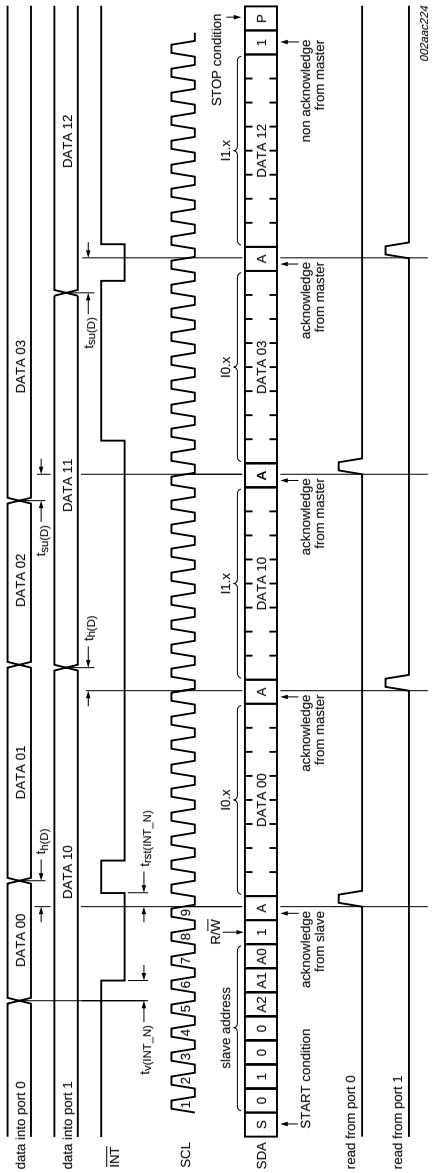
Remark: Transfer can be stopped at any time by a STOP condition.

Fig 10. Read from register



Remark: Transfer of data can be stopped at any moment by a STOP condition. When this occurs, data present at the latest acknowledge phase is valid (output mode). It is assumed that the command byte has previously been set to '00' (read Input Port register).

Fig 11. Read Input Port register, scenario 1



Remark: Transfer of data can be stopped at any moment by a STOP condition. When this occurs, data present at the latest acknowledge phase is valid (output mode). It is assumed that the command byte has previously been set to '00' (read Input Port register).

Fig 12. Read Input Port register, scenario 2

6.5.3 Interrupt output

The open-drain interrupt output is activated when one of the port pins change state and the pin is configured as an input. The interrupt is deactivated when the input returns to its previous state or the Input Port register is read (see Figure 11). A pin configured as an output cannot cause an interrupt. Since each 8-bit port is read independently, the interrupt caused by Port 0 will not be cleared by a read of Port 1 or the other way around.

Remark: Changing an I/O from an output to an input may cause a false interrupt to occur if the state of the pin does not match the contents of the Input Port register.

7. Characteristics of the I²C-bus

The I²C-bus is for 2-way, 2-line communication between different ICs or modules. The two lines are a serial data line (SDA) and a serial clock line (SCL). Both lines must be connected to a positive supply via a pull-up resistor when connected to the output stages of a device. Data transfer may be initiated only when the bus is not busy.

7.1 Bit transfer

One data bit is transferred during each clock pulse. The data on the SDA line must remain stable during the HIGH period of the clock pulse as changes in the data line at this time will be interpreted as control signals (see Figure 13).

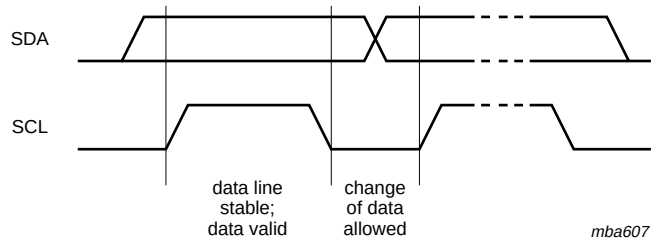


Fig 13. Bit transfer

7.1.1 START and STOP conditions

Both data and clock lines remain HIGH when the bus is not busy. A HIGH-to-LOW transition of the data line while the clock is HIGH is defined as the START condition (S). A LOW-to-HIGH transition of the data line while the clock is HIGH is defined as the STOP condition (P) (see Figure 14).

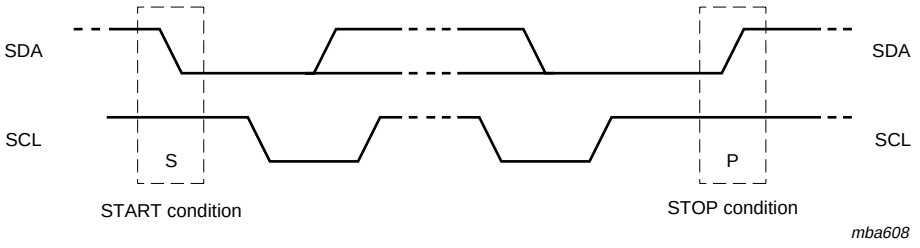


Fig 14. Definition of START and STOP conditions

7.2 System configuration

A device generating a message is a ‘transmitter’; a device receiving is the ‘receiver’. The device that controls the message is the ‘master’ and the devices which are controlled by the master are the ‘slaves’ (see Figure 15).

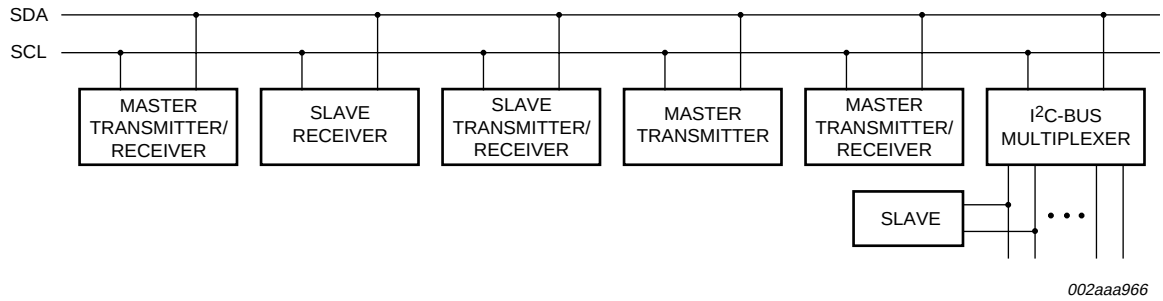


Fig 15. System configuration

7.3 Acknowledge

The number of data bytes transferred between the START and the STOP conditions from transmitter to receiver is not limited. Each byte of eight bits is followed by one acknowledge bit. The acknowledge bit is a HIGH level put on the bus by the transmitter, whereas the master generates an extra acknowledge related clock pulse.

A slave receiver which is addressed must generate an acknowledge after the reception of each byte. Also a master must generate an acknowledge after the reception of each byte that has been clocked out of the slave transmitter. The device that acknowledges has to pull down the SDA line during the acknowledge clock pulse, so that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse; set-up time and hold time must be taken into account.

A master receiver must signal an end of data to the transmitter by not generating an acknowledge on the last byte that has been clocked out of the slave. In this event, the transmitter must leave the data line HIGH to enable the master to generate a STOP condition.

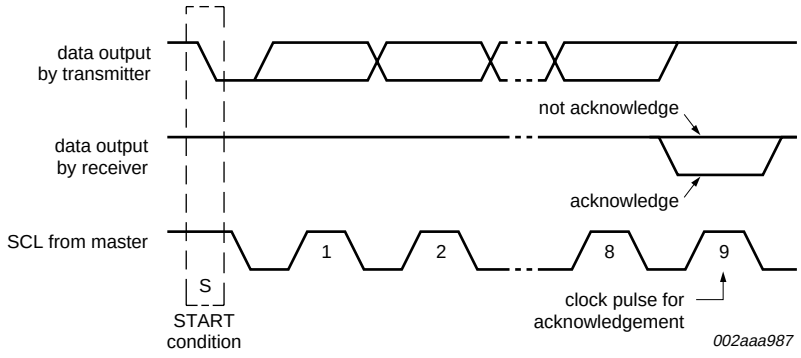
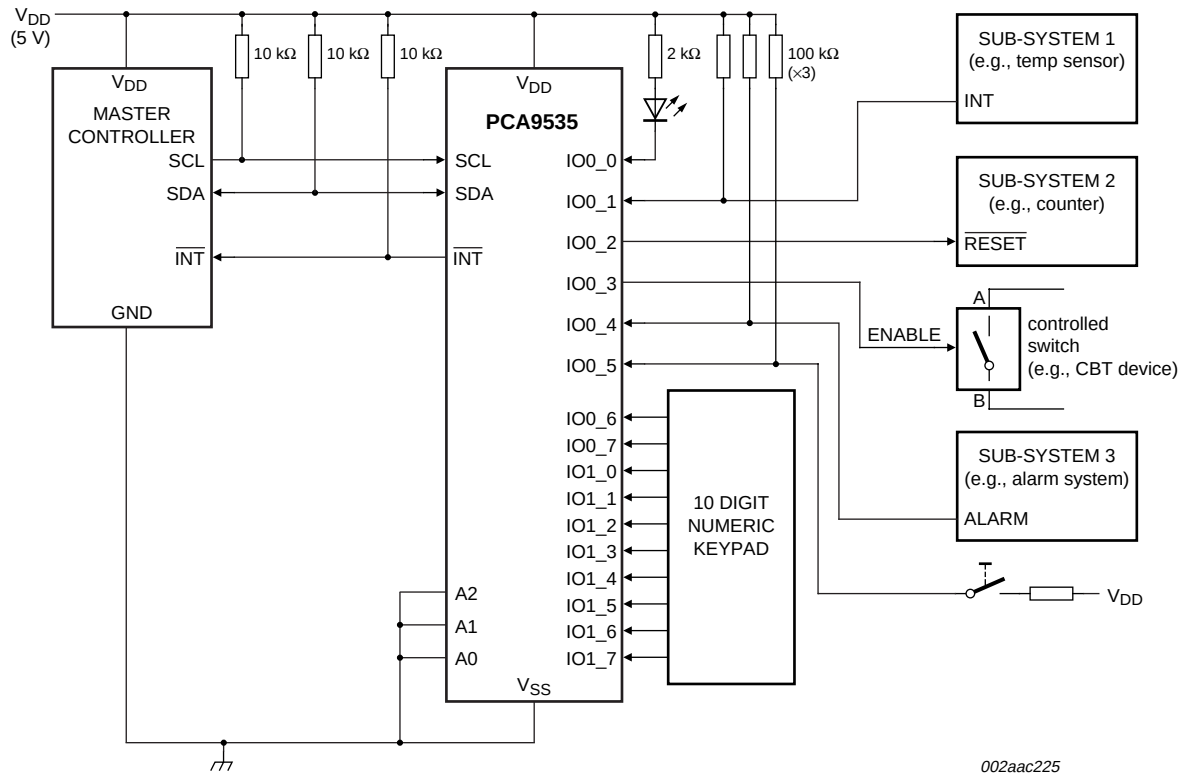


Fig 16. Acknowledgement on the I²C-bus

8. Application design-in information



Device address configured as 1110 100Xb for this example.

IO0_0, IO0_2, IO0_3 configured as outputs.

IO0_1, IO0_4, IO0_5 configured as inputs.

IO0_6, IO0_7, and IO1_0 to IO1_7 configured as inputs.

Fig 17. Typical application

8.1 Minimizing I_{DD} when the I/Os are used to control LEDs

When the PCA9535 I/Os are used to control LEDs, they are normally connected to V_{DD} through a resistor as shown in Figure 17. Since the LED acts as a diode, when the LED is off the I/O V_I is about 1.2 V less than V_{DD}. The supply current, I_{DD}, increases as V_I becomes lower than V_{DD}.

Designs needing to minimize current consumption, such as battery power applications, should consider maintaining the I/O pins greater than or equal to V_{DD} when the LED is off. Figure 18 shows a high value resistor in parallel with the LED. Figure 19 shows V_{DD} less than the LED supply voltage by at least 1.2 V. Both of these methods maintain the I/O V_I at or above V_{DD} and prevents additional supply current consumption when the LED is off.

This concern does not occur in the case of PCA9535C because the I/O pins are open-drain.

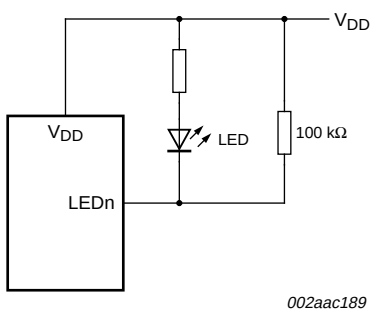


Fig 18. High value resistor in parallel with the LED

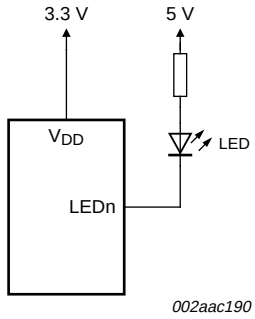


Fig 19. Device supplied by a lower voltage

9. Limiting values

Table 13. Limiting values

In accordance with the Absolute Maximum Rating System (IEC 60134).

Symbol	Parameter	Conditions	Min	Max	Unit
V_{DD}	supply voltage		–0.5	+6.0	V
$V_{I/O}$	voltage on an input/output pin		V_{SS} – 0.5	6	V
I_O	output current	on an I/O pin	-	±50	mA
I_I	input current		-	±20	mA
I_{DD}	supply current		-	160	mA
I_{SS}	ground supply current		-	200	mA
P_{tot}	total power dissipation		-	200	mW
T_{stg}	storage temperature		–65	+150	°C
T_{amb}	ambient temperature	operating	–40	+85	°C

10. Static characteristics

Table 14. Static characteristics

V_{DD} = 2.3 V to 5.5 V; V_{SS} = 0 V; T_{amb} = −40 °C to +85 °C; unless otherwise specified.

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
Supplies							
V_{DD}	supply voltage			2.3	-	5.5	V
I_{DD}	supply current	Operating mode; V_{DD} = 5.5 V; no load; f_{SCL} = 100 kHz; I/O = inputs		-	135	200	µA
I_{stb}	standby current	Standby mode; V_{DD} = 5.5 V; no load; $V_I = V_{SS}$; f_{SCL} = 0 kHz; I/O = inputs		-	0.25	1	µA
		Standby mode; V_{DD} = 5.5 V; no load; $V_I = V_{DD}$; f_{SCL} = 0 kHz; I/O = inputs		-	0.25	1	µA
V_{POR}	power-on reset voltage[1]	no load; $V_I = V_{DD}$ or V_{SS}		-	1.5	1.65	V
Input SCL; input/output SDA							
V_{IL}	LOW-level input voltage			−0.5	-	+0.3V_{DD}	V
V_{IH}	HIGH-level input voltage			0.7V_{DD}	-	5.5	V
I_{OL}	LOW-level output current	V_{OL} = 0.4 V		3	-	-	mA
I_L	leakage current	$V_I = V_{DD} = V_{SS}$		−1	-	+1	µA
C_i	input capacitance	$V_I = V_{SS}$		-	6	10	pF
I/Os							
V_{IL}	LOW-level input voltage			−0.5	-	+0.3V_{DD}	V
V_{IH}	HIGH-level input voltage			0.7V_{DD}	-	5.5	V
I_{OL}	LOW-level output current	V_{DD} = 2.3 V to 5.5 V; V_{OL} = 0.5 V	[2]	8	10	-	mA
		V_{DD} = 2.3 V to 5.5 V; V_{OL} = 0.7 V	[2]	10	14	-	mA
V_{OH}	HIGH-level output voltage	PCA9535 only					
		I_{OH} = −8 mA; V_{DD} = 2.3 V	[3]	1.8	-	-	V
		I_{OH} = −10 mA; V_{DD} = 2.3 V	[3]	1.7	-	-	V
		I_{OH} = −8 mA; V_{DD} = 3.0 V	[3]	2.6	-	-	V
		I_{OH} = −10 mA; V_{DD} = 3.0 V	[3]	2.5	-	-	V
		I_{OH} = −8 mA; V_{DD} = 4.75 V	[3]	4.1	-	-	V
		I_{OH} = −10 mA; V_{DD} = 4.75 V	[3]	4.0	-	-	V
I_{LIH}	HIGH-level input leakage current	V_{DD} = 5.5 V; $V_I = V_{DD}$		-	-	1	µA
I_{LIL}	LOW-level input leakage current	V_{DD} = 5.5 V; $V_I = V_{SS}$		-	-	−1	µA
C_i	input capacitance			-	3.7	5	pF
C_o	output capacitance			-	3.7	5	pF
Interrupt $\overline{INT}$							
I_{OL}	LOW-level output current	V_{OL} = 0.4 V		3	-	-	mA
Select inputs A0, A1, A2							
V_{IL}	LOW-level input voltage			−0.5	-	+0.3V_{DD}	V
V_{IH}	HIGH-level input voltage			0.7V_{DD}	-	5.5	V
I_{LI}	input leakage current			−1	-	+1	µA

[1] V_{DD} must be lowered to 0.2 V for at least 5 µs in order to reset part.

[2] Each I/O must be externally limited to a maximum of 25 mA and each octal (IO0_0 to IO0_7 and IO1_0 to IO1_7) must be limited to a maximum current of 100 mA for a device total of 200 mA.

[3] The total current sourced by all I/Os must be limited to 160 mA. PCA9535C does not source current and does not have the V_{OH} specification.

11. Dynamic characteristics

Table 15. Dynamic characteristics

Symbol	Parameter	Conditions	Standard-mode I²C-bus		Fast-mode I²C-bus		Unit
			Min	Max	Min	Max	
f_{SCL}	SCL clock frequency		0	100	0	400	kHz
t_{BUF}	bus free time between a STOP and START condition		4.7	-	1.3	-	µs
$t_{HD;STA}$	hold time (repeated) START condition		4.0	-	0.6	-	µs
$t_{SU;STA}$	set-up time for a repeated START condition		4.7	-	0.6	-	µs
$t_{SU;STO}$	set-up time for STOP condition		4.0	-	0.6	-	µs
$t_{VD;ACK}$	data valid acknowledge time	[1]	0.3	3.45	0.1	0.9	µs
$t_{HD;DAT}$	data hold time		0	-	0	-	ns
$t_{VD;DAT}$	data valid time	[2]	300	-	50	-	ns
$t_{SU;DAT}$	data set-up time		250	-	100	-	ns
t_{LOW}	LOW period of the SCL clock		4.7	-	1.3	-	µs
t_{HIGH}	HIGH period of the SCL clock		4.0	-	0.6	-	µs
t_f	fall time of both SDA and SCL signals		-	300	$20 + 0.1C_b$ [3]	300	ns
t_r	rise time of both SDA and SCL signals		-	1000	$20 + 0.1C_b$ [3]	300	ns
t_{SP}	pulse width of spikes that must be suppressed by the input filter		-	50	-	50	ns
Port timing							
$t_{v(Q)}$	data output valid time	[4]	-	200	-	200	ns
$t_{su(D)}$	data input set-up time		150	-	150	-	ns
$t_{h(D)}$	data input hold time		1	-	1	-	µs
Interrupt timing							
$t_{v(INT\_N)}$	valid time on pin $\overline{INT}$		-	4	-	4	µs
$t_{rst(INT\_N)}$	reset time on pin $\overline{INT}$		-	4	-	4	µs

[1] $t_{VD;ACK}$ = time for acknowledgement signal from SCL LOW to SDA (out) LOW.

[2] $t_{VD;DAT}$ = minimum time for SDA data out to be valid following SCL LOW.

[3] C_b = total capacitance of one bus line in pF.

[4] $t_{v(Q)}$ measured from $0.7V_{DD}$ on SCL to 50 % I/O output (PCA9535). For PCA9535C, use load circuit shown in Figure 24 and measure from $0.7V_{DD}$ on SCL to 30 % I/O output.

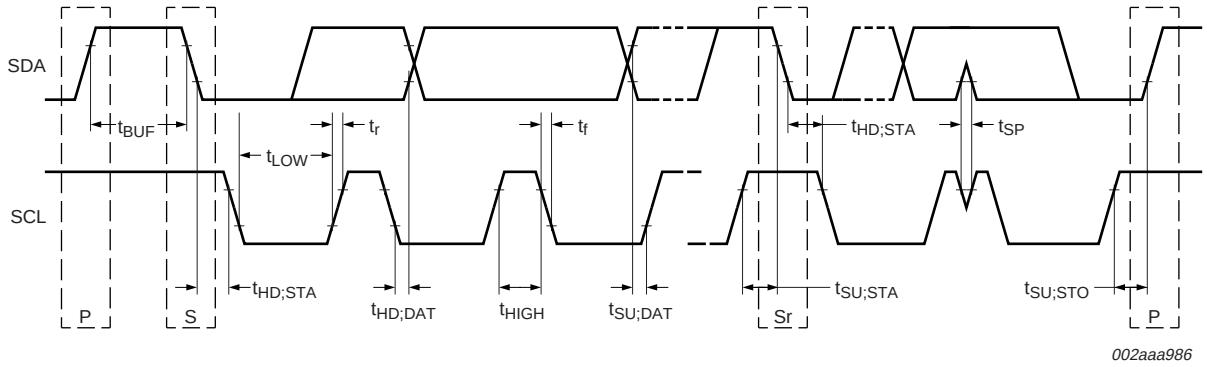
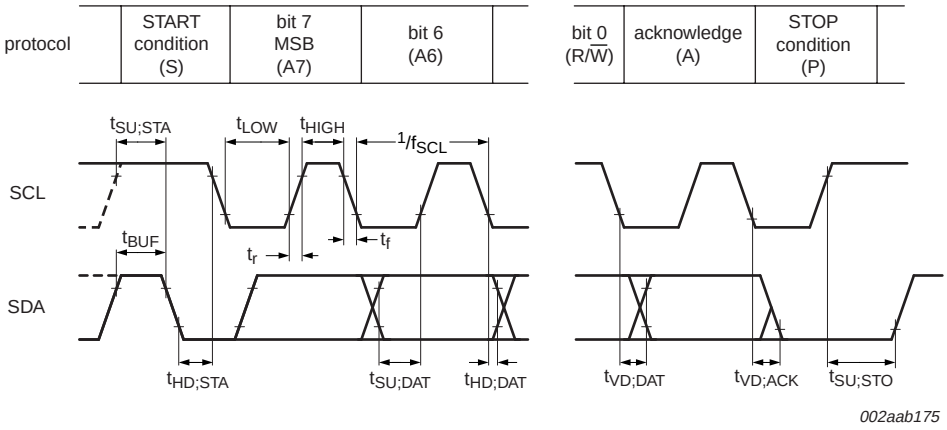


Fig 20. Definition of timing on the I²C-bus



Rise and fall times refer to V_{IL} and V_{IH}.

Fig 21. I²C-bus timing diagram

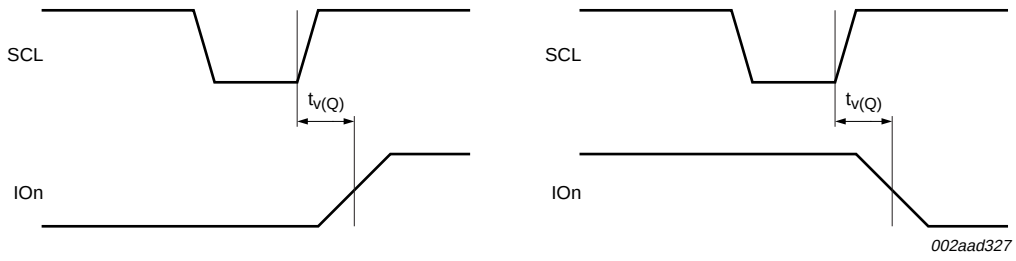
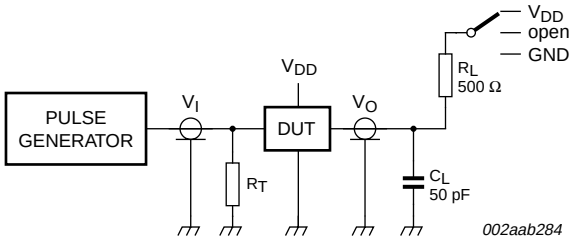


Fig 22. $t_{v(Q)}$ timing

12. Test information



R_L = load resistor.

C_L = load capacitance includes jig and probe capacitance.

R_T = termination resistance should be equal to the output impedance of Z_o of the pulse generators.

Fig 23. Test circuitry for switching times

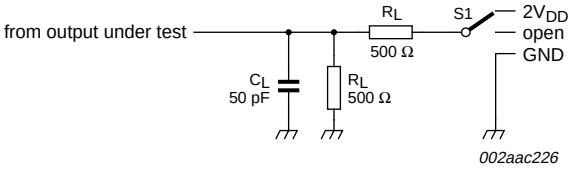
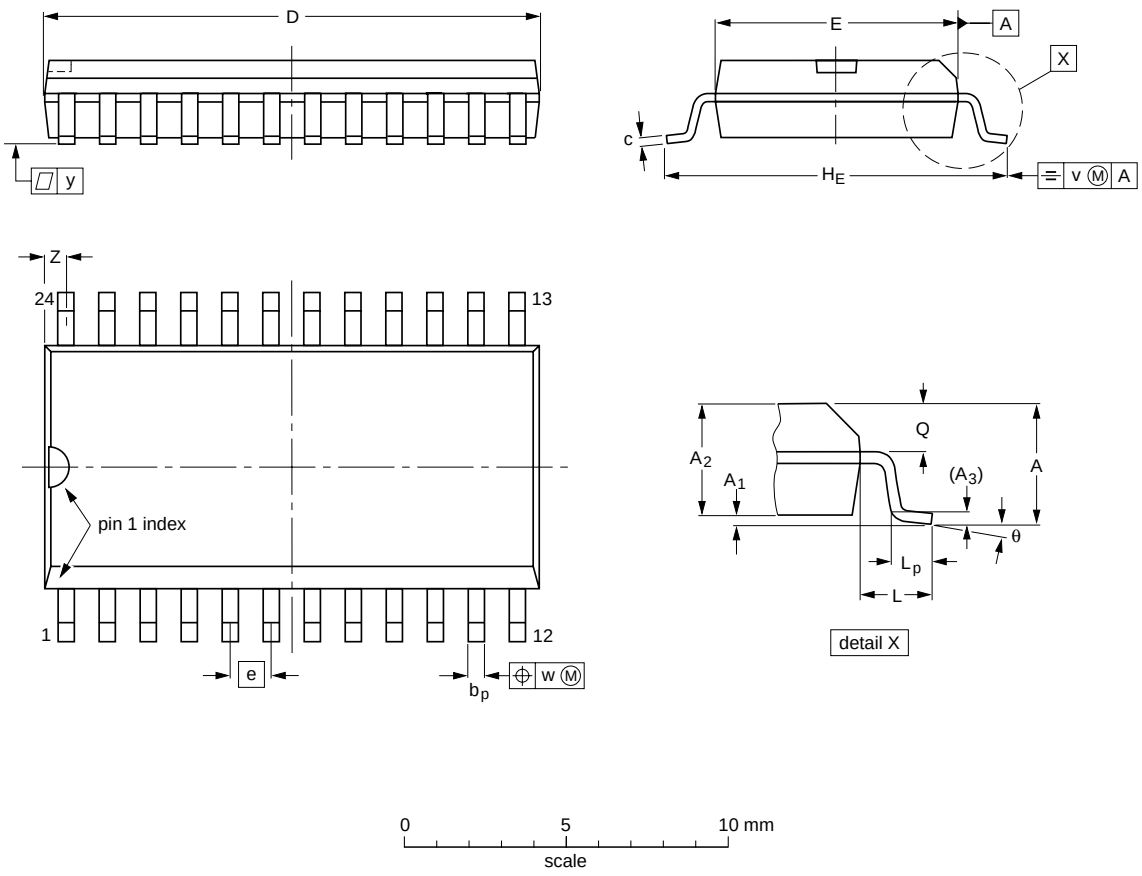


Fig 24. Load circuit

13. Package outline

SO24: plastic small outline package; 24 leads; body width 7.5 mm **SOT137-1**



DIMENSIONS (inch dimensions are derived from the original mm dimensions)

UNIT	A max.	A_1	A_2	A_3	b_p	c	D (1)	E (1)	e	H_E	L	L_p	Q	v	w	y	Z (1)	θ
mm	2.65	0.3 0.1	2.45 2.25	0.25	0.49 0.36	0.32 0.23	15.6 15.2	7.6 7.4	1.27	10.65 10.00	1.4	1.1 0.4	1.1 1.0	0.25	0.25	0.1	0.9 0.4	8^o 0^o
inches	0.1	0.012 0.004	0.096 0.089	0.01	0.019 0.014	0.013 0.009	0.61 0.60	0.30 0.29	0.05	0.419 0.394	0.055	0.043 0.016	0.043 0.039	0.01	0.01	0.004	0.035 0.016	

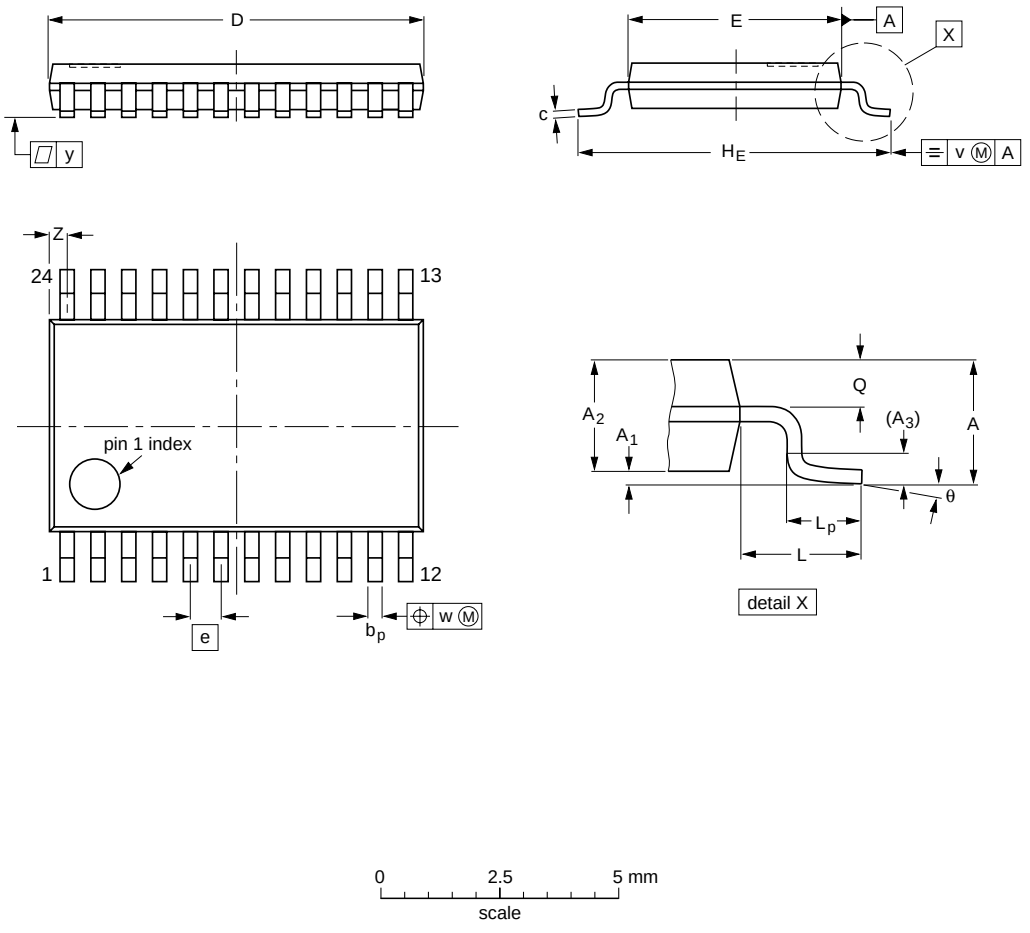
Note

1. Plastic or metal protrusions of 0.15 mm (0.006 inch) maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	JEITA			
SOT137-1	075E05	MS-013				~~99-12-27~~ 03-02-19

Fig 25. Package outline SOT137-1 (SO24)

TSSOP24: plastic thin shrink small outline package; 24 leads; body width 4.4 mm **SOT355-1**



DIMENSIONS (mm are the original dimensions)

UNIT	A max.	A_1	A_2	A_3	b_p	c	$D^{(1)}$	$E^{(2)}$	e	H_E	L	L_p	Q	v	w	y	$Z^{(1)}$	θ
mm	1.1	0.15 0.05	0.95 0.80	0.25	0.30 0.19	0.2 0.1	7.9 7.7	4.5 4.3	0.65	6.6 6.2	1	0.75 0.50	0.4 0.3	0.2	0.13	0.1	0.5 0.2	8^o 0^o

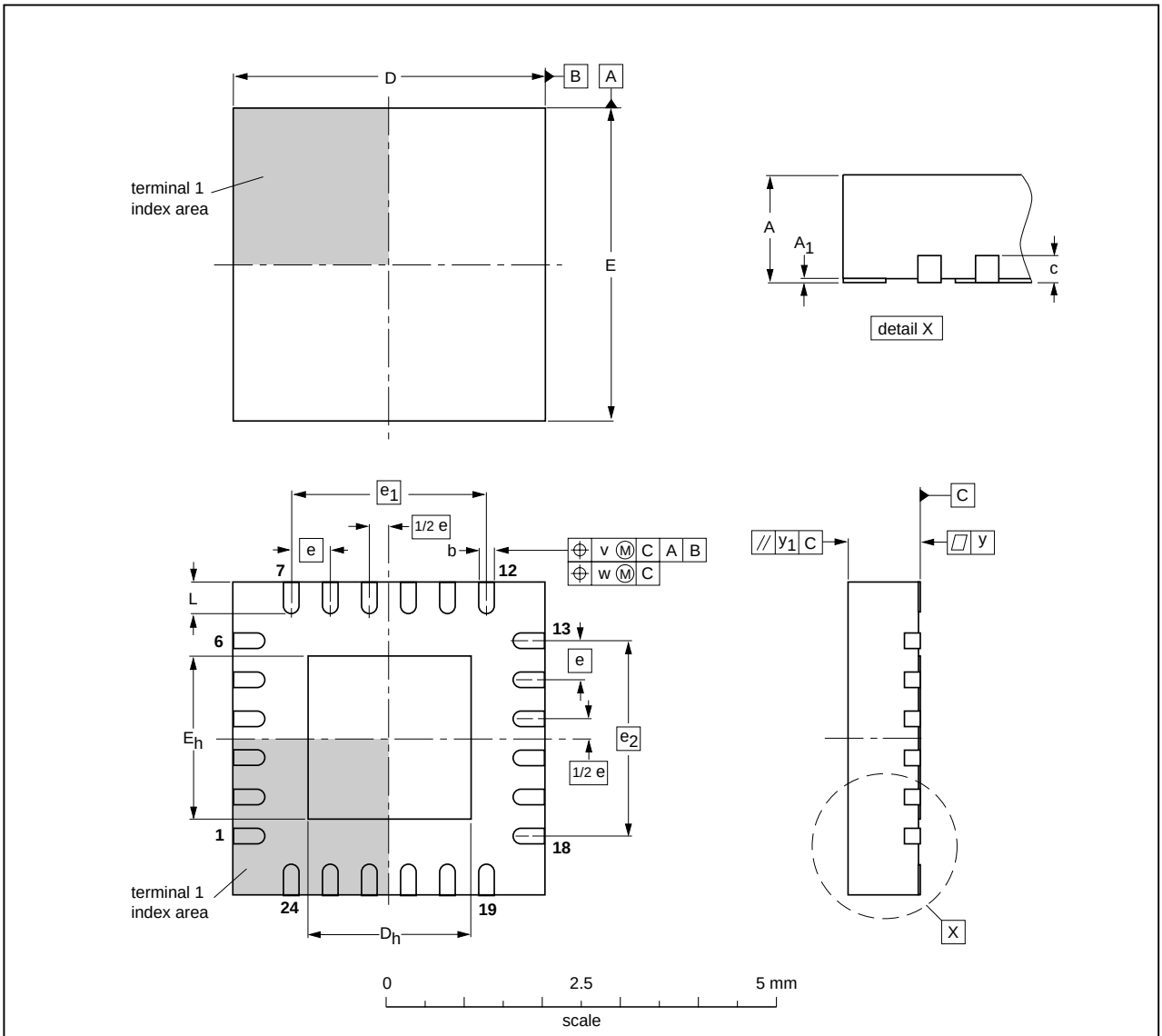
Notes

1. Plastic or metal protrusions of 0.15 mm maximum per side are not included.
2. Plastic interlead protrusions of 0.25 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	JEITA			
SOT355-1		MO-153				~~99-12-27~~ 03-02-19

Fig 26. Package outline SOT355-1 (TSSOP24)

HVQFN24: plastic thermal enhanced very thin quad flat package; no leads; 24 terminals; body 4 x 4 x 0.85 mm **SOT616-1**



DIMENSIONS (mm are the original dimensions)

UNIT	A(1) max.	A_1	b	c	D(1)	D_h	E(1)	E_h	e	e_1	e_2	L	v	w	y	y_1
mm	1	0.05 0.00	0.30 0.18	0.2	4.1 3.9	2.25 1.95	4.1 3.9	2.25 1.95	0.5	2.5	2.5	0.5 0.3	0.1	0.05	0.05	0.1

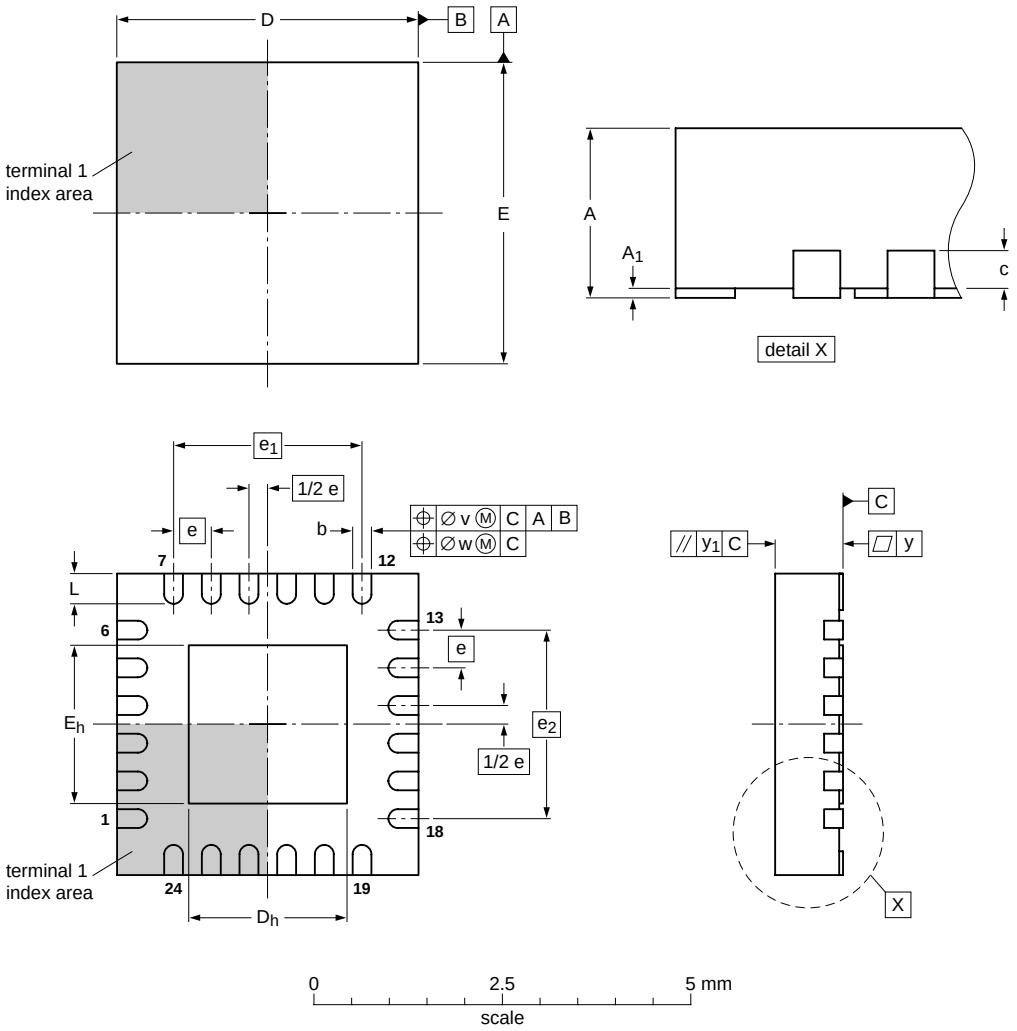
Note

1. Plastic or metal protrusions of 0.075 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	JEITA			
SOT616-1	- - -	MO-220	- - -			~~01-08-08~~ 02-10-22

Fig 27. Package outline SOT616-1 (HVQFN24)

HWQFN24: plastic thermal enhanced very very thin quad flat package; no leads; 24 terminals; body 4 x 4 x 0.75 mm **SOT994-1**



DIMENSIONS (mm are the original dimensions)

UNIT	A(1) max	A_1	b	c	D(1)	D_h	E(1)	E_h	e	e_1	e_2	L	v	w	y	y_1
mm	0.8	0.05 0.00	0.30 0.18	0.2	4.1 3.9	2.25 1.95	4.1 3.9	2.25 1.95	0.5	2.5	2.5	0.5 0.3	0.1	0.05	0.05	0.1

Note

1. Plastic or metal protrusions of 0.075 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	JEITA			
SOT994-1	- - -	MO-220	- - -			~~07-02-07~~ 07-03-03

Fig 28. Package outline SOT994-1 (HWQFN24)

14. Handling information

Inputs and outputs are protected against electrostatic discharge in normal handling. However, to be completely safe you must take normal precautions appropriate to handling integrated circuits.

15. Soldering of SMD packages

This text provides a very brief insight into a complex technology. A more in-depth account of soldering ICs can be found in Application Note *AN10365 "Surface mount reflow soldering description"*.

15.1 Introduction to soldering

Soldering is one of the most common methods through which packages are attached to Printed Circuit Boards (PCBs), to form electrical circuits. The soldered joint provides both the mechanical and the electrical connection. There is no single soldering method that is ideal for all IC packages. Wave soldering is often preferred when through-hole and Surface Mount Devices (SMDs) are mixed on one printed wiring board; however, it is not suitable for fine pitch SMDs. Reflow soldering is ideal for the small pitches and high densities that come with increased miniaturization.

15.2 Wave and reflow soldering

Wave soldering is a joining technology in which the joints are made by solder coming from a standing wave of liquid solder. The wave soldering process is suitable for the following:

- Through-hole components
- Leaded or leadless SMDs, which are glued to the surface of the printed circuit board

Not all SMDs can be wave soldered. Packages with solder balls, and some leadless packages which have solder lands underneath the body, cannot be wave soldered. Also, leaded SMDs with leads having a pitch smaller than ~0.6 mm cannot be wave soldered, due to an increased probability of bridging.

The reflow soldering process involves applying solder paste to a board, followed by component placement and exposure to a temperature profile. Leaded packages, packages with solder balls, and leadless packages are all reflow solderable.

Key characteristics in both wave and reflow soldering are:

- Board specifications, including the board finish, solder masks and vias
- Package footprints, including solder thieves and orientation
- The moisture sensitivity level of the packages
- Package placement
- Inspection and repair
- Lead-free soldering versus SnPb soldering

15.3 Wave soldering

Key characteristics in wave soldering are:

- Process issues, such as application of adhesive and flux, clinching of leads, board transport, the solder wave parameters, and the time during which components are exposed to the wave
- Solder bath specifications, including temperature and impurities

15.4 Reflow soldering

Key characteristics in reflow soldering are:

- Lead-free versus SnPb soldering; note that a lead-free reflow process usually leads to higher minimum peak temperatures (see Figure 29) than a SnPb process, thus reducing the process window
- Solder paste printing issues including smearing, release, and adjusting the process window for a mix of large and small components on one board
- Reflow temperature profile; this profile includes preheat, reflow (in which the board is heated to the peak temperature) and cooling down. It is imperative that the peak temperature is high enough for the solder to make reliable solder joints (a solder paste characteristic). In addition, the peak temperature must be low enough that the packages and/or boards are not damaged. The peak temperature of the package depends on package thickness and volume and is classified in accordance with Table 16 and 17

Table 16. SnPb eutectic process (from J-STD-020C)

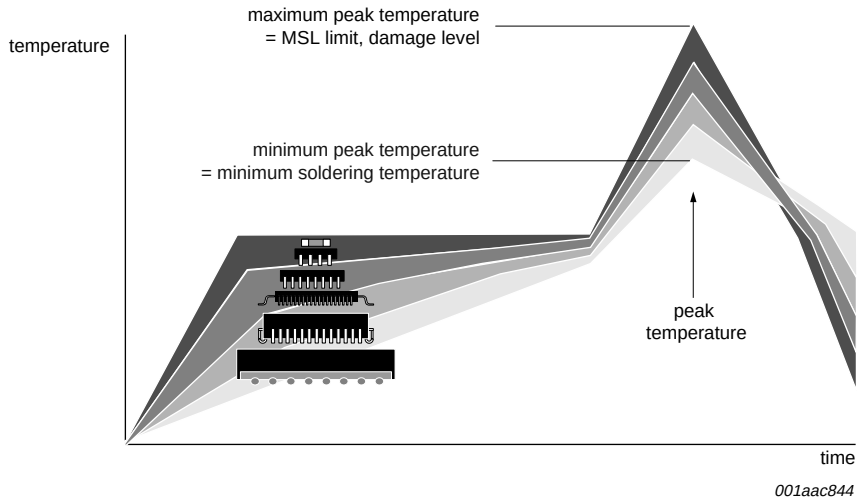
Package thickness (mm)	Package reflow temperature (°C)	
	Volume (mm^3)	
	< 350	≥ 350
< 2.5	235	220
≥ 2.5	220	220

Table 17. Lead-free process (from J-STD-020C)

Package thickness (mm)	Package reflow temperature (°C)		
	Volume (mm^3)		
	< 350	350 to 2000	> 2000
< 1.6	260	260	260
1.6 to 2.5	260	250	245
> 2.5	250	245	245

Moisture sensitivity precautions, as indicated on the packing, must be respected at all times.

Studies have shown that small packages reach higher temperatures during reflow soldering, see Figure 29.



MSL: Moisture Sensitivity Level

Fig 29. Temperature profiles for large and small components

For further information on temperature profiles, refer to Application Note *AN10365 "Surface mount reflow soldering description"*.

16. Abbreviations

Table 18. Abbreviations

Acronym	Description
ACPI	Advanced Configuration and Power Interface
CBT	Cross Bar Technology
CDM	Charged-Device Model
CMOS	Complementary Metal-Oxide Semiconductor
DUT	Device Under Test
ESD	ElectroStatic Discharge
FET	Field-Effect Transistor
GPIO	General Purpose Input/Output
HBM	Human Body Model
I/O	Input/Output
I²C-bus	Inter-Integrated Circuit bus
IC	Integrated Circuit
LED	Light Emitting Diode
MM	Machine Model
PCB	Printed-Circuit Board
SMBus	System Management Bus

17. Revision history

Table 19. Revision history

Document ID	Release date	Data sheet status	Change notice	Supersedes
PCA9535_PCA9535C_5	20080915	Product data sheet	-	PCA9535_PCA9535C_4
Modifications:	• Table 3 “Pin description”: Table note [2] re-written; added its reference at port 1 input/output			
PCA9535_PCA9535C_4	20080731	Product data sheet	-	PCA9535_PCA9535C_3
PCA9535_PCA9535C_3	20071004	Product data sheet	-	PCA9535_2
PCA9535_2 (9397 750 12896)	20040930	Product data sheet	-	PCA9535_1
PCA9535_1 (9397 750 11681)	20030627	Product data	853-2430 30019 of 11 June 2003	-

18. Legal information

18.1 Data sheet status

Document status[1][2]	Product status[3]	Definition
Objective [short] data sheet	Development	This document contains data from the objective specification for product development.
Preliminary [short] data sheet	Qualification	This document contains data from the preliminary specification.
Product [short] data sheet	Production	This document contains the product specification.

[1] Please consult the most recently issued document before initiating or completing a design.

[2] The term 'short data sheet' is explained in section "Definitions".

[3] The product status of device(s) described in this document may have changed since this document was published and may differ in case of multiple devices. The latest product status information is available on the Internet at URL http://www.nxp.com.

18.2 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

Short data sheet — A short data sheet is an extract from a full data sheet with the same product type number(s) and title. A short data sheet is intended for quick reference only and should not be relied upon to contain detailed and full information. For detailed and full information see the relevant full data sheet, which is available on request via the local NXP Semiconductors sales office. In case of any inconsistency or conflict with the short data sheet, the full data sheet shall prevail.

18.3 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Limiting values — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) may cause permanent damage to the device. Limiting values are stress ratings only and operation of the device at these or any other conditions above those given in the Characteristics sections of this document is not implied. Exposure to limiting values for extended periods may affect device reliability.

Terms and conditions of sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, including those pertaining to warranty, intellectual property rights infringement and limitation of liability, unless explicitly otherwise agreed to in writing by NXP Semiconductors. In case of any inconsistency or conflict between information in this document and such terms and conditions, the latter will prevail.

No offer to sell or license — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

18.4 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

I²C-bus — logo is a trademark of NXP B.V.

19. Contact information

For more information, please visit: **http://www.nxp.com**

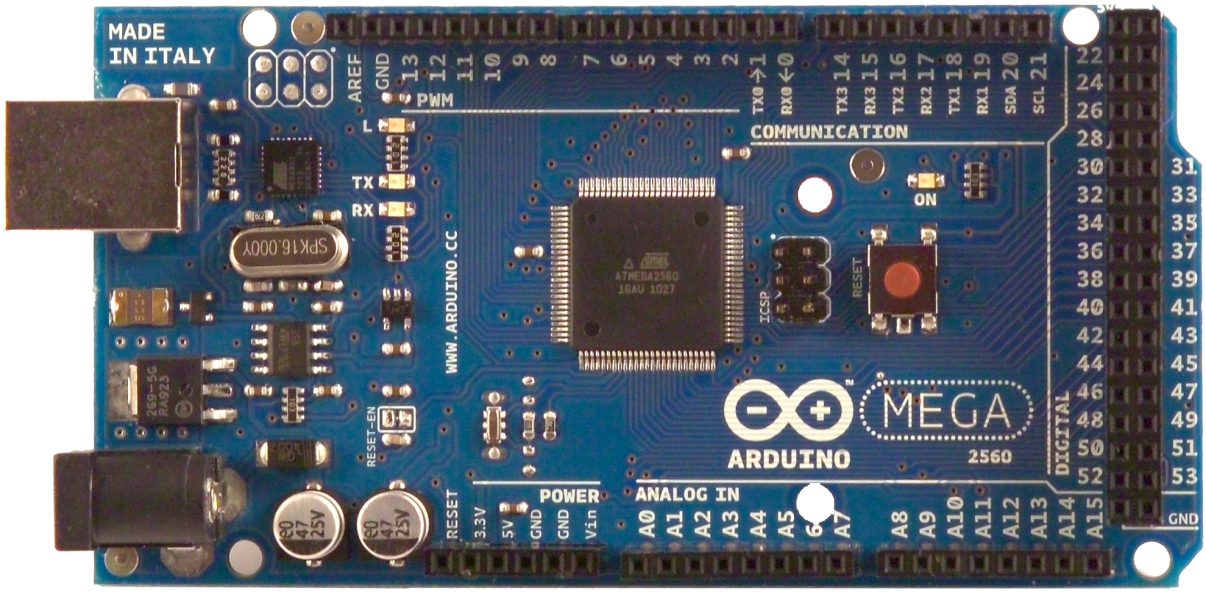
For sales office addresses, please send an email to: **salesaddresses@nxp.com**

20. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 15 September 2008
Document identifier: PCA9535_PCA9535C_5

Arduino MEGA 2560



Product Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Index

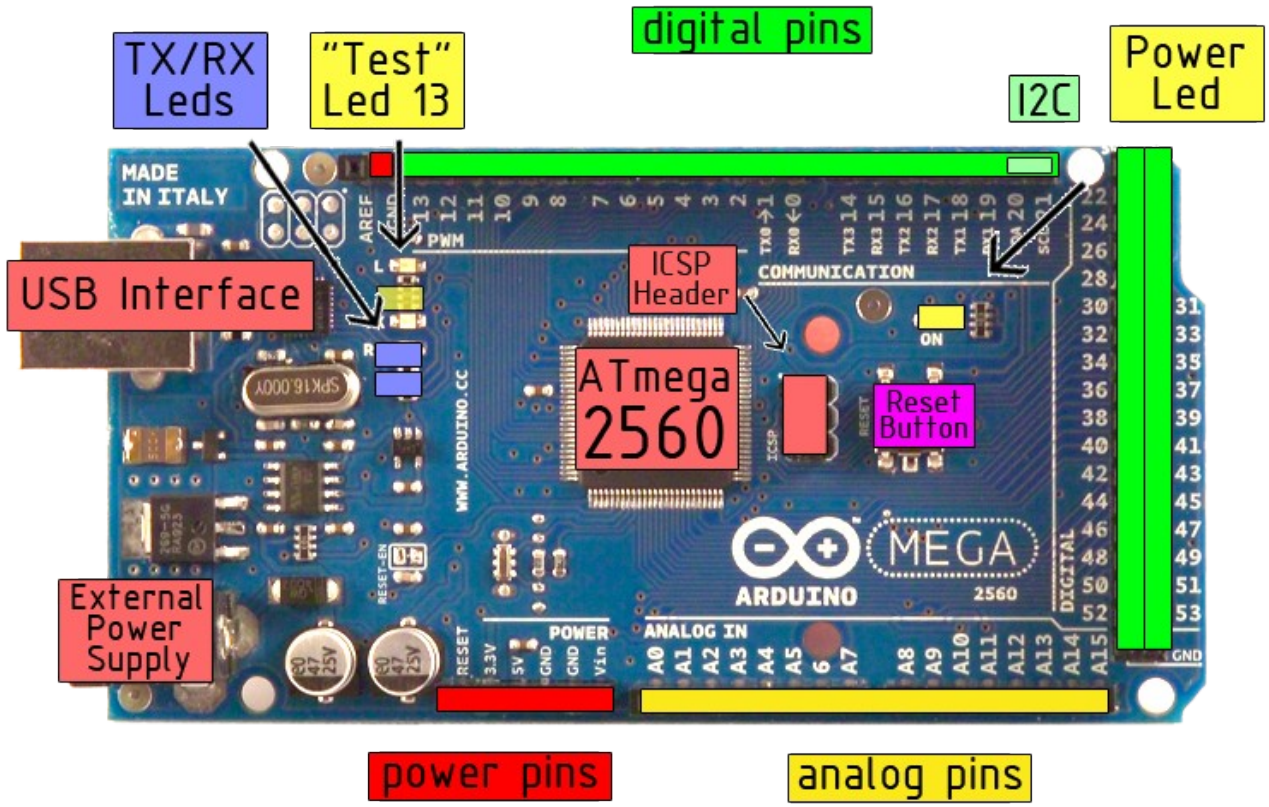
Technical Specification

EAGLE files: arduino-mega2560-reference-design.zip Schematic: arduino-mega2560-schematic.pdf

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

the board



Power

The Arduino Mega2560 can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **I^2C: 20 (SDA) and 21 (SCL).** Support I^2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I^2C pins on the Duemilanove.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Mega's digital pins.

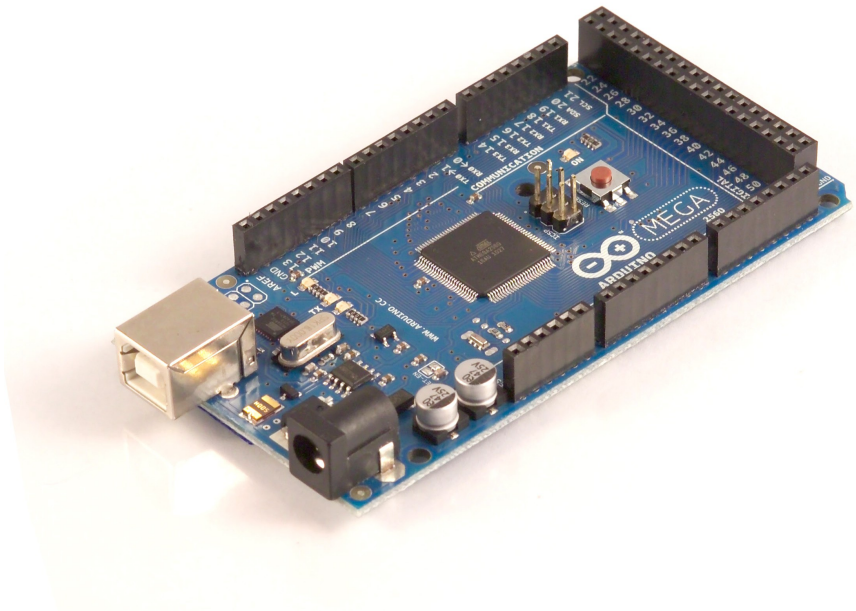
The ATmega2560 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. To use the SPI communication, please see the ATmega2560 datasheet.

Programming

The Arduino Mega2560 can be programmed with the Arduino software (download). For details, see the reference and tutorials.

The Atmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.



Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega is designed to be compatible with most shields designed for the Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega and Duemilanove / Diecimila. **Please note that I^2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).**

How to use Arduino

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your personal OS. Check on the Arduino site for the latest instructions. *http://arduino.cc/en/Guide/HomePage*

Linux Install

Windows Install

Mac Install

Once you have downloaded/unzipped the arduino IDE, you can Plug the Arduino to your PC via USB cable.

Blink led

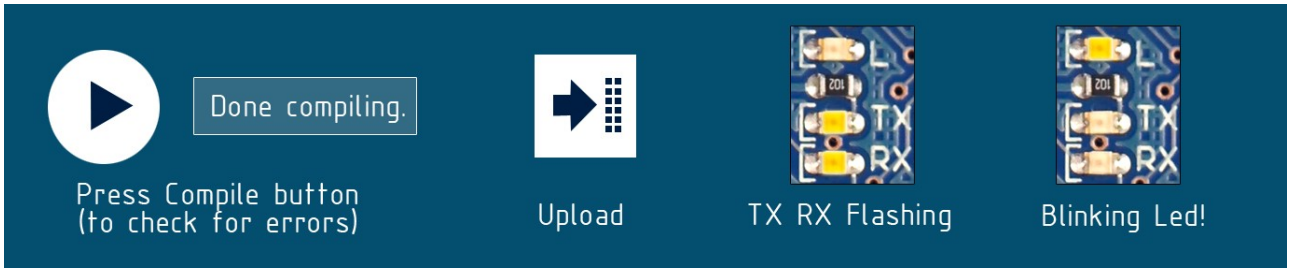
Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well known programming "hello world", select

File>Sketchbook>
Arduino-0017>Examples>
Digital>Blink

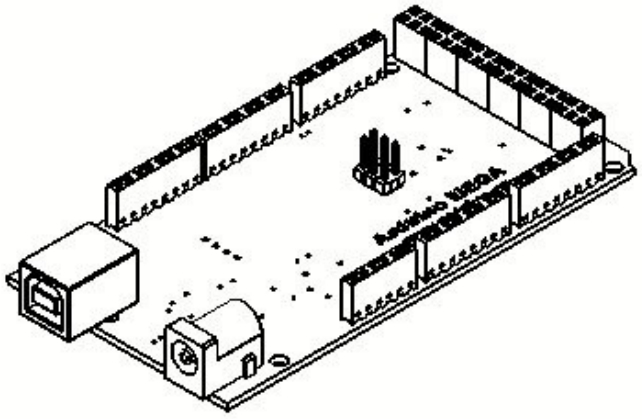
Once you have your skecth you'll see something very close to the screenshot on the right.

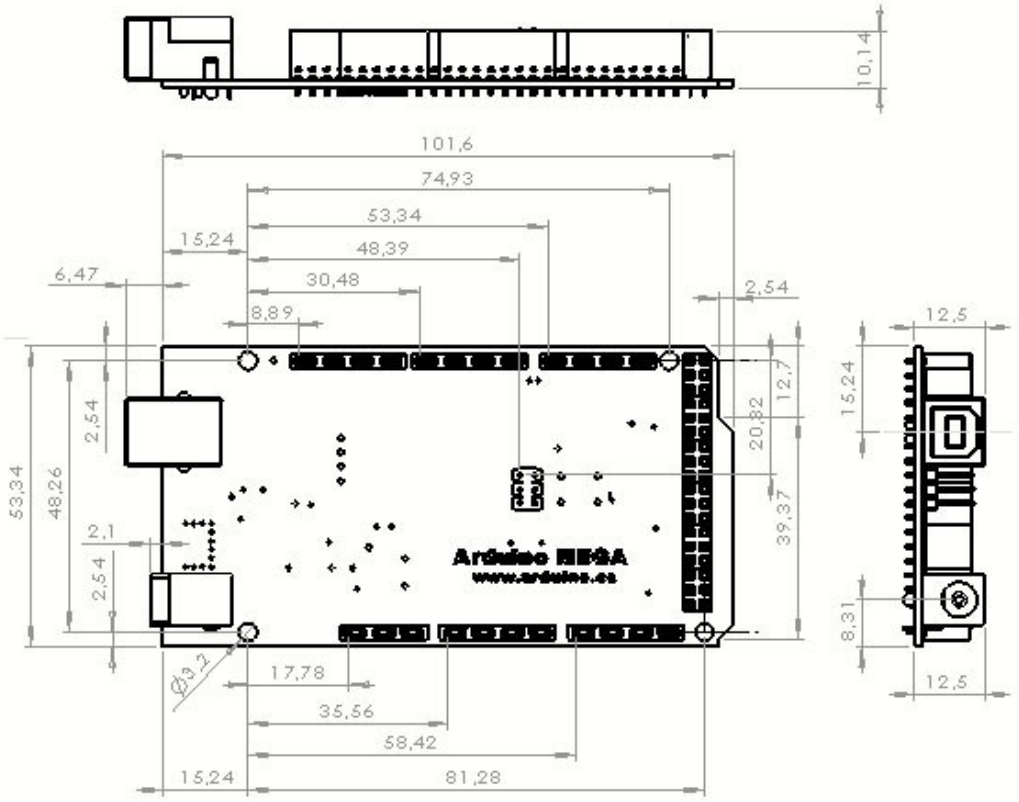
In **Tools>Board** select MEGA

Now you have to go to
Tools>SerialPort
and select the right serial port, the one arduino is attached to.



Dimensioned Drawing





Terms & Conditions

1. Warranties

1.1 The producer warrants that its products will conform to the Specifications. This warranty lasts for one (1) years from the date of the sale. The producer shall not be liable for any defects that are caused by neglect, misuse or mistreatment by the Customer, including improper installation or testing, or for any products that have been altered or modified in any way by a Customer. Moreover, The producer shall not be liable for any defects that result from Customer's design, specifications or instructions for such products. Testing and other quality control techniques are used to the extent the producer deems necessary.

1.2 If any products fail to conform to the warranty set forth above, the producer's sole liability shall be to replace such products. The producer's liability shall be limited to products that are determined by the producer not to conform to such warranty. If the producer elects to replace such products, the producer shall have a reasonable time to replacements. Replaced products shall be warranted for a new full warranty period.

1.3 EXCEPT AS SET FORTH ABOVE, PRODUCTS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS." THE PRODUCER DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

1.4 Customer agrees that prior to using any systems that include the producer products, Customer will test such systems and the functionality of the products as used in such systems. The producer may provide technical, applications or design advice, quality characterization, reliability data or other services. Customer acknowledges and agrees that providing these services shall not expand or otherwise alter the producer's warranties, as set forth above, and no additional obligations or liabilities shall arise from the producer providing such services.

1.5 The Arduino™ products are not authorized for use in safety-critical applications where a failure of the product would reasonably be expected to cause severe personal injury or death. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Arduino™ products are neither designed nor intended for use in military or aerospace applications or environments and for automotive applications or environment. Customer acknowledges and agrees that any such use of Arduino™ products which is solely at the Customer's risk, and that Customer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

1.6 Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products and any use of Arduino™ products in Customer's applications, notwithstanding any applications-related information or support that may be provided by the producer.

2. Indemnification

The Customer acknowledges and agrees to defend, indemnify and hold harmless the producer from and against any and all third-party losses, damages, liabilities and expenses it incurs to the extent directly caused by: (i) an actual breach by a Customer of the representation and warranties made under this terms and conditions or (ii) the gross negligence or willful misconduct by the Customer.

3. Consequential Damages Waiver

In no event the producer shall be liable to the Customer or any third parties for any special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of the products provided hereunder, regardless of whether the producer has been advised of the possibility of such damages. This section will survive the termination of the warranty period.

4. Changes to specifications

The producer may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." The producer reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

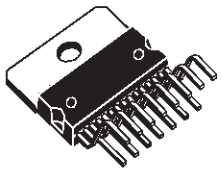
Enviromental Policies

The producer of Arduino™ has joined the Impatto Zero® policy of LifeGate.it. For each Arduino board produced is created / looked after half squared Km of Costa Rica's forest's.

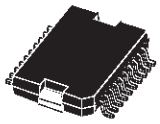
L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)



Multiwatt15



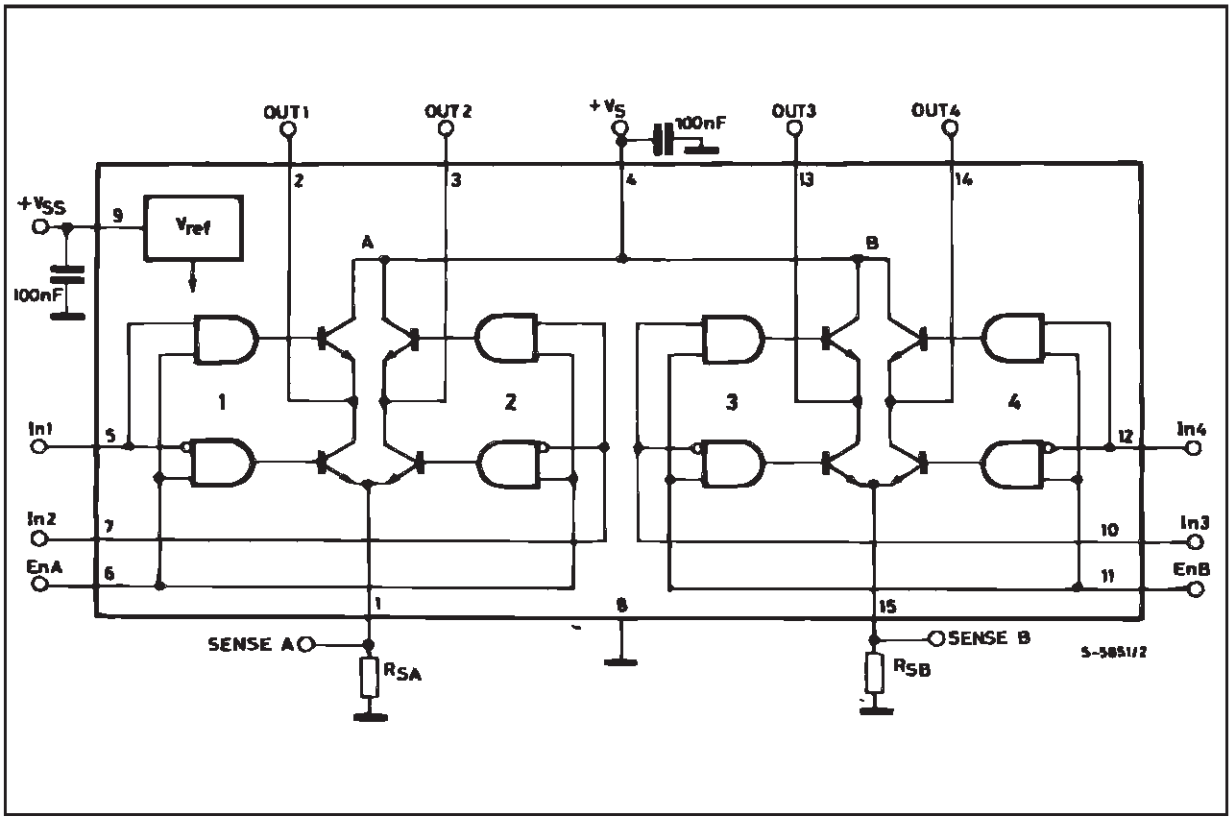
PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

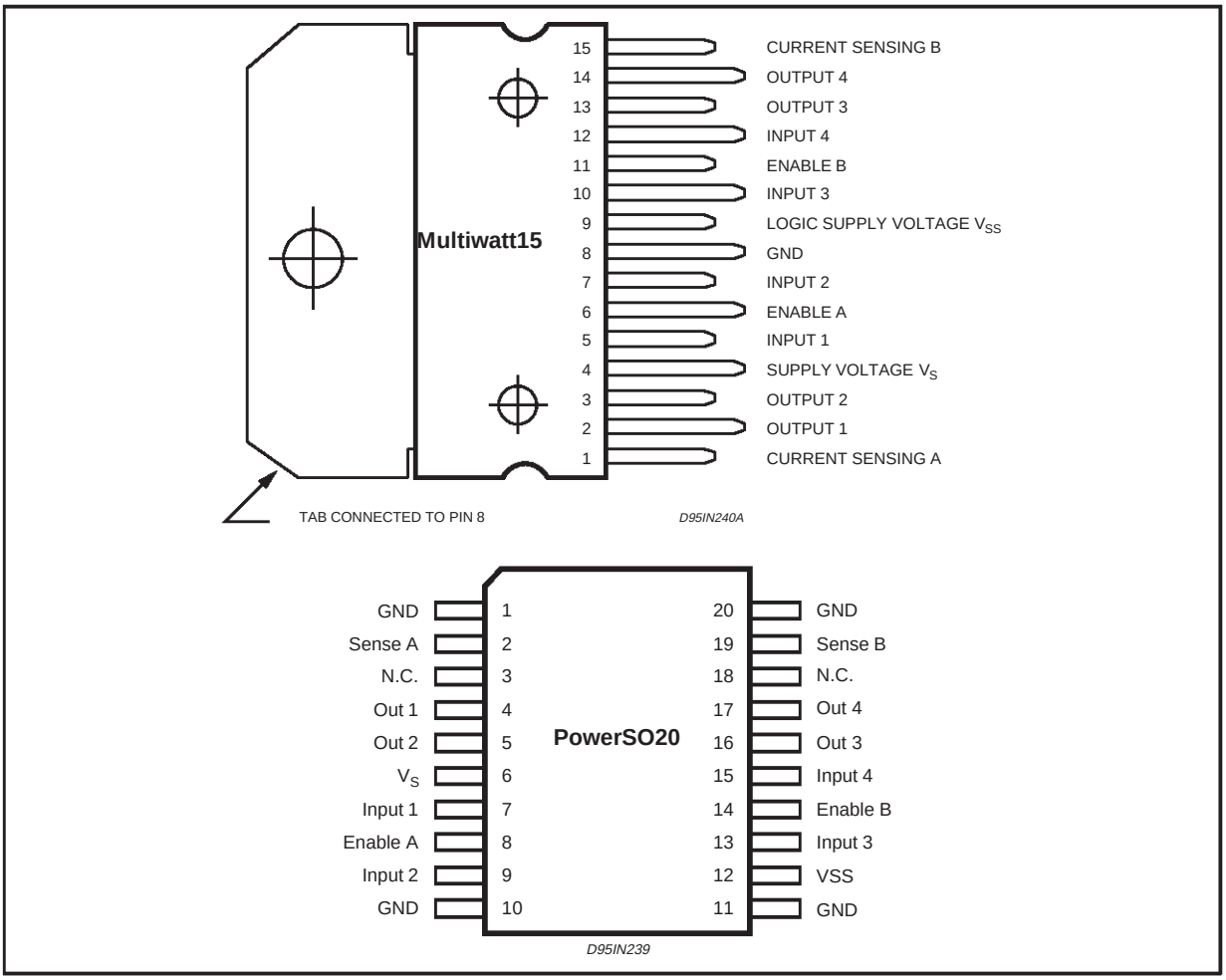
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I, V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)



THERMAL DATA

Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	Max.	–	3	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	°C/W

(*) Mounted on aluminum substrate

PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V_S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3;18	N.C.	Not Connected

ELECTRICAL CHARACTERISTICS (V_S = 42V; V_{SS} = 5V, T_j = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 4)	Operative Condition	V_{IH} +2.5		46	V
V_{SS}	Logic Supply Voltage (pin 9)		4.5	5	7	V
I_S	Quiescent Supply Current (pin 4)	V_{en} = H; I_L = 0 V_i = L V_i = H		13 50	22 70	mA mA
		V_{en} = L V_i = X			4	mA
I_{SS}	Quiescent Current from V_{SS} (pin 9)	V_{en} = H; I_L = 0 V_i = L V_i = H		24 7	36 12	mA mA
		V_{en} = L V_i = X			6	mA
V_{iL}	Input Low Voltage (pins 5, 7, 10, 12)		–0.3		1.5	V
V_{iH}	Input High Voltage (pins 5, 7, 10, 12)		2.3		VSS	V
I_{iL}	Low Voltage Input Current (pins 5, 7, 10, 12)	V_i = L			–10	µA
I_{iH}	High Voltage Input Current (pins 5, 7, 10, 12)	Vi = H ≤ V_{SS} –0.6V		30	100	µA
V_{en} = L	Enable Low Voltage (pins 6, 11)		–0.3		1.5	V
V_{en} = H	Enable High Voltage (pins 6, 11)		2.3		V_{SS}	V
I_{en} = L	Low Voltage Enable Current (pins 6, 11)	V_{en} = L			–10	µA
I_{en} = H	High Voltage Enable Current (pins 6, 11)	V_{en} = H ≤ V_{SS} –0.6V		30	100	µA
$V_{CEsat(H)}$	Source Saturation Voltage	I_L = 1A I_L = 2A	0.95	1.35 2	1.7 2.7	V V
$V_{CEsat(L)}$	Sink Saturation Voltage	I_L = 1A (5) I_L = 2A (5)	0.85	1.2 1.7	1.6 2.3	V V
V_{CEsat}	Total Drop	I_L = 1A (5) I_L = 2A (5)	1.80		3.2 4.9	V V
V_{sens}	Sensing Voltage (pins 1, 15)		–1 (1)		2	V

ELECTRICAL CHARACTERISTICS (continued)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
T_1 (V_i)	Source Current Turn-off Delay	0.5 V_i to 0.9 I_L (2); (4)		1.5		µs
T_2 (V_i)	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		0.2		µs
T_3 (V_i)	Source Current Turn-on Delay	0.5 V_i to 0.1 I_L (2); (4)		2		µs
T_4 (V_i)	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.7		µs
T_5 (V_i)	Sink Current Turn-off Delay	0.5 V_i to 0.9 I_L (3); (4)		0.7		µs
T_6 (V_i)	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.25		µs
T_7 (V_i)	Sink Current Turn-on Delay	0.5 V_i to 0.9 I_L (3); (4)		1.6		µs
T_8 (V_i)	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.2		µs
fc (V_i)	Commutation Frequency	I_L = 2A		25	40	KHz
T_1 (V_{en})	Source Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (2); (4)		3		µs
T_2 (V_{en})	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		1		µs
T_3 (V_{en})	Source Current Turn-on Delay	0.5 V_{en} to 0.1 I_L (2); (4)		0.3		µs
T_4 (V_{en})	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.4		µs
T_5 (V_{en})	Sink Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (3); (4)		2.2		µs
T_6 (V_{en})	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.35		µs
T_7 (V_{en})	Sink Current Turn-on Delay	0.5 V_{en} to 0.9 I_L (3); (4)		0.25		µs
T_8 (V_{en})	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.1		µs

1) 1)Sensing voltage can be –1 V for t ≤ 50 µsec; in steady state V_{sens} min ≥ –0.5 V.
2) See fig. 2.
3) See fig. 4.
4) The load must be a pure resistor.

Figure 1 : Typical Saturation Voltage vs. Output Current.

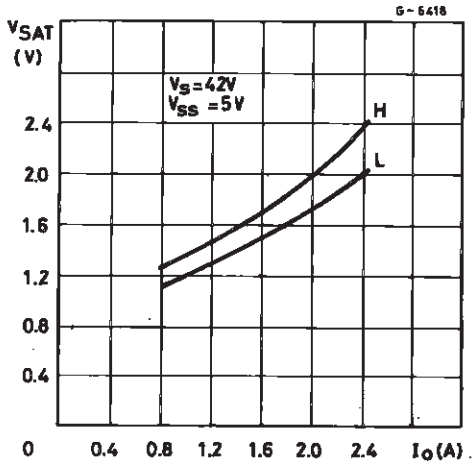
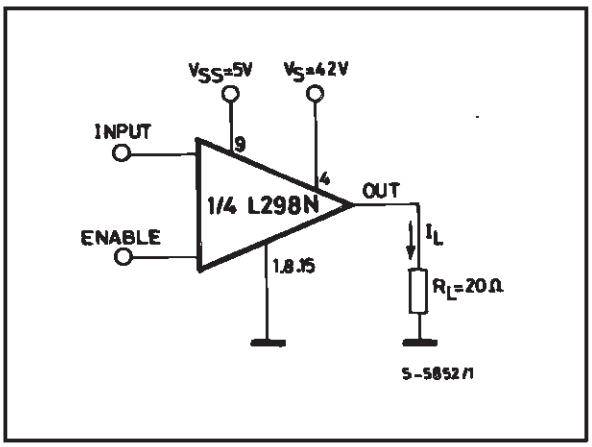


Figure 2 : Switching Times Test Circuits.



Note : For INPUT Switching, set EN = H
For ENABLE Switching, set IN = H

Figure 3 : Source Current Delay Times vs. Input or Enable Switching.

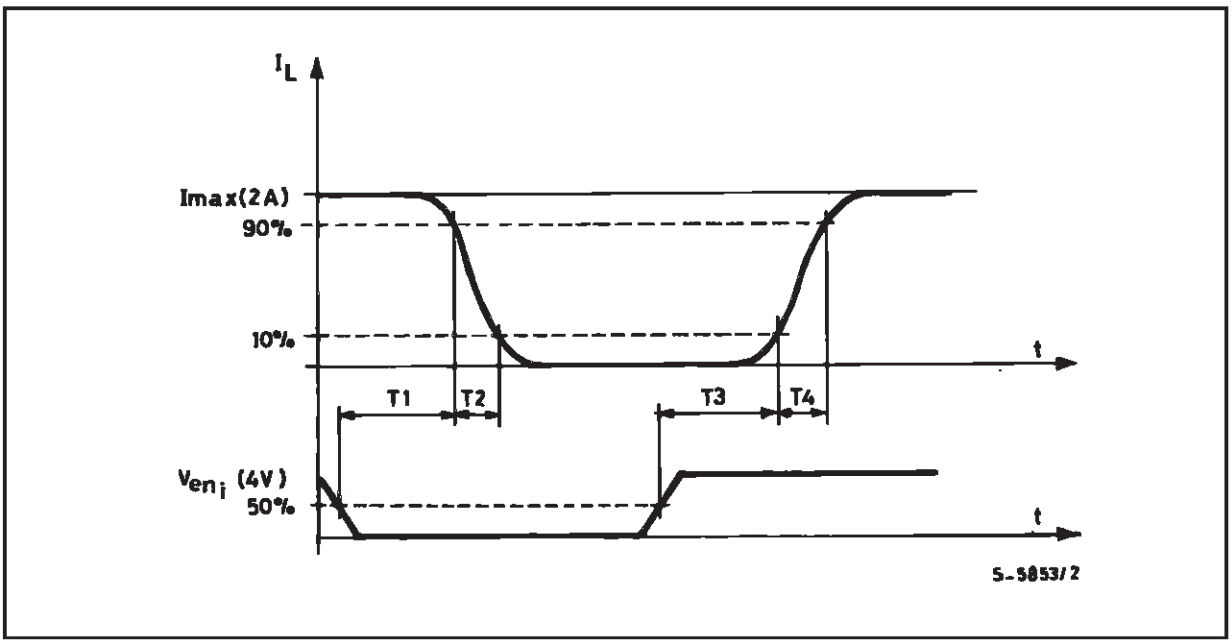
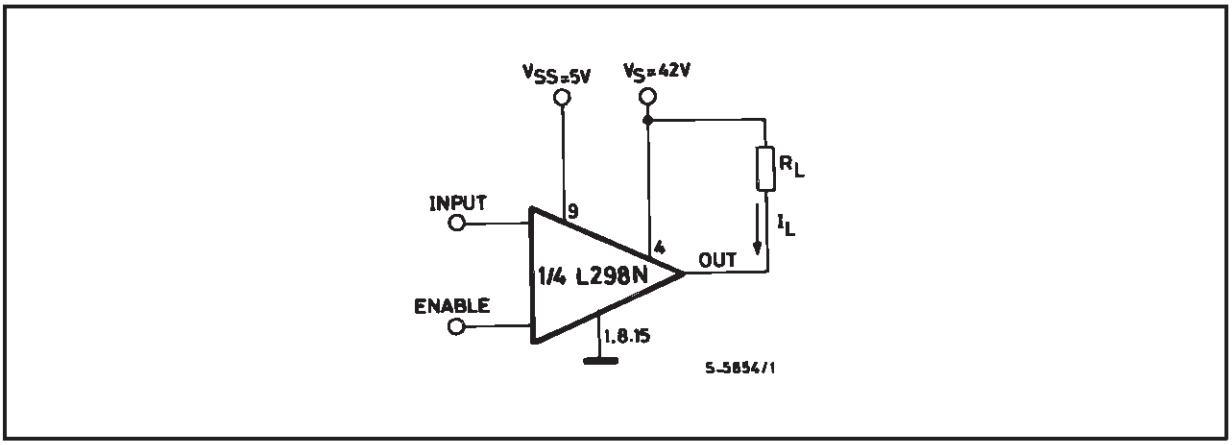


Figure 4 : Switching Times Test Circuits.



Note : For INPUT Switching, set EN = H
For ENABLE Switching, set IN = L

Figure 5 : Sink Current Delay Times vs. Input 0 V Enable Switching.

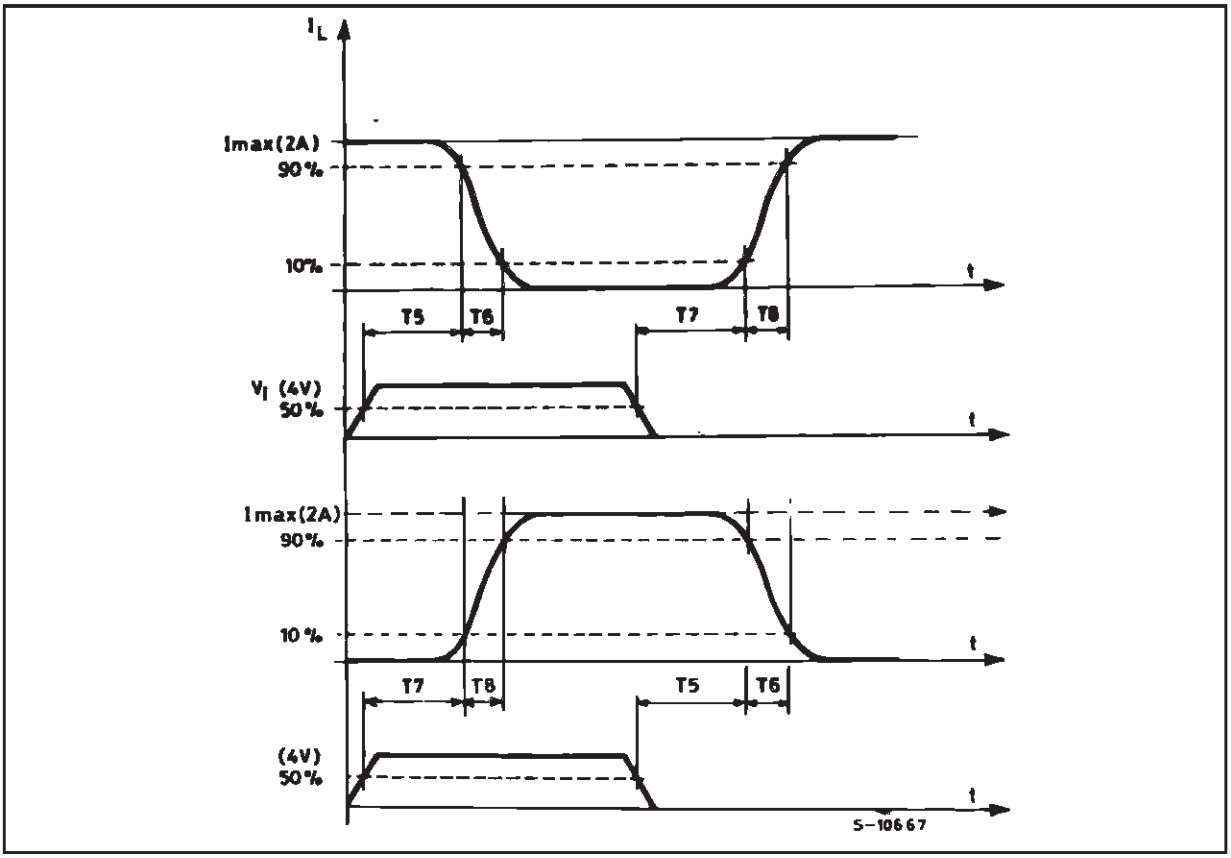
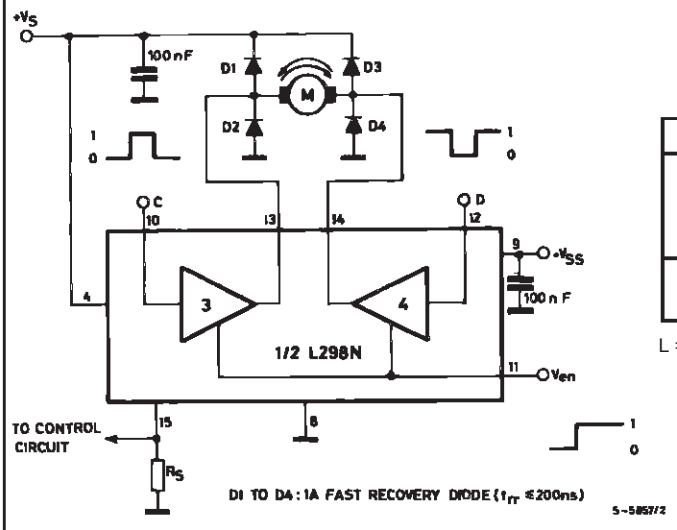


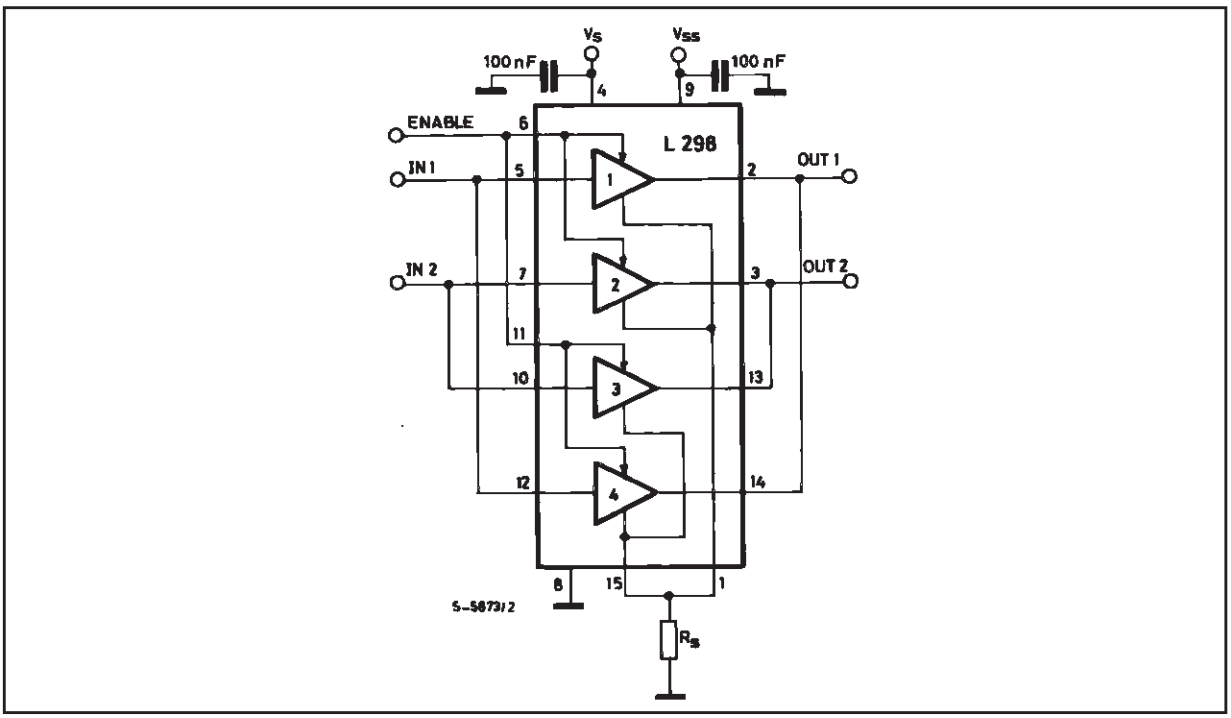
Figure 6 : Bidirectional DC Motor Control.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Figure 7 : For higher currents, outputs can be paralleled. Take care to parallel channel 1 with channel 4 and channel 2 with channel 3.



APPLICATION INFORMATION (Refer to the block diagram)

1.1. POWER OUTPUT STAGE

The L298 integrates two power output stages (A ; B). The power output stage is a bridge configuration and its outputs can drive an inductive load in common or differenzial mode, depending on the state of the inputs. The current that flows through the load comes out from the bridge at the sense output : an external resistor (R_{SA} ; R_{SB}.) allows to detect the intensity of this current.

1.2. INPUT STAGE

Each bridge is driven by means of four gates the input of which are In1 ; In2 ; EnA and In3 ; In4 ; EnB. The In inputs set the bridge state when The En input is high ; a low state of the En input inhibits the bridge. All the inputs are TTL compatible.

2. SUGGESTIONS

A non inductive capacitor, usually of 100 nF, must be foreseen between both Vs and Vss, to ground, as near as possible to GND pin. When the large capacitor of the power supply is too far from the IC, a second smaller one must be foreseen near the L298.

The sense resistor, not of a wire wound type, must be grounded near the negative pole of Vs that must be near the GND pin of the I.C.

Each input must be connected to the source of the driving signals by means of a very short path.

Turn-On and Turn-Off : Before to Turn-ON the Supply Voltage and before to Turn it OFF, the Enable input must be driven to the Low state.

3. APPLICATIONS

Fig 6 shows a bidirectional DC motor control Schematic Diagram for which only one bridge is needed. The external bridge of diodes D1 to D4 is made by four fast recovery elements (trr $\leq$ 200 nsec) that must be chosen of a VF as low as possible at the worst case of the load current.

The sense output voltage can be used to control the current amplitude by chopping the inputs, or to provide overcurrent protection by switching low the enable input.

The brake function (Fast motor stop) requires that the Absolute Maximum Rating of 2 Amps must never be overcome.

When the repetitive peak current needed from the load is higher than 2 Amps, a paralleled configuration can be chosen (See Fig.7).

An external bridge of diodes are required when inductive loads are driven and when the inputs of the IC are chopped ; Shottky diodes would be preferred.

This solution can drive until 3 Amps In DC operation and until 3.5 Amps of a repetitive peak current.

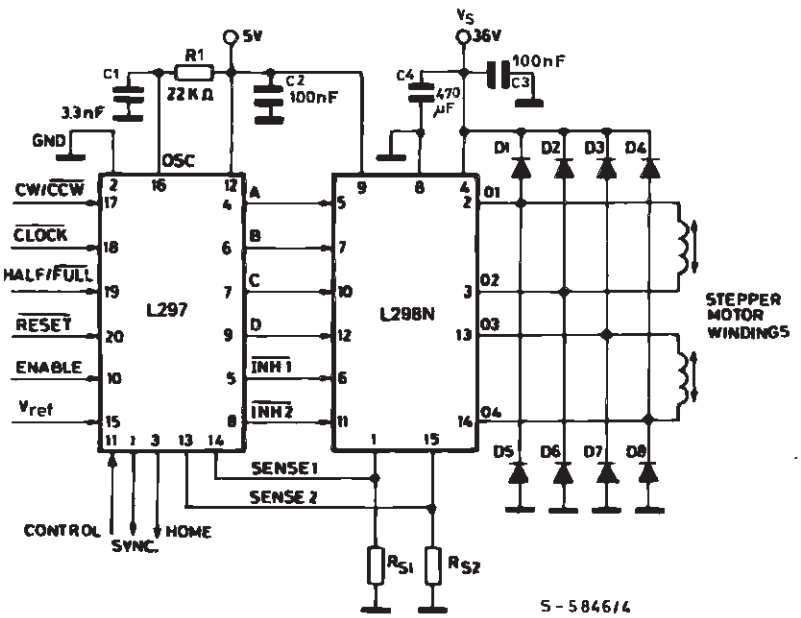
On Fig 8 it is shown the driving of a two phase bipolar stepper motor ; the needed signals to drive the inputs of the L298 are generated, in this example, from the IC L297.

Fig 9 shows an example of P.C.B. designed for the application of Fig 8.

Fig 10 shows a second two phase bipolar stepper motor control circuit where the current is controlled by the I.C. L6506.

Figure 8 : Two Phase Bipolar Stepper Motor Circuit.

This circuit drives bipolar stepper motors with winding currents up to 2 A. The diodes are fast 2 A types.



$R_{S1} = R_{S2} = 0.5\ \Omega$

D1 to D8 = 2 A Fast diodes { $V_F \leq 1.2$ V @ I = 2 A ; trr ≤ 200 ns

Figure 9 : Suggested Printed Circuit Board Layout for the Circuit of fig. 8 (1:1 scale).

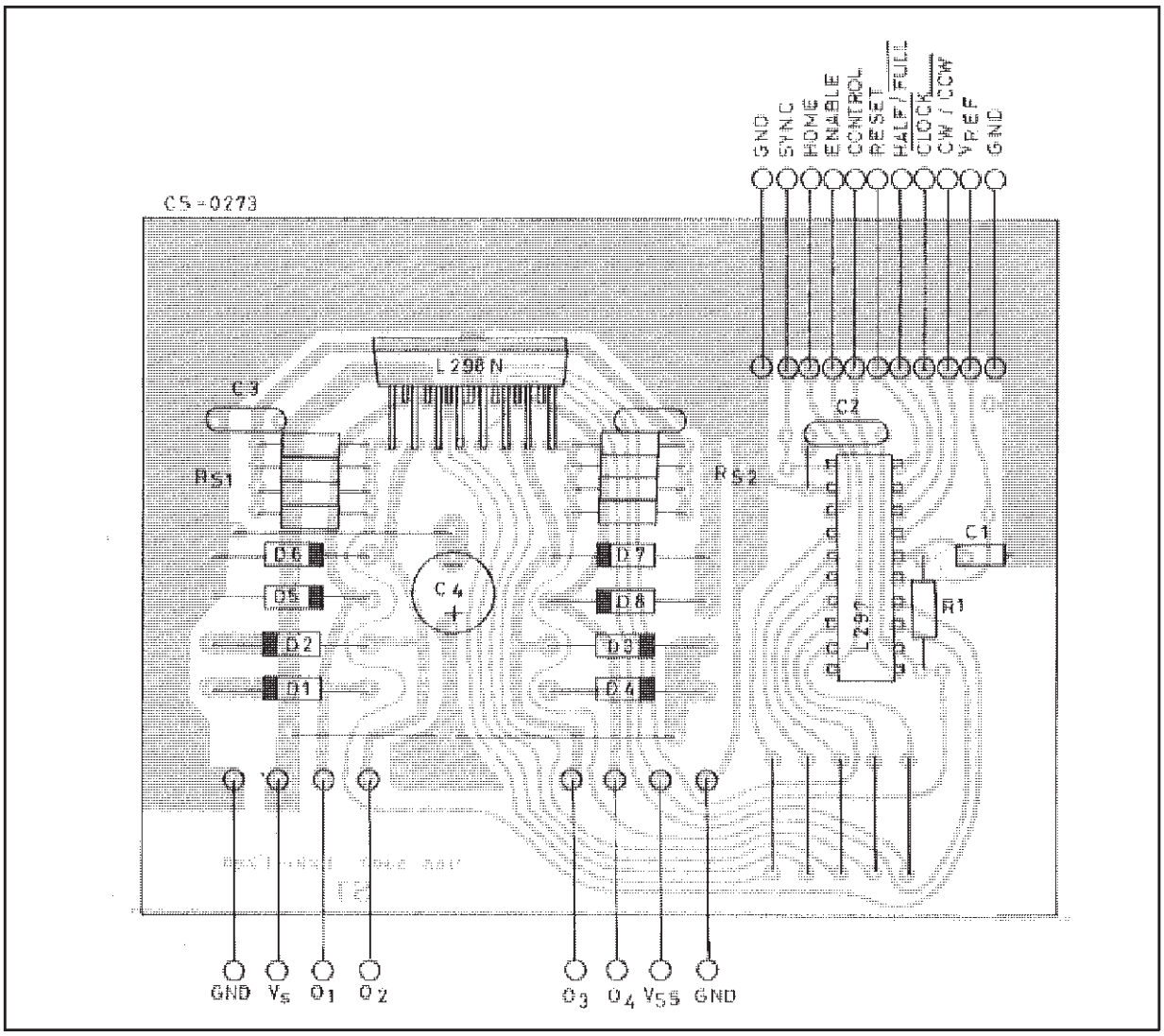
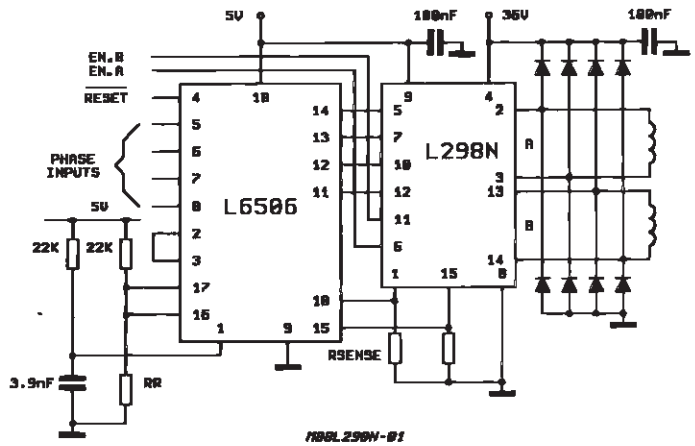


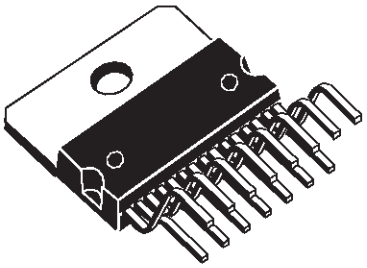
Figure 10 : Two Phase Bipolar Stepper Motor Control Circuit by Using the Current Controller L6506.



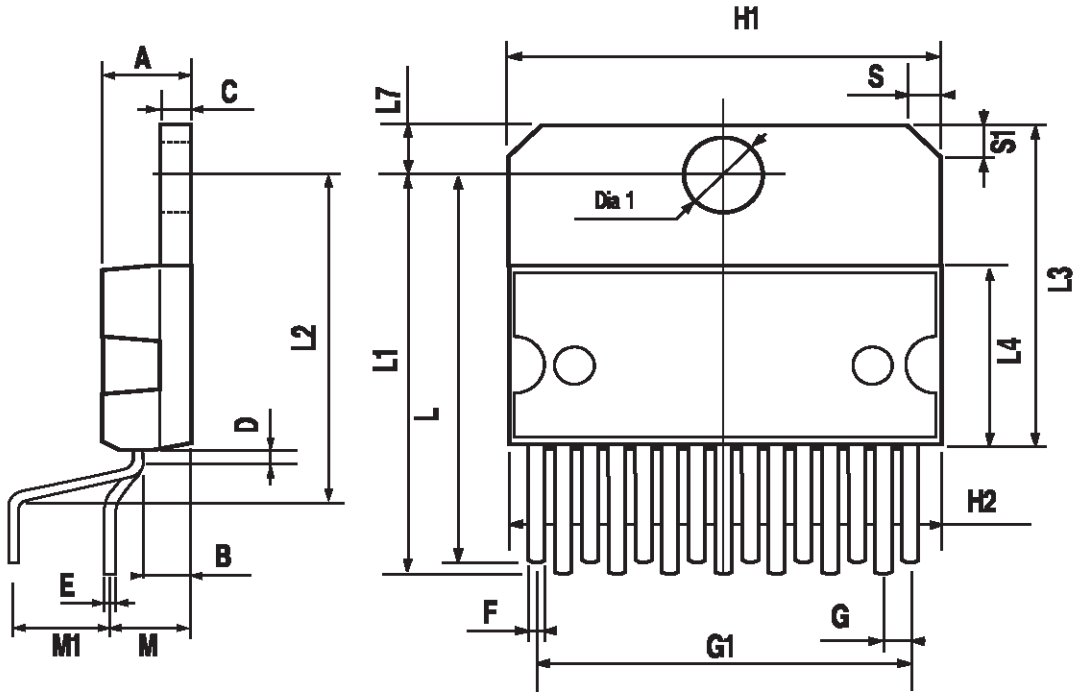
R_R and R_{sense} depend from the load current

DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A			5			0.197
B			2.65			0.104
C			1.6			0.063
D		1			0.039	
E	0.49		0.55	0.019		0.022
F	0.66		0.75	0.026		0.030
G	1.02	1.27	1.52	0.040	0.050	0.060
G1	17.53	17.78	18.03	0.690	0.700	0.710
H1	19.6			0.772		
H2			20.2			0.795
L	21.9	22.2	22.5	0.862	0.874	0.886
L1	21.7	22.1	22.5	0.854	0.870	0.886
L2	17.65		18.1	0.695		0.713
L3	17.25	17.5	17.75	0.679	0.689	0.699
L4	10.3	10.7	10.9	0.406	0.421	0.429
L7	2.65		2.9	0.104		0.114
M	4.25	4.55	4.85	0.167	0.179	0.191
M1	4.63	5.08	5.53	0.182	0.200	0.218
S	1.9		2.6	0.075		0.102
S1	1.9		2.6	0.075		0.102
Dia1	3.65		3.85	0.144		0.152

OUTLINE AND MECHANICAL DATA

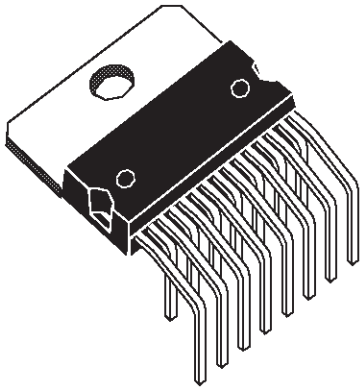


Multiwatt15 V

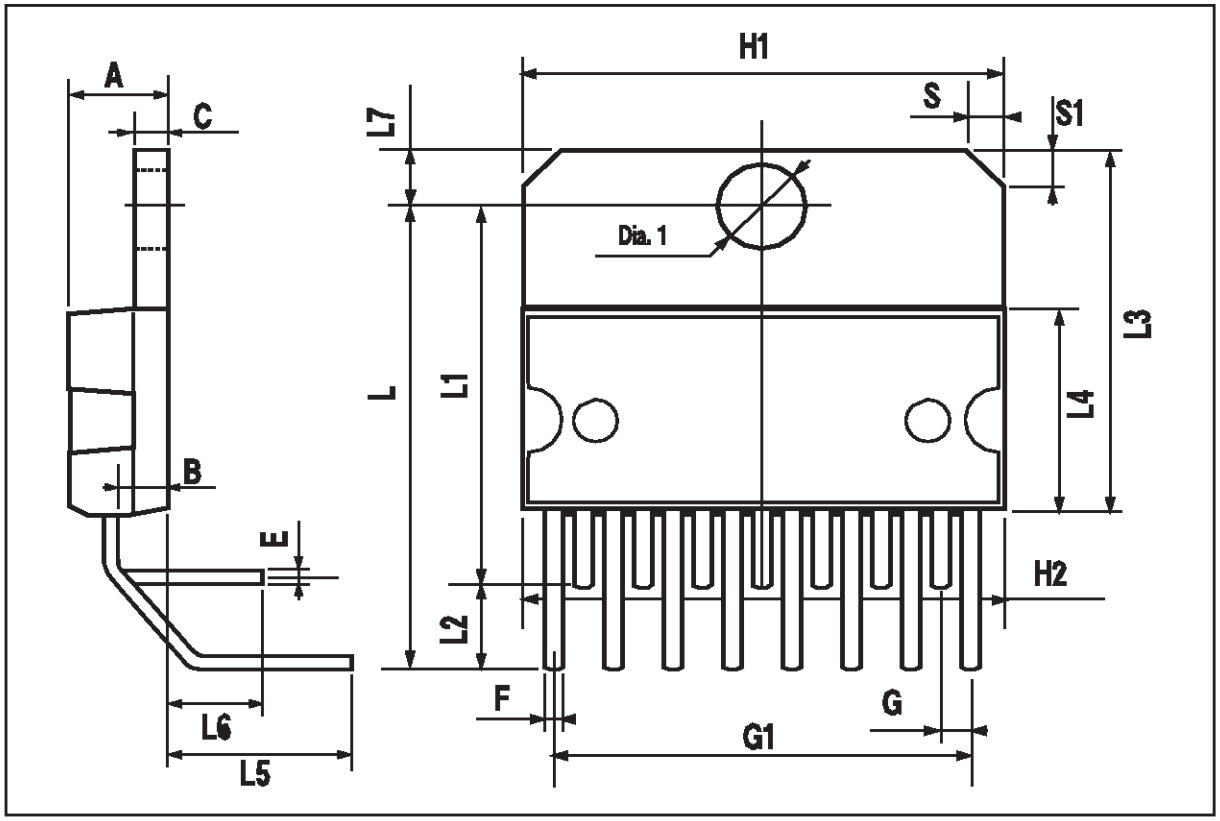


DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A			5			0.197
B			2.65			0.104
C			1.6			0.063
E	0.49		0.55	0.019		0.022
F	0.66		0.75	0.026		0.030
G	1.14	1.27	1.4	0.045	0.050	0.055
G1	17.57	17.78	17.91	0.692	0.700	0.705
H1	19.6			0.772		
H2			20.2			0.795
L		20.57			0.810	
L1		18.03			0.710	
L2		2.54			0.100	
L3	17.25	17.5	17.75	0.679	0.689	0.699
L4	10.3	10.7	10.9	0.406	0.421	0.429
L5		5.28			0.208	
L6		2.38			0.094	
L7	2.65		2.9	0.104		0.114
S	1.9		2.6	0.075		0.102
S1	1.9		2.6	0.075		0.102
Dia1	3.65		3.85	0.144		0.152

OUTLINE AND MECHANICAL DATA



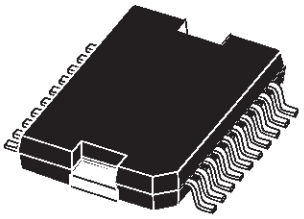
Multiwatt15 H



DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A			3.6			0.142
a1	0.1		0.3	0.004		0.012
a2			3.3			0.130
a3	0		0.1	0.000		0.004
b	0.4		0.53	0.016		0.021
c	0.23		0.32	0.009		0.013
D (1)	15.8		16	0.622		0.630
D1	9.4		9.8	0.370		0.386
E	13.9		14.5	0.547		0.570
e		1.27			0.050	
e3		11.43			0.450	
E1 (1)	10.9		11.1	0.429		0.437
E2			2.9			0.114
E3	5.8		6.2	0.228		0.244
G	0		0.1	0.000		0.004
H	15.5		15.9	0.610		0.626
h			1.1			0.043
L	0.8		1.1	0.031		0.043
N	10° (max.)					
S	8° (max.)					
T		10			0.394	

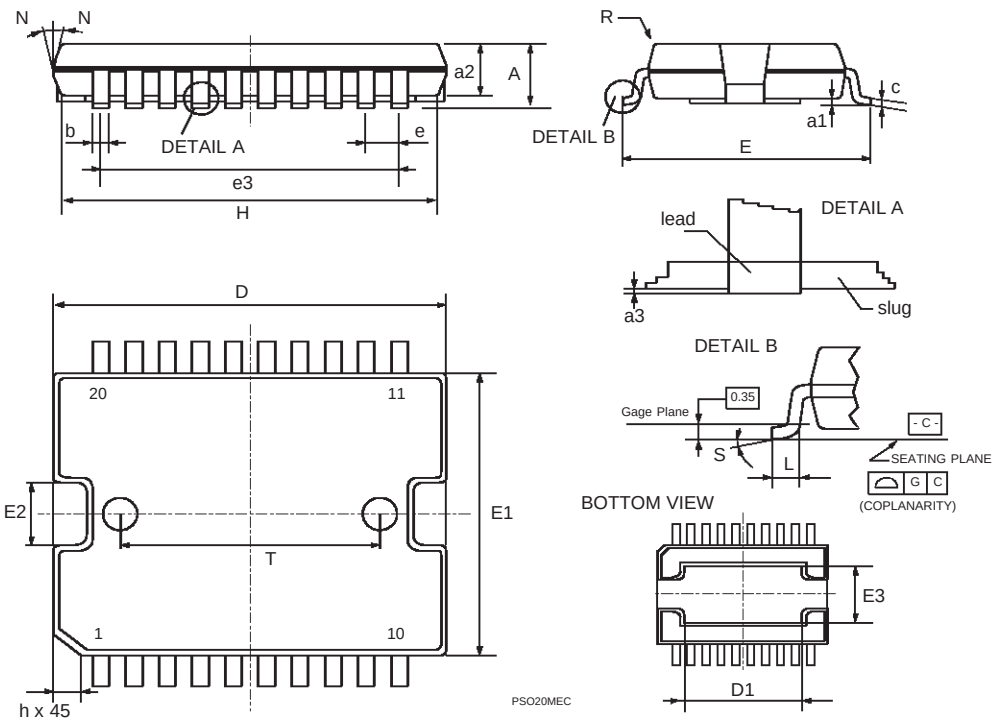
(1) "D and F" do not include mold flash or protrusions.
- Mold flash or protrusions shall not exceed 0.15 mm (0.006").
- Critical dimensions: "E", "G" and "a3"

OUTLINE AND MECHANICAL DATA



JEDEC MO-166

PowerSO20



CONECTOR D50