Trabajo Fin de Grado del Grado en Ingeniería Electrónica y Automática

Samuel Estévez Rodríguez

SISTEMA DE RIEGO INTELIGENTE DE BAJO COSTE

Trabajo dirigido por

Jonay Tomás Toledo Carrillo

y como coodirectora

Sara González Pérez

curso 2017-2018

Agradecimientos

Aunque llevo cuatro años en la carrera y aun hoy no está claro si la acabaré, quiero agradecerles a mis padres el increíble esfuerzo y sacrificio que han hecho para que un zoquete como yo se meta en vereda y pudiera estar aquí. Esa paciencia que han tenido en aguantar mis formas y métodos pocos ortodoxos de estudio. Y tiene mérito que siendo de familia humilde, no pararais de trabajar nunca para conseguir que vuestros hijos tuvieran la educación que no tuvisteis, para darnos la oportunidad de decidir que ser y no estar obligados por las circunstancias. Muchas gracias a los dos por esto y por lo que me queda por venir.

También quiero agradecer a Corina, una persona que ha sido compañera, amiga y actualmente mi pareja, por ayudarme cuando necesitaba, por todo el apoyo que me ha dado, todo el tiempo que ha invertido en motivarme cuando veía las cosas mal y sobre todo por estar ahí cuando mas lo necesitaba.

Mencionar cómo no, a mis compañeros de equipo, de trabajo y de juegos, Alby y Kevin, dos personas que en ningún momento dudaron en ofrecerme su ayuda y su consejo a lo largo de la carrera, y de éste mismo trabajo. Muchas gracias chicos por hacer que este camino que estaba en pendiente lo tomáramos juntos como una bajada.

Por otra parte, agradecer a mi tutor de proyecto Jonay Toledo, las enseñanzas que me ha transmitido, siempre tenía un punto de vista que yo no había pensado. Además, supo ver cuando estaba bloqueado y estresado para mostrarme una solución más sencilla y como dividir los problemas en pequeño.

Resumen

Con el presente proyecto se ha implementado un sistema de riego inteligente de bajo consumo, explicando paso a paso su desarrollo y los problemas encontrados para su realización.

En primer lugar se se ha llevado a cabo la programación de Arduino para una comunicación alámbrica de tipo serial sin sincronización en el envío de bytes de información. En esta sección se explica el programa realizado y los problemas que se han solventando, explicando en cada caso las alternativas y opciones que se presentaron en el momento de resolverlas.

A continuación se expone la manera en la que se transformo el código de la sección alámbrica a inalámbrica por medio de los módulos de radiofrecuencia, explicando también las distintas alternativas y soluciones que se tomaron para la elaboración del programa definitivo.

Por ultimo se explican los distintos elementos usados para implementar el sistema (Arduino, Módulos de radiofrecuencia, sensores ...) donde se describe su uso, sus características y limitaciones para nuestro objetivo. Así como se exponen las ventajas que tiene este modo de comunicación frente a modos de comunicación como WIFI o Bluetooth

Abstract

In this project an intelligent irrigation system of low consumption is explained. It is done explaining step by step the development and the problems found for its realization.

Firstly, this project follows the steps to program Arduino for a wired communication of serial type without synchronization in the sending of information bytes. Furthermore, this section explains the program carried out and the different problems which have been solved. It is done explaining in each case the several alternatives and options that were presented at the time, trying to solve them.

In the second part of the project will explain the way in which the code of the wired to wireless section was transformed by the radio frequency modules. It is done explaining the different alternatives and solutions that were taken into account to present the final program.

At last, the different elements which are used to implement the system (Arduino, Radio Frequency Modules, sensors ...) will be explained in a way that describes their use, their characteristics and their limitations for our purpose. As well as, the advantages of this communication mode will be analyzed in comparison to another ways of communication such as WIFI or Bluetooth.

Trabajo Fin de Grado del Grado en Ingeniería Electrónica y Automática

curso 2017-2018

SISTEMA DE RIEGO INTELIGENTE DE BAJO COSTE

Memoria

Índice

1. INTRODUCCIÓN GENERAL. OBJETIVOS

1.1. Introducción

Todo ser humano necesita alimentos para su subsistencia y hace miles de años cuando las sociedades vieron que la ganadería y la agricultura era una forma eficaz de abastecimiento cambiaron su forma de vida a asentamientos que ha sido la mayor revolución de la humanidad.

Estos asentamientos fueron cada vez más grandes y para abastecer a todos sus individuos hubo que mejorar las técnicas de cultivo que se conocían como único objetivo el producir más en menos tiempo y en menos espacio.

Empezaron con técnicas tan básicas como usar las crecidas de los ríos, crear canalizaciones para llevar el agua a distintas zonas y solventando desniveles, etc. Con la revolución industrial problemas como los anteriores se resolvieron con la tecnología de aquella época, con motores, haciendo uso de bombas de regadío y fabricando aquellas máquinas y herramientas que les fueran útiles y les proporcionaran menos esfuerzo en el trabajo de la agricultura.

Hoy, en la era moderna los problemas surgen con: cambios climáticos, el aumento exponencial de la población y la escasez de agua, que hace imprescindible hacer un control eficiente de su consumo. La ventaja, es que actualmente podemos hacer uso de diversas tecnologías para solventar estos problemas.

Este proyecto propone una solución a algunos de estos problemas mediante el uso de las tecnologías disponibles actualmente.Con estas buscamos un consumo eficiente del agua, optimizando así no sólo la cantidad de agua sino también el tiempo empleado en el riego

1.2. Objetivos

El objetivo de este proyecto consiste en la automatización del riego de una plantación de cultivo, que trabaje de forma autónoma e inteligente, capaz de adaptarse a las diferentes condiciones climatológicas y cuyo consumo sea mínimo o de bajo consumo.

La solución para este sistema ha sido utilizar diferentes placas Arduino, ya que, éstas nos permiten programar e implementar sistemas tan complejos como estos sin exceder en energía ni costo permitiéndonos añadir mas funcionalidades a un sistema de riego convencional.

Una de las placas será el maestro de nuestro sistema, el cual sera el encargado de pedir al resto de placas esclavo, la información de los diferentes sensores y enviar,

después de analizar los diferentes datos, la petición de riego al esclavo pertinente.

Los esclavos, que estarán repartidos en la plantación a controlar, se compondrán de una batería que tendremos que optimizar, de los diferentes sensores para medir las condiciones ambientales las cuales nos darán la información que tendremos que transmitir al maestro y la válvula o válvulas que el maestro de forma remota activará de forma que pueda controlarse para optimizar el riego.

Como nos interesa optimizar la batería de cada Arduino, tendremos que tenerlos en modo ahorro lo máximo posible y evitar hacer uso excesivo de las baterías teniendo componentes eternamente encendidos. Ya que mantener los sensores y actuadores encendidos un largo tiempo, supone gastar energía, los programaremos de forma que solo se active pocos milisegundos antes de pedirles información reduciendo su tiempo de consumo de forma notable.

En la siguiente figura vemos un esquema general del sistema.

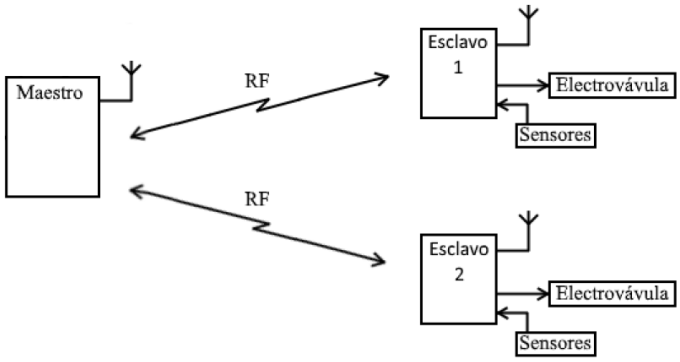


Figura 1: Esquema general

1.3. Estructura de la memoria

Esta memoria la vamos a dividir en varias partes bien identificadas las cuales vamos a describir ahora.

En primer lugar se describen los **Elementos electrónicos** que se han utilizado en la práctica y en el análisis de los sistemas compuestos por: sensores, actuadores, microcontroladores y, en segundo lugar, el uso de los diferentes circuitos integrados usados.

A continuación se ha dispuesto el **Entorno de desarrollo** donde se explicará el entorno de software dentro del cual estamos trabajando, que en nuestro caso es

Arduino. Ademas se explicara la evolución del código tanto del maestro, como del esclavo, en los diferentes modos de comunicación.

En tercer lugar se detallará el **Presupuesto del material**,describiendo el coste del trabajo realizado, de los elementos usados y de la mano de obra dedicada para este proyecto.

Y finalmente, en **Conclusión** se expondrá las conclusiones, experiencia y observaciones del autor respecto al trabajo. También se detallará el resultado obtenido,funcionalidad y especificaciones de éste.

1.4. Conceptos generales

Comunicación Serial

La comunicación Serial que usaremos es un protocolo muy usado (no se confunda con el Bus Serial de Comunicación, ni con el USB) para comunicación entre diferentes dispositivos, que se incorpora de manera estándar en casi cualquier ordenador hoy en día. La mayoría de ordenadores tienen incorporado dos puertos seriales RS-232. Ademas, este tipo de comunicación serial puede ser usada para las adquisición de datos si se usa en conjunto con un dispositivo de muestreo.

La idea de la comunicación serial es muy sencilla. Un puerto serial envía y recibe bytes de información bit a bit. Aunque la comunicación en paralelo es mas rápida que ésta debido a que nos permite enviar un byte completo, este tipo de comunicación es mas sencillo y os permite transmitir a mayores distancias. Para ser mas concretos, la especificación *IEEE 488* para la comunicación en paralelo define que lo largo del cable para el sistema no puede ser mayor a 20 metros con una distancia no mayor de 2 metros entre dos dispositivos, mientras que usando la comunicación serial el largo puede llegar hasta los 1200 metros.

Normalmente, este tipo de comunicación serial se utiliza para la trasmisión de datos en formato ASCII. Lo común para llevar a cabo esta comunicación es usar 3 lineas de transmisión: (1)Transmitir, (2)Recibir y (3)Tierra (o punto común). Como esta comunicación es asíncrona, se puede enviar y recibir datos al mismo tiempo. También se puede incorporar otras lineas que nos permitan hacer *handshaking* o intercambio de pulsos de sincronización, que e nuestro caso no usaremos. Para pueda haber comunicación entre dos puertos, deben de tener las mismas propiedades, la velocidad de transmisión, los bits de datos, los bits de parada, y la paridad.

- **Velocidad de transmisión (baud rate):** Se refiere al numero de bits por segundo que se transmiten y estos se mide en baudios. Por ejemplo 400 baudios son 400 bits por segundo. Si se menciona los ciclos de reloj, se esta hablando de la velocidad de transmisión, si decimos 4800 ciclos de reloj, os referimos a que el puerto serial se esta muestreando a razón de 4800 Hz. En arduino

las velocidades mas comunes son de 1200, 2400, 4800, 9600, 14400, siendo en nuestro caso la de 9600 baudios.

- **Bits de datos:** Nos referimos con esto a la cantidad de bits en la trasmisión. Las cantidades mas comunes de bits por paquete que enviamos en la transmisión son de 5, 7 y 8, no tiene que ser necesariamente 8 como se afirma en algunos foros. Ademas el numero de bits que se envía también depende del tipo de información que se este trasmitiendo, por ejemplo el formato ASCII del que hablábamos antes tiene un rango de 0 a 127, es decir, utiliza 7 bits; para ASCII extendido es de 0 a 255, lo que utiliza 8 bits. Cuando hablamos de paquete, nos referimos a una transferencia de un byte, incluyendo los bit de inicio, parada y de paridad.

- **Bits de parada:** Estos bits son usados para indicar el fin de la comunicación de un solo paquete. Los valores mas comunes son 1, 1.5 o 2 bits. Como existen diferentes maneras de trasmitir información a través de las lineas de comunicación y de que cada sistema posee su propio reloj, existe la posibilidad de que ambos no estén sincronizados y estos bits de parada no sólo indiquen el fin de la transmisión sino que también dan un margen de tolerancia para esa diferencia de los relojes. Por lo que cuanto mas bits de parada se usen, mas lenta sera la transmisión pero mayor su tolerancia.

- **Paridad:** La paridad nos permite verificar de forma muy sencilla si existe errores en la transmisión serial. La opción de no usar ningún tipo de paridad también se puede usar e este tipo de comunicación. Para los tipos de paridad par e impar, en el puerto serial se fijara el bit de paridad (el último bit después de los bits de datos) a un valor que nos indique si en la comunicación ha habido una transmisión con un número par o impar de bits en estado alto lógico. La manera en la que funciona esto es, si la paridad es de tipo par, el bit de paridad debe de mantener el numero de bits en estado alto par, luego para un 101 el bit seria 0, y para 010 seria 1. La paridad impar funciona igual, manteniendo el numero de bits en estado lógico, impar. Hacer esto, nos permite conocer si ha habido ruido en el mensaje, por que conociendo el tipo de paridad y conociendo el valor del bit, nos hacemos una idea de la secuencia de bits que deberíamos recibir.

- **Protocolo RS-232:** Es una norma o estándar mundial que regula los parámetros de uno de los modos de comunicación serial. Gracias a este protocolo se estandarizan las velocidades de transferencia, los niveles de voltajes utilizados, el tipo de cable permitido, la forma de control que utiliza la transferencia, las distancias entre equipos, los conectores, etc. Junto a las líneas de transmisión (Tx) y recepción (Rx), las comunicaciones seriales también poseen otras líneas de control de flujo (Hands-hake), las cuales son opcionales dependiendo del dispositivo a conectar.

Comunicación por Radio Frecuencia

Para definir la radiofrecuencia debemos entender el concepto de Telecomunicación. La Telecomunicación es la transmisión entre un receptor y un emisor a determinada distancia a través de señales.

Este estudio de las telecomunicaciones es muy amplio y varía según la magnitud de la frecuencia que nos encontremos. Debemos tener en cuenta, que existe una relación inversamente proporcional entre frecuencia y longitud de onda.

A frecuencias muy bajas del orden de 3 - 30 Hz, nos encontramos con la comunicación neuronal y de submarinos. Más adelante, de 30 - 300 KHz, está la frecuencia AM que permite el control del tráfico aéreo. Le sigue, la FM de 300 - 300 MHz, utilizada en telefonía móvil y radioaficionados. La siguiente es la ultra altra frecuencia (UHF), de 300 a 3000 MHz, que incluye desde los microondas, la televisión, bluetooth y redes inalámbricas. De hecho, trabajaremos en esta banda con un valor de 2,4 GHz. Hay tres tipos de bandas superiores con aplicaciones destacables como la teledetección (satélites, resonancia magnética, radares) e incluso espectroscopia.

2. ELEMENTOS ELECTRÓNICOS

2.1. Introducción

En esta sección se procederá a nombrar y a describir cada uno de los elementos usados a lo largo de la ejecución del proyecto, tanto elementos para el montaje, como las distintas herramientas usadas para el análisis de los diferentes sistemas realizados.

2.2. Multímetro

Uno de las herramientas que mas se usan en la electronica es posiblemente el multímetro digital, el cual nos permite hacer análisis de nuestros sistemas obteniendo medidas de voltios, intensidad, ohmios y permitiéndonos también localizar fallos en los componentes y en la conductividad. En nuestro caso estas dos ultimas funciones fueron las que mas usamos, ya que debido a la gran cantidad de conexiones que se han realizado, se ha tenido que comprobar la conductividad de cada conexión como también el buen funcionamiento de los elementos.

Figura 2: Multímetro Digital

2.3. Arduino

Las placas de arduino tanto las de tipo UNO, comos las de tipo NANO son un tipo de placas que contienen integradas un microcontrolador ATmega328P, creado por Atmel, que tiene una memoria flash (memoria donde se almacenara de programa) de 32KB y una 1KB de memoria EEPROM, que es un tipo de memoria ROM. Ambas pueden ser programadas, con su propio lenguaje de programación y mediante software libre ,es decir, existen esquemas y programas de acceso público disponibles en Internet y cualquiera puede fabricarlos, además posee puertos de entrada y salida de tipo digital en los que podemos conectar sensores y actuadores binarios, como también puertos de entrada y salida analógicos.

Para este proyecto se ha decidido trabajar con este tipo de placas por su versatilidad y su bajo precio, además de poderse programar de forma muy sencilla, con un lenguaje muy similar al C y que gracias al software libre se puede programar desde cualquier ordenador con gran variedad de ejemplos generados por la comunidad.

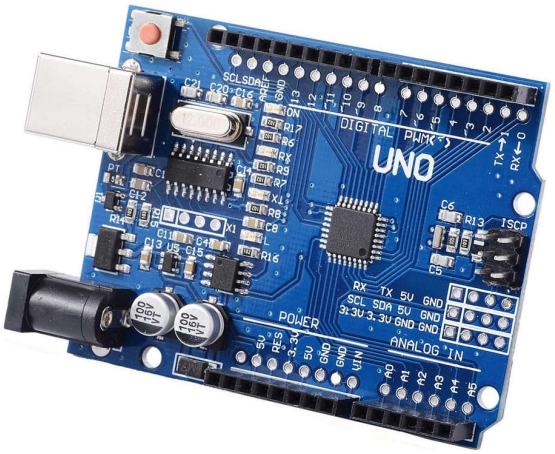


Figura 3: Arduino Uno

Las características principales de este tipo de placas son:

- Alimentación: Estas placas necesitan de una alimentación de 5V de continua. Puede alimentarse directamente a través del propio cable USB o mediante una fuente de alimentación externa, como puede ser un pequeño transformador o, por ejemplo una pila de 9V. Y también la alimentación puede conectarse mediante un conector de 2,1mm con el positivo en el centro o directamente a los pines Vin y GND marcados sobre la placa.

- Entradas y salidas digitales (14 en total): estos pines trabajan de forma binaria entre 0 a 5 Voltios y es donde se conectarán los diferentes sensores, actuadores y módulos de comunicación.

- Entradas analógicas: Estas entradas disponen de canales de 10 bits de resolución, trabajando entre valores de 0 a 5 voltios, que usando solo valores extremos los podemos tener como digitales.
- Los pines RX y TX: Se usan para las transmisiones serie de señales TTL.
- : Los pines digitales 10, 11, 12 y 13 pueden utilizarse para llevar a cabo comunicaciones SPI, que permiten trasladar información full dúplex en un entorno Maestro/Esclavo.

Un resumen de las características del arduino UNO utilizado seria:

Microcontrolador	Atmega328
Voltaje de operación	5V
Voltaje de entrada (Recomendado)	7 – 12V
Voltaje de entrada (Límite)	6 – 20V
Pines para entrada- salida digital.	14 (6 pueden usarse como salida de PWM)
Pines de entrada analógica.	6
Corriente continua por pin IO	40 mA
Corriente continua en el pin 3.3V	50 mA
Memoria Flash	32 KB (0,5 KB ocupados por el bootloader)
SRAM	2 KB
EEPROM	1 KB
Frecuencia de reloj	16 MHz

Figura 4: Características generales

2.4. Sensores

Sensor de Humedad y Temperatura

El sensor DHT22(AM2302) es un sensor que nos permite realizar la medición de temperatura y humedad de forma simultánea. Estos sensores disponen de un procesador interno que realiza la medición proporcionando una señal digital como salida.

Este sensor utiliza un sensor capacitivo de humedad y un termistor para medir valores de temperatura y humedad en el aire circundante y sólo un pin para la lectura de los datos. La desventaja de estos sensores es la velocidad de las lecturas y el tiempo que hay que esperar para tomar nuevas lecturas (nueva lectura después de 2 segundos, utilizar un periodos menores puede ocasionar que los datos no sean precisos), pero esto no es tan importante puesto que la Temperatura y la Humedad son variables que no cambian muy rápido en el tiempo.

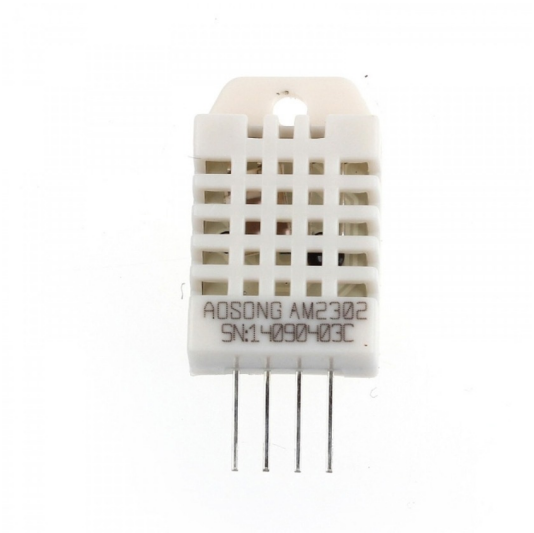


Figura 5: Sensor de humedad y temperatura

EL DHT22 (sin llegar a ser en absoluto un sensor de alta precisión) tiene unas características aceptables para que sea posible emplearlo en nuestro proyecto:

- Medición de temperatura entre -40 a 125ºC, con una precisión de 0.5ºC.
- Medición de humedad entre 0 a 100 por ciento, con precisión del 2 a 5 por ciento.
- Frecuencia de muestreo de 2 muestras por segundo (2 Hz)

Sensor de Humedad del suelo

El higrómetro de suelo FC-28, es un sensor que mide la humedad del suelo. Este tipo de sensores son ampliamente usados en sistemas automáticos de riego como es en este proyecto.

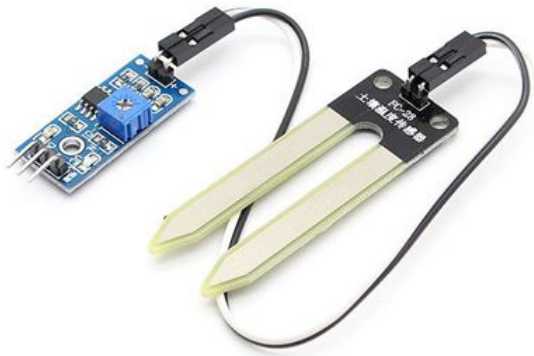


Figura 6: Sensor de humedad del suelo

Éste, es un sensor sencillo que mide la humedad del suelo por conductividad. No tiene la precisión suficiente como para realizar una medición absoluta del terreno, pero para nuestro caso no nos es del todo necesario.

El FC-28 se distribuye con una placa de medición estándar capaz de obtener las medición de la humedad de dos formas; una como valor analógico, es decir, nos da una aproximación del valor que existe que va desde 0 sumergido en agua, a 1023 en el aire y de forma digital,de manera que si la humedad del terreno no alcanza un cierto valor de consigna, nos activa una señal LOW como que el suelo no está húmedo, y HIGH cuando la humedad supera el valor de consigna.

Sensor de Luminosidad

El sensor de luminosidad usado es el TSL2561. Éste es un sensor que nos permite medir en un rango de 0,1 a 40.000 lux. No sólo tienen un amplio rango de medida sino que puede medirnos tanto la luz ambiente en el espectro visible como también la luz infrarroja.

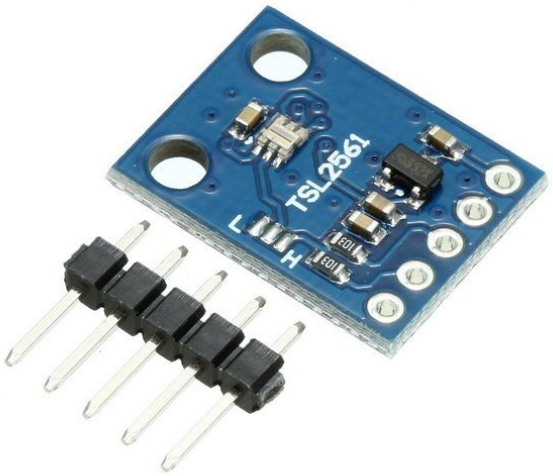


Figura 7: Sensor de luminosidad ambiental

El sensor está diseñado para funcionar tanto a 3.3 como a 5V, así que lo podemos utilizar en nuestro Arduino sin componentes adicionales. Además, funciona mediante el bus I2C, por lo que es muy sencillo de utilizar junto a otros sensores en un mismo bus.

Sus características principales son:

- Medición precisa en diversos ambientes lumínicos
- Temperatura de funcionamiento: -30 a 80 *C
- Rango dinámico (Lux): 0.1 a 40,000 Lux
- Interfaz: I2C (Dirección: 0x39, 0x29 o 0x49, seleccionable)

Sensor de Lluvia

Este sensor es el YL38 el cual es capaz de detectar la presencia de lluvia por la variación de conductividad cuando la placa auxiliar del sensor entra en contacto con el agua.

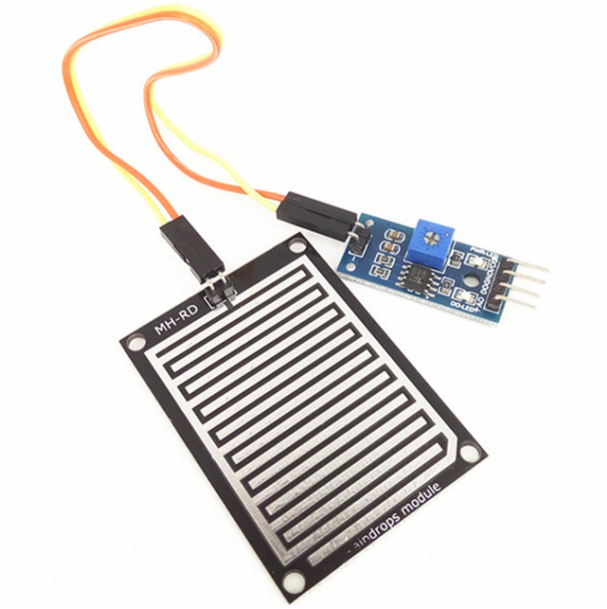


Figura 8: Sensor de lluvia

Estos sensores se componen de una placa con unas pistas entrelazadas a una cierta distancia. Al caerle, desde pequeñas gotas a incluso sumergirse en agua, estas pistas se ponen en contacto entre ellas siendo esto, detectado por el sensor.

Esta información se envía a una placa de medición con el LM393 integrado, que nos permite tener una lectura tanto como valor analógico y como valor digital. Como valor analógico (varían desde 0 para una placa totalmente empapada, a 1023 para una placa totalmente seca). Y como valor digital(obtendremos una señal LOW en ausencia de lluvia, y HIGH con presencia de lluvia en función de un valor de consigna). Como vemos es muy similar al funcionamiento del sensor de humedad del suelo.

2.5. Modulos de Radiofrecuencia

Uno de los tipos de comunicación que usamos en el proyecto es la comunicación inalambrica y el método usado para esta, es el de radiofrecuencia mediante los módulos de radio NRF2401. Estos contienen un chip de comunicación inalámbrica creado por Nordic Semiconductor y puede ser conectado a una placa Arduino a través del bus SPI (Serial Peripheral Interface).

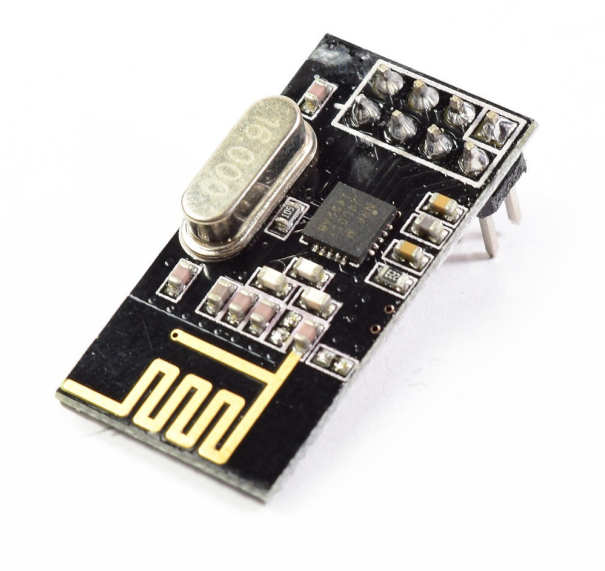


Figura 9: Modulos de radio frecuencia.

Estos módulos integran en un dispositivo toda la electrónica para establecer comunicación radio frecuencia entre dos dispositivos a diferentes velocidades (Hasta 2 Mb/seg).

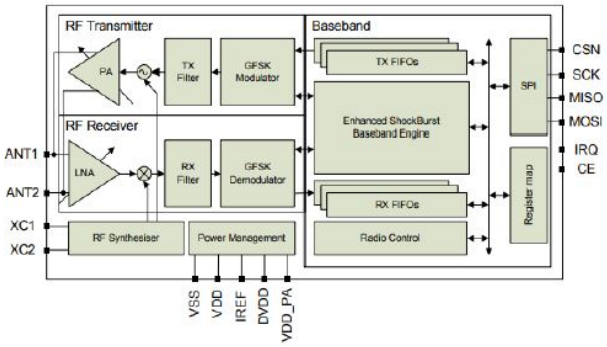


Figura 10: Diagrama de bloques de los módulos NRF2401.

El motivo por el que usamos este tipo de módulos frente a los de Blue Tooth o los de WIFI es que estos, tienen unas limitaciones bastante significativas respecto a las distancias a la que podemos usarlo, ambos no pasan mas allá de 20 metros. Los módulos RF con antena integrada tienen un alcance de entre 20-30 metros como máximo y la versión de alta potencia, que contiene un amplificador y antena externa, alcanza como máximo de 700-1000 metros.

Sin embargo, el alcance real se ve limitado por muchos factores, incluso en condiciones visibilidad directa sin obstáculos. Con el módulo de antena integrada y alimentación desde Arduino y velocidad de transmisión de 2 Mbps el alcance será apenas de 2-3 metros.

Las principales características de este tipo de módulos son:

- Operan en la banda de 2.4Ghz, que es de libre uso a nivel mundial.
- Velocidad de transmisión configurable de 250kb, 1 Mb o 2Mb por segundo.
- Consumo muy bajo en Stand By (Cuando no se esta comunicando).
- El alcance de los modulos depende de si hay visión directa entre modulos, pero nos ofrece un mínimo de entre unos 20 m hasta un máximo en **80m en óptimas condiciones, en el modelo básico con la antena integrada**.
- También podemos encontrar modelos con antenas de mayor rendimiento por un coste no muy superior que aumentan de forma importante el alcance hasta casi un km.

2.6. Integrados

Una de las partes del proyecto, como más adelante veremos, ha sido la de realizar el sistema de forma que la comunicación entre esclavo y maestro fuera totalmente inalámbrica, como también, que la alimentación se realizara mediante el uso de baterías para así evitar el uso de cables.

La razón de que se evitara el uso de los cables tanto en el la comunicación, como en la alimentación, era suprimir en lo máximo posible el coste de éste, tanto del precio del cable, como de la instalación de canalización que hubiera tenido que hacerse. Ademas se necesitaba que este sistema consumiera lo mínimo posible para prolongar la vida de las baterías, por lo que fue necesario hacer uso de los siguientes elementos.

Regulador de Tensión

El regulador de tensión es quizás el elemento mas común en sistemas donde existen varios elementos conectados, ya que, lo normal es que no todos éstos trabajen en el mismo margen de tensiones. Es por ello, por lo que hemos hecho uso de este elemento en nuestro trabajo.

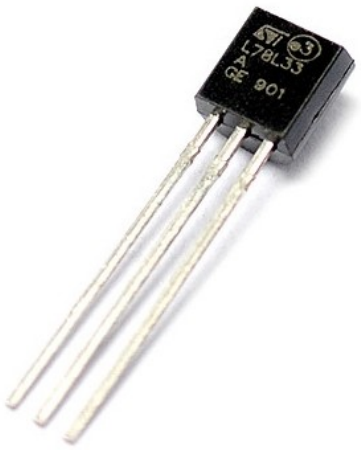


Figura 11: Regulador de Tensión

Para ser más exactos, el regulador se usó para establecer la tensión de alimentación de los módulos de radiofrecuencia necesarios para establecer la comunicación inalámbrica en el conjunto de arduinos. Éstos a diferencia del resto de sensores y elementos, necesita una tension de 3,3V que obtendremos a partir de los 5V que nos proporciona arduino haciendo uso del regulador.

Como necesitamos reducir el consumo de los elementos del sistema, estos 5V de los que hablábamos, no se obtendrán de la salida de 5V que nos da Arduino, sino de una salida digital que programaremos, consiguiendo así que los módulos sólo se ejecuten cuando es totalmente necesario y evitado el consumo en tiempos muertos o de suspensión.

Mosfet

El mosfet, es también uno de los elementos mas usado en la electrónica, sobre todo cuando este nos puede funcionar como interruptor controlado por tension. Nuestro mosfet es el TN0702, este a diferencia de lo normal, tiene 3 patillas ya que la pata sustrato esta unida al terminal surtidor.

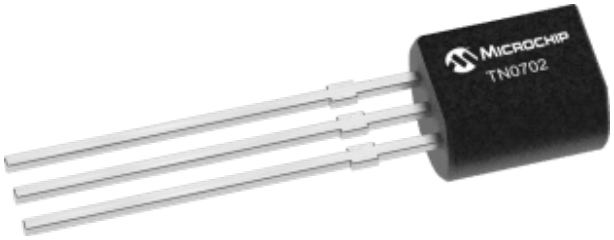


Figura 12: Mosfet TN0702

Para nuestro caso, la función del mosfet no es otra que controlar la alimentación de los módulos de comunicación inalambrica de los que ya se hablo anteriormente y que nos permitirá la comunicación entre maestro y esclavo, haciendo que este trabaje como interruptor controlado por tension.

Como uno de los objetivos del proyecto es reducir en lo máximo posible, el consumo del conjunto de dispositivos que conforman cada uno de los esclavos y el maestro, y también necesitamos una comunicación constante entre maestro y esclavo, este mosfet solo se va a implementar en los esclavos los cuales van a operar durante menos tiempo a lo largo del programa.

Cada vez que cada uno de los esclavos entre en modo sleep, entra en un estado de reposos en el cual no ejecuta programa, su única función es guardar el programa insertado y almacenar las variables que hayamos guardado en la memoria no volátil. Es por ello que no va dar señal de activación al mosfet y en consecuencia en este estado se va a alimentar a los módulos de comunicación.

Puente en H

En muchas de las aplicaciones electrónicas orientadas a la Robotica nos surge la necesidad de generar y controlar la activación de algún sistema anexo al principal. Según la naturaleza usamos Reles, microcontroladores, pulsadores, etc.

En nuestro caso el elemento a controlar es una electroválvula. Esta determina su estado en funcion de la tension entre sus terminales, si conectamos el terminal 1 de la válvula al **positivo** de la pila y el terminal 2 al **negativo** de la pila, obtendremos la apertura de la válvula y si lo conectamos de forma opuesta, obtendremos el cierre de ésta.

Para lograr esto, lo podemos hacer con reles y pulsadores e interruptores pero seria tedioso teniendo a nuestra disposicion un circuito integrado conocido como puente en H, el cual nos permite el cambio de polaridad que necesitamos para el control de nuestra valvula. Este integrado es el L293D, el cual es una disposición circuital de transistores y diodos que nos permite controlar la polaridad de dos terminales de salida en función de unas entradas lógicas.

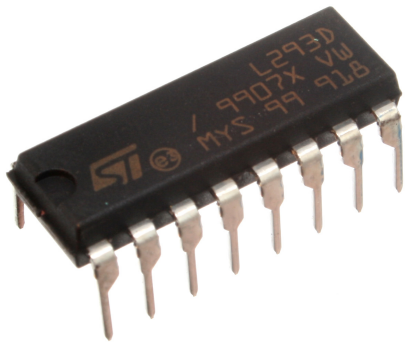


Figura 13: Puente en H

Este integrado nos facilita el control de dos válvulas de forma simultanea, los pines de la primera válvula conectados a los pines 3 y 6 y los terminales de la segunda válvula a los terminales 11 y 14.

Es importante tener en cuenta que este integrado se alimenta con dos niveles de tensión diferente,según vemos en el **Datasheet**, uno corresponde a la alimentación propia de integrado, que no debe ser superior a 7V(VSS) y otra es la tensión con la que alimentaremos las válvulas, pudiendo en este ultimo hacerlo con hasta 36V(VC), para nuestro caso quedara en 5V.

3. ENTORNO DE DESARROLLO

3.1. Introducción

En esta sección vamos a explicar el proceso que se ha llevado a cabo para la realización de cada uno de los programas realizados desde el inicio del proyecto, hasta la última versión de éste, analizando las distintas mejoras y explicando los distintos problemas que hemos tenido en las diferentes versiones. Como estamos trabajando con las placas Arduino, habláremos también del entorno de desarrollo de Arduino.

El entorno de desarrollo de Arduino es un software de tipo libre y para multiplataforma , es decir que sirve para diferentes sistemas operativos, además dicho entorno nos permite compilar y cargar programas en la memoria flash del microcontrolador que contiene las placas Arduino. Esto se consigue mediante la conexión USB y una vez programada la placa ya no necesitaría ningún tipo de comunicación con el Pc a menos que necesitáramos ver el proceso u obtener valores directamente de la placa.

El lenguaje usado para la programación de los distintos programas dentro del entrono Arduino es básicamente C de manera mas sencilla y simplificada. Esto quiere decir, que usaremos elementos típicos de este lenguaje como bucles, funciones, condicionales, variables y demás.

3.2. Programación para una comunicación tipo RS-232

En el comienzo de este proyecto, se planteo el uso de Arduinos para implementar un riego inteligente de bajo consumo, capaz de tomar las medidas necesarias, de forma autónoma, para el riego de una plantación agrícola. Para esto empezamos por un sistema, en el cual todos los Arduinos estuvieran comunicados por cable, ya que de esta forma las probabilidades de perder información en el transcurso de la comunicación serian muy bajas.

Una vez tomada ésta decisión, estudiamos los distintos tipos de interfaces de comunicación (y protocolos asociados) entre Arduinos que se habían usado en otros proyectos del mismo estilo, como por ejemplo el SPI(protocolo síncrono que envía y recibe un bit por cada señal de reloj, un solo cable), el I2C(protocolo síncrono que usa señal de reloj por un cable y los datos por otro) y la UART(es uno de los protocolos mas usados, que usa una linea para enviar y otra para recibir datos, normalmente de 8 bits usando a parte un bit de inicio y un bit de parada).

Vistas las ventajas y desventajas de las distintas interfaces y de los distintos protocolos, decidimos usar un estandar de la UART, el RS-232. Este nos permite la comunicación mediante dos cables RX y TX, los cuales nos envían los datos en

separados por bytes, si hacer en ningún momento uso de un cable con una señal de reloj, ni usar este para la sincronización de los distintos bits enviados en cada byte.

Como uno de nuestro objetivos es reducir costes, con este protocolo y esta interfaz reducimos el coste de un cable, que aunque a priori no es nada, en una plantación son varios metros de los que se tiene que hacer uso y es este coste, uno de los motivos por lo que se puede encarecer bastante el proyecto. Como en este caso, solo vamos a estudiar el rendimiento que nos da este tipo comunicación en un sistema pequeño con pequeñas distancias, el uso de los cables TX y RX no nos son un problema.

3.2.1. Programación del Maestro

La programación del Arduino Maestro es sin duda la mas compleja de los códigos generados para este proyecto, debido a que este debe de contemplar varias circunstancias de comunicación, debe de analizar los diferentes datos recibidos por cada uno de los Esclavos y de cada uno distintos sensores que estos tienen implementados. No solo debe de analizarlos sino que debe de dar una respuesta distinta para cada una de las situaciones, de ahí, que lo estemos denominando inteligente.

Para conseguir este Maestro, es evidente que no se consiguió a la primera, hubo que realizar muchos programas, con circunstancias y situaciones diferentes y siempre paso a paso. Se empezó desde programas sencillos en los que solo pasábamos una letra hasta ya programas mas complejos en los que pasábamos la totalidad de los datos, interáctuando con distintos esclavos de forma secuencial.

A continuación se hablara de los distintos pasos que se han llevado a cabo, para alcanzar un maestro que mediante comunicación RS-232 controle a varios esclavos y riego que se necesite, ademas se hablara de los problema que han surgido y la soluciones que se han tomado. Para explicarlo, tomaremos 3 de los muchos programas realizados, explicando el avance desde a version anterior.

Maestro 1:
En el primer código que se realizó, lo único que se consiguió fue una comunicación sencilla en la que solamente se trasmitía un valor numérico de tipo *int*, que era el valor del dato recogido por el sensor gobernado por el esclavo y que era enviado al maestro, y un carácter de tipo *char*, que enviábamos desde el maestro para indicarle al esclavo de que sensor queríamos obtener información.

Unos de los problemas con los que nos encontramos para este código, fue que la comunicación serie de Arduino junto a la librería para esta comunicación es muy limitada. Como hablábamos antes, una de las características que tiene este tipo de comunicación, es el paso de la información por canal serie que lo hace en bits, pero ¿ Cuanta información es capaz de trasmitir?

Si estudiamos la librería y hacemos pruebas, vemos que lo único que podemos hacer es enviar mensajes de longitud un byte, eso nos reduce el numero de caracteres en el envió a una sola letra ó a un numero con un valor máximo de $2^8 = 256$ que es lo máximo representable en un byte. Recordemos que se toma en cuenta el 0 por lo cual 255 equivaldrá a 256.

Para este problema surgieron dos soluciones viables:

- Separar de forma binaria los distintos datos e información que se quisiera transmitir en el origen y en el destino volver a unir ambos paquetes. Con esto lo que vimos que conseguíamos era la posibilidad de transmitir valores numéricos de 255 a números de un valor máximo $2^{16} - 1 = 65{,}535$ que para valores de un sensor ya es mas aceptable, teniendo en cuenta que el valor mas alto de un sensor que íbamos a tener es el de luminosidad que si recordamos sus características era de 40.000 lux. El problema para esta forma nos surgió en el envío de variables tipo float, donde enviar la para enviar aumentaba las lineas de código.

- Enviar paquetes que almacenaran *Strings*, es decir secuencias de caracteres de tipo *int, char, float ...* La ventaja de esta forma respecto a la anterior es que nos permite enviar cualquier cosa, siempre que antes del envío se le cambie el tipo de variable a *String*. La desventaja surge cuando pensamos en envíos inalambricos, que al ser una secuencia de mas de un byte es muy fácil perder uno de los muchos bits que los componen, por lo que el mensaje podía ser erróneo.

Como para nuestro caso, estamos usando cables físicos para la comunicación y el problema de perdida de bits es casi que inexistente (debido a que las tierras de los elementos se encuentran conectadas en común y la diferencia de potencial va a existir), elegimos la segunda opción, ya que nos da mas juego en el envió de datos y podemos pasar la parte decimal de los datos.

Con esta forma, no solo podemos enviar números de cualquier tipo, sino que podemos llamar a los sensores y a los actuadores de una forma concreta, ya que con ver el *String* recibido, podemos ver a cual se refiere, el tipo y actuar en consecuencia, por lo que nos ayuda a etiquetar estos sensores y programar de forma mas sencilla.

Maestro 2:

En esta segunda version del maestro, ya teníamos solventado el problema de transmisión de datos y el programa básicamente consistía en recibir los diferentes datos del esclavo, tanto valores de los sensores implementados en este, como también, el estado de los actuadores (que por seguridad, no deben activarse estando ya activos).

Una vez recibía los distintos datos del esclavo, el maestro analizaba los datos comparándolos con unos valores prefijados que hacían de valores de consigna y si se cumplía que estos estaban dentro de los valores de consigna, se aprobaba el riego enviando un valor de riego estándar también prefijado en el código.

Unas de las cuestiones que nos surgió en el modo de transferencia de datos, fue el ¿ cúantos enviamos en el mensaje? Cuando elegimos en la primera version el *String* por su versatilidad, lo hicimos teniendo en mente la posibilidad de enviar en un *String* la totalidad de información que el maestro pudiera necesitar del esclavo. Es decir, podíamos enviar una serie de números y letras en un solo mensaje separando cada dato por un símbolo (corchetes, paréntesis, almohadilla, ...) y en el maestro separarlos.

Aunque en primera instancia se llego a hacer así consiguiendo incluso que funcionara, se opto por enviar y recibir paquetes de un solo dato. La razón de esta decisión fue dejar el código listo para trabajar en futuras versiones en las que solo se enviara un byte y como mucho dos, evitando así la perdida de datos y en consecuencia pedir un riego erróneo.

El problema en este caso fue el ¿ cómo diferenciamos los datos de cada sensor, si no podemos etiquetarlos como apriori teníamos pensado ? La solución que se nos ocurrió, fue usar mensajes como etiquetas, es decir que para poder pedir un determinado dato al esclavo, tengo que enviar una determinada letra o numero y para este, leer y procesarlo de manera correcta, o como numero o como letra.

Maestro 3:

La ultima version del maestro para este tipo de comunicación, consistía en un maestro capaz de analizar los distintos datos de los distintos Arduinos esclavos que estuvieran en el sistema, de enviar el tiempo que se necesitaba que estuviera el riego activo y la parte mas nueva, que era la asignación de una identificación a cada Arduino esclavo nuevo que se conectara.

La forma de trabajar de esta asignación era, que al conectar un nuevo Arduino al puerto serie, generado para la comunicación RS-232, se le asignara un numero identificador por el cual se debía de llamar al esclavo cada vez que necesitáramos pedirle información. Así, de esta forma, podíamos pedir la información de un dato especifico a un esclavo especifico, sabiendo en todo momento que y quien nos estaba enviando datos al maestro.

La forma en la que se implemento esto en el código fue crear una variable que se fuera incrementando a medida que se conectaran los esclavos. Como no teníamos forma de identificar cuales estaban ya conectados y cuales tenían un numero identificador, lo que hicimos fue que solo dábamos identificador a aquellos esclavos que nos enviaran una 'I' como mensaje de inicio. Es decir que los que no nos lo enviaran

ya tenían un identificador asignado y a los que si, le asignábamos la variable que comentábamos mas uno.

Uno de los problemas que nos surgía en este caso, fue el ¿ cómo saber que dato estoy recibiendo y de quién, si hay varios hablando? La solución a éste fue, por un lado adaptar el programa para que cada vez que pidiéramos un dato, teníamos que enviar el numero identificador y el mensaje etiqueta del que hablábamos, indicando el sensor que queríamos leer. Y por otro lado, para asegurar que el mensaje fuera el correcto y que la petición de lectura de sensor también lo fuera, lo que hicimos es que cada mensaje que se enviara tenia que tener una respuesta, si estaba dentro de las posibles ejecutaba el código, si no, salia un mensaje de error diciendo que se el mensaje no era correcto.

3.2.2. Programación del Esclavo

La programación del Arduino Esclavo es a diferencia del de el Maestro mucho mas simple, debido a que éste no tiene sino ejecutar la parte de código que el maestro necesite que ejecute. Es decir para cada mensaje que el esclavo reciba de parte del maestro, el esclavo tendrá asignado un código especifico para este, que en nuestro sera la lectura de cada sensor o la activación de una válvula especifica como ya comentamos en secciones anteriores.

Al igual que pasaba con el maestro, el esclavo ha pasado por una serie de versiones, cada cual mejor que la anterior, en las cuales íbamos solventando los problemas que nos iban surgiendo en las versiones anteriores.

Debido al planteamiento y planificación que se hizo para la programación del maestro y de los esclavo, estos debían programarse de forma paralela,es decir, los programas se hicieron de forma secuencial, de programas sencillos a mas complejos y para cada programa que se hiciera para el maestro, debía haber uno para el esclavo con los que poder hacer pruebas y corregir fallos.

Es por ello que a continuación procederemos a explicar las tres versiones adyacentes a las versiones de los maestros ya comentados anteriormente, con sus mejoras y explicando los problemas que han surgieron en la elaboración de dichos programas.

Esclavo 1:
La función que tenia nuestro primer programa que generado para nuestro esclavo, era la de leer una variable de tipo *char*, recibida por la conexión serie y si esta era la correcta pedir lectura a un sensor, que para nuestro caso fue el de humedad y temperatura debido a que la librería es muy simple y no nos exige demasiado código para pedir valores.

A continuación se enviaba al maestro el valor que nos devolvía el sensor y también por medio de la conexión serie. Como comentábamos en la primera version del

maestro uno de los problemas surgieron fue el como pasar este dato y como vimos, la forma de solventarlo fue enviar por medio de un *String* no solo entero sino decimal del dato recogido por el sensor. Ya que las funciones seriales de *.write* no nos permitían enviar números decimales, de tipo *float*.

Para que la comunicación entre maestro y esclavo fuera eficiente en esta version, se debían de asegurar dos cosas, la primera que no se nos mezclaran diferentes mensajes, es decir, que se enviara los valores exactos que queríamos que se enviaran, y en segundo lugar que no se diera la circunstancia de que tanto maestro como esclavo estuvieran ni enviando ni haciendo lectura ambos simultáneamente, ya que podría paralizarse el proceso o saturar el buffer.

Para ello lo que hicimos fue establecer y crear un funciones para la comunicación, las cuales obligaban a seguir un proceso secuencial, en el cual, una vez que enviábamos un mensaje fuera en la dirección que fuera, debía de esperar un mensaje devuelta antes de de volver a enviar otro. Aunque parece una cosa sencilla y lógica, cuando se ejecuta lo ciclos del Arduino si no se programa bien pueden surgir fallos o descoordinacion entre sistemas difíciles de detectar.

Esclavo 2:
La segunda version del programa generado para el esclavo, consistía en la implementacion de todos los sensores nombrados anteriormente en la sección de Elementos Electrónicos, en el código, de forma que hicieran una lectura en el caso en el que recibiera una letra. Como ya vimos, para que se leyera un sensor especifico se debía de recibir una letra especifica de forma que no se pidieran simultáneamente muchos datos y en consecuencia enviar muchos datos seguidos saturando el buffer del maestro.

El problema que surgió en esta version fue específicamente ese que comentamos, que el buffer del maestro se saturaba al enviar muchos datos seguidos o de diferentes tamaños de byte, por que como dijimos en la version anexa del maestro la forma en la que enviábamos los mensajes era mediante *Strings* y ello conlleva que por cada carácter del *String* que se envié, se este enviando un byte que era lo que queríamos evitar.

La manera en la que se veía reflejado este problema, era que a la hora de hacer la lectura de datos por el serial en el maestro, los números e información que transmitíamos se veía distorsionada, debido a que al realizar la lectura de de los datos, en el buffer del maestro solo se sobreescribian tantos bytes como bytes tuviera el mensaje de entrada. Luego se mostraría parte del mensaje anterior si éste tuviera mas bytes que el nuevo mensaje.

La manera en la que solucionamos este problema fue por un lado, con la creación de una función que se encargaba de limpiar el buffer del maestro antes de cada

lectura, así el buffer estaba vacío para la nuevas lecturas procedentes del canal serial. Esta función la creamos de dos tipos, una que leyera los 8 bytes que sabíamos que tenia el buffer y otra mas dinámica que borrara solo los bytes del mensaje anterior de forma que no hiciera mas trabajo de lo que debía. Y por otro lado realizar un conmutador *switch...case* para las lecturas de sensores, así asegurábamos que se leía y enviaba información de un solo sensor.

Una de las peculiaridades de esta version, es que al incorporarse todos los sensores hubo que hacer un estudio previo de cada uno de ellos, y se vio que todos eran diferentes, por lo que la forma de usarlos es igualmente diferente y en consecuencia la manera de programarlos también lo es. En este código se usaron 2 librerías, una especifica para el sensor de humedad y temperatura , que nos permite obtener datos de una forma muy sencilla llamado a una función de esta, y otra especifica para el sensor de luminosidad que necesita mas funciones para la obtención de los valores ya que se necesita especificar la ganancia y el tiempo de integración que nos proporciona precision en función del tiempo empleado en la medida.

Esclavo 3:

En la ultima version del esclavo para este tipo de comunicación, consistió en un esclavo que enviara información de todos los sensores siempre que el maestro lo solicitara y activar las electrovalvulas que tengan bajo su control, e igualmente siempre que el maestro se lo solicite. Ademas se incorpora la petición de entrada al sistema, que lo que hace es asignarle un identificador al esclavo mediante el cual se va a comunicar el maestro con este.

La forma en la cual trabaja esta petición es mediante un bucle, que envía de forma insistente una variable de tipo *char* al maestro y que en caso de que este envíe un identificador al esclavo si este recibe una variable de tipo *int* recogía este valor y lo guardaba en una variable interna del esclavo con la que compararíamos cada mensaje del maestro para identificar si el mensaje corresponde al esclavo correspondiente. Es decir, que el esclavo únicamente podría comunicarse con el maestro si este le envía su identificador, en caso contrario obvia todos los mensajes recibidos.

Otro de los problemas que surgieron fue encontrar la manera de ejecutar programas de forma paralela. Debido a que los esclavos no solo hace las lecturas sino que también activan las electrovalvulas que controlan el riego, debemos de hacer ambos procesos en paralelo ya que no debe de pararse el riego mientras se leen sensores o viceversa. Para solucionar esto, creamos una función que trabaja con la función *millis()* de Arduino en la que se compara el tiempo en el que recibe la orden de riego y la compara con el tiempo en el que se debe de parar el riego, así de esta forma este en el momento que este siempre se compara.

3.2.3. Programa Final Alámbrico

Como veníamos hablando en los apartados anteriores el programa final para la comunicación serial con el método tipo RS-232 consiste en dos programas: el programa del Maestro y el programa del Esclavo, los cuales vamos a describir a continuación.

Descripción del Maestro: (mirar sección 7.2)

Con lo primero que nos encontramos en el codigo del maestro es con la llamada a las distintas librerías usadas para el conjunto de funciones y procesos que ocurren en nuestro maestro, ademas de las inicializaciones de las variables y consignas que va a usar en nuestro programa.

```
// LIBRERRIAS DEL PROGRAMA -----------------------------
#include <SoftwareSerial.h>

// INICIALIZACIONES ------------------------------------
SoftwareSerial mySerial(10, 11);

// CONSTANTES DEL PROGRAMA -----------------------------
const int TIEMPO_RIEGO = 10;
const int HUMEDAD = 65;
const int TEMPERATURA_FRIO = 15;
const int TEMPERATURA_CALOR = 45;
const int HUMEDAD_SUELO = 600 ;
const int LUMINOSIDAD = 70;

//Identificador de arduinos conectados al maestro--------
int esclavosConectados = 0;
```

Ademas el Maestro consiste en un programa formado por un conjunto de funciones que hemos creado para simplificar, estructurar y organizar mejor nuestro programa, de modo que, cada una de éstas tenga una función específica. La primera función que encontramos es para generar un*Array* dinámico que, vaya aumentando su capacidad cada vez que la llamemos. Esta función la usamos para que cada vez que entre un nuevo esclavo al sistema, agregue un nuevo espacio en el array, en el que almacenaremos el identificador de éste. Ese identificador se asignara por otra función y sera ese numero el que se guarde en ese espacio de memoria.

```
//FUNCIONES PARA EL TRATAMIETO DE LA LISTA DINAMICA/////,

int* list;
size_t count;
size_t capacity;

// Crear una nueva lista
void CreateList(size_t _capacity)
{
  list = new int[_capacity];
  capacity = _capacity;
  count = 0;
}
```

Las siguientes funciones que nos encontramos dentro del Maestro, son para llevar a cabo la función anterior. La primera de ellas nos permite añadir un nuevo elemento dentro del *Array*.

```
// Añadir elemento al final de la lista//////////////////,
void AddItem(int item)
{
  ++count;

  if (count > capacity)
  {
    size_t newSize = capacity * 2;
    resize(newSize);
  }

  list[count - 1] = item;
  resize(count);
}
```

La siguiente nos permite eliminar el último elemento del array, y finalmente, respecto a la creación del array, una función que nos muestra la capacidad, el número de posiciones ocupadas y los elementos de cada posición.

```
// Eliminar ultimo elemento de la lista//////////////////,
void RemoveTail()
{
  --count;
  resize(count);
}
```

```
// Muestra la lista por Serial para debug///////////////
void printList()
{
  Serial.print("Capacity:");
  Serial.print(capacity);

  Serial.print("  Count:");
  Serial.print(count);

  Serial.print("  Items:");
  for (size_t index = 0; index < count; index++)
  {
    Serial.print(list[index]);
    Serial.print(' ');
  }
  Serial.println();
}
```

Después de éstas, nos encontramos dos nuevas funciones que trabajan también con el *Array* nombrado anteriormente. La primera de ellas devuelve un número a razón de, cada vez que la llamemos crea una nueva posición en el array y la ocupa con el número de la posición más uno, sacando este valor, que será el identificador de un esclavo nuevo que se conecte. Si en alguna de las posiciones del array existiera un cero, no se añadiría una nueva posición, sino que se ocuparía ésta del mismo modo. Y la segunda, hace todo lo contrario, dándole un valor numérico, busca de forma simétrica (esa posición menos uno) en el array y la pone a cero, que será una posición libre para un identificador nuevo de algún esclavo que se conecte. Ya que, en la función anterior se encarga de buscar éste y darle un nuevo identificador.

```
//Añade un elemento nuevo, si hay ceros los ocupa y si no crea uno nuevo////
int newId(){
  int ceros = 0;
  int idex[ceros];
  for (size_t index = 0; index < count; index++){
    if (list[index]== 0){
      idex[ceros]= index;
       Serial.println(idex[ceros]);
      ceros++;
    }
  }
  if(ceros == 0){
    AddItem(count+1);
    printList();
    return (count+1);
  }else{
```

```
    }else{
      int idnew = idex[0] + 1 ;
      list[idex[0]]= idnew;
      printList();
      return idnew;
    }
  }

  //Elimina el elemento de la lista que le indiquemos//////////////////////.
  void idCero(int id){
    list[id -1]= 0;
    printList();
  }
```

A continuación, lo que nos encontramos son dos funciones que nos limpian el buffer de entrada del serial, la primera de ellas limpia los 8 bytes del buffer y la segunda, limpia el número de bytes que insertemos en la función. Ésta la generamos debido que, en la transferencia de datos el buffer quedaba lleno con información de mensajes anteriores y nos provocaban una lectura errónea de datos, monstrándonos información que no era lógica.

```
//FUNCION DE LIMPIAR LOS 8 BITES DEL BUFFER:////////////////////////////////
void limpiarBuffer (){
  for(int i=0; i<10; i++){
    mySerial.read();
  }
}


//FUNCION DE LIMPIAR LOS N BITES DEL BUFFER:////////////////////////////////
void limpiarBuffer(int bytes){
  for(int i= 0; i<bytes; i++){
    mySerial.read();
  }
}
```

La siguiente función que encontramos, lo que hace es leer cada byte que esté en el buffer y añadirlo a un *String*, que a priori está vacío. Con esta función lo que conseguimos es mostrar todo el mensaje que nos llegue por el serial.

```
//FUNCION LEER LOS 8 BITES DEL BUFFER:////////////////////////////////////.
String Buffer(){
  String bufferleido = "";
  for(int i= 0; i<8; i++){
    bufferleido = bufferleido + mySerial.peek();
  }
  return bufferleido;
}
```

Ahora comenzaremos a explicar las funciones de control y de comunicación que hemos generado. En primer lugar empezamos con una función de confirmación, ésta lo que hace es enviar el identificador del esclavo con el que nos queremos comunicar y decirle qué tipo de dato queremos leer o envíar. Si hay comunicación esta función limpia el buffer con las funciones anteriores y muestra los datos recibidos.

```
//FUNCION DE CONFIRMACION DE ACTUADOR/SENSORES////////////////////////////////
 char Confirmacion(char cosa, int id){
  mySerial.write(id);
  Serial.println("id");
  while(!mySerial.available()){
    Serial.println("esperando R Act");
  }

  Serial.println("Buffer: " + Buffer());
  char confirmacion =  mySerial.read();
  limpiarBuffer(8);
  Serial.println(confirmacion);

  if(confirmacion == 'R'){
    Serial.println("recibo r");
    mySerial.write(cosa);
  }
  while(!mySerial.available()){
    Serial.println("esperando ");
    Serial.println(cosa);
  }

  Serial.println("Tenemos respuesta");
  Serial.println("Buffer: " + Buffer());
  confirmacion =  mySerial.read();
  limpiarBuffer(8);
  return confirmacion;
}
```

En segundo lugar, podemos ver dos funciones en las cuales, si el dato recibido en la función anterior es correcto, envía al esclavo qué sensor o actuador quiere leer o ejecutar, y devuelve el mensaje que nos da como resultado a esto. Para hacer esto usamos también la función de lectura de buffer, que nombrábamos antes, realizando un pequeño cambio, y es que como el valor que nos da el sensor es enviado como un *String* entre corchetes, necesitamos sacar lo que está dentro de éstos, por lo que, usamos la función *.indexOf* para separarlo y mostrarlo.

```
//FUNCION COMUNICACION PARA ENVIO DE ACTUADOR ESCLAVO:////////////////////////,
void comunActu(int timeAc, int identificador){

  char confirmacion =Confirmacion('A', identificador);

    if(confirmacion == 'A'){
      Serial.println("recibo A");
      mySerial.write(timeAc);
    }
}


//FUNCION COMUNICACION PARA LECTURA DE VARIABLE SENSOR ESCLAVO://///////////////
float lectEsclavo(char sensor, int identificador){
  float dato;
  char confirmacion = Confirmacion('D', identificador);
  if(confirmacion == 'D'){
    Serial.println("recibo D");
    mySerial.write(sensor);
    while(!mySerial.available()){
      Serial.println("espero 2");
    }

    String lecturaDatos = "";
    lecturaDatos = mySerial.readString();
    //limpiarBuffer(8);
    Serial.println("buffer bruto : " + String(lecturaDatos));

    int primerCorchete = lecturaDatos.indexOf('[');
    int segundoCorchete = lecturaDatos.indexOf(']' , primerCorchete + 1 );
    lecturaDatos = lecturaDatos.substring(primerCorchete +1, segundoCorchete);

    limpiarBuffer(8);
    Serial.println("Buffer: " + Buffer());
    Serial.println("LA LECTURA FUE: " + lecturaDatos);
    dato = lecturaDatos.toFloat();

    return dato;
  }
  return -1;
}
```

Posteriormente, tenemos una función que compara cada valor obtenido de las funciones anteriores con unos valores de consigna, si éstos están dentro se permitiría el riego, devolviendo la función un uno. Y si éstos están fuera de los valores de consigna tendríamos un error y nos daría como resultado un menos uno. Para ello, lo que hacemos es usar la función anterior para envíar cada sensor que queremos leer, enviando una letra específica (H=Humedad, Z=Luz, L=Lluvia,...). La forma en la qué decidimos si se riega o no, es por negación, si todos los casos que niegan el

riego se han superado, es decir, no se devuelve ningún menos uno, entonces queda el caso favorable devolviendo el uno.

```
//FUNCION ANALISIS DE SENSORES/////////////////////////////////////////////////
int controlAct(int identificador){

  float luz = lectEsclavo( 'Z' , identificador);
  Serial.println("leemos si es de dia");
  delay (100);

  float lluvia = lectEsclavo( 'L' , identificador);
  Serial.println("leemos si esta lloviendo");
  delay (100);

  float humSul = lectEsclavo( 'S' , identificador);
  Serial.println("leemos humedad del suelo");
  delay (100);

  float humedad = lectEsclavo( 'H' , identificador);
  Serial.println("leemos humedad");
  delay (100);

  float temperatura = lectEsclavo( 'T' , identificador);
  Serial.println("leemos temperatura");
  delay (100);

  /* Si es de noche */
  if (luz <= LUMINOSIDAD){
    Serial.println("No se riega porque es de noche");
    return -1;
  }


 /* Si el suelo está húmedo */
 if(humSul <= HUMEDAD_SUELO){
   Serial.println("No se riega porque es el suelo está demasiado humedo");
   return -1;
 }

 /* Si la humedad del aire es alta */
 if(humedad >= HUMEDAD){
   Serial.println("No se riega porque es el tiene suficiente humedad");
   return -1;
 }

 /* Si la temperatura es muy alta, no se riega */
 if(temperatura >= TEMPERATURA_CALOR ){
   Serial.println("No se riega porque peligra el estado de la plantacion");
   return -1;
 }
```

```
  /* Si la temperatura es muy baja no se riega */
  if(temperatura <= TEMPERATURA_FRIO ){
    Serial.println("No se riega porque se puede congelar");
    return -1;
  }

  /* Si no esta lloviendo */
  if(lluvia == 0 ){
    Serial.println("Si no está lloviendo, se riega");
    return 1;
  }
  return -1;
}
```

Y por último, dentro del conjunto de las funciones de control y comunicación se halla una función, cuyo objetivo es según el error o resultado de la función anterior, enviar un tiempo de riego o indicarnos el tipo de error que ha surgido a lo largo del proceso del programa.

```
//FUNCION INFORMACION SALIDA/ACTUADORES ESCLAVO:////////////////////////////////
void enviarActuador(int tipOrden , int identificador){
 int stopRiego=0;
  switch (tipOrden) {
    /* Si la orden es 'no regar' */
    case -1:
      Serial.println("ERROR: Alguna variable contrarresta el riego");
      comunActu( stopRiego , identificador);
      break;

    /* Si la orden es 'regar' */
    case 1:
      Serial.println("OK: Se pide riego");
      comunActu( TIEMPO_RIEGO, identificador);
      break;

    default:
      Serial.println("ERROR: Valor de entrada nulo, no tienen funcion asignada");
  }
}
```

Ahora vamos a hablar de las partes *void setup* y *void loop* del código, que como ya sabemos, son por las que se rige cualquier estructura de código dentro del entorno de arduino. Por un lado, en el *void setup* encontramos la inicialización del Serial y del mySerial a una velocidad de 9600 baudios (bps), además de hacer una espera de un segundo para efectuar estos cambios.

```
///////////////SETUP////////////////////////////////////////////////////////////

void setup() {
  pinMode(8,OUTPUT);
  mySerial.begin(9600);
  Serial.begin(9600);
  delay(1000);
}
```

En el *void loop* lo que podemos observar es en primera instancia una sentencia que se ejecuta si se recibe una petición de identificador, en este caso, haciendo uso de las funciones de creación de *Array* que nombrábamos al principio, generamos un identificador nuevo añadiéndolo al *Array* y enviándoselo al esclavo pertinente. Y en segundo lugar, lo que tenemos es un bucle que se repite tantas veces como número de esclavos halla,en el cual se llama a la función de control de consigna dentro de la cual se ejecutan todas las funciones para ver que se debe ejecutar y la de control de error, que es la que nos indica si ha habido algún error y en caso de que no, envía el tiempo de riego.

```
///////////////LOOP/////////////////////////////////////////////////////////////

void loop() {
  //Creamos sincronizacion con el nuevo esclavo y lo añadimos
  Serial.println("Hay esclavos conectados?");
  if(mySerial.read()=='I'){
        Serial.println("Detectamos esclavo, lo agregamos");
        int nuevo=newId();
        esclavosConectados=count;
        Serial.println("Limpiamos el buffer");
        limpiarBuffer();

        Serial.println("Esperamos 5s para sincronizar.");
        delay (5000);
        Serial.print("Retornamos el numero de esclavo: ");
        Serial.println(nuevo);
        mySerial.write(nuevo);
  }


  for  (int i=1; i <= esclavosConectados; i++){
   int accion = controlAct(i);
   enviarActuador(accion , i);
  }


  delay(5000);
}
```

Descripción del Esclavo: (mirar sección 7.2)

En el programa realizado para la ejecución del esclavo no existen tantas funciones como habían en el maestro, debido a que éste, no tiene que hacer el control de ninguna clase respecto a ninguna variable. Por lo que, lo primero que vamos a encontrar es la declaración de librerías y la inicializaciones de variables tanto propias del programa como también de los distintos sensores, y a continuación se vera la parte del *void loop* y *void setup*.

```
// LIBRERRIAS DEL PROGRAMA -----------------------------------------------------
#include <SoftwareSerial.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2561_U.h>
#include "DHT.h"

// CONSTANTES DEL PROGRAMA -----------------------------------------------------
const String GANANCIA_SENSOR_LUZ = "AUTO";       // 1, 16, AUTO
const int TIEMPO_INTEGRACION_SENSOR_LUZ = 101;   // 101, 13, 402
#define DHTPIN 2
#define DHTTYPE DHT22         // DHT 22  (AM2302)

 int tiempoRiego;
 long tiempoPrevio;

//Identificador del arduino, asignada por el maestro
int identificador = -1;

//Declaracion de Pines y Variables asociadas-----------------------------------
const int sensorHS = 6;
int valorHS = 0;
const int sensorLL = 4;
int valorLL = 0;
// Actuadores virtuales
int rojo = 9;
int valvulaDig = 8;


// INICIALIZACIONES DE SENSORES-------------------------------------------------
Adafruit_TSL2561_Unified tsl = Adafruit_TSL2561_Unified(TSL2561_ADDR_FLOAT, 12345);
DHT dht(DHTPIN, DHTTYPE);     // LED pins


// Declaracion de un nuevo puerto para la comunicacion serie.
SoftwareSerial mySerial(10, 11);    // RX, TX.
```

En el *void setup* se ejecuta una vez se inicia el arduino como ya sabemos, por ello, lo que hacemos en éste es, aparte de inicilizar el Serial y el mySerial como mismo

hicimos en el maestro, vamos a pedir de forma insistente un identificador al maestro, y si existe algún dato de entrada, lo recogemos y lo tomamos como identificador. Una vez guardado éste en la variable correspondiente inicializamos los pines de los actuadores y de los sensores como salida y entrada respectivamente, y de la salida, si están en estado *High* o *Low*, según corresponda.

```
void setup() {

  analogReference(INTERNAL);

  // Configuramos el puerto serial a 9600 bps.
  mySerial.begin(9600); // Comienzo de la comunicaciÃ³n serie.
  Serial.begin(9600);

  dht.begin();

  Serial.println("Soy el esclavo: mando 'I' para sumarme al sistema");
  mySerial.write('I');
  Serial.println("Limpio la memoria...");
  limpiarBuffer();
  mySerial.listen();
  Serial.println("Espero 10s ...");
  delay (10000);

  //Escucho para si me esta enviando identificador y lo muestro por pantalla.
  if(mySerial.peek() > 0){
      Serial.println("Hay dato...");
      Serial.println("Se asigna como identificador...");
      identificador = mySerial.read() ;
      Serial.print("El numero asignado es: ");
      Serial.println(identificador);
      delay (1000);
  }


  //Defino los pines de los sensores como salida y primer nivel bajo.
  pinMode(sensorHS, INPUT);  //definir pin HS como entrada
  pinMode(sensorLL, INPUT);  //definir pin LL como entrada



  //Declaro los pines de actuadores como salida y primer nivel bajo.
  pinMode(rojo, OUTPUT);
  pinMode(valvulaDig, OUTPUT);
  digitalWrite(rojo, LOW);
  digitalWrite(valvulaDig, LOW);

  delay(1000);                  // Retardo para establecer configuracion del setup.
}
```

A continuación, en el *void loop* lo primero que hacemos es comprobar si tenemos un mensaje de entrada y si lo hay, lo guardamos en una variable. Si éste es igual al identificador del esclavo, ejecutamos el programa pertinente, el cual envía un mensaje de respuesta solicitando una orden. En este caso, la forma de organizar las diferentes órdenes fue con un *Case*, donde según la letra de llegada (H=Sensor de Humedad, Z=Sensor de Luz, L=Sensor de Lluvia,...), entrábamos dentro del switch que nos interesara, dentro del cual leíamos y enviábamos el valor recogido por estos sensores.

```
void loop() {

  // Si hay datos disponibles
  if (mySerial.available()>0){
    Serial.println("Buffer: " + Buffer());
    // Leerlos
    int id = mySerial.read();
    limpiarBuffer();
    Serial.println("id: " + String(id));

    // Si la informacion no es para mi, salgo
    if(id = identificador){

      // Confirmacion al maestro de que lo escuché
      Serial.println("Escribo R");
      mySerial.write('R');              //Confirmo que lo recibi
      while(!mySerial.available()){
        Serial.println("esperando dato");
      }
      // Leo la petición que me manda el maestro
      Serial.println("dato recibido");
      Serial.println("Buffer: " + Buffer());
      char mensaje = mySerial.read();
      limpiarBuffer();
      Serial.println("mensaje: " + String(mensaje));



      // Procesamos la petición
      switch (mensaje) {
        case 'D':       // Si quiere algo de los sensores
          // Le envío confirmación al maestro
          Serial.println("Escribo D");
          mySerial.write('D');

          // Espero una órden del maestro
          while(!mySerial.available()){
            Serial.println("esperando dato");
          }
```

```
// Leo dato recibido
Serial.println("dato recibido");
Serial.println("Buffer: " + Buffer());
mensaje = mySerial.read();
limpiarBuffer();              //Leo que sensor quiere
Serial.println("mensaje: " + String(mensaje));

switch(mensaje){

  case 'S':
    //Lectura del sensor humedad del suelo
    valorHS = analogRead(sensorHS );  //lectura analogica de sensor HS
    mySerial.print("[" + String(valorHS) + "]");
    return;


  case 'L':
    //Lectura del sensor de lluvia
    valorLL = digitalRead(sensorLL );  //lectura digital de sensor LL
    if (valorLL == LOW)
      mySerial.print("[" + (String)1 + "]");
    else
      mySerial.print("[" +(String)0 + "]");
    return;


 case 'H':
   //Lectura del sensor de Humedad
   float h;
   h = dht.readHumidity();
   mySerial.print("[" + String(h) + "]");
   return;


 case 'T':
   //Lectura del sensor de Temperatura
   float t;
   t = dht.readTemperature();
   mySerial.print("[" + String(t) + "]");
   return;
```

```
    case 'Z':
      //Lectura del sensor de Luminosidad
      sensor_t sensor;
      tsl.getSensor(&sensor);

      // Establece la ganancia deseada de acuerdo a la constante
      //GANANCIA_SENSOR_LUZ
      if (GANANCIA_SENSOR_LUZ == "AUTO")
        tsl.enableAutoRange(true);
      if (GANANCIA_SENSOR_LUZ == "16")
        tsl.setGain(TSL2561_GAIN_16X);
      if (GANANCIA_SENSOR_LUZ == "1")
        tsl.setGain(TSL2561_GAIN_1X);

      // Establece el tiempo de integración deseado de acuerdo a
      // la constante TIEMPO_INTEGRACION_SENSOR_LUZ
      if (TIEMPO_INTEGRACION_SENSOR_LUZ == 101)
        tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_101MS);
      if (TIEMPO_INTEGRACION_SENSOR_LUZ == 402)
        tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_402MS);
      if (TIEMPO_INTEGRACION_SENSOR_LUZ == 13)
        tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_13MS);

      sensors_event_t event;
      tsl.getEvent(&event);

      mySerial.print("[" + String(event.light) + "]");
      Serial.println("TOTAL A ENVIAR: [" + String(event.light) + "]");
      return;


  break;
case 'A':      // Si quiere algo de los actuadores

  // Le mando un mensaje de 'recibido'
  Serial.println("Escribo A");
  mySerial.write('A');

  // Espero la órden del maestro
  while(!mySerial.available()){
    Serial.println("esperando dato");
  }
```

```
          // Leo la órden del maestro
          Serial.println("dato recibido");
          Serial.println("Buffer: " + Buffer());
          tiempoRiego = mySerial.read(); //Recibo tiempo en minutos
          tiempoPrevio= millis();
          limpiarBuffer();
          Serial.println("mensaje: " + String(tiempoRiego));
          break;
      } //Fin del primer switch
    }
  }

  if(tiempoRiego > 0){
    Actuador1(tiempoPrevio,tiempoRiego);
  }else{
    Serial.println("No se ha pedido riego");
    digitalWrite(valvulaDig, LOW);
  }

}// Fin de la funcion LOOP
```

Finalmente, como este programa no sólo recibe órdenes sino que de forma paralela ejecuta el riego de una plantación, lo que encontramos es una función que compara el tiempo actual en el que se recibe una orden y el tiempo que debe estar activo el actuador, para así, sin hacer uso de delays que paren el proceso del programa, controlamos el riego de forma paralela a la ejecución del resto del programa anteriormente explicado.

```
void Actuador1(long previo,int timeR){

  long RiegoMSeg = timeR*60*1000;
  unsigned long timeActual = millis();

   if(timeActual - previo <= RiegoMSeg) {
     Serial.println("Se esta Regando en este momento");
     digitalWrite(valvulaDig, HIGH);
   }else{
     Serial.println("Riego finalizado");
     digitalWrite(valvulaDig, LOW);
   }
}
```

3.3. Programación para una comunicación tipo Radio Frecuencia

La segunda parte en la que se divide nuestro proyecto, es en la obtención de un Maestro y un Esclavo, que se comunicaran de una manera inalámbrica, ya que con ésta conseguiríamos suprimir el coste, que nos provoca el cableado de un sistema de comunicación de tales dimensiones, tal y como era el que se había generado anteriormente.

Para conseguir esto se estudió las diferentes formas de comunicación inalámbrica que existen hoy en día en la industria. Las más comunes para este tipo de sistemas son la de comunicación tipo Wifi (las cuales necesitan para su comunicación un punto de acceso y un dispositivo wifi que se conecte a éste, por lo que requeriría de Internet para funcionar), comunicación por infrarrojos (que genera comunicación entre dos nodos usando ledes infrarrojos), comunicación Bluetooth (transmite la información por radiofrecuencia en la banda ISM)y radiofrecuencia. Estas ultimas nos permiten

3.3.1. Programa Final Inalámbrico

Tal y como se trata en nuestro sistema se han elaborado dos programas, como venimos describiendo. Uno por el Maestro que hará control de las variables recogidas por los sensores y otro el del Esclavo que enviará estas variables al Maestro después de pedirlas a los sensores correspondientes. A continuación se describirán los diferentes programas.

Pero en este apartado no vamos a describir con tanto detalle el funcionamiento de las distintas funciones que conforman los distintos programas, sino que se hablará de las modificaciones que se llevaron a cabo en cada una de éstas para pasar de un sistema serial a un sistema por comunicación de radiofrecuencia.

Descripción del Maestro (mirar sección 7.2)

Al igual que en todos los códigos que hemos realizado lo primero que escribimos en un nuestro código es la llamada a las distintas librerías de las que vamos a hacer uso. En este código incorporamos una nueva librería que nos serviría para utilizar las distintas funciones y argumentos para la comunicación inalámbrica, en nuestro caso las destinadas al control de los módulos de radiofrecuencia, los RF24.

```
// LIBRERRIAS DEL PROGRAMA ----------------------------------------------------------------
#include <SoftwareSerial.h>

#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>
```

La primera modificación que hacemos en el código es definir los pines que necesita nuestros módulos de comunicación inalámbrica para enviar y recibir datos, y crear una instancia con dichos pines. Además, creamos los canales de transmisión, a los cuales les daremos un nombre hexadecimal y por el que se comunicarán maestro y esclavo.

```
const int pinCE = 7;                // PINES ASOCIADOS A LA COMUNICACION INALAMBRICA
const int pinCSN = 8;
RF24 radio(pinCE, pinCSN);  //Creamos instancia de la radio con sus pines de control

|
// INICIALIZACIONES -----------------------------------------------------------------
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };
```

A continuación, en el Setup necesitamos ejecutar unas determinadas funciones. En primer lugar, la *.begin* empieza creando el canal que definimos anteriormente. La siguiente, *.setRetries* establece la velocidad de escritura y de lectura de ambos canales. Las dos que nos encontramos después *.openWritingPipe* y *.openReadingPipe*, indican que canal será de escritura y cual será el encargado de leer los mensajes. Y a continuación, la función *.startListening* lo que hace es empezar a escuchar mensajes dado que, con las funciones anteriores ya quedaría definido el canal de comunicación.

```
void setup() {
  //mySerial.begin(9600);              // Comienzo de la comunicacion serie

  Serial.begin(9600);

  radio.begin();
  radio.setRetries(15,15);
  radio.openWritingPipe(pipes[0]);
  radio.openReadingPipe(1,pipes[1]);

  radio.startListening();

  delay(100);
}
```

Como este programa necesita establecer y ejecutar ciertas funciones para leer y escribir los mensajes, teniéndoles que decir el tamaño de la variable que va a almacenar, hemos tenido que crear de forma diferente de los programas de comunicación Serial unas funciones que nos envíen y reciban una determinadas variables. Por ello, decidimos crear cuatro funciones: la primera envía una variable tipo *char*, la segunda envía una variable tipo *int*, la tercera leería una variable tipo *char* y la cuarta leería una variable tipo *int*.

La manera en la que funciona la función *envio(char letra)*, es en primer lugar, parar de escuchar cualquier mensaje, es decir, anula el canal de lectura, a continua-

ción, crea una variable en la que almacena el carácter a enviar y con la función *.write* enviamos ese carácter diciéndole el número de bytes que ocupa. Una vez hecho esto y habiendo enviado el mensaje se vuelve a activar el canal de lectura y vuelve a realizarse la comunicación. La función de *envio(int letra)* hace exactamente lo mismo pero envía dos paquetes de un byte, en uno envíamos los bits más significativos y a continuación, los menos significativos para seguir ejecutando el mismo proceso que la función anterior.

```
void envio(char letra){
  radio.stopListening();
  char msgl[2];
  msgl[0]= letra;
  radio.write(msgl, 1);
  radio.startListening();

}

void envio(int lectura){
  radio.stopListening();
  uint8_t Envio[2];
  Envio[0] = (lectura >> 8);
  Envio[1] = (lectura & 0xFF);
  radio.write((uint8_t *)Envio, 2);
  radio.startListening();

}
```

Las dos siguientes funciones, recogen cualquier mensaje de entrada siempre y cuando exista en el canal de lectura, en caso de que no lo hubiera no se leería nada, tanto en la función *leoNum* como en la de *leoLetr*.

```
char leoLetr(){
    if (radio.available()) { //Si hay datos disponibles
     // Serial.println("Leemos si nos llega alguna letra como respuesta");
    uint8_t msj_maestro[2];
    radio.read(msj_maestro, sizeof(msj_maestro));
    char recibido= (char)msj_maestro[0];
    return recibido;
  }
}
```

```
int leoNum(){ //Leo el mensaje y miro si es superior o inferior a los bytes esperados
  uint8_t dato_temp[2];//Dos elementos
  int result;
  if (radio.available()) { //Si hay datos disponibles
    radio.read(dato_temp, 2);
    result = dato_temp[0] << 8 | dato_temp[1];
    return result;
  }
}
```

La modificación que le hicimos a la función confirmación de la que ya se habló en el programa del Maestro para la comunicación alámbrica, fue, en primer lugar usar la función *envio* de la que estamos hablando anteriormente para enviar mensajes, en segundo lugar, comprobar si hay algún mensaje de entrada y en caso afirmativo mostrarlo, en caso negativo se reenviaría el mensaje y se comprobaría nuevamente. Si se llegara a enviar más de cinco veces y no hubiera respuesta, se mostraría un mensaje de error y finalizaría el programa. Este proceso lo haría dos veces, una para pedir identificador y otra para verificación de respuesta correcta, que en nuestro caso es 'R'.

```
char Confirmacion(char cosa, int id){
  char confirmacion;
//Envio identificador
  Serial.print("Esperando Respuesta al id: ");
  Serial.println(id);
  envio(id);

  int reenvios = 0;
  unsigned long tiempo_actual = millis();
  while ((reenvios <= 5)){

    if(radio.available()) {
      uint8_t msj_maestro[2];
      radio.read(msj_maestro, sizeof(msj_maestro));
      confirmacion = (char)msj_maestro[0];
      Serial.print("Respuesta a id: ");
      Serial.println(confirmacion);
      break;
    }
    if ((reenvios < 5) && ( millis() - tiempo_actual > 200)){
      reenvios++;
      tiempo_actual = millis();
      envio(id);
      Serial.print("Intentando conectar ");
      Serial.println(reenvios);
    }
```

```
    if (reenvios >= 5) {
      Serial.println("Se ha supeado el numero de intentos");
      Serial.print("No se ha conectado ");
      Serial.print(id);
      Serial.println(" en envio de id");
//      idCero(id);
      return 'E';
    }
  }


  if(confirmacion == 'R'){
   Serial.print("Recibo r: ");
   Serial.println("Esperando respuesta a act/sen ");
   envio(cosa);
   int reenvios2 = 0;
   tiempo_actual = millis();
   int RR;
   while ((reenvios2 <= 5)){

    if(radio.available()){
      uint8_t msj_maestro[2];
      radio.read(msj_maestro, sizeof(msj_maestro));
      confirmacion = (char)msj_maestro[0];

      if(confirmacion == 'R'){
         Serial.println("Se sigue recibiendo R");
         RR++;
         if(RR == 4){
           return 'E' ;
         }
      }else{
        Serial.print("Respuesta a tipo: ");
        Serial.println(confirmacion);
       // reenvios = 0;
        break;
      }
    }
```

```
    if((reenvios2 < 5) && ( millis() - tiempo_actual > 200)){
      reenvios2++;
      tiempo_actual = millis();
      envio(cosa);
      Serial.print("Intentando conectar ");
      Serial.println(reenvios2);
    }if (reenvios >= 5) {
      Serial.println("Se ha supeado el numero de intentos");
      Serial.print("No se ha conectado ");
      Serial.print(id);
      Serial.println(" en envio de act/sen");
//      idCero(id);
      return 'E';
    }
   }


   return confirmacion;
  }else{
    Serial.print("No se ha conectado ");
    Serial.print(id);
    Serial.println(" en recepcion de R");
//    idCero(id);
    return 'E';
  }
 }
```

Como también van a ejecutarse más de un esclavo y el maestro tiene que leerlo en su totalidad debemos de establecer unas velocidades de comunicación, no del canal, que ya lo tenemos, sino de las veces que enviamos mensajes y también de los que debemos leer. Para hacer esto hacemos una comparación entre el tiempo en el que pedimos o enviamos un mensaje y cuando se supone que tenemos que leerlo. Para hacer esto, comparamos el tiempo en el que se envía el mensaje y el tiempo que transcurre desde ese entonces y si este tiempo es superior a uno prefijado se vuelve a enviar el mensaje y esperamos a recibir alguno. Esto lo conseguimos haciendo que este hecho sea una condición para enviar mensajes, en conjunto con la de enviar más de cinco veces que comentábamos antes.

```
    if ((reenvios < 5) && ( millis() - tiempo_actual > 200)){
      reenvios++;
      tiempo_actual = millis();
      envio(id);
      Serial.print("Intentando conectar ");
      Serial.println(reenvios);
    }
```

Para las siguientes funciones como la de *lecEsclavo*, *comunActu*, *controlAct*, se realizaron mejoras como las comentadas en este apartado y para el método de comunicación del que estamos hablando. Es decir, su funcionamiento no difiere del de modo alámbrico, ya que solo se adaptaron con la librería de radiofrecuencia.

Otra de las mejoras que realizamos en el modo inalámbrico surgió a raíz de un problema que ya hemos comentado anteriormente. Lo que hace el *void loop* a parte del bucle con el que llamamos a las funciones anteriores y el cual ya hemos explicado, es que en este caso, es el maestro el que envía o el que busca si algún esclavo se ha conectado recientemente y mira si no está registrado. En caso de que no estuviera registrado, lo registra y le asigna un nuevo identificador.

Para lograr esto, creamos un *while* que busca información diez veces cada quinientos milisegundos.Para ello, si esta en el momento de *esperando por esclavos* envía el carácter *'I'* esperando a que algún esclavo que este por esperando por asignación de identificador responda. En caso de que le respondan con un *'I'* éste envía un número nuevo. En caso de que supere la búsqueda en diez veces se sale del *while* mostrándonos que no hay esclavos.

```
while ((reen <= 10)&&(waitForSlaves == true)){
  if ((reen < 10) && ( millis() - tiempo_actual > 500)){
    reen++;
    tiempo_actual = millis();
    envio('I');
  }
  if (reen >= 10){
    Serial.println("No hay esclavos a la espera");
    break ;
  }
  if(radio.available()){
    radio.read(msj_maestro, sizeof(msj_maestro));
    confirm = (char)msj_maestro[0];
    Serial.print("Algun esclavo respondio: ");
    Serial.println(confirm);
  }
  if(confirm == 'I'){
    Serial.println("Detectamos esclavo, lo agregamos");
    int nuevo = newId();
    esclavosConectados=count;
    Serial.print("Retornamos el numero de esclavo: ");
    Serial.println(nuevo);
    envio( nuevo);
    intentos[nuevo]= 0;
    break;
  }
}
```

La última mejora que añadimos fue la del reseteo de variables de tiempo, es decir, cuando la función *millis()* se va a desbordar surge un problema debido a que las variables de almacenamiento de tiempo quedan con unas variables superiores de las que nos va a dar esta función, lo que provocaría que las comparaciones de tiempo fueran erróneas, para esto decidimos poner todas las variables a cero y aunque perdamos precisión conseguimos actualizar el programa eliminando un error que provocaría un fallo desde ese entonces hasta que se apagara el arduino.

La solución de este problema fue comparar el tiempo actual con el tiempo de saturación y si estaba muy próximo reseteábamos estas variables o en otro caso, si la primera variable que almacenamos en el programa es mayor al tiempo actual de cuando finaliza, querría decir que, se ha saturado la función a lo largo del programa y se necesitaría el reseteo de las variables.

```
if((waitForSlaves == true)&&(millis() > 4294963295)){
  delay(5000);
  for(int i = 0; i< CAPACIDAD_SLAVES; i++){
    intentos[i] = 0;
  }
}

if((waitForSlaves == true)&&(control_49 > millis())){
  for(int i = 0; i< CAPACIDAD_SLAVES; i++){
    intentos[i] = 0;
  }
}
```

Descripción del Esclavo (mirar sección 7.2)

Para adaptar el código del esclavo alámbrico necesitamos incluir un conjunto de librerías nuevas que unas nos ayudaran a controlar los módulos de radiofrecuencia y otra que nos permitirá controlar el modo suspensión o modo bajo consumo del arduino y si conseguimos este modo, necesitamos guardar en la memoria no volátil del esclavo por lo que también incluiremos la librería destinada a este fin. El resto de librerias ya se han comentado en otros apartados y son únicamente para controlar los diferentes sensores.

```
// LIBRERRIAS DEL PROGRAMA ---------------------------------------------------------------
#include <SoftwareSerial.h>

#include <Wire.h>

#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2561_U.h>

#include "DHT.h"

#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>

#include <LowPower.h>

#include <EEPROM.h>
```

Además de las variables y las constantes fijas que definimos al principio de nuestro programa y cuyo código ya hemos comentado en otros apartados, para este nuevo esclavo generamos otro tipo de variables que nos permitirán generar unos canales de comunicación inalámbrica como mismo generábamos en el maestro. A diferencia del esclavo alambrico, este controla cuando se activan los sensores teniendo que activar las diferentes variables de manera digital.

```
//Declaracion de Pines y Variables asociadas para Comunicacion Inalam-----------------------
const int pinCE = 7;
const int pinCSN = 8;
RF24 radio(pinCE, pinCSN); //Creamos instancia de la radio con sus pines de control
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };


//Asignacion de pines para alimentacion seccionada--------------------------------------

const int ali_HS  = 5 ;
const int ali_LL  = 4 ;
const int ali_LX  = 3 ;
const int ali_HT  = 2 ;
const int ali_CO  = 1 ;
```

Las cuatro funciones que vemos a continuación en el código son las mismas que las que se explicaron en el apartado anterior del Maestro, que son las de envío y recepción de mensajes datos numéricos como de caracteres. Como esto es así no explicaremos el uso, ni su utilidad en este apartado.

Después de encontrarnos con estas cuatro funciones, vemos una nueva llamada a *comproba maestro*. La utilidad de esta función es escuchar de forma indefinida si algún maestro está buscando algún esclavo nuevo, en caso afirmativo se recibiría una *'I'* para la cual le responderíamos con el mismo carácter solicitando así un identificador. Si recibiéramos el identificador lo guardaríamos en la memoria *EEPROM* , la cual leeríamos cada vez que necesitáramos usar o comprobar este dato.

```
int comproba_maestro(char lectura){
int result = 0 ;
 int reenvios = 0;
 uint8_t dato_temp[2];
 bool waiting = true;
 char recibido = 'P';
 uint8_t msj_maestro[2];

 while(1){
  Serial.print("Esperando al maestro.");
  Serial.println(recibido);
  if(radio.available()){
     radio.read(msj_maestro, sizeof(msj_maestro));
     recibido = (char)msj_maestro[0];
     Serial.print("Esperando nuevo mensaje:  ");
     Serial.println(recibido);
   }
```

Si observamos bien vemos que de igual manera que hacíamos en el maestro y en

cualquier envío de mensaje, comprobamos si se ha enviado cinco veces la información y en caso de que no hubiera respuesta, se solicitaría el reseteo del esclavo, porque si no se mantendría en un bucle infinito.

```
   if(recibido == 'I'){
      envio(lectura);
      long tiempo_actual = millis();
      while ((reenvios <= 5)){
        if ((reenvios < 5) && ( millis() - tiempo_actual > 200)){
          reenvios++;
          tiempo_actual = millis();
          envio(lectura);
          Serial.print("Esperando a que el maestro nos asigne id. ");
          Serial.println(reenvios);
        }
        if (reenvios >= 5) {
          Serial.println("Se ha superado el numero de intentos. ");
          Serial.print("No se ha podido conectar al maestro. Reiniciar.");
          break ;
        }
        if(radio.available()) {
          radio.read(dato_temp, 2);
          result = dato_temp[0] << 8 | dato_temp[1];
          Serial.print("Recibimos id: ");
          Serial.println(result);
          return result;
        }
      }
    }
  }
}
```

Una modificación que realizamos en el esclavo a diferencia de los esclavos alambricos y del maestro que acabamos de describir es que en este caso en el *setup*, abrimos los canales de comunicación, cosa que en los alambricos no hacia falta y que respecto al maestro anterior de hace de una manera distinta, ya que los canales cambian. El motivo es simple y es que, si el maestro usa un canal para enviar información ese mismo canal será el canal de recepción para el esclavo, y el de recepción del maestro, el de enviar información del esclavo.

```
void setup() {

  analogReference(INTERNAL);
  Serial.begin(9600);

  radio.begin();
  radio.setRetries(15,15);
  radio.openWritingPipe(pipes[1]);
  radio.openReadingPipe(1,pipes[0]);
  radio.startListening();
```

A continuación lo que nos encontramos es un bucle, que usa la función *comproba* para lo descrito anteriormente, en el cual, entramos desde que se ejecuta el programa y se queda dentro hasta que algún maestro lo detecte y le envíe un identificador que mas adelante guardara en la memoria EEPROM

```
Serial.println("Soy el esclavo: mando 'I' para sumarme...");
while (!(identificador = comproba_maestro('I')))
  delay(50);

Serial.print("recibimos id: ");
Serial.println(identificador);
comu = true;

// Guardamos el valor del identificador en la memoria eeprom del arduino para tenerlo despues
// del modo sleep

EEPROM.write(Direccion, identificador);
```

En el código del esclavo, el*void loop* es muy similar al de modo alambrico. Como mismo pasaba en este, el *void loop* esta conformado por un *switch case* en el que para cada caso realizamos una comprobación de mensaje. Esta comprobación consiste en esperar a recibir cualquier tipo de mensaje, el cual no interfiera con otros esclavos. Si el mensaje es una de las opciones del switch se ejecuta esa parte del código, en caso negativo, si cumple que en una comparación de tiempo se supera un tiempo consigna, se desecha la comunicación y el esclavo pedirá reiniciar el sistema ya que se quedará en un bucle infinito.

```
while(1){
  if (radio.available()) {
    radio.read(msje_maestro, sizeof(msje_maestro));
    mensaje= (char)msje_maestro[0];
    Serial.print("mensaje recibido para tipo: ");
     Serial.println(mensaje);
    break;
  }

  if (millis()-time_ree > 1000) {
    Serial.println("Reenvio R de confirmacion");
    time_ree=millis();
    envio('R');
  }
  if(millis()-timecom > 10000){
    comu = false;
    while(comu == false){
      Serial.println("Se ha perdido comunicacion, se solicita reseteo del esclavo");
    }
  }
}
```

Finalmente lo que encontramos es como se ejecuta el actuador. La forma en la que lo activamos es mediante un solo mensaje tipo *char* 'A' que recibe del maestro y que a diferencia del de el modo alámbrico, este carácter no esta precedido de el tiempo que va a estar encendido. La razón por la que no se envía este valor de tiempo desde el maestro, surgió de un problema que apareció en esos casos y es que, a como teníamos el programa, solo podíamos enviar números que no fueran identificadores ya que los esclavos podían recogerlos como su identificador y ejecutarse sin nosotros necesitarlos ni quererlo. Por lo que como resultado eliminamos la parte de lectura de ese valor y como nuevo añadido, lo que hicimos es que si el actuador ya esta encendido, no necesitaría volver a activarse, ni leer si se le pide, o no, nuevo riego, estableciendo también este con un valor fijo ya prefijado.

```
    case 'A':      // Si quiere algo de los actuadores

      // Le mando un mensaje de 'recibido'
      Serial.println("Escribo A");
      envio('A');
      Serial.println("esperando dato para actuador");

     if(regando == false){

        tiempoPrevio = millis();
        regando = true;
        break;
      }else{
       Serial.println("Ya se esta regando, obviamos la peticion");
       break;
      }
  }
}

if(regando == true){
  riego = Actuador1(tiempoPrevio,TIEMPO_RIEGO);
    if(riego == 0 ){
      regando = false;
    }
 }else{
  Serial.println("No se ha pedido riego");
}
```

4. PRESUPUESTO DEL MATERIAL

A continuación se presenta una tabla con los precios de los componentes usados y la cantidad necesaria para implementar un sistema como el que se ha ido describiendo en el proyecto. La cantidad total del material alcanza los 99 €, situándose por debajo de la media de precios de programadores de riego.

Tabla de precios de los componentes			
DESCRIPCIÓN	CANTIDAD	COSTE UNITARIO (€/unidad)	COSTE (€)
Arduino UNO	3	6,42	19,26
Arduino Nano	1	4,28	4,28
Multímetro digital	1	9,49	9,49
Sensor DTH22(AM2302)	2	5,35	10,7
Sensor FC-28	2	0,85	1,7
Sensor TSL2561	2	4,28	8,56
Sensor YL38	2	2,2	8,56
Módulo NRF2401	3	3,7	6,6
Regulador de tensión L78L33	2	0,4	7,4
MOSFET-TN0702	2	0,47	0,8
Electroválvula	2	6,5	0,94
Puente en H L293D	2	4,74	13
Cables Dupont Macho-Hembra	20	0,35	7
COSTE TOTAL			98,29

El que este precio de coste sea tan bajo frete a otros en el mercado, es bueno ya que este es un controlador que riega en función de las condiciones ambientales y no del día u hora que son lo que encontramos de forma mas común, por lo que en mercado un sistema como el nuestro puede ser muy competitivo. Ademas, al ser Arduino un elemento de bajo consumo, seria interesante hacer un estudio de viabilidad de este sistema con baterías y fuentes de alimentación renovables, que permitan su uso sin necesidad de alimentación de red, ya que de esta forma entraría muy bien en el mercado actual.

En cuanto a la tabla del coste mano de obra, hemos usado las horas que nos ha llevado realizar cada parte del proyecto a un coste meramente orientativo con el que poder evaluar precios. El precio total de esta elaboración llega hasta los 6600 €.

Tabla de precios de mano de obra			
DESCRIPCIÓN	HORAS	COSTE UNITARIO (€/hora)	COSTE (€)
Tiempo de documentacion	50	15	750
Tiempo de programacion	200	15	3000
Tiempo de Implementacion	190	15	2850
COSTE TOTAL			6600

5. CONCLUSIONES

Lo que se ha llevado a cabo en este proyecto ha sido la elaboración de un código para Arduino, el cual nos permitiera controlar un riego de forma autónoma. Para ello, se ha realizado de forma autodidacta un estudio previo para la viabilidad de las diferentes formas de comunicación para este tipo de proyecto.

Una de las conclusiones que podríamos sacar en claro es la diferencia significativa de precios de un sistema con comunicación alámbrica y otro de comunicación inalámbrica. Está claro que el cableado que conlleva una comunicación alámbrica es un gasto que con la tecnología actual se puede suprimir.

Por otro lado, hemos visto que para este tipo de sistema de fácil instalación, es decir, darle alimentación y que funcione solo, no era necesario implementarlo con métodos de comunicación inalámbrica como WIFI o Bluetooh, que necesitan una previa configuración para su uso. Con esto hemos facilitado que cualquier usuario pueda adquirirlo sin tener necesidad de conocer el sistema, ni tener conocimientos previos de comunicación.

También podemos destacar, que, aunque es cierto que para este sistema es recomendable la radiofrecuencia como tipo de comunicación, por la distancia en la que se pueden comunicar los distintos elementos, la sencillez de uso de los módulos y las librerías para programarlo, tiene unos inconvenientes también notables. Uno de ellos es que necesitan de un entorno sin muchas interferencias de otros equipos para una comunicación eficiente. Como también que necesita un entorno libre, es decir, que entre módulos no exista elementos que bloqueen las ondas, que para plantaciones planificadas va muy bien, pero existirán entornos para los que no funcione correctamente.

Para este proyecto, se propuso el uso de las placas Arduino para implementarlo. Esto, como todo, tienen ventajas e inconvenientes. Ventajas pueden ser, un entorno de programación no muy complicado, mucha información por parte de la comunidad en la red, amplia variedad de librerías para sensores y módulos etc. Pero también tenemos que centrarnos en que este tipo de placas, en especial las de tipo UNO, tiene unas limitaciones que impiden el ampliar el proyecto. Entradas y salidas escasas, una memoria con la que no podemos hacer un control tan exhaustivo, límite de guardado en memoria etc. Estaría bien buscar otro tipo de placas que nos permitan mayor capacidad.

Después de hacer comparativa precios, podemos ver reflejado que este tipo de proyecto puede ser muy competitivo en el mercado actual. Los sistemas que se están comercializando, son en su mayoría de programación de riego, es decir, ponemos que días y horas queremos riego y se ejecuta, y rondan precios desde 30 € en adelante. Este proyecto lo que plantea, es un sistema capaz de ejecutar un control que decida

si es recomendable o no el riego y en caso afirmativo realizarlo. Este tipo de sistema de control, los encontramos por precios que superan 130 €, por lo que, dado que el coste de nuestro sistema no lo supera, sería interesante hacer un estudio de mercado para ver qué beneficio se le podría sacar a una placa programada como la nuestra.

5.1. Conclusions

In this project has been carried out the development of a code for Arduino, which would allow us to control irrigation autonomously. For this, a previous study has been carried out in a self-taught way in order to demonstrate the viability of the different forms of communication in this type of project.

One of the conclusions that we could make clear is the difference of prices between a system with wired and wireless communication. It is clear that the wiring that involves a wired communication is an expense that with the current technology can be suppressed.

On the other hand, we have seen that, for this type of system of easy installation. In other words, it gives it power and that it works alone. Furthermore, it is not necessary to implement it with wireless communication methods such as WIFI or Bluetooh. These type of methods need a previous cofiguration for its use. It facilitates that any user can acquire it without having to know the system, that he or she has a previous knowledge of communication.

We can also highlight that, although it is true that radiofrequency as a type of communication is recommended for this system, because of the distance in which the different elements can be communicated, the simplicity of use of the modules and the libraries to program it, it has also notable drawbacks. One of them needs an environment without many interferences from other systems, for an efficient communication. As well as it needs a free environment. In other words, an environment between modules, have no elements that block the waves. For planned plantations works very well, but there will be environments for which it does not work correctly.

For this project, the use of Arduino boards was proposed to implement it and it has advantages and disadvantages. Advantages can be, an environment of programming not very complicated, a lot of information on the part of the community in the network, wide variety of libraries for sensors and modules etc. However, we have to focus on the fact that this type of plates, especially those of the UNO type, has limitations that prevent the project from expanding. Entries and exits scarce, a memory with which we cannot make such an exhaustive control, limit of saving in memory etc. It would be good to look for another type of plates that allow us greater capacity.

After comparing prices, we can see reflected that this type of project can be very competitive in the current market. The systems that are being marketed are; irrigation scheduling. That is to say, we put days and hours we want to water and run, and they run prices from 30 € onwards. This project raises a system which is capable of executing a control that decides if it is advisable or not the irrigation and in the affirmative case to do it. This type of control system will have prices that

exceed 130€. This means that the cost of our system does not exceed it. For this reason, it would be interesting to do a market study to see what benefit could be taken to a plate programmed as ours

6. Bibliografia y Referencias

- Información sobre los distintos sensores y la manera de usarlos en Arduino:

- https://www.luisllamas.es/
- https://learn.adafruit.com/tsl2561/arduino-code
- https://programarfacil.com/blog/arduino-blog/sensor-dht11-temperatura-humedad-arduino/

- Información sobre el uso de librerías, y características de Arduino:

- https://www.prometec.net/
- https://aprendiendoarduino.wordpress.com/

- Información sobre librerías usadas en Arduino:

- https://www.arduino.cc/

- Modo bajo consumo de Arduino y su programacion:

- https://ricveal.com/blog/arduino-modo-sleep/
- https://www.prometec.net/el-modo-sleep-en-arduino/
- https://www.prometec.net/consumos-arduino/

7. Anexos

7.1. Anexo I: Esquemas y conexiones

Los siguientes esquemas y conexiones están realizados con el programa Fritzing que es un programa de software libre que nos permite realizar circuitos electrónicos de forma sencilla y de fácil entendimiento. Ademas representa con gran exactitud los elementos que se usen en los circuitos pudiendo no solo mostrar el conexionado real sino también como esquema eléctrico tal y como mostraremos a continuación.

Conexionado Comunicación Serial

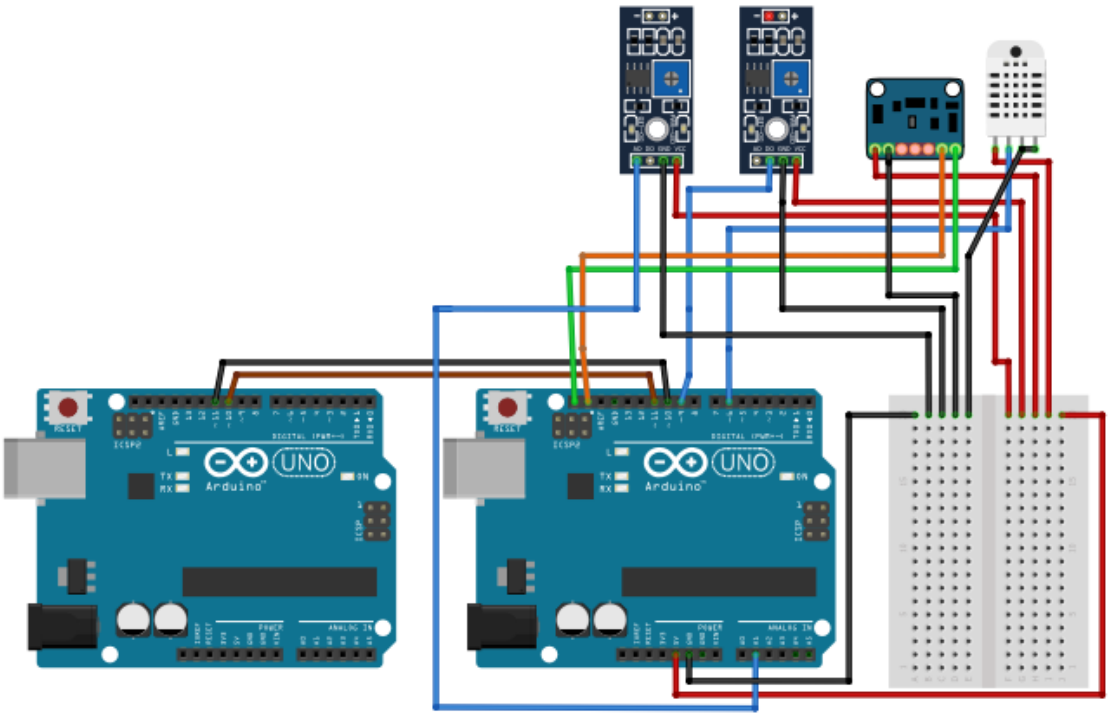


Figura 14: Conexionado del sistema Serial

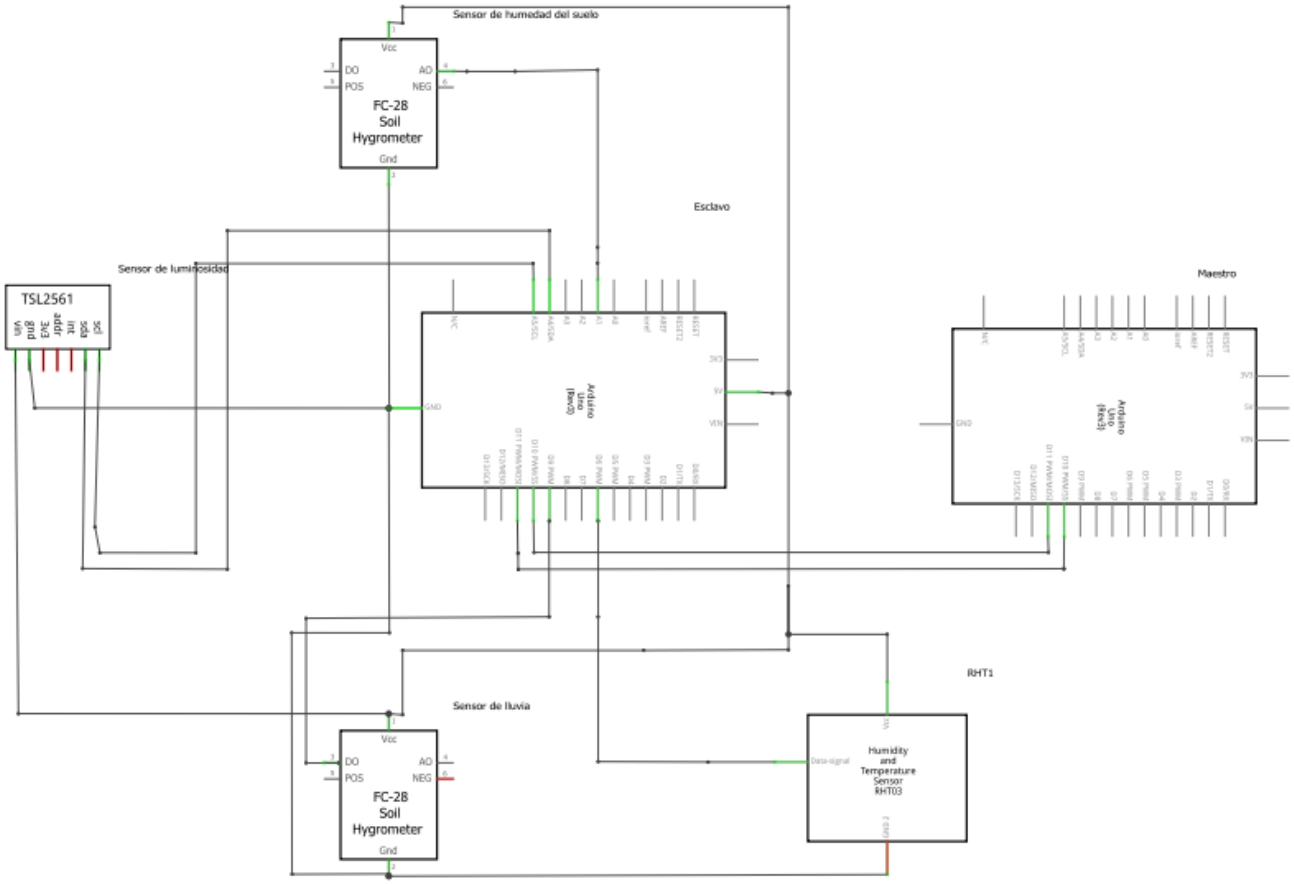


Figura 15: Esquema del sistema Serial

Conexionado del Maestro por Radiofrecuencia

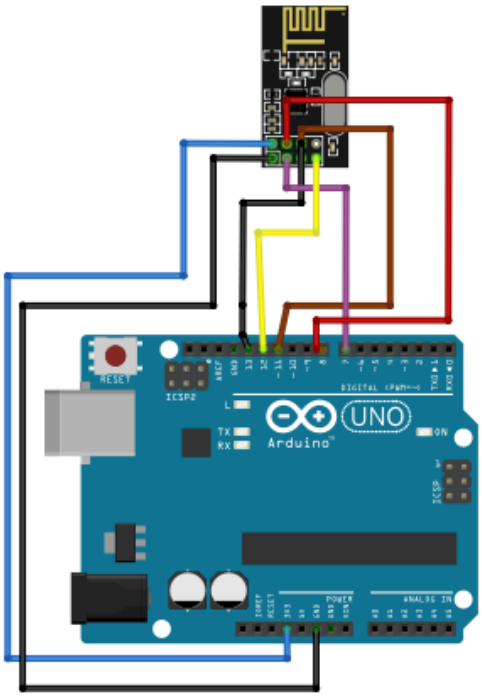


Figura 16: Conexionado del Maestro Inalambrico

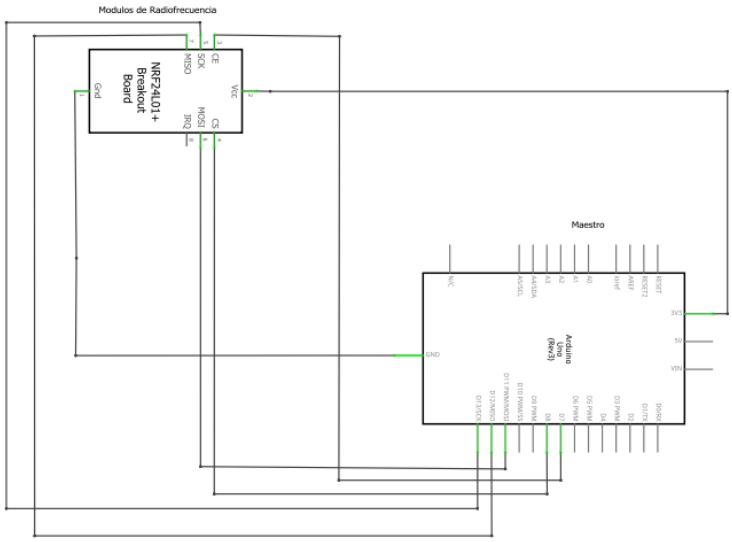


Figura 17: Esquema del Maestro Inalámbrico

Conexionado del Esclavo por Radiofrecuencia

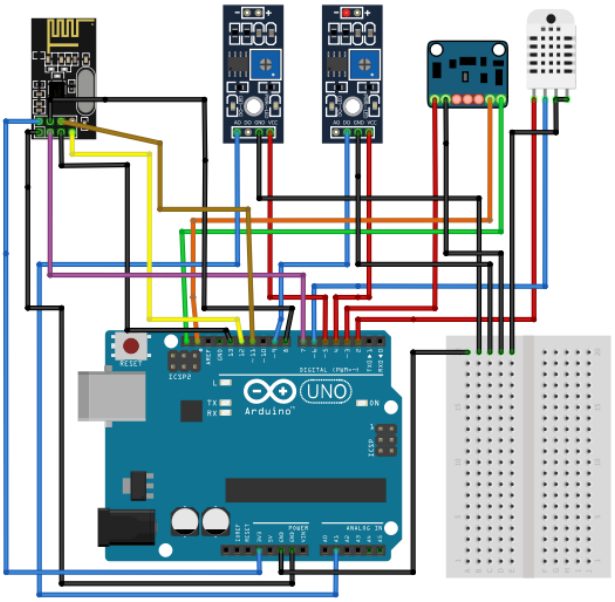


Figura 18: Conexionado del Esclavo Inalambrico

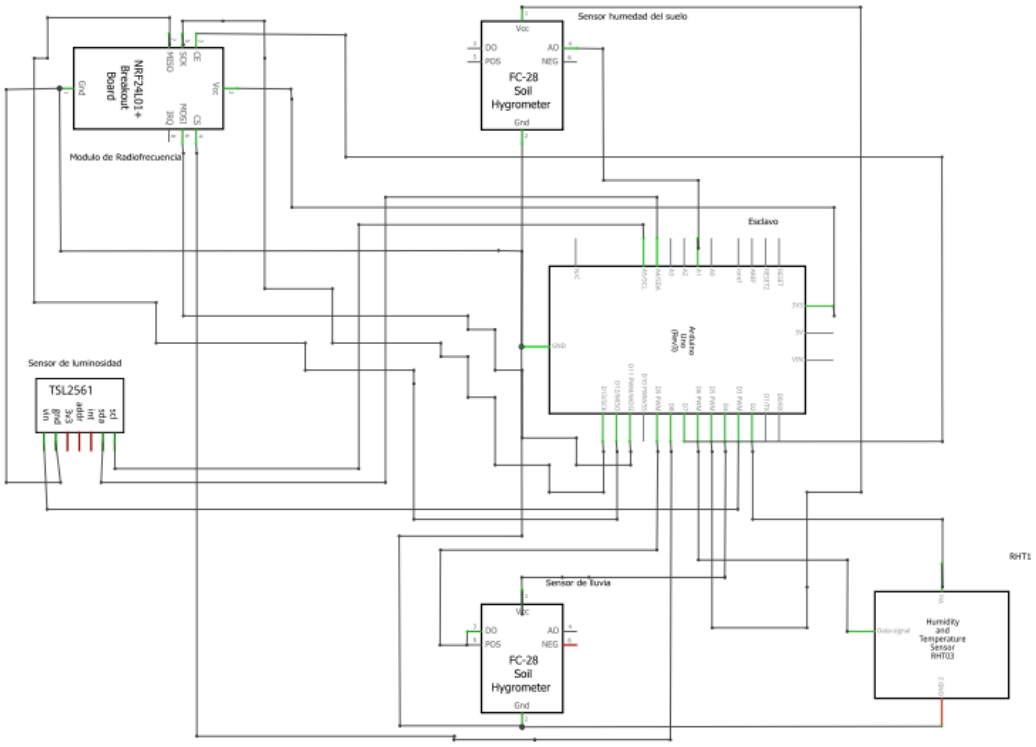


Figura 19: Esquema del Esclavo Inalámbrico

7.2. Anexo II: Códigos

Maestro por Serial

```
/*
Programa: Creamos programa de la maquina Maestro por Serial.
Autor: Samuel Estevez Rodriguez <samuesro@gmail.com>
Institucion: Universidad de La Laguna
Codigo de Dominio Pivado
*/

// LIBRERRIAS DEL PROGRAMA ------------------------------------------------------------
---
#include <SoftwareSerial.h>

// INICIALIZACIONES -------------------------------------------------------------------
---
SoftwareSerial mySerial(10, 11);                                        // RX, TX

// CONSTANTES DEL PROGRAMA ------------------------------------------------------------
---
const int TIEMPO_RIEGO = 10;                                             //en Minutos
const int HUMEDAD = 65;                                                  //%
const int TEMPERATURA_FRIO = 15;                                         //°C
const int TEMPERATURA_CALOR = 45;
const int HUMEDAD_SUELO = 600 ;                                          //Valor
analogico de suelo humedo. 0 en agua, a 1023 en aire
const int LUMINOSIDAD = 70;                                              // LUX

//Identificador de arduinos conectados al maestro-----------------------------------
---
int esclavosConectados = 0;

///////////////FUNCIONES/////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////////////
////////////////

//FUNCIONES PARA EL TRATAMIETO DE LA LISTA
DINAMICA//////////////////////////////////////

int* list;
size_t count;
size_t capacity;

// Crear una nueva lista
void CreateList(size_t _capacity)
{
  list = new int[_capacity];
  capacity = _capacity;
  count = 0;
}

// Reescalar
lista/////////////////////////////////////////////////////////////////
void resize(size_t newCapacity)
{
  int* newList = new int[newCapacity];
  memmove(newList, list, count  * sizeof(int));
  delete[] list;
  capacity = newCapacity;
  list = newList;
}

// Añadir elemento al final de la
lista////////////////////////////////////////////
void AddItem(int item)
{
  ++count;

  if (count > capacity)
  {
```

```
    size_t newSize = capacity * 2;
    resize(newSize);
  }

  list[count - 1] = item;
  resize(count);                                          //esto es para ajustar el
array y no pedir mas memoria que la utilizada
}

// Eliminar ultimo elemento de la lista//////////////////////////////////////////////
void RemoveTail()
{
  --count;
  resize(count);
}

// Muestra la lista por Serial para debug//////////////////////////////////////////////
void printList()
{
  Serial.print("Capacity:");
  Serial.print(capacity);

  Serial.print("  Count:");
  Serial.print(count);

  Serial.print("  Items:");
  for (size_t index = 0; index < count; index++)
  {
    Serial.print(list[index]);
    Serial.print(' ');
  }
  Serial.println();
}

//Añade un elemento nuevo, si hay ceros los ocupa y si no crea uno nuevo/////////////
int newId(){
  int ceros = 0;
  int idex[ceros];
  for (size_t index = 0; index < count; index++){
    if (list[index]== 0){
      idex[ceros]= index;
       Serial.println(idex[ceros]);
      ceros++;
    }
  }
  if(ceros == 0){
    AddItem(count+1);
    printList();
    return (count+1);
  }else{
    int idnew = idex[0] + 1 ;
    list[idex[0]]= idnew;
    printList();
    return idnew;
  }
}

//Elimina el elemento de la lista que le indiquemos//////////////////////////
void idCero(int id){
  list[id -1]= 0;
  printList();
}

//FUNCION DE LIMPIAR LOS 8 BITES DEL
BUFFER:////////////////////////////////////////////////
void limpiarBuffer (){
  for(int i=0; i<10; i++){
```

```
    mySerial.read();
  }
}


//FUNCION DE LIMPIAR LOS N BITES DEL
BUFFER:////////////////////////////////9/////////////
void limpiarBuffer(int bytes){
  for(int i= 0; i<bytes; i++){
    mySerial.read();
  }
}



//FUNCION LEER LOS 8 BITES DEL
BUFFER:////////////////////////////////////////////////
String Buffer(){
  String bufferleido = "";
  for(int i= 0; i<8; i++){
    bufferleido = bufferleido + mySerial.peek();
  }
  return bufferleido;
}


///////////////FUNCIONES/////////////////////////////////////////////////////////9///////
////////////////////////////////////////////////////////////////////////////////////////
/////////////////


//FUNCION DE CONFIRMACION DE
ACTUADOR/SENSORES////////////////////////////////9/////////////
 char Confirmacion(char cosa, int id){
  mySerial.write(id);
  Serial.println("id");
  while(!mySerial.available()){
    Serial.println("esperando R Act");
  }

  Serial.println("Buffer: " + Buffer());
  char confirmacion =  mySerial.read();
  limpiarBuffer(8);
  Serial.println(confirmacion);

  if(confirmacion == 'R'){
    Serial.println("recibo r");
    mySerial.write(cosa);
  }
  while(!mySerial.available()){
    Serial.println("esperando ");
    Serial.println(cosa);
  }

  Serial.println("Tenemos respuesta");
  Serial.println("Buffer: " + Buffer());
  confirmacion =  mySerial.read();
  limpiarBuffer(8);
  return confirmacion;
}


//FUNCION COMUNICACION PARA LECTURA DE VARIABLE SENSOR
ESCLAVO:////////////////////////////
float lectEsclavo(char sensor, int identificador){
  float dato;
  char confirmacion = Confirmacion('D', identificador);
```

```
  if(confirmacion == 'D'){
    Serial.println("recibo D");
    mySerial.write(sensor);
    while(!mySerial.available()){
      Serial.println("espero 2");
    }

    String lecturaDatos = "";
    lecturaDatos = mySerial.readString();
    //limpiarBuffer(8);
    Serial.println("buffer bruto : " + String(lecturaDatos));

    int primerCorchete = lecturaDatos.indexOf('[');
    int segundoCorchete = lecturaDatos.indexOf(']' , primerCorchete + 1 );
    lecturaDatos = lecturaDatos.substring(primerCorchete +1, segundoCorchete);

    limpiarBuffer(8);
    Serial.println("Buffer: " + Buffer());
    Serial.println("LA LECTURA FUE: " + lecturaDatos);
    dato = lecturaDatos.toFloat();

    return dato;
  }
  return -1;
}


//FUNCION COMUNICACION PARA ENVIO DE ACTUADOR
ESCLAVO:////////////////////////////////////
void comunActu(int timeAc, int identificador){

  char confirmacion =Confirmacion('A', identificador);

    if(confirmacion == 'A'){
      Serial.println("recibo A");
      mySerial.write(timeAc);                                                    ////
///////// WARNING:::::::::::::::::::::Aqui va el tiempo de riego no el
actuador//////////////////////
    }
}


//FUNCION ANALISIS DE
SENSORES//////////////////////////////////////////////////////////
int controlAct(int identificador){

  float luz = lectEsclavo( 'Z' , identificador);
  Serial.println("leemos si es de dia");
  delay (100);

  float lluvia = lectEsclavo( 'L' , identificador);
  Serial.println("leemos si esta lloviendo");
  delay (100);

  float humSul = lectEsclavo( 'S' , identificador);
  Serial.println("leemos humedad del suelo");
  delay (100);

  float humedad = lectEsclavo( 'H' , identificador);
  Serial.println("leemos humedad");
  delay (100);

  float temperatura = lectEsclavo( 'T' , identificador);
  Serial.println("leemos temperatura");
  delay (100);

  /* Si es de noche */
```

```
  if (luz <= LUMINOSIDAD){
    Serial.println("No se riega porque es de noche");
    return -1;
  }

  /* Si el suelo está húmedo */
  if(humSul <= HUMEDAD_SUELO){
    Serial.println("No se riega porque es el suelo está demasiado humedo");
    return -1;
  }

  /* Si la humedad del aire es alta */
  if(humedad >= HUMEDAD){
    Serial.println("No se riega porque tiene suficiente humedad");
    return -1;
  }

  /* Si la temperatura es muy alta, no se riega */
  if(temperatura >= TEMPERATURA_CALOR ){
    Serial.println("No se riega porque peligra el estado de la plantacion");
    return -1;
  }

  /* Si la temperatura es muy baja no se riega */
  if(temperatura <= TEMPERATURA_FRIO ){
    Serial.println("No se riega porque se puede congelar");
    return -1;
  }

  /* Si no esta lloviendo */
  if(lluvia == 0 ){
    Serial.println("Si no está lloviendo, se riega");
    return 1;
  }
  return -1;
}



//FUNCION INFORMACION SALIDA/ACTUADORES
ESCLAVO:////////////////////////////////////////
void enviarActuador(int tipOrden , int identificador){
 int stopRiego=0;
  switch (tipOrden) {
    /* Si la orden es 'no regar' */
    case -1:
      Serial.println("ERROR: Alguna variable contrarresta el riego");
      comunActu( stopRiego , identificador);
      break;

    /* Si la orden es 'regar' */
    case 1:
      Serial.println("OK: Se pide riego");
      comunActu( TIEMPO_RIEGO, identificador);
      break;

    default:
      Serial.println("ERROR: Valor de entrada nulo, no tienen funcion asignada");
  }
}

///////////////SETUP////////////////////////////////////////////////////////9//////////
////////////////////////////////////////////////////////////////////////////////////////
//////////////////

void setup() {
  pinMode(8,OUTPUT);
```

```
  mySerial.begin(9600);                                                    //
Comienzo de la comunicacion serie
  Serial.begin(9600);
  delay(1000);
}


///////////////LOOP/////////////////////////////////////////////////////////9////////
////////////////////////////////////////////////////////////////////////////////////
//////////////////

void loop() {
  //Creamos sincronizacion con el nuevo esclavo y lo añadimos
  Serial.println("Hay esclavos conectados?");
  if(mySerial.read()=='I'){
        Serial.println("Detectamos esclavo, lo agregamos");
        int nuevo=newId();
        esclavosConectados=count;
        Serial.println("Limpiamos el buffer");
        limpiarBuffer();

        Serial.println("Esperamos 5s para sincronizar.");
        delay (5000);
        Serial.print("Retornamos el numero de esclavo: ");
        Serial.println(nuevo);
        mySerial.write(nuevo);
  }


  for  (int i=1; i <= esclavosConectados; i++){
   int accion = controlAct(i);
   enviarActuador(accion , i);
  }


  delay(5000);
}
```

Maestro por Radiofrecuencia

```
/*
Programa: Creamos programa de la maquina Maestro para Radiofrecuencia.
Autor: Samuel Estevez Rodriguez <samuesro@gmail.com>
Institucion: Universidad de La Laguna
Codigo de Dominio Pivado
*/

// LIBRERRIAS DEL PROGRAMA ----------------------------------------
--------------------
#include <SoftwareSerial.h>

#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>


// CONSTANTES DEL PROGRAMA ----------------------------------------
--------------------
const int TIEMPO_RIEGO = 1;      //en Minutos
const int HUMEDAD = 65;           //%
const int TEMPERATURA_FRIO = 0;  //°C
const int TEMPERATURA_CALOR = 45;
const int HUMEDAD_SUELO = 600 ;       //Valor analogico de suelo
humedo. 0 en agua, a 1023 en aire
const int LUMINOSIDAD = 0;       // LUX

const int pinCE = 7;         // PINES ASOCIADOS A LA COMUNICACION
INALAMBRICA
const int pinCSN = 8;
RF24 radio(pinCE, pinCSN);   //Creamos instancia de la radio con sus
pines de control

#define idnuevo



// INICIALIZACIONES -----------------------------------------------
--------------------
// SoftwareSerial mySerial(10, 11);         // RX, TX

const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

//Identificador de arduinos conectados al maestro----------------------
--------------------
int esclavosConectados = 0;


int intentos[5];
//intentos[1]= 1;


///////////////SETUP//////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////

void setup() {
  //mySerial.begin(9600);              // Comienzo de la comunicacion
serie

  Serial.begin(9600);
```

```
  radio.begin();
  radio.setRetries(15,15);
  radio.openWritingPipe(pipes[0]);
  radio.openReadingPipe(1,pipes[1]);

  radio.startListening();

  delay(100);
}

//////////////FUNCIONES///////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////

//////////////FUNCIONES///////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////

//FUNCIONES PARA EL TRATAMIETO DE LA LISTA
DINAMICA//////////////////////////////////////

int* list;
size_t count;
size_t capacity;

// Crear una nueva lista
void CreateList(size_t _capacity)
{
  list = new int[_capacity];
  capacity = _capacity;
  count = 0;
}

// Reescalar
lista/////////////////////////////////////////////////////////////////
/////
void resize(size_t newCapacity)
{
  int* newList = new int[newCapacity];
  memmove(newList, list, count  * sizeof(int));
  delete[] list;
  capacity = newCapacity;
  list = newList;
}

// Añadir elemento al final de la
lista/////////////////////////////////////////////
void AddItem(int item)
{
  ++count;

  if (count > capacity)
  {
    size_t newSize = capacity * 2;
    resize(newSize);
  }

  list[count - 1] = item;
  resize(count); //esto es para ajustar el array y no pedir mas
memoria que la utilizada
```

```
}

// Eliminar ultimo elemento de la
lista////////////////////////////////////////////
void RemoveTail()
{
  --count;
  resize(count);
}

// Muestra la lista por Serial para
debug////////////////////////////////////////////
void printList()
{
  Serial.print("Capacity:");
  Serial.print(capacity);

  Serial.print("  Count:");
  Serial.print(count);

  Serial.print("  Items:");
  for (size_t index = 0; index < count; index++)
  {
    Serial.print(list[index]);
    Serial.print(' ');
  }
  Serial.println();
}

//Añade un elemento nuevo, si hay ceros los ocupa y si no crea uno
nuevo/////////////
int newId(){
  int ceros = 0;
  int idex[ceros];
  for (size_t index = 0; index < count; index++){
    if (list[index]== 0){
      idex[ceros]= index;
      Serial.println("Sustituimos a un arduino que se ha
desconectado");
      ceros++;
    }
  }
  if(ceros == 0){
    AddItem(count+1);
    printList();
    return (count);
  }else{
    int idnew = idex[0] + 1 ;
    list[idex[0]]= idnew;
    printList();
    return idnew;
  }
}

void idCero(int id){
  list[id -1]= 0;
  printList();
}

//////FUNCIONES PARA LA COMUNICACION
INALAMBRICA/////////////////////////////////////////
```

```
//fUNCIONES DE ENVIO DE MENSAJE
////////////////////////////////////////////////////////
void envio(char letra){
  radio.stopListening();
  char msg1[2];
  msg1[0]= letra;
  radio.write(msg1, 1); //Enviamos el mensajemensaje
  radio.startListening(); //Volvemos a la escucha

}

void envio(int lectura){
  radio.stopListening();
  uint8_t Envio[2]; //Dos elementos
  Envio[0] = (lectura >> 8); // Mueve hacia la derecha
  Envio[1] = (lectura & 0xFF); //AND (producto) con 255 (número más
pequeño que puede representar)
  radio.write((uint8_t *)Envio, 2); //Enviamo los dos elementos
  radio.startListening(); //Volvemos a la escucha

}

// FUNCIONES DE LECTURA DE
MENSAJE////////////////////////////////////////////////////
char leoLetr(){
    if (radio.available()) { //Si hay datos disponibles
     // Serial.println("Leemos si nos llega alguna letra como
respuesta");
    uint8_t msj_maestro[2];
    radio.read(msj_maestro, sizeof(msj_maestro));
    char recibido= (char)msj_maestro[0];
    return recibido;
  }
}

int leoNum(){ //Leo el mensaje y miro si es superior o inferior a los
bytes esperados
  uint8_t dato_temp[2];//Dos elementos
  int result;
  if (radio.available()) { //Si hay datos disponibles
    radio.read(dato_temp, 2);
    result = dato_temp[0] << 8 | dato_temp[1];
    return result;
  }
}

//FUNCION DE CONFIRMACION DE
ACTUADOR/SENSORES////////////////////////////////////////////
char Confirmacion(char cosa, int id){
  char confirmacion;
//Envio identificador
  Serial.print("Esperando Respuesta al id: ");
  Serial.println(id);
  envio(id);

  int reenvios = 0;
  long tiempo_actual = millis();
  while ((reenvios <= 5)){

    if(radio.available()) {
```

```
      uint8_t msj_maestro[2];
      radio.read(msj_maestro, sizeof(msj_maestro));
      confirmacion = (char)msj_maestro[0];
      Serial.print("Respuesta a id: ");
      Serial.println(confirmacion);
      break;
    }
    if ((reenvios < 5) && ( millis() - tiempo_actual > 200)){
      reenvios++;
      tiempo_actual = millis();
      envio(id);
      Serial.print("Intentando conectar ");
      Serial.println(reenvios);
    }
    if (reenvios >= 5) {
      Serial.println("Se ha supeado el numero de intentos");
      Serial.print("No se ha conectado ");
      Serial.print(id);
      Serial.println(" en envio de id");
//      idCero(id);
      return 'E';
    }
  }
  if(confirmacion == 'R'){
   Serial.print("Recibo r: ");
   Serial.println("Esperando respuesta a act/sen ");
   envio(cosa);
   int reenvios2 = 0;
   tiempo_actual = millis();
   int RR;
   while ((reenvios2 <= 5)){

    if(radio.available()){
      uint8_t msj_maestro[2];
      radio.read(msj_maestro, sizeof(msj_maestro));
      confirmacion = (char)msj_maestro[0];

      if(confirmacion == 'R'){
         Serial.println("Se sigue recibiendo R");
         RR++;
         if(RR == 4){
           return 'E' ;
         }
      }else{
        Serial.print("Respuesta a tipo: ");
        Serial.println(confirmacion);
       // reenvios = 0;
        break;
      }
    }

    if((reenvios2 < 5) && ( millis() - tiempo_actual > 200)){
      reenvios2++;
      tiempo_actual = millis();
      envio(cosa);
      Serial.print("Intentando conectar ");
      Serial.println(reenvios2);
    }if (reenvios >= 5) {
      Serial.println("Se ha supeado el numero de intentos");
      Serial.print("No se ha conectado ");
      Serial.print(id);
```

```
      Serial.println(" en envio de act/sen");
//      idCero(id);
      return 'E';
    }
   }
   return confirmacion;
  }else{
    Serial.print("No se ha conectado ");
    Serial.print(id);
    Serial.println(" en recepcion de R");
//    idCero(id);
    return 'E';
  }
 }


//FUNCION COMUNICACION PARA LECTURA DE VARIABLE SENSOR
ESCLAVO:////////////////////////////
float lectEsclavo(char sensor, int identificador){
  int dato;

  char confirmacion = Confirmacion('D', identificador);
  if(confirmacion == 'D'){
    Serial.print("Recibo D: ");
    Serial.println("Esperando dato sensor");
    envio(sensor);
    long tiempo_actual = millis();
    int reenvios = 0;

    while ((reenvios <= 5)){
      if(radio.available()) {
        uint8_t dato_temp[2];//Dos elementos
        radio.read(dato_temp, 2);
        Serial.print("Primer byte: ");
        Serial.println(dato_temp[0]);
        Serial.print("Segundo byte: ");
        Serial.println(dato_temp[1]);
        dato = dato_temp[0] << 8 | dato_temp[1];
        Serial.print("Respuesta del sensor ");
        Serial.print(sensor);
        Serial.print(" :");
        Serial.println(dato);
       // reenvios = 0;
        break;
      }
      if((reenvios < 5)&& ( millis() - tiempo_actual > 500)){
        reenvios++;
        tiempo_actual = millis();
        envio(sensor);
        Serial.print("Intentando conectar ");
        Serial.println(reenvios);
      }if (reenvios >= 5){
        Serial.println("Se ha supeado el numero de intentos");
        Serial.print("No se ha conectado ");
        Serial.print(identificador);
        Serial.println(" en envio de sensor");
//        idCero(identificador);
        return -2;
      }
    }
    return dato;
```

```
    }else{
        Serial.print("No se ha conectado ");
        Serial.print(identificador);
        Serial.println(" en recepcion de D");
//        idCero(identificador);
        return -2;
    }
}


//FUNCION COMUNICACION PARA ENVIO DE ACTUADOR
ESCLAVO:////////////////////////////////////
void comunActu(int timeAc, int identificador){
  char confirmacion =Confirmacion('A', identificador);
  if(confirmacion == 'A'){
    Serial.println("recibo A");
    envio(timeAc);
  }else{
   Serial.println("No se ha recibido peticion de tiempo");
  }
}


//FUNCION ANALISIS DE
SENSORES//////////////////////////////////////////////////////////
int controlAct(int identificador){

  Serial.println("leemos si es de dia");
  float luz = lectEsclavo( 'Z' , identificador);
  if(luz == -2){
    return -2;
  }
  delay (50);

  Serial.println("leemos si esta lloviendo");
  float lluvia = lectEsclavo( 'L' , identificador);
  if(lluvia == -2){
    return -2;
  }
  delay (50);

  Serial.println("leemos humedad del suelo");
  float humSul = lectEsclavo( 'S' , identificador);
  if(humSul == -2){
    return -2;
  }
  delay (50);

  Serial.println("leemos humedad");
  float humedad = lectEsclavo( 'H' , identificador);
  if(humedad == -2){
    return -2;
 }
  delay (50);

  Serial.println("leemos temperatura");
  float temperatura = lectEsclavo( 'T' , identificador);
  if(temperatura == -2){
    return -2;
  }
```

```
  delay (50);

  /* Si es de noche */
  if (luz <= LUMINOSIDAD){
    Serial.println("No se riega porque es de noche");
    return -1;
  }

  /* Si el suelo está húmedo */
  if(humSul <= HUMEDAD_SUELO){
    Serial.println("No se riega porque es el suelo está humedo");
    return -1;
  }

  /* Si la humedad del aire es alta */
  if(humedad >= HUMEDAD){
    Serial.println("No se riega porque es el aire está humedo");
    return -1;
  }

  /* Si la temperatura es muy alta, no se riega */
  if(temperatura >= TEMPERATURA_CALOR ){
    Serial.println("No se riega porque hay alta temperatura");
    return -1;
  }

  /* Si la temperatura es muy baja no se riega */
  if(temperatura <= TEMPERATURA_FRIO ){
    Serial.println("No se riega porque hay baja temperatura");
    return -1;
  }

  /* Si no esta lloviendo */
  if(lluvia == 0 ){
    Serial.println("Si no está lloviendo, se riega");
    return 1;
  }

  return -1;
}


//FUNCION INFORMACION SALIDA/ACTUADORES
ESCLAVO:////////////////////////////////////////
void enviarActuador(int tipOrden , int identificador){
 int stopRiego=2;
  switch (tipOrden) {
    /* Si la orden es 'no regar' */

    case -2:
      Serial.print("ERROR: En algun momento se perdio la comunicacion
con el esclavo: ");
      Serial.println(identificador);
      intentos[identificador]++;
      if(intentos[identificador] == 5){
        Serial.println(" Se ha intentado conectar mas de 5 veces. Lo
damos de baja.");
        idCero(identificador);
      }
      break;
```

```
    case -1:
      Serial.println("ERROR: Alguna variable contrarresta el riego");
      comunActu( stopRiego , identificador);
      break;

    /* Si la orden es 'regar' */
    case 1:
      Serial.println("OK: Se pide riego");
      comunActu( TIEMPO_RIEGO, identificador);
      break;

    default:
      Serial.println("ERROR: Valor de entrada nulo, no tienen funcion
asignada");
  }
}


///////////////LOOP/////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////

void loop() {
  bool waitForSlaves = false;
  int reen = 0;
  uint8_t msj_maestro[2];
  char confirm = 'P';
  long tiempo_actual = millis();

  for  (int i=1; i <= esclavosConectados; i++){
    Serial.println("Ejecutamos programa de lectura y control de
valvula ");
    if(!list[i-1]==0){
      int accion = controlAct(i);
      enviarActuador(accion , i);
    }else{
       Serial.println("Este esclavo esta desconectado");
    }
  }

  waitForSlaves = true;
  delay(100);


   //Creamos sincronizacion con el nuevo esclavo y lo añadimos
  Serial.println("Hay esclavos conectados?");

  while ((reen <= 10)&&(waitForSlaves == true)){
    if ((reen < 10) && ( millis() - tiempo_actual > 500)){
      reen++;
      tiempo_actual = millis();
      envio('I');
      Serial.print("Vemos si hay esclavos ");
      Serial.println(reen);
    }

    if (reen >= 10){
      Serial.println("No hay esclavos a la espera");
      break ;
    }
```

```
    if(radio.available()){
      radio.read(msj_maestro, sizeof(msj_maestro));
      confirm = (char)msj_maestro[0];
      Serial.print("Algun esclavo respondio: ");
      Serial.println(confirm);
    }

    if(confirm == 'I'){
      Serial.println("Detectamos esclavo, lo agregamos");
      int nuevo = newId();
      esclavosConectados=count;
      Serial.print("Retornamos el numero de esclavo: ");
      Serial.println(nuevo);
      envio( nuevo);
      intentos[nuevo]= 0;
      break;
    }


  }


}
```

Esclavo por Serial

```
/*
Programa: Creamos programa de la maquina Esclavo para Serial.
Autor: Samuel Estevez Rodriguez <samuesro@gmail.com>
Institucion: Universidad de La Laguna
Codigo de Dominio Pivado
*/

// LIBRERRIAS DEL PROGRAMA ------------------------------------------------------------
---
#include <SoftwareSerial.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2561_U.h>
#include "DHT.h"

// CONSTANTES DEL PROGRAMA ------------------------------------------------------------
---
const String GANANCIA_SENSOR_LUZ = "AUTO";       // 1, 16, AUTO
const int TIEMPO_INTEGRACION_SENSOR_LUZ = 101;   // 101, 13, 402

              /*
               * ------------------------------------------------------------------
               * No gain ... use in bright light to avoid sensor saturation:
               * tsl.setGain(TSL2561_GAIN_1X);
               * 16x gain ... use in low light to boost sensitivity:
               * tsl.setGain(TSL2561_GAIN_16X);
               * Auto-gain ... switches automatically between 1x and 16x:
               * tsl.enableAutoRange(true);
               * ------------------------------------------------------------------
               * Changing the integration time gives you better sensor resolution
               * (402ms = 16-bit data)
               * ------------------------------------------------------------------
               * Medium resolution and speed:
               * tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_101MS)
               * 16-bit data but slowest conversions:
               * tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_402MS);
               * Fast but low resolution:
               * tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_13MS);
               */

#define DHTPIN 2
#define DHTTYPE DHT22         // DHT 22  (AM2302)

 int tiempoRiego;
 long tiempoPrevio;

//Identificador del arduino, asignada por el maestro
int identificador = -1;

//Declaracion de Pines y Variables asociadas-----------------------------------------
----
const int sensorHS = 6;                                                          //
Variable asociada al pin de entrada del sensor HS (6)
int valorHS = 0;                                                                 //
Variable para el almacenamiento del sensor HS
const int sensorLL = 4;                                                          //
Variable asociada al pin de entrada del sensor LL (4)
int valorLL = 0;                                                                 //
Variable para el almacenamiento del sensor LL
// Actuadores virtuales
int rojo = 9;
int valvulaDig = 8;

// INICIALIZACIONES DE SENSORES-------------------------------------------------------
--------------
Adafruit_TSL2561_Unified tsl = Adafruit_TSL2561_Unified(TSL2561_ADDR_FLOAT, 12345);
DHT dht(DHTPIN, DHTTYPE);     // LED pins
```

```
// Declaracion de un nuevo puerto para la comunicacion serie.
SoftwareSerial mySerial(10, 11);    // RX, TX.



//////////////FUNCIONES//////////////////////////////////////////////////////////////
/

void limpiarBuffer (){
  for(int i=0; i<8; i++){
    //Serial.print(serialLimpiar.peek());
    mySerial.read();
  }
}



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

void setup() {

  analogReference(INTERNAL);

  // Configuramos el puerto serial a 9600 bps.
  mySerial.begin(9600); // Comienzo de la comunicaciÃ³n serie.
  Serial.begin(9600);

  dht.begin();

  Serial.println("Soy el esclavo: mando 'I' para sumarme al sistema");
  mySerial.write('I');
  Serial.println("Limpio la memoria...");
  limpiarBuffer();
  mySerial.listen();
  Serial.println("Espero 10s ...");
  delay (10000);

  //Escucho para si me esta enviando identificador y lo muestro por pantalla.
  if(mySerial.peek() > 0){
      Serial.println("Hay dato...");
      Serial.println("Se asigna como identificador...");
      identificador = mySerial.read() ;
      Serial.print("El numero asignado es: ");
      Serial.println(identificador);
      delay (1000);
  }

  //Defino los pines de los sensores como salida y primer nivel bajo.
  pinMode(sensorHS, INPUT);  //definir pin HS como entrada
  pinMode(sensorLL, INPUT);  //definir pin LL como entrada



  //Declaro los pines de actuadores como salida y primer nivel bajo.
  pinMode(rojo, OUTPUT);
  pinMode(valvulaDig, OUTPUT);
  digitalWrite(rojo, LOW);
  digitalWrite(valvulaDig, LOW);

  delay(1000);                  // Retardo para establecer configuracion del setup.
}

//////////////FUNCIONES//////////////////////////////////////////////////////////////
/
```

```
String Buffer(){
  String bufferleido = "";
  for(int i= 0; i<8; i++){
    bufferleido = bufferleido + mySerial.peek();
  }
  return bufferleido;
}



void loop() {

  // Si hay datos disponibles
  if (mySerial.available()>0){
    Serial.println("Buffer: " + Buffer());
    // Leerlos
    int id = mySerial.read();
    limpiarBuffer();
    Serial.println("id: " + String(id));

    // Si la informacion no es para mi, salgo
    if(id = identificador){

      // Confirmacion al maestro de que lo escuché
      Serial.println("Escribo R");
      mySerial.write('R');              //Confirmo que lo recibi
      while(!mySerial.available()){
        Serial.println("esperando dato");
      }
      // Leo la petición que me manda el maestro
      Serial.println("dato recibido");
      Serial.println("Buffer: " + Buffer());
      char mensaje = mySerial.read();
      limpiarBuffer();
      Serial.println("mensaje: " + String(mensaje));



      // Procesamos la petición
      switch (mensaje) {
        case 'D':       // Si quiere algo de los sensores
          // Le envío confirmación al maestro
          Serial.println("Escribo D");
          mySerial.write('D');

          // Espero una órden del maestro
          while(!mySerial.available()){
            Serial.println("esperando dato");
          }


          // Leo dato recibido
          Serial.println("dato recibido");
          Serial.println("Buffer: " + Buffer());
          mensaje = mySerial.read();
          limpiarBuffer();              //Leo que sensor quiere
          Serial.println("mensaje: " + String(mensaje));

          switch(mensaje){

            case 'S':
              //Lectura del sensor humedad del suelo
              valorHS = analogRead(sensorHS );  //lectura analogica de sensor HS
              mySerial.print("[" + String(valorHS) + "]");
              return;
```

```
    case 'L':
      //Lectura del sensor de lluvia
      valorLL = digitalRead(sensorLL );  //lectura digital de sensor LL
      if (valorLL == LOW)
        mySerial.print("[" + (String)1 + "]");
      else
        mySerial.print("[" +(String)0 + "]");
      return;

    case 'H':
      //Lectura del sensor de Humedad
      float h;
      h = dht.readHumidity();
      mySerial.print("[" + String(h) + "]");
      return;


    case 'T':
      //Lectura del sensor de Temperatura
      float t;
      t = dht.readTemperature();
      mySerial.print("[" + String(t) + "]");
      return;

    case 'Z':
      //Lectura del sensor de Luminosidad
      sensor_t sensor;
      tsl.getSensor(&sensor);

      // Establece la ganancia deseada de acuerdo a la constante
      //GANANCIA_SENSOR_LUZ
      if (GANANCIA_SENSOR_LUZ == "AUTO")
        tsl.enableAutoRange(true);
      if (GANANCIA_SENSOR_LUZ == "16")
        tsl.setGain(TSL2561_GAIN_16X);
      if (GANANCIA_SENSOR_LUZ == "1")
        tsl.setGain(TSL2561_GAIN_1X);

      // Establece el tiempo de integración deseado de acuerdo a
      // la constante TIEMPO_INTEGRACION_SENSOR_LUZ
      if (TIEMPO_INTEGRACION_SENSOR_LUZ == 101)
        tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_101MS);
      if (TIEMPO_INTEGRACION_SENSOR_LUZ == 402)
        tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_402MS);
      if (TIEMPO_INTEGRACION_SENSOR_LUZ == 13)
        tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_13MS);

      sensors_event_t event;
      tsl.getEvent(&event);

      mySerial.print("[" + String(event.light) + "]");
      Serial.println("TOTAL A ENVIAR: [" + String(event.light) + "]");
      return;
  }//Fin del segundo switch

  break;
case 'A':      // Si quiere algo de los actuadores

  // Le mando un mensaje de 'recibido'
  Serial.println("Escribo A");
  mySerial.write('A');

  // Espero la órden del maestro
  while(!mySerial.available()){
    Serial.println("esperando dato");
```

```
            }


            // Leo la órden del maestro
            Serial.println("dato recibido");
            Serial.println("Buffer: " + Buffer());
            tiempoRiego = mySerial.read(); //Recibo tiempo en minutos
            tiempoPrevio= millis();
            limpiarBuffer();
            Serial.println("mensaje: " + String(tiempoRiego));
            break;
        } //Fin del primer switch
      }
    }

    if(tiempoRiego > 0){
      Actuador1(tiempoPrevio,tiempoRiego);
    }else{
      Serial.println("No se ha pedido riego");
      digitalWrite(valvulaDig, LOW);
    }

}// Fin de la funcion LOOP

void Actuador1(long previo,int timeR){

    long RiegoMSeg = timeR*60*1000;
    unsigned long timeActual = millis();

     if(timeActual - previo <= RiegoMSeg) {
       Serial.println("Se esta Regando en este momento");
       digitalWrite(valvulaDig, HIGH);
     }else{
       Serial.println("Riego finalizado");
       digitalWrite(valvulaDig, LOW);
     }
}
```

Esclavo por Radiofrecuencia

```
/*
Programa: Creamos programa de la maquina Esclavo por Radiofrecuencia.
Autor: Samuel Estevez Rodriguez <samuesro@gmail.com>
Institucion: Universidad de La Laguna
Codigo de Dominio Pivado
*/

// LIBRERRIAS DEL PROGRAMA ------------------------------------------
--------------------
#include <SoftwareSerial.h>

#include <LowPower.h>


#include <Wire.h>

#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2561_U.h>

#include "DHT.h"

#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>

#include <EEPROM.h>

// CONSTANTES DEL PROGRAMA ------------------------------------------
--------------------
const String GANANCIA_SENSOR_LUZ = "AUTO"; // 1, 16, AUTO
const int TIEMPO_INTEGRACION_SENSOR_LUZ = 101;  // 101, 13, 402
const int TIEMPO_RIEGO = 2;


#define DHTPIN 6
#define DHTTYPE DHT22         // DHT 22  (AM2302)

//Declaracion de Pines y Variables asociadas para Comunicacion Inalam-
-----------------------------------------------
const int pinCE = 7;
const int pinCSN = 8;
RF24 radio(pinCE, pinCSN); //Creamos instancia de la radio con sus
pines de control
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };



//Declaracion de Pines y Variables asociadas------------------------
---------------------

const int sensorHS = 1;       // Variable asociada al pin de entrada
del sensor HS (6)
int valorHS = 0;              // Variable para el almacenamiento del
sensor HS
const int sensorLL = 9;       // Variable asociada al pin de entrada
del sensor LL (9)
int valorLL = 0;              // Variable para el almacenamiento del
sensor LL
```

```
//Asignacion de pines para alimentacion seccionada--------------------
---------------------

const int ali_HS  = 5 ;
const int ali_LL  = 4 ;
const int ali_LX  = 3 ;
const int ali_HT  = 2 ;
const int ali_CO  = 1 ;

// Actuadores
int valvulaDig = 10;

//Identificador del arduino, asignada por el maestro------------------
----------------------
int identificador = -1;       // Variabe para almacenar el
identificador

// INICIALIZACIONES ----------------------------------------------------
--------------------
Adafruit_TSL2561_Unified tsl =
Adafruit_TSL2561_Unified(TSL2561_ADDR_FLOAT, 12345);
DHT dht(DHTPIN, DHTTYPE);     // LED pins

int luz;                  // Variable para lectura de sensor de luz
long tiempoRiego ;
long tiempoPrevio ;
bool comu;

int timesleep  ;          //Tiempo de sleep que tendra nuestro esclavo
en segundos

int Direccion = 0;         // Direccon de memoria de almacenamiento en
la eeprom
byte IdByte;               // Byte de lectura de la memoria eeprom


 long time_ree;
 long timecom ;

 uint8_t msje_maestro[2];
 char mensaje;
 uint8_t dato_temp[2];

 bool regando;
 int riego;




//fUNCIONES DE ENVIO DE MENSAJE
////////////////////////////////////////////////////////
void envio(char letra){
  radio.stopListening();
  char msg1[2];
  msg1[0]= letra;
  radio.write(msg1, 1); //Enviamos el mensaje
  radio.startListening(); //Volvemos a la escucha
}


void envio(int lectura){
```

```
  radio.stopListening();
  uint8_t Envio[2]; //Dos elementos
  Envio[0] = (lectura >> 8); // Mueve hacia la derecha
  Envio[1] = (lectura & 0xFF); //AND (producto) con 255 (número más
pequeño que puede representar)
  radio.write((uint8_t *)Envio, 2); //Enviamo los dos elementos
  radio.startListening(); //Volvemos a la escucha
}


// FUNCIONES DE LECTURA DE
MENSAJE//////////////////////////////////////////////////////
char leoLetr(){
    if (radio.available()) { //Si hay datos disponibles
    uint8_t msj_maestro[2];
    radio.read(msj_maestro, sizeof(msj_maestro));
    char recibido= (char)msj_maestro[0];
    return recibido;
  }
}

int leoNum(){
  uint8_t dato_temp[2];//Dos elementos
  int result;
  if (radio.available()) { //Si hay datos disponibles
    radio.read(dato_temp, 2);
    result = dato_temp[0] << 8 | dato_temp[1];
    return result;
  }
  return -1;
}


///fUNCIONES DE COMPROBACION DE
MENSAJE/////////////////////////////////////////////////

int comproba_maestro(char lectura){
int result = 0 ;
 int reenvios = 0;
 uint8_t dato_temp[2];
 bool waiting = true;
 char recibido = 'P';
 uint8_t msj_maestro[2];

 while(1){
  Serial.print("Esperando al maestro.");
  Serial.println(recibido);
  if(radio.available()){
     radio.read(msj_maestro, sizeof(msj_maestro));
     recibido = (char)msj_maestro[0];
     Serial.print("Esperando nuevo mensaje:  ");
     Serial.println(recibido);
   }

   if(recibido == 'I'){
      envio(lectura);
      long tiempo_actual = millis();
      while ((reenvios <= 5)){
        if ((reenvios < 5) && ( millis() - tiempo_actual > 200)){
          reenvios++;
          tiempo_actual = millis();
```

```
          envio(lectura);
          Serial.print("Esperando a que el maestro nos asigne id. ");
          Serial.println(reenvios);
        }
        if (reenvios >= 5) {
          Serial.println("Se ha superado el numero de intentos. ");
          Serial.print("No se ha podido conectar al maestro.
Reiniciar.");
          break ;
        }
        if(radio.available()) {
          radio.read(dato_temp, 2);
          result = dato_temp[0] << 8 | dato_temp[1];
          Serial.print("Recibimos id: ");
          Serial.println(result);
          return result;
        }
      }
    }
  }
}


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

void setup() {

  analogReference(INTERNAL);
  Serial.begin(9600);

  radio.begin();
  radio.setRetries(15,15);
  radio.openWritingPipe(pipes[1]);
  radio.openReadingPipe(1,pipes[0]);
  radio.startListening();

  dht.begin();

  //Mando una I para sumarme.
  Serial.println("Esperando respuesta del maestro");
  while (!(identificador = comproba_maestro('I')))
    delay(50);

  Serial.print("recibimos id: ");
  Serial.println(identificador);
  comu = true;

  // Guardamos el valor del identificador en la memoria eeprom del
arduino para tenerlo despues
  // del modo sleep

  EEPROM.write(Direccion, identificador);


  //Defino los pines de los sensores como salida y primer nivel bajo.
  pinMode(sensorHS, INPUT);  //definir pin HS como entrada
  pinMode(sensorLL, INPUT);  //definir pin LL como entrada
```

```
  //Declaro los pines de actuadores y alimentacion como salida y
primer nivel bajo.
  pinMode(valvulaDig, OUTPUT);
  digitalWrite(valvulaDig, LOW);
//--------------------------------------------------------------------
-------------

  pinMode(ali_HS, OUTPUT);
  digitalWrite(ali_HS, LOW);

  pinMode(ali_LL, OUTPUT);
  digitalWrite(ali_LL, LOW);

  pinMode(ali_LX, OUTPUT);
  digitalWrite(ali_LX, LOW);

  pinMode(ali_HT, OUTPUT);
  digitalWrite(ali_HT, LOW);

  pinMode(ali_CO, OUTPUT);
  digitalWrite(ali_CO, LOW);

  delay(10);                    // Retardo para establecer
configuracion del setup.
}

///////////////loop/////////////////////////////////////////////////////
/////////////

void loop() {

   IdByte = EEPROM.read(Direccion);

  int SS = leoNum();
  Serial.println(SS);
  // Si hay datos disponibles
  if (SS == IdByte){
    Serial.println("Recibo mi identificador");
    // Confirmacion al maestro de que lo escuché
    Serial.println("Escribo R");
    envio('R'); //Confirmo que lo recibi
    delay (10);

    //ciclos = 0;


    timecom = millis();
    time_ree=millis();

    while(1){
      if (radio.available()) {
        radio.read(msje_maestro, sizeof(msje_maestro));
        mensaje= (char)msje_maestro[0];
        Serial.print("mensaje recibido para tipo: ");
         Serial.println(mensaje);
        break;
      }
//          Serial.println("Escribo D de estar en sensores");
//

      if (millis()-time_ree > 1000) {
```

```
        Serial.println("Reenvio R de confirmacion");
        time_ree=millis();
        envio('R');
      }
      if(millis()-timecom > 10000){
        comu = false;
        while(comu == false){
          Serial.println("Se ha perdido comunicacion, se solicita
reseteo del esclavo");
        }
      }
    }


    // Procesamos la petición
    switch (mensaje) {


      case 'D':       // Si quiere algo de los sensores

        // Le envío confirmación al maestro
        Serial.println("Escribo D de estar en sensores");
        envio('D');
        delay (10);

        // Espero una órden del maestro
        timecom = millis();
        time_ree=millis();

        while(1){
          if (radio.available()) {
            radio.read(msje_maestro, sizeof(msje_maestro));
            mensaje= (char)msje_maestro[0];
            Serial.print("mensaje recibido para sensor: ");
             Serial.println(mensaje);
            break;
          }

          if (millis()-time_ree > 1000) {
            Serial.println("Reenvio D para sensores");
            time_ree=millis();
            envio('D');
          }
          if(millis()-timecom > 10000){
            comu = false;
            while(comu == false){
              Serial.println("Se ha perdido comunicacion, se solicita
reseteo del esclavo");
            }
          }
        }


        // Leo dato recibido
//        mensaje = leoLetr();


        switch(mensaje){

          case 'S':
            //Lectura del sensor humedad del suelo
```

```
            digitalWrite(ali_HS, HIGH);
            delay(500);
            valorHS = analogRead(sensorHS );  //lectura analogica de
sensor HS
            Serial.println("TOTAL A ENVIAR: " + String(valorHS));
            envio((int)valorHS);
            digitalWrite(ali_HS, LOW);
            return;


          case 'L':
            //Lectura del sensor de lluvia
            digitalWrite(ali_LL, HIGH);
            delay(500);
            valorLL = digitalRead(sensorLL );  //lectura digital de
sensor LL
            if (valorLL == LOW){
              Serial.println("TOTAL A ENVIAR: " + (String)1);
              envio(1);
              digitalWrite(ali_LL, LOW);
            }else{
              Serial.println("TOTAL A ENVIAR: " +(String)0 );
              envio(0);
              digitalWrite(ali_LL, LOW);
            }
            return;


          case 'Z':
            //Lectura del sensor de Luminosidad


            /*
             * ------------------------------------------------------
----------
             * No gain ... use in bright light to avoid sensor
saturation:
             * tsl.setGain(TSL2561_GAIN_1X);
             * 16x gain ... use in low light to boost sensitivity:
             * tsl.setGain(TSL2561_GAIN_16X);
             * Auto-gain ... switches automatically between 1x and
16x:
             * tsl.enableAutoRange(true);
             * ------------------------------------------------------
----------
             * Changing the integration time gives you better sensor
resolution
             * (402ms = 16-bit data)
             * ------------------------------------------------------
-----------
             * Medium resolution and speed:
             * tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_101MS)
             * 16-bit data but slowest conversions:
             * tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_402MS);
             * Fast but low resolution:
             * tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_13MS);
             */
            digitalWrite(ali_LX, HIGH);
            delay(500);
            sensor_t sensor;
            tsl.getSensor(&sensor);
```

```
            // Establece la ganancia deseada de acuerdo a la constante
GANANCIA_SENSOR_LUZ
            if (GANANCIA_SENSOR_LUZ == "AUTO")
              tsl.enableAutoRange(true);

            if (GANANCIA_SENSOR_LUZ == "16")
              tsl.setGain(TSL2561_GAIN_16X);

            if (GANANCIA_SENSOR_LUZ == "1")
              tsl.setGain(TSL2561_GAIN_1X);


            // Establece el tiempo de integración deseado de acuerdo a
            // la constante TIEMPO_INTEGRACION_SENSOR_LUZ
            if (TIEMPO_INTEGRACION_SENSOR_LUZ == 101){
              tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_101MS);
            }
            if (TIEMPO_INTEGRACION_SENSOR_LUZ == 402)
              tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_402MS);

            if (TIEMPO_INTEGRACION_SENSOR_LUZ == 13)
              tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_13MS);


            sensors_event_t event;
            tsl.getEvent(&event);
            luz= (int)event.light;
            envio(luz);
            Serial.println("TOTAL A ENVIAR: " + String(event.light) );
            digitalWrite(ali_LX, LOW);
            return;


          case 'H':
            //Lectura del sensor de Humedad
              digitalWrite(ali_HT, HIGH);
              delay(500);
            float h;
            h = dht.readHumidity();
            Serial.println("TOTAL A ENVIAR: " + String(h));
            envio((int)h);
            digitalWrite(ali_HT, LOW);
            return;


          case 'T':
            //Lectura del sensor de Temperatura
            digitalWrite(ali_HT, HIGH);
            delay(500);
            float t;
            t = dht.readTemperature();
            Serial.println("TOTAL A ENVIAR:" + String(t));
            envio((int)t);
            digitalWrite(ali_HT, LOW);
            return;
        }

        break;
```

```
      case 'A':      // Si quiere algo de los actuadores

        // Le mando un mensaje de 'recibido'
        Serial.println("Escribo A");
        envio('A');
        Serial.println("esperando dato para actuador");

       if(regando == false){

          tiempoPrevio = millis();
          regando = true;
          break;
       }else{
        Serial.println("Ya se esta regando, obviamos la peticion");
        break;
       }
    }
  }

  if(regando == true){
    riego = Actuador1(tiempoPrevio,TIEMPO_RIEGO);
      if(riego == 0 ){
        regando = false;
      //  digitalWrite(valvulaDig, LOW);
     //   sleep(timesleep);

      }
   }else{
    Serial.println("No se ha pedido riego");
  }
}


int Actuador1(long previo,long timeR){

  Serial.println("Datos de la funcion");
  Serial.println(previo);
  Serial.println(timeR);

  unsigned long RiegoMSeg;
  RiegoMSeg = timeR*60*1000;

  Serial.println(RiegoMSeg);
  unsigned long timeActual = millis();
  Serial.println(timeActual);
  long resta;
  resta = (timeActual - previo) ;
  Serial.println(resta);

   if((resta) <= RiegoMSeg) {
     Serial.println("Se esta Regando en este momento");
     digitalWrite(valvulaDig, HIGH);
     return 1;
   }
   Serial.println("Riego finalizado");
   digitalWrite(valvulaDig, LOW);
   return 0;
}
```

```
void sleep(int sec) {
  while (sec >= 8) {
    LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
    sec -= 8;
  }
  if (sec >= 4) {
    LowPower.powerDown(SLEEP_4S, ADC_OFF, BOD_OFF);
    sec -= 4;
  }
  if (sec >= 2) {
    LowPower.powerDown(SLEEP_2S, ADC_OFF, BOD_OFF);
    sec -= 2;
  }
  if (sec >= 1) {
    LowPower.powerDown(SLEEP_1S, ADC_OFF, BOD_OFF);
    sec -= 1;
  }
}
```

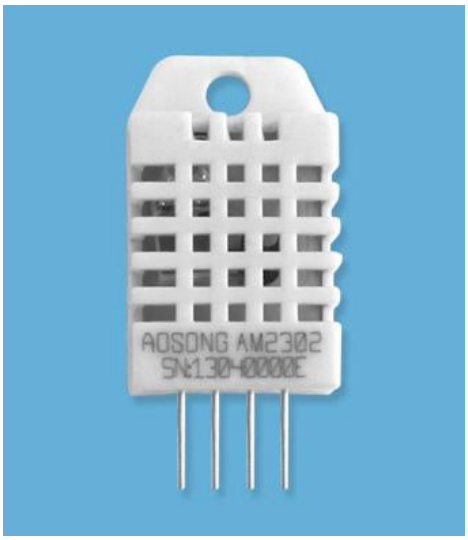
7.3. Anexo III: Datasheets

Sensor de temperatura y humedad:

AOSONG

Temperature and humidity module

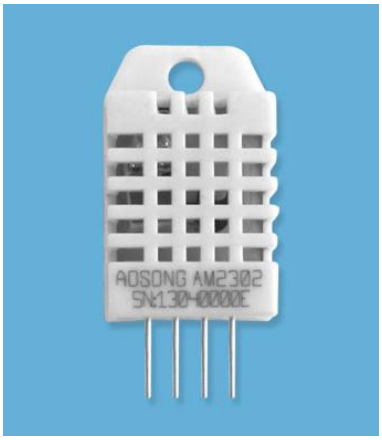
AM2302 Product Manual



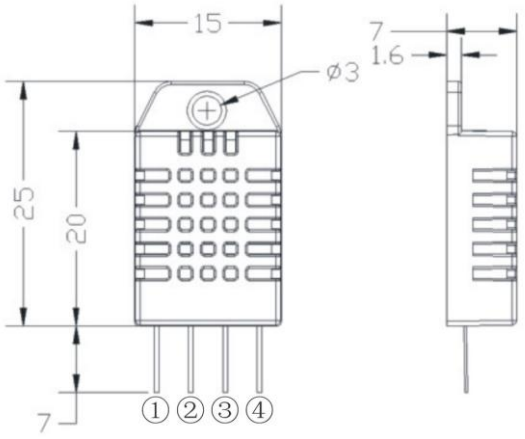
www.aosong.com

1、Product Overview

AM2302 capacitive humidity sensing digital temperature and humidity module is one that contains the compound has been calibrated digital signal output of the temperature and humidity sensors. Application of a dedicated digital modules collection technology and the temperature and humidity sensing technology, to ensure that the product has high reliability and excellent long-term stability. The sensor includes a capacitive sensor wet components and a high-precision temperature measurement devices, and connected with a high-performance 8-bit microcontroller. The product has excellent quality, fast response, strong anti-jamming capability, and high cost. Each sensor is extremely accurate humidity calibration chamber calibration. The form of procedures, the calibration coefficients stored in the microcontroller, the sensor within the processing of the heartbeat to call these calibration coefficients. Standard single-bus interface, system integration quick and easy. Small size, low power consumption, signal transmission distance up to 20 meters, making it the best choice of all kinds of applications and even the most demanding applications. Products for the 3-lead (single-bus interface) connection convenience. Special packages according to user needs.



Physical map



Dimensions (unit: mm)

2、Applications

HVAC, dehumidifier, testing and inspection equipment, consumer goods, automotive, automatic control, data loggers, home appliances, humidity regulator, medical, weather stations, and other humidity measurement and control and so on.

3、Features

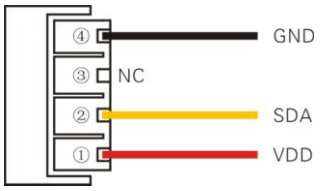
Ultra-low power, the transmission distance, fully automated calibration, the use of capacitive humidity sensor, completely interchangeable, standard digital single-bus output, excellent long-term stability, high accuracy temperature measurement devices.

4、The definition of single-bus interface

4.1 AM2302 Pin assignments

Table 1： AM2302 Pin assignments

Pin	Name	Description
①	VDD	Power (3.3V-5.5V)
②	SDA	Serial data, bidirectional port
③	NC	Empty
④	GND	Ground



PIC1： AM2302 Pin Assignment

4.2 Power supply pins（VDD GND）

AM2302 supply voltage range 3.3V － 5.5V, recommended supply voltage is 5V.

4.3 Serial data（SDA）

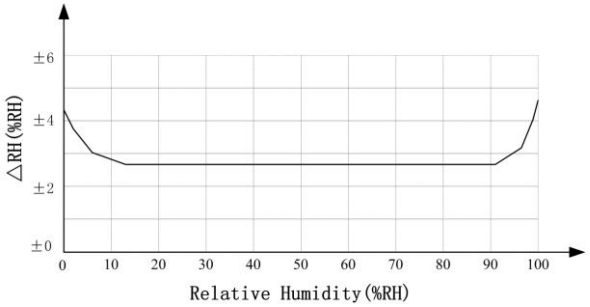
SDA pin is tri structure for reading, writing sensor data. Specific communication timing, see the detailed description of the communication protocol.

5、Sensor performance

5.1 Relative humidity

Table 2： AM2302 Relative humidity performance table

Parameter	Condition	min	typ	max	Unit
Resolution			0.1		%RH
Range		0		99.9	%RH
Accuracy [1]	25℃		± 2		%RH
Repeatability			± 0.3		%RH
Exchange		Completely interchangeable			
Response [2]	1/e(63%)		<5		S
Sluggish			<0.3		%RH
Drift [3]	Typical		<0.5		%RH/yr

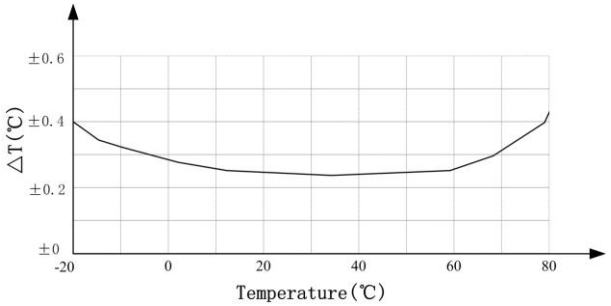


Pic2: At25℃ The error of relative humidity

5.2 Temperature

Table 3： AM2302 Relative temperature performance

Parameter	Condition	min	typ	max	Unit
Resolution			0.1		℃
			16		bit
Accuracy			± 0.5	± 1	℃
Range		-40		80	℃
Repeat			± 0.2		℃
Exchange		Completely interchangeable			
Response	1/e(63%)		<10		S
Drift			± 0.3		℃/yr



Pic3： The maximum temperature error

6、Electrical Characteristics

Electrical characteristics, such as energy consumption, high, low, input, output voltage, depending on the power supply. Table 4 details the electrical characteristics of the AM2302, if not identified, said supply voltage of 5V. To get the best results with the sensor, please design strictly in accordance with the conditions of design in Table 4.

Table 4： AM2302 DC Characteristics

Parameter	Condition	min	typ	max	Unit
Voltage		3.3	5	5.5	V
Power consumption [4]	Dormancy	10	15		μA
	Measuring		500		μA
	Average		300		μA
Low level output voltage	I_{OL}[5]	0		300	mV
High output voltage	Rp<25 kΩ	90%		100%	VDD
Low input voltage	Decline	0		30%	VDD
Input High Voltage	Rise	70%		100%	VDD
Rpu[6]	VDD = 5V VIN = VSS	30	45	60	kΩ
Output current	turn on		8		mA
	turn off	10	20		μA
Sampling period		2			S

[1] the accuracy of the factory inspection, the sensor 25℃ and 5V, the accuracy specification of test conditions, it does not include hysteresis and nonlinearity, and is only suitable for non-condensing environment.

[2] to achieve an order of 63% of the time required under the conditions of 25℃ and 1m / s airflow.

[3] in the volatile organic compounds, the values may be higher. See the manual application to store information.

[4] this value at VDD = 5.0V when the temperature is 25℃, 2S / time, under the conditions of the average.

[5] low output current.

[6] that the pull-up resistor.

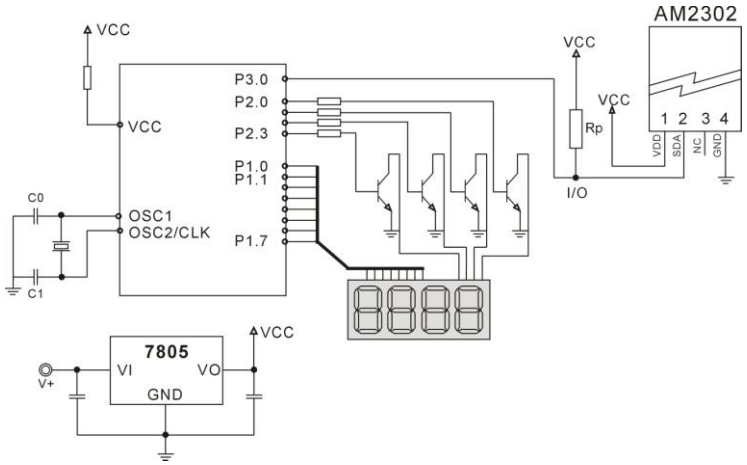
7、Single-bus communication（ONE-WIRE）

7.1 Typical circuits for single bus

Microprocessor and AM2302 connection typical application circuit is shown in Figure 4. Single bus communication mode, pull the SDA microprocessor I / O port is connected.

Special instructions of the single-bus communication：

1. Typical application circuit recommended in the short cable length of 30 meters on the 5.1K pull-up resistor pullup resistor according to the actual situation of lower than 30 m.
2. With 3.3V supply voltage, cable length shall not be greater than 100cm. Otherwise, the line voltage drop will lead to the sensor power supply, resulting in measurement error.
3. Read the sensor minimum time interval for the 2S; read interval is less than 2S, may cause the temperature and humidity are not allowed or communication is unsuccessful, etc..
4. Temperature and humidity values are each read out the results of the last measurement For real-time data that need continuous read twice, we recommend repeatedly to read sensors, and each read sensor interval is greater than 2 seconds to obtain accuratethe data.



Pic4： AM2302 Typical circuits for single bus

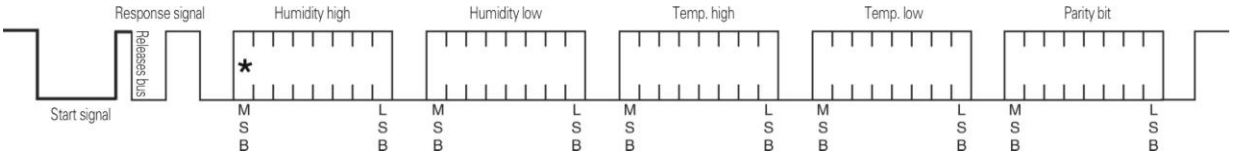
7.2、Single-bus communication protocol

◎Single bus Description

AM2302 device uses a simplified single-bus communication. Single bus that only one data line, data exchange system, controlled by the data line to complete. Equipment (microprocessor) through an open-drain or tri-state port connected to the data line to allow the device does not send data to release the bus, while other devices use the bus; single bus usually require an external about 5.1kΩ pull-up resistor, so when the bus is idle, its status is high. Because they are the master-slave structure, only the host calls the sensor, the sensor will answer, so the hosts to access the sensor must strictly follow the sequence of single bus, if there is a sequence of confusion, the sensor will not respond to the host.

◎Single bus to send data definition

SDA For communication and synchronization between the microprocessor and the AM2302, single-bus data format, a transmission of 40 data, the high first-out. Specific communication timing shown in Figure 5, the communication format is depicted in Table 5.



Pic5： AM2302 Single-bus communication protocol

Table 5： AM2302 Communication format specifier

Name	Single-bus format definition
Start signal	Microprocessor data bus (SDA) to bring down a period of time (at least 800μ s) [1] notify the sensor to prepare the data.
Response signal	Sensor data bus (SDA) is pulled down to 80μ s, followed by high-80μ s response to host the start signal.
Data format	Host the start signal is received, the sensor one-time string from the data bus (SDA) 40 data, the high first-out.
Humidity	Humidity resolution of 16Bit, the previous high; humidity sensor string value is 10 times the actual humidity values.
Temp.	Temperature resolution of 16Bit, the previous high; temperature sensor string value is 10 times the actual temperature value; The temperature is the highest bit (Bit15) is equal to 1 indicates a negative temperature, the temperature is the highest bit (Bit15) is equal to 0 indicates a positive temperature; Temperature in addition to the most significant bit (Bit14 ~ bit 0) temperature values.
Parity bit	Parity bit = humidity high + humidity low + temperature high + temperature low

◎**Single-bus data calculation example**

Example 1： 40 Data received：

0000 0010	1001 0010	0000 0001	0000 1101	1010 0010
High humidity 8	Low humidity 8	High temp. 8	Low temp. 8	Parity bit

Calculate：

0000 0010+1001 0010 +0000 0001+0000 1101= 1010 0010（Parity bit）

Received data is correct：

humidity： 0000 0010 1001 0010 = 0292H (Hexadecimal)= 2×256 + 9×16 + 2 = 658

=> Humidity = 65.8%RH

Temp.： 0000 0001 0000 1101 = 10DH(Hexadecimal) = 1×256 + 0×16 + 13 = 269

=> Temp.= 26.9℃

◎**Special Instructions：**

When the temperature is below 0 ℃, the highest position of the temperature data.

Example： −10.1 ℃ Expressed as 1 000 0000 0110 0101

Temp.： 0000 0000 0110 0101 = 0065H(Hexadecimal)= 6×16 +5 = 101

=> Temp. = −10.1℃

Example 2： 40 received data：

0000 0010	1001 0010	0000 0001	0000 1101	1011 0010
High humidity 8	Low humidity 8	High temp. 8	Low temp. 8	Parity bit

Calculate：

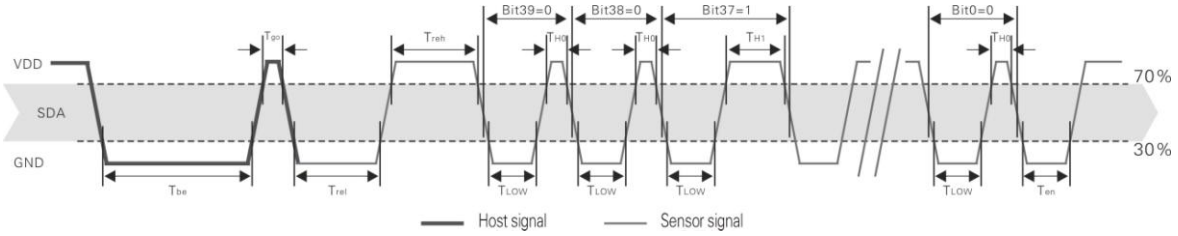
0000 0010+1001 0010 +0000 0001+0000 1101= 1010 0010 ≠ 1011 0010 （Validation error）

The received data is not correct, give up, to re-receive data.

7.3 Single-bus communication timing

User host (MCU) to send a start signal (data bus SDA line low for at least 800μ s) after AM2302 from Sleep mode conversion to high-speed mode.The host began to signal the end of the AM2302 send a response signal sent from the data bus SDA serial 40Bit's data, sends the byte high; data sent is followed by: Humidity high、Humidity low、Temperature high、Temperature low、Parity bit，Send data to the end of trigger information collection, the collection end of the sensor is automatically transferred to the sleep mode, the advent until the next communication.

Detailed timing signal characteristics in Table 6， Single-bus communication timing diagram Pic 6：



Pic 6： AM2302 Single-bus communication timing

Note: the temperature and humidity data read by the host from the AM2302 is always the last measured value, such as the two measurement interval is very long, continuous read twice to the second value of real-time temperature and humidity values, while two readtake minimum time interval be 2S.

Table 6: Single bus signal characteristics

Symbol	Parameter	min	typ	max	Unit
T_{be}	Host the start signal down time	0.8	1	20	mS
T_{go}	Bus master has released time	20	30	200	μS
T_{rel}	Response to low time	75	80	85	μS
T_{reh}	In response to high time	75	80	85	μS
T_{LOW}	Signal "0", "1" low time	48	50	55	μS
T_{H0}	Signal "0" high time	22	26	30	μS
T_{H1}	Signal "1" high time	68	70	75	μS
T_{en}	Sensor to release the bus time	45	50	55	μS

Note： To ensure the accurate communication of the sensor, the read signal, in strict accordance with the design parameters and timing in Table 6 and Figure 6.

7.4 Peripherals read step example

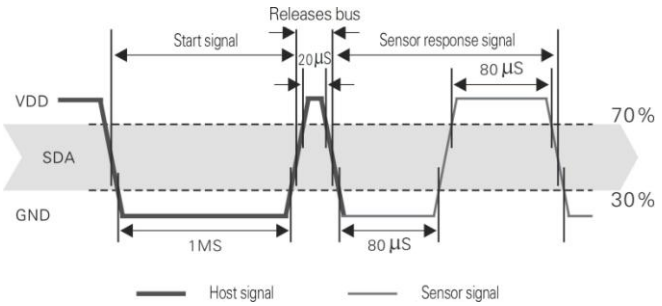
Communication between the host and the sensor can read data through the following three steps to complete.

Step 1

AM2302 have to wait for the power (on AM2302 power 2S crossed the unstable state, the device can not send any instructions to read during this period), the test environment temperature and humidity data, and record data, since the sensor into a sleep state automatically. AM2302 The SDA data line from the previous pull-up resistor pulled up is always high, the AM2302 the SDA pin is in input state, the time detection of external signal.

Step 2

Microprocessor I/O set to output, while output low, and low hold time can not be less than 800us, typical values are down 1MS, then the microprocessor I/O is set to input state, the release of the bus, due to the pull-up resistor, the microprocessor I/O AM2302 the SDA data line also will be high, the bus master has released the AM2302 send a response signal, that is, the output 80 microseconds low as the response signal, tightthen output high of 80 microseconds notice peripheral is ready to receive data signal transmission as shown to Pic7 :



Pic7： Single bus decomposition of the timing diagram

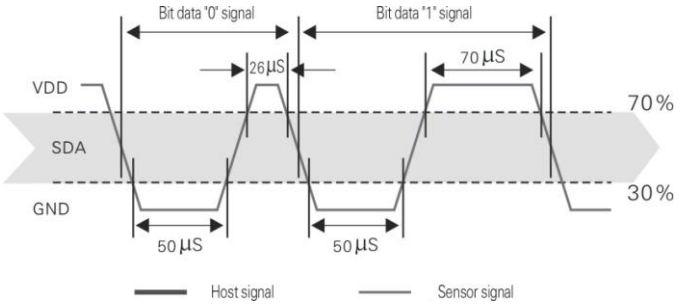
Step 3

AM2302 sending the response, followed by the data bus SDA continuous serial output 40 data, the microprocessor receives 40 data I/O level changes.

Bit data "0" format: 26-28 microseconds 50 microseconds low plus high;

Bit data "1" format: the high level of low plus, 50 microseconds to 70 microseconds;

Bit data "0" bit data "1" format signal shown to pic 8:

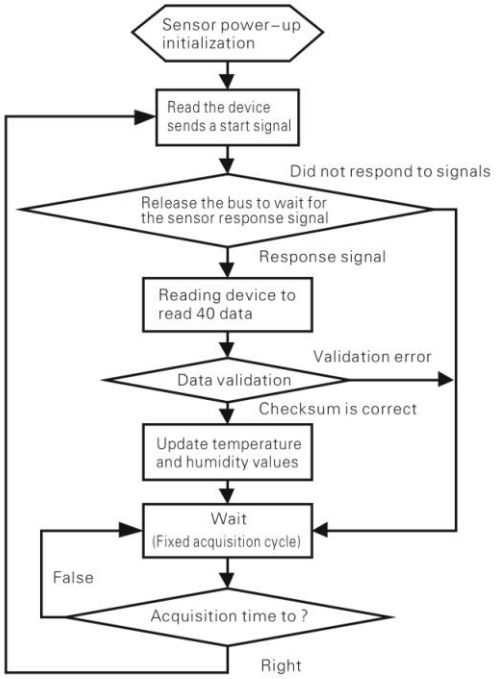


Pic 8： The single bus break down the timing diagram

AM2302 data bus SDA output 40 data continue to output the low 50 microseconds into the input state, followed by pull-up resistor goes high. AM2302 internal re-test environmental temperature and humidity data, and record the data, the end of the test records, the microcontroller automatically into hibernation. Microcontroller only after receipt of the start signal of the host wake-up sensor, into the working state.

7.5 Peripheral to read flow chart

AM2302 sensor read single bus flow chart diagram shown in Figure 9, we also provide the C51 read the code examples, customers need to download, please visit our website (www.aosong.com) related to downloadthis manual does not provide the code description.



Pic9：Single-bus to read the flow chart

8、Application of information

1. Work and storage conditions

Outside the sensor the proposed scope of work may lead to temporary drift of the signal up to 300% RH. Return to normal working conditions, sensor calibration status will slowly toward recovery. To speed up the recovery process may refer to "resume processing". Prolonged use of non-normal operating conditions, will accelerate the aging of the product.

Avoid placing the components on the long-term condensation and dry environment, as well as the following environment.

A, salt spray

B, acidic or oxidizing gases such as sulfur dioxide, hydrochloric acid

Recommended storage environment

Temperature: 10 ~ 40 ℃ Humidity: 60% RH or less

2. The impact of exposure to chemicals

The capacitive humidity sensor has a layer by chemical vapor interference, the proliferation of chemicals in the sensing layer may lead to drift and decreased sensitivity of the measured values. In a pure environment, contaminants will slowly be released. Resume processing as described below will accelerate this process. The high concentration of chemical pollution (such as ethanol) will lead to the complete damage of the sensitive layer of the sensor.

3. The temperature influence

Relative humidity of the gas to a large extent dependent on temperature. Therefore, in the measurement of humidity,

should be to ensure that the work of the humidity sensor at the same temperature. With the release of heat of electronic components share a printed circuit board, the installation should be as far as possible the sensor away from the electronic components and mounted below the heat source, while maintaining good ventilation of the enclosure. To reduce the thermal conductivity sensor and printed circuit board copper plating should be the smallest possible, and leaving a gap between the two.

4. Light impact

Prolonged exposure to sunlight or strong ultraviolet radiation, and degrade performance.

5. Resume processing

Placed under extreme working conditions or chemical vapor sensor, which allows it to return to the status of calibration by the following handler. Maintain two hours in the humidity conditions of 45℃ and <10% RH (dry); followed by 20-30℃ and> 70% RH humidity conditions to maintain more than five hours.

6. Wiring precautions

The quality of the signal wire will affect the quality of the voltage output, it is recommended to use high quality shielded cable.

7. Welding information

Manual welding, in the maximum temperature of 300℃ under the conditions of contact time shall be less than 3 seconds.

8. Product upgrades

Details, please the consultation Aosong electronics department.

9、The license agreement

Without the prior written permission of the copyright holder, shall not in any form or by any means, electronic or mechanical (including photocopying), copy any part of this manual, nor shall its contents be communicated to a third party. The contents are subject to change without notice.

The Company and third parties have ownership of the software, the user may use only signed a contract or software license.

10、Warnings and personal injury

This product is not applied to the safety or emergency stop devices, as well as the failure of the product may result in injury to any other application, unless a particular purpose or use authorized. Installation, handling, use or maintenance of the product refer to product data sheets and application notes. Failure to comply with this recommendation may result in death and serious personal injury. The Company will bear all damages resulting personal injury or death, and waive any claims that the resulting subsidiary company managers and employees and agents, distributors, etc. that may arise, including: a variety of costs, compensation costs, attorneys' fees, and so on.

11、Quality Assurance

The company and its direct purchaser of the product quality guarantee period of three months (from the date of delivery). Publishes the technical specifications of the product data sheet shall prevail. Within the warranty period, the product was confirmed that the quality is really defective, the company will provide free repair or replacement. The user must satisfy the following conditions:

① The product is found defective within 14 days written notice to the Company;

② The product shall be paid by mail back to the company;

③ The product should be within the warranty period.

The Company is only responsible for those used in the occasion of the technical condition of the product defective product. Without any guarantee, warranty or written statement of its products used in special applications. Company for its products applied to the reliability of the product or circuit does not make any commitment.

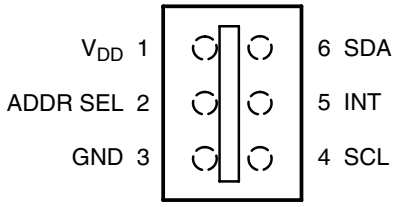
Sensor de luminosidad:

TSL2560, TSL2561
LIGHT-TO-DIGITAL CONVERTER

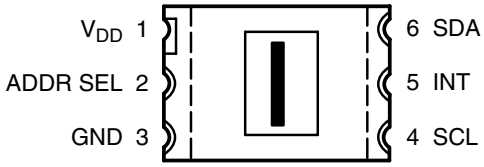
TAOS059N – MARCH 2009

- **Approximates Human Eye Response**
- **Programmable Interrupt Function with User-Defined Upper and Lower Threshold Settings**
- **16-Bit Digital Output with SMBus (TSL2560) at 100 kHz or I²C (TSL2561) Fast-Mode at 400 kHz**
- **Programmable Analog Gain and Integration Time Supporting 1,000,000-to-1 Dynamic Range**
- **Automatically Rejects 50/60-Hz Lighting Ripple**
- **Low Active Power (0.75 mW Typical) with Power Down Mode**
- **RoHS Compliant**

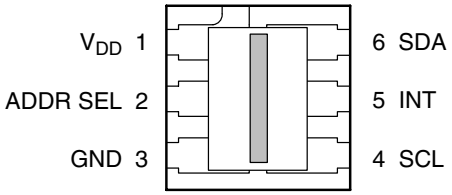
PACKAGE CS
6-LEAD CHIPSCALE
(TOP VIEW)



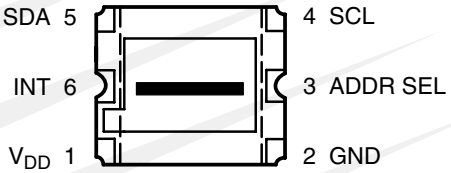
PACKAGE T
6-LEAD TMB
(TOP VIEW)



PACKAGE FN
DUAL FLAT NO-LEAD
(TOP VIEW)



PACKAGE CL
6-LEAD ChipLED
(TOP VIEW)



Package Drawings are Not to Scale

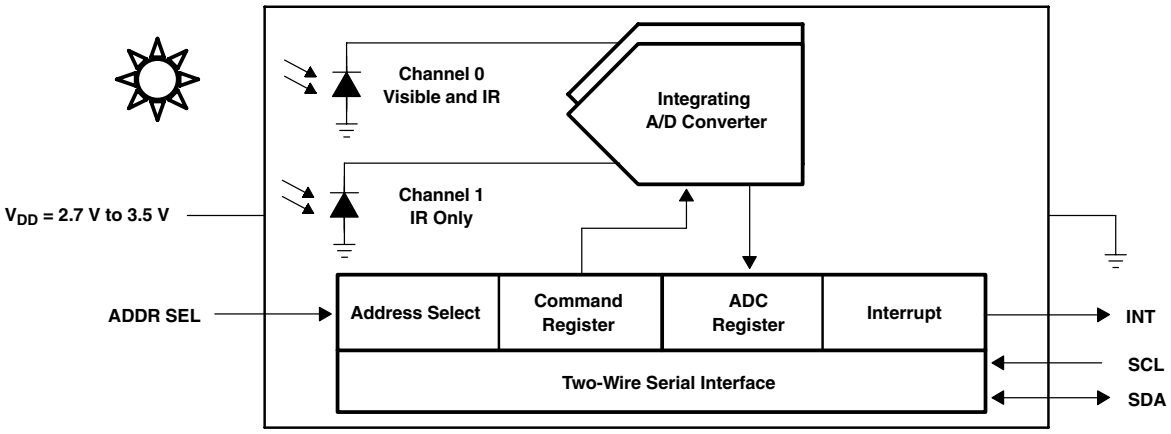
Description

The TSL2560 and TSL2561 are light-to-digital converters that transform light intensity to a digital signal output capable of direct I²C (TSL2561) or SMBus (TSL2560) interface. Each device combines one broadband photodiode (visible plus infrared) and one infrared-responding photodiode on a single CMOS integrated circuit capable of providing a near-photopic response over an effective 20-bit dynamic range (16-bit resolution). Two integrating ADCs convert the photodiode currents to a digital output that represents the irradiance measured on each channel. This digital output can be input to a microprocessor where illuminance (ambient light level) in lux is derived using an empirical formula to approximate the human eye response. The TSL2560 device permits an SMB-Alert style interrupt, and the TSL2561 device supports a traditional level style interrupt that remains asserted until the firmware clears it.

While useful for general purpose light sensing applications, the TSL2560/61 devices are designed particularly for display panels (LCD, OLED, etc.) with the purpose of extending battery life and providing optimum viewing in diverse lighting conditions. Display panel backlighting, which can account for up to 30 to 40 percent of total platform power, can be automatically managed. Both devices are also ideal for controlling keyboard illumination based upon ambient lighting conditions. Illuminance information can further be used to manage exposure control in digital cameras. The TSL2560/61 devices are ideal in notebook/tablet PCs, LCD monitors, flat-panel televisions, cell phones, and digital cameras. In addition, other applications include street light control, security lighting, sunlight harvesting, machine vision, and automotive instrumentation clusters.

Texas Advanced Optoelectronic Solutions Inc.
1001 Klein Road • Suite 300 • Plano, TX 75074 • (972) 673-0759
www.taosinc.com

Functional Block Diagram



Detailed Description

The TSL2560 and TSL2561 are second-generation ambient light sensor devices. Each contains two integrating analog-to-digital converters (ADC) that integrate currents from two photodiodes. Integration of both channels occurs simultaneously. Upon completion of the conversion cycle, the conversion result is transferred to the Channel 0 and Channel 1 data registers, respectively. The transfers are double-buffered to ensure that the integrity of the data is maintained. After the transfer, the device automatically begins the next integration cycle.

Communication to the device is accomplished through a standard, two-wire SMBus or I^2C serial bus. Consequently, the TSL256x device can be easily connected to a microcontroller or embedded controller. No external circuitry is required for signal conditioning, thereby saving PCB real estate as well. Since the output of the TSL256x device is digital, the output is effectively immune to noise when compared to an analog signal.

The TSL256x devices also support an interrupt feature that simplifies and improves system efficiency by eliminating the need to poll a sensor for a light intensity value. The primary purpose of the interrupt function is to detect a meaningful change in light intensity. The concept of a *meaningful change* can be defined by the user both in terms of light intensity and time, or persistence, of that change in intensity. The TSL256x devices have the ability to define a threshold above and below the current light level. An interrupt is generated when the value of a conversion exceeds either of these limits.

Available Options

DEVICE	INTERFACE	PACKAGE – LEADS	PACKAGE DESIGNATOR	ORDERING NUMBER
TSL2560	SMBus	Chipscale	CS	TSL2560CS
TSL2560	SMBus	TMB-6	T	TSL2560T
TSL2560	SMBus	Dual Flat No-Lead – 6	FN	TSL2560FN
TSL2560	SMBus	ChipLED-6	CL	TSL2560CL
TSL2561	I^2C	Chipscale	CS	TSL2561CS
TSL2561	I^2C	TMB-6	T	TSL2561T
TSL2561	I^2C	Dual Flat No-Lead – 6	FN	TSL2561FN
TSL2561	I^2C	ChipLED-6	CL	TSL2561CL

Terminal Functions

TERMINAL NAME	CS, T, FN PKG NO.	CL PKG NO.	TYPE	DESCRIPTION
ADDR SEL	2	3	I	SMBus device select — three-state
GND	3	2		Power supply ground. All voltages are referenced to GND.
INT	5	6	O	Level or SMB Alert interrupt — open drain.
SCL	4	4	I	SMBus serial clock input terminal — clock signal for SMBus serial data.
SDA	6	5	I/O	SMBus serial data I/O terminal — serial data I/O for SMBus.
V_{DD}	1	1		Supply voltage.

Absolute Maximum Ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage, V_{DD} (see Note 1) . . . . . 3.8 V
Digital output voltage range, V_O . . . . . –0.5 V to 3.8 V
Digital output current, I_O . . . . . –1 mA to 20 mA
Storage temperature range, T_{stg} . . . . . –40°C to 85°C
ESD tolerance, human body model . . . . . 2000 V

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

NOTE 1: All voltages are with respect to GND.

Recommended Operating Conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V_{DD}	2.7	3	3.6	V
Operating free-air temperature, T_A	–30		70	°C
SCL, SDA input low voltage, V_{IL}	–0.5		0.8	V
SCL, SDA input high voltage, V_{IH}	2.1		3.6	V

Electrical Characteristics over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
I_{DD}	Supply current	Active		0.24	0.6	mA
		Power down		3.2	15	µA
V_{OL}	INT, SDA output low voltage	3 mA sink current	0		0.4	V
		6 mA sink current	0		0.6	V
I_{LEAK}	Leakage current		–5		5	µA

Operating Characteristics, High Gain (16×), V_{DD} = 3 V, T_A = 25°C, (unless otherwise noted) (see Notes 2, 3, 4, 5)

PARAMETER		TEST CONDITIONS	CHANNEL	TSL2560T, FN, & CL TSL2561T, FN & CL			TSL2560CS, TSL2561CS			UNIT
				MIN	TYP	MAX	MIN	TYP	MAX	
f_{osc}	Oscillator frequency			690	735	780	690	735	780	kHz
	Dark ADC count value	E_e = 0, T_{int} = 402 ms	Ch0	0		4	0		4	counts
			Ch1	0		4	0		4	
	Full scale ADC count value (Note 6)	T_{int} > 178 ms	Ch0			65535			65535	counts
			Ch1			65535			65535	
		T_{int} = 101 ms	Ch0			37177			37177	
			Ch1			37177			37177	
		T_{int} = 13.7 ms	Ch0			5047			5047	
			Ch1			5047			5047	
	ADC count value	λ_p = 640 nm, T_{int} = 101 ms, E_e = 36.3 µW/cm²	Ch0	750	1000	1250				counts
			Ch1		200					
		λ_p = 940 nm, T_{int} = 101 ms, E_e = 119 µW/cm²	Ch0	700	1000	1300				
			Ch1		820					
		λ_p = 640 nm, T_{int} = 101 ms, E_e = 41 µW/cm²	Ch0				750	1000	1250	counts
			Ch1					190		
		λ_p = 940 nm, T_{int} = 101 ms, E_e = 135 µW/cm²	Ch0				700	1000	1300	
			Ch1					850		
	ADC count value ratio: Ch1/Ch0	λ_p = 640 nm, T_{int} = 101 ms		0.15	0.20	0.25	0.14	0.19	0.24	
		λ_p = 940 nm, T_{int} = 101 ms		0.69	0.82	0.95	0.70	0.85	1	
R_e	Irradiance responsivity	λ_p = 640 nm, T_{int} = 101 ms	Ch0		27.5			24.4		counts/(µW/cm²)
			Ch1		5.5			4.6		
		λ_p = 940 nm, T_{int} = 101 ms	Ch0		8.4			7.4		
			Ch1		6.9			6.3		
R_v	Illuminance responsivity	Fluorescent light source: T_{int} = 402 ms	Ch0		36			35		counts/lux
			Ch1		4			3.8		
		Incandescent light source: T_{int} = 402 ms	Ch0		144			129		
			Ch1		72			67		
	ADC count value ratio: Ch1/Ch0	Fluorescent light source: T_{int} = 402 ms			0.11			0.11		
		Incandescent light source: T_{int} = 402 ms			0.5			0.52		
R_v	Illuminance responsivity, low gain mode (Note 7)	Fluorescent light source: T_{int} = 402 ms	Ch0		2.3			2.2		counts/lux
			Ch1		0.25			0.24		
		Incandescent light source: T_{int} = 402 ms	Ch0		9			8.1		
			Ch1		4.5			4.2		
	(Sensor Lux) / (actual Lux), high gain mode (Note 8)	Fluorescent light source: T_{int} = 402 ms		0.65	1	1.35	0.65	1	1.35	
		Incandescent light source: T_{int} = 402 ms		0.60	1	1.40	0.60	1	1.40	

NOTES:
2. Optical measurements are made using small-angle incident radiation from light-emitting diode optical sources. Visible 640 nm LEDs and infrared 940 nm LEDs are used for final product testing for compatibility with high-volume production.
3. The 640 nm irradiance E_e is supplied by an AlInGaP light-emitting diode with the following characteristics: peak wavelength λp = 640 nm and spectral halfwidth $\Delta\lambda$½ = 17 nm.
4. The 940 nm irradiance E_e is supplied by a GaAs light-emitting diode with the following characteristics: peak wavelength λp = 940 nm and spectral halfwidth $\Delta\lambda$½ = 40 nm.
5. Integration time T_{int}, is dependent on internal oscillator frequency (f_{osc}) and on the integration field value in the timing register as described in the *Register Set* section. For nominal f_{osc} = 735 kHz, nominal T_{int} = (number of clock cycles)/f_{osc}.
   Field value 00: $T_{int} = (11 \times 918)/f_{osc}$ = 13.7 ms
   Field value 01: $T_{int} = (81 \times 918)/f_{osc}$ = 101 ms
   Field value 10: $T_{int} = (322 \times 918)/f_{osc}$ = 402 ms
   Scaling between integration times vary proportionally as follows: 11/322 = 0.034 (field value 00), 81/322 = 0.252 (field value 01), and 322/322 = 1 (field value 10).
6. Full scale ADC count value is limited by the fact that there is a maximum of one count per two oscillator frequency periods and also by a 2-count offset.
   Full scale ADC count value = ((number of clock cycles)/2 – 2)
   Field value 00: Full scale ADC count value = $((11 \times 918)/2 - 2)$ = 5047
   Field value 01: Full scale ADC count value = $((81 \times 918)/2 - 2)$ = 37177
   Field value 10: Full scale ADC count value = 65535, which is limited by 16 bit register. This full scale ADC count value is reached for 131074 clock cycles, which occurs for T_{int} = 178 ms for nominal f_{osc} = 735 kHz.
7. Low gain mode has 16× lower gain than high gain mode: (1/16 = 0.0625).
8. The sensor Lux is calculated using the empirical formula shown on p. 22 of this data sheet based on measured Ch0 and Ch1 ADC count values for the light source specified. Actual Lux is obtained with a commercial luxmeter. The range of the (sensor Lux) / (actual Lux) ratio is estimated based on the variation of the 640 nm and 940 nm optical parameters. Devices are not 100% tested with fluorescent or incandescent light sources.

AC Electrical Characteristics, V_{DD} = 3 V, T_A = 25°C (unless otherwise noted)

PARAMETER†		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_{(CONV)}$	Conversion time		12	100	400	ms
$f_{(SCL)}$	Clock frequency (I^2C only)		0		400	kHz
	Clock frequency (SMBus only)		10		100	kHz
$t_{(BUF)}$	Bus free time between start and stop condition		1.3			µs
$t_{(HDSTA)}$	Hold time after (repeated) start condition. After this period, the first clock is generated.		0.6			µs
$t_{(SUSTA)}$	Repeated start condition setup time		0.6			µs
$t_{(SUSTO)}$	Stop condition setup time		0.6			µs
$t_{(HDDAT)}$	Data hold time		0		0.9	µs
$t_{(SUDAT)}$	Data setup time		100			ns
$t_{(LOW)}$	SCL clock low period		1.3			µs
$t_{(HIGH)}$	SCL clock high period		0.6			µs
$t_{(TIMEOUT)}$	Detect clock/data low timeout (SMBus only)		25		35	ms
t_F	Clock/data fall time				300	ns
t_R	Clock/data rise time				300	ns
C_i	Input pin capacitance				10	pF

† Specified by design and characterization; not production tested.

PARAMETER MEASUREMENT INFORMATION

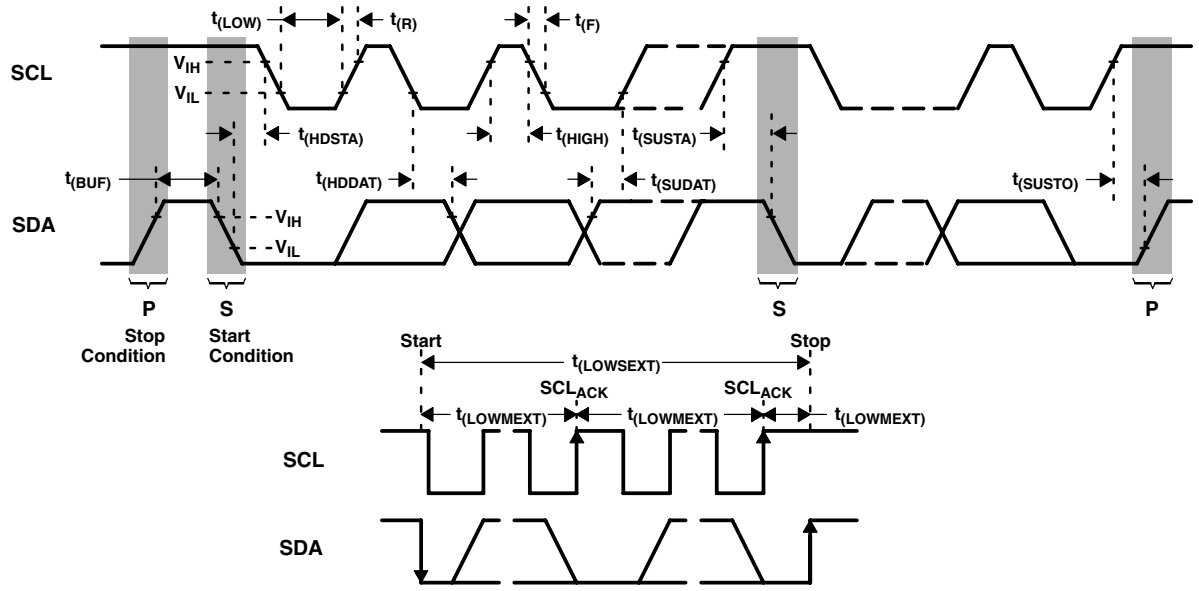


Figure 1. Timing Diagrams

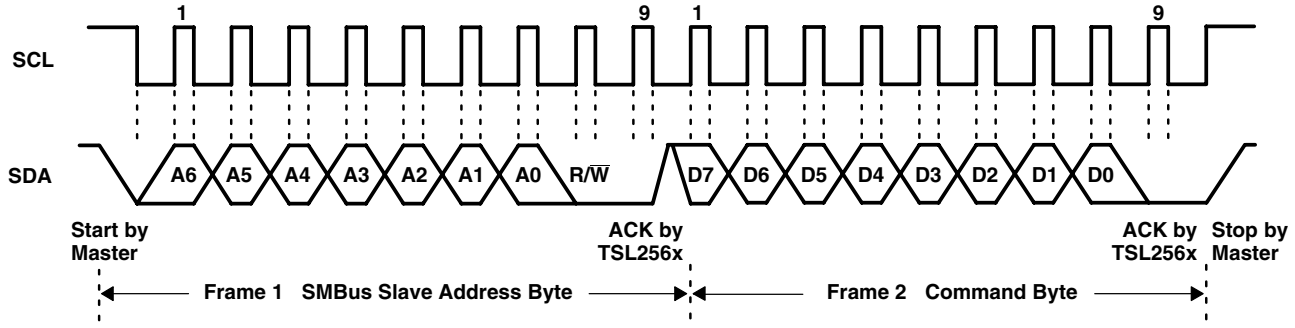


Figure 2. Example Timing Diagram for SMBus Send Byte Format

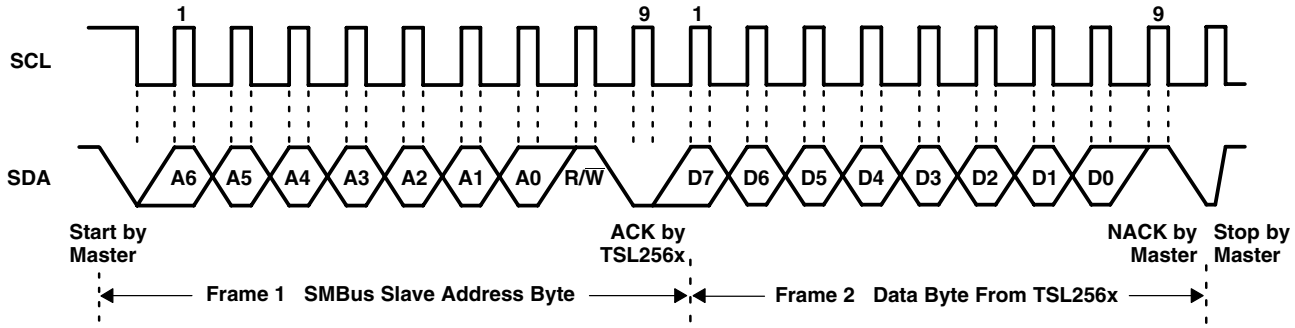


Figure 3. Example Timing Diagram for SMBus Receive Byte Format

TYPICAL CHARACTERISTICS

SPECTRAL RESPONSIVITY

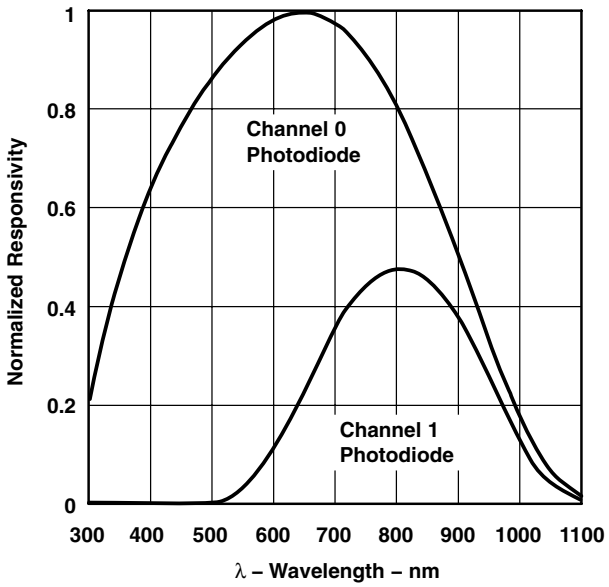


Figure 4

NORMALIZED RESPONSIVITY vs. ANGULAR DISPLACEMENT — CS PACKAGE

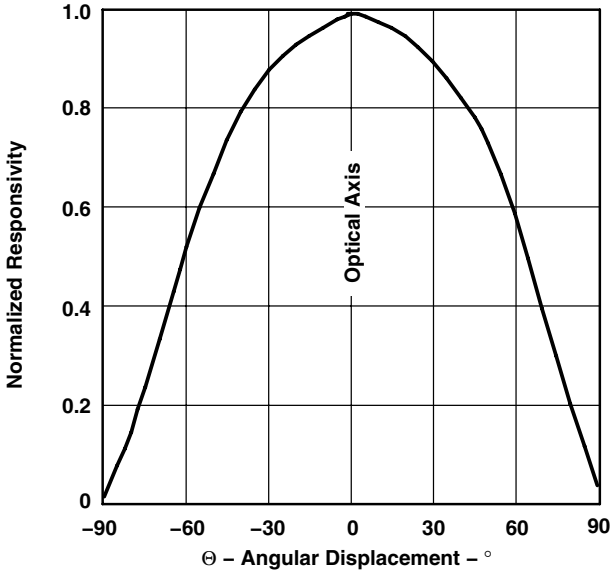


Figure 5

NORMALIZED RESPONSIVITY vs. ANGULAR DISPLACEMENT — T PACKAGE

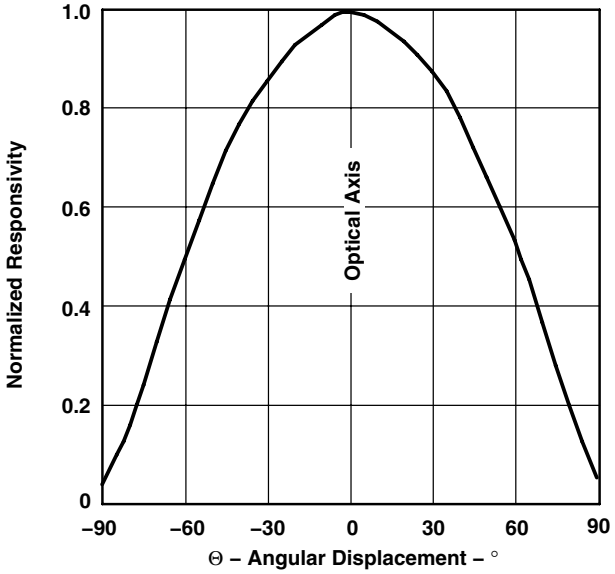


Figure 6

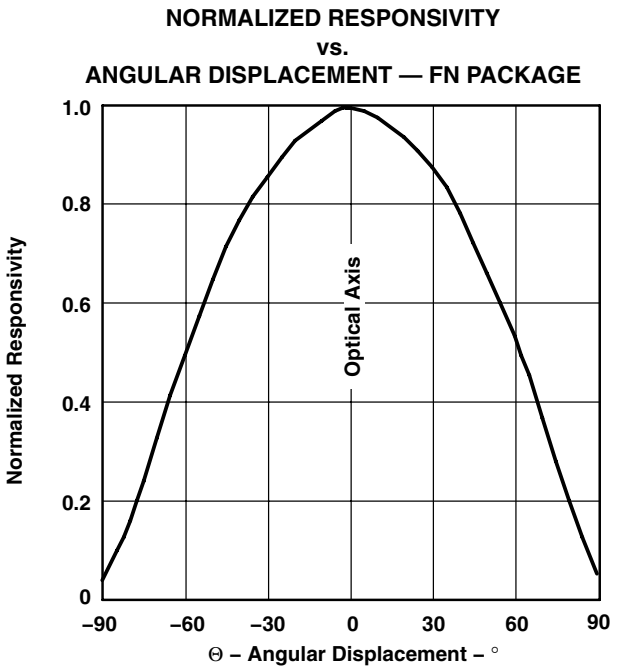


Figure 7

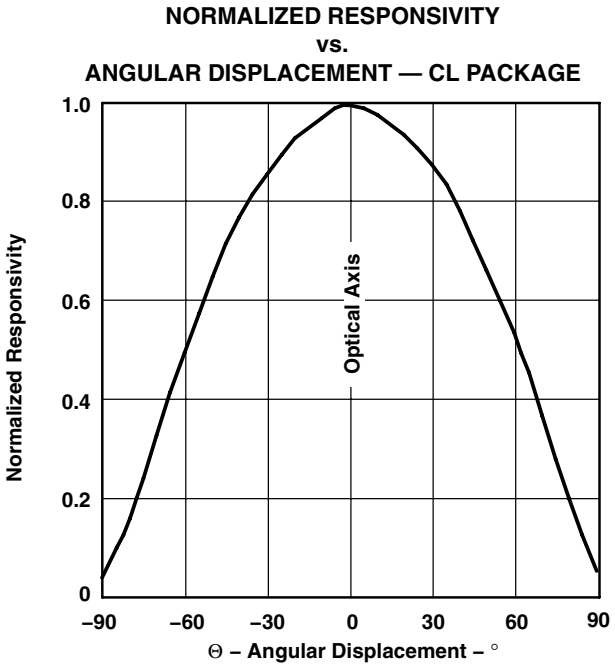


Figure 8

PRINCIPLES OF OPERATION

Analog-to-Digital Converter

The TSL256x contains two integrating analog-to-digital converters (ADC) that integrate the currents from the channel 0 and channel 1 photodiodes. Integration of both channels occurs simultaneously, and upon completion of the conversion cycle the conversion result is transferred to the channel 0 and channel 1 data registers, respectively. The transfers are double buffered to ensure that invalid data is not read during the transfer. After the transfer, the device automatically begins the next integration cycle.

Digital Interface

Interface and control of the TSL256x is accomplished through a two-wire serial interface to a set of registers that provide access to device control functions and output data. The serial interface is compatible with System Management Bus (SMBus) versions 1.1 and 2.0, and I^2C bus Fast-Mode. The TSL256x offers three slave addresses that are selectable via an external pin (ADDR SEL). The slave address options are shown in Table 1.

Table 1. Slave Address Selection

ADDR SEL TERMINAL LEVEL	SLAVE ADDRESS	SMB ALERT ADDRESS
GND	0101001	0001100
Float	0111001	0001100
VDD	1001001	0001100

NOTE: The Slave and SMB Alert Addresses are 7 bits. Please note the SMBus and I^2C protocols on pages 9 through 12. A read/write bit should be appended to the slave address by the master device to properly communicate with the TSL256X device.

SMBus and I^2C Protocols

Each *Send* and *Write* protocol is, essentially, a series of bytes. A byte sent to the TSL256x with the most significant bit (MSB) equal to 1 will be interpreted as a COMMAND byte. The lower four bits of the COMMAND byte form the register select address (see Table 2), which is used to select the destination for the subsequent byte(s) received. The TSL256x responds to any Receive Byte requests with the contents of the register specified by the stored register select address.

The TSL256X implements the following protocols of the SMB 2.0 specification:

- Send Byte Protocol
- Receive Byte Protocol
- Write Byte Protocol
- Write Word Protocol
- Read Word Protocol
- Block Write Protocol
- Block Read Protocol

The TSL256X implements the following protocols of the Philips Semiconductor I^2C specification:

- I^2C Write Protocol
- I^2C Read (Combined Format) Protocol

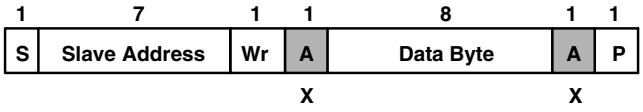
When an SMBus Block Write or Block Read is initiated (see description of COMMAND Register), the byte following the COMMAND byte is ignored but is a requirement of the SMBus specification. This field contains the byte count (i.e. the number of bytes to be transferred). The TSL2560 (SMBus) device ignores this field and extracts this information by counting the actual number of bytes transferred before the Stop condition is detected.

When an I^2C Write or I^2C Read (Combined Format) is initiated, the byte count is also ignored but follows the SMBus protocol specification. Data bytes continue to be transferred from the TSL2561 (I^2C) device to Master until a NACK is sent by the Master.

The data formats supported by the TSL2560 and TSL2561 devices are:

- Master transmitter transmits to slave receiver (SMBus and I^2C):
  - The transfer direction in this case is not changed.
- Master reads slave immediately after the first byte (SMBus only):
  - At the moment of the first acknowledgment (provided by the slave receiver) the master transmitter becomes a master receiver and the slave receiver becomes a slave transmitter.
- Combined format (SMBus and I^2C):
  - During a change of direction within a transfer, the master repeats both a START condition and the slave address but with the R/W bit reversed. In this case, the master receiver terminates the transfer by generating a NACK on the last byte of the transfer and a STOP condition.

For a complete description of SMBus protocols, please review the SMBus Specification at http://www.smbus.org/specs. For a complete description of I^2C protocols, please review the I^2C Specification at http://www.semiconductors.philips.com.



A Acknowledge (this bit position may be 0 for an ACK or 1 for a NACK)

P Stop Condition

Rd Read (bit value of 1)

S Start Condition

Sr Repeated Start Condition

Wr Write (bit value of 0)

X Shown under a field indicates that that field is required to have a value of X

... Continuation of protocol

Master-to-Slave

Slave-to-Master

Figure 9. SMBus and I^2C Packet Protocol Element Key

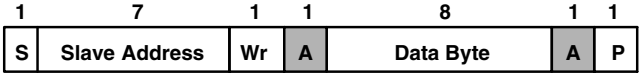


Figure 10. SMBus Send Byte Protocol

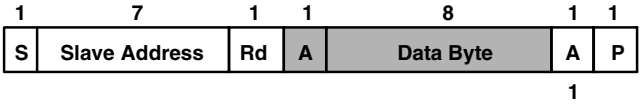


Figure 11. SMBus Receive Byte Protocol

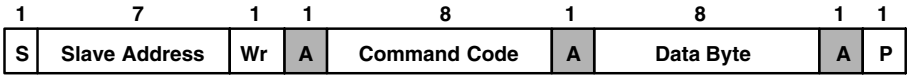


Figure 12. SMBus Write Byte Protocol

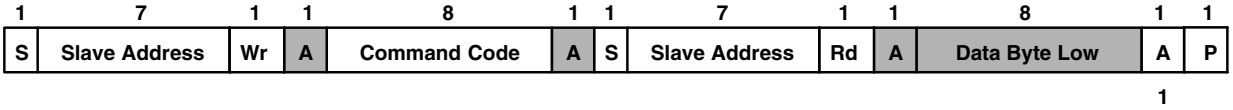


Figure 13. SMBus Read Byte Protocol

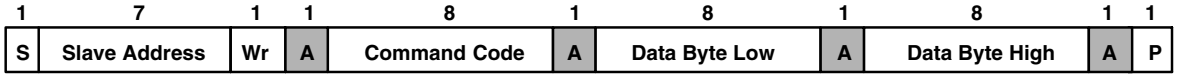


Figure 14. SMBus Write Word Protocol

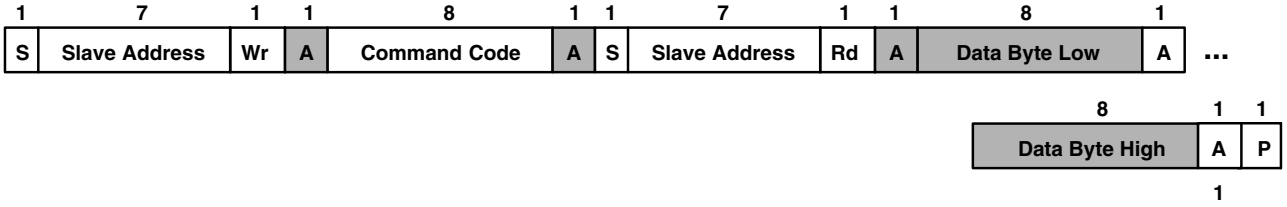


Figure 15. SMBus Read Word Protocol

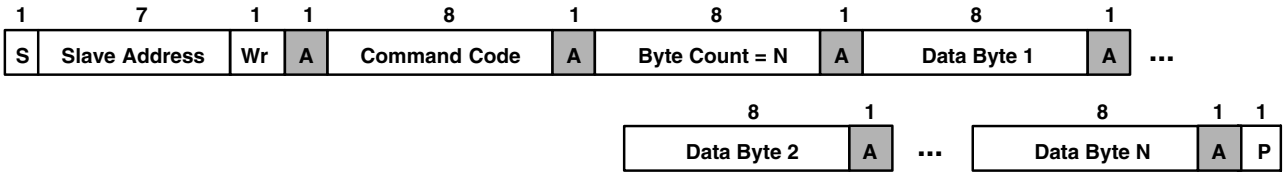


Figure 16. SMBus Block Write or I²C Write Protocols

NOTE: The I²C write protocol does not use the Byte Count packet, and the Master will continue sending Data Bytes until the Master initiates a Stop condition. See the Command Register on page 13 for additional information regarding the Block Read/Write protocol.

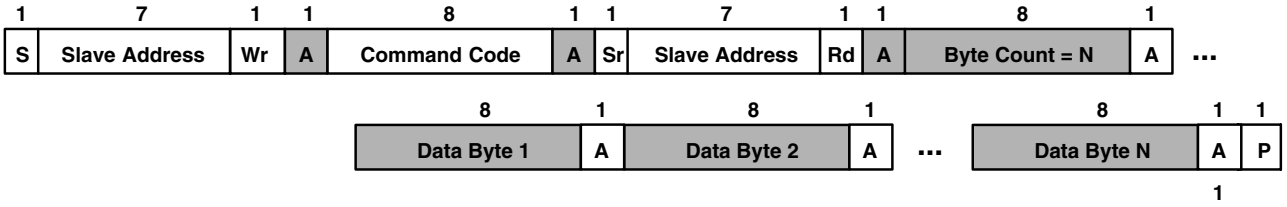


Figure 17. SMBus Block Read or I²C Read (Combined Format) Protocols

NOTE: The I²C read protocol does not use the Byte Count packet, and the Master will continue receiving Data Bytes until the Master initiates a Stop Condition. See the Command Register on page 13 for additional information regarding the Block Read/Write protocol.

Register Set

The TSL256x is controlled and monitored by sixteen registers (three are reserved) and a command register accessed through the serial interface. These registers provide for a variety of control functions and can be read to determine results of the ADC conversions. The register set is summarized in Table 2.

Table 2. Register Address

ADDRESS	RESISTER NAME	REGISTER FUNCTION
––	COMMAND	Specifies register address
0h	CONTROL	Control of basic functions
1h	TIMING	Integration time/gain control
2h	THRESHLOWLOW	Low byte of low interrupt threshold
3h	THRESHLOWHIGH	High byte of low interrupt threshold
4h	THRESHHIGHLOW	Low byte of high interrupt threshold
5h	THRESHHIGHHIGH	High byte of high interrupt threshold
6h	INTERRUPT	Interrupt control
7h	––	Reserved
8h	CRC	Factory test — not a user register
9h	––	Reserved
Ah	ID	Part number/ Rev ID
Bh	––	Reserved
Ch	DATA0LOW	Low byte of ADC channel 0
Dh	DATA0HIGH	High byte of ADC channel 0
Eh	DATA1LOW	Low byte of ADC channel 1
Fh	DATA1HIGH	High byte of ADC channel 1

The mechanics of accessing a specific register depends on the specific SMB protocol used. Refer to the section on SMBus protocols. In general, the COMMAND register is written first to specify the specific control/status register for following read/write operations.

Command Register

The command register specifies the address of the target register for subsequent read and write operations. The Send Byte protocol is used to configure the COMMAND register. The command register contains eight bits as described in Table 3. The command register defaults to 00h at power on.

Table 3. Command Register

	7	6	5	4	3	2	1	0	
	CMD	CLEAR	WORD	BLOCK	ADDRESS				COMMAND
Reset Value:	0	0	0	0	0	0	0	0	

FIELD	BIT	DESCRIPTION
CMD	7	Select command register. Must write as 1.
CLEAR	6	Interrupt clear. Clears any pending interrupt. This bit is a write-one-to-clear bit. It is self clearing.
WORD	5	SMB Write/Read Word Protocol. 1 indicates that this SMB transaction is using either the SMB Write Word or Read Word protocol.
BLOCK	4	Block Write/Read Protocol. 1 indicates that this transaction is using either the Block Write or the Block Read protocol. See Note below.
ADDRESS	3:0	Register Address. This field selects the specific control or status register for following write and read commands according to Table 2.

NOTE: An I^2C block transaction will continue until the Master sends a stop condition. See Figure 16 and Figure 17. Unlike the I2C protocol, the SMBus read/write protocol requires a Byte Count. All four ADC Channel Data Registers (Ch through Fh) can be read simultaneously in a single SMBus transaction. This is the only 32-bit data block supported by the TSL2560 SMBus protocol. The BLOCK bit must be set to 1, and a read condition should be initiated with a COMMAND CODE of 9Bh. By using a COMMAND CODE of 9Bh during an SMBus Block Read Protocol, the TSL2560 device will automatically insert the appropriate Byte Count (Byte Count = 4) as illustrated in Figure 17. A write condition should not be used in conjunction with the Bh register.

Control Register (0h)

The CONTROL register contains two bits and is primarily used to power the TSL256x device up and down as shown in Table 4.

Table 4. Control Register

	7	6	5	4	3	2	1	0	
0h	Resv	Resv	Resv	Resv	Resv	Resv	POWER		CONTROL
Reset Value:	0	0	0	0	0	0	0	0	

FIELD	BIT	DESCRIPTION
Resv	7:2	Reserved. Write as 0.
POWER	1:0	Power up/power down. By writing a 03h to this register, the device is powered up. By writing a 00h to this register, the device is powered down. ***NOTE:*** *If a value of 03h is written, the value returned during a read cycle will be 03h. This feature can be used to verify that the device is communicating properly.*

Timing Register (1h)

The TIMING register controls both the integration time and the gain of the ADC channels. A common set of control bits is provided that controls both ADC channels. The TIMING register defaults to 02h at power on.

Table 5. Timing Register

	7	6	5	4	3	2	1	0	
1h	Resv	Resv	Resv	GAIN	Manual	Resv	INTEG		TIMING
Reset Value:	0	0	0	0	0	0	1	0	

FIELD	BIT	DESCRIPTION
Resv	7–5	Reserved. Write as 0.
GAIN	4	Switches gain between low gain and high gain modes. Writing a 0 selects low gain (1×); writing a 1 selects high gain (16×).
Manual	3	Manual timing control. Writing a 1 begins an integration cycle. Writing a 0 stops an integration cycle. ***NOTE:*** *This field only has meaning when INTEG = 11. It is ignored at all other times.*
Resv	2	Reserved. Write as 0.
INTEG	1:0	Integrate time. This field selects the integration time for each conversion.

Integration time is dependent on the INTEG FIELD VALUE and the internal clock frequency. Nominal integration times and respective scaling between integration times scale proportionally as shown in Table 6. See Note 5 and Note 6 on page 5 for detailed information regarding how the scale values were obtained; see page 22 for further information on how to calculate lux.

Table 6. Integration Time

INTEG FIELD VALUE	SCALE	NOMINAL INTEGRATION TIME
00	0.034	13.7 ms
01	0.252	101 ms
10	1	402 ms
11	––	N/A

The manual timing control feature is used to manually start and stop the integration time period. If a particular integration time period is required that is not listed in Table 6, then this feature can be used. For example, the manual timing control can be used to synchronize the TSL256x device with an external light source (e.g. LED). A start command to begin integration can be initiated by writing a 1 to this bit field. Correspondingly, the integration can be stopped by simply writing a 0 to the same bit field.

Interrupt Threshold Register (2h – 5h)

The interrupt threshold registers store the values to be used as the high and low trigger points for the comparison function for interrupt generation. If the value generated by channel 0 crosses below or is equal to the low threshold specified, an interrupt is asserted on the interrupt pin. If the value generated by channel 0 crosses above the high threshold specified, an interrupt is asserted on the interrupt pin. Registers THRESHLOWLOW and THRESHLOWHIGH provide the low byte and high byte, respectively, of the lower interrupt threshold. Registers THRESHHIGHLOW and THRESHHIGHHIGH provide the low and high bytes, respectively, of the upper interrupt threshold. The high and low bytes from each set of registers are combined to form a 16-bit threshold value. The interrupt threshold registers default to 00h on power up.

Table 7. Interrupt Threshold Register

REGISTER	ADDRESS	BITS	DESCRIPTION
THRESHLOWLOW	2h	7:0	ADC channel 0 lower byte of the low threshold
THRESHLOWHIGH	3h	7:0	ADC channel 0 upper byte of the low threshold
THRESHHIGHLOW	4h	7:0	ADC channel 0 lower byte of the high threshold
THRESHHIGHHIGH	5h	7:0	ADC channel 0 upper byte of the high threshold

NOTE: Since two 8-bit values are combined for a single 16-bit value for each of the high and low interrupt thresholds, the Send Byte protocol should not be used to write to these registers. Any values transferred by the Send Byte protocol with the MSB set would be interpreted as the COMMAND field and stored as an address for subsequent read/write operations and not as the interrupt threshold information as desired. The Write Word protocol should be used to write byte-paired registers. For example, the THRESHLOWLOW and THRESHLOWHIGH registers (as well as the THRESHHIGHLOW and THRESHHIGHHIGH registers) can be written together to set the 16-bit ADC value in a single transaction.

Interrupt Control Register (6h)

The INTERRUPT register controls the extensive interrupt capabilities of the TSL256x. The TSL256x permits both SMB-Alert style interrupts as well as traditional level-style interrupts. The interrupt persist bit field (PERSIST) provides control over when interrupts occur. A value of 0 causes an interrupt to occur after every integration cycle regardless of the threshold settings. A value of 1 results in an interrupt after one integration time period outside the threshold window. A value of *N* (where *N* is 2 through15) results in an interrupt only if the value remains outside the threshold window for *N* consecutive integration cycles. For example, if *N* is equal to 10 and the integration time is 402 ms, then the total time is approximately 4 seconds.

When a level Interrupt is selected, an interrupt is generated whenever the last conversion results in a value outside of the programmed threshold window. The interrupt is active-low and remains asserted until cleared by writing the COMMAND register with the CLEAR bit set.

In SMBAlert mode, the interrupt is similar to the traditional level style and the interrupt line is asserted low. To clear the interrupt, the host responds to the SMBAlert by performing a modified Receive Byte operation, in which the Alert Response Address (ARA) is placed in the slave address field, and the TSL256x that generated the interrupt responds by returning its own address in the seven most significant bits of the receive data byte. If more than one device connected on the bus has pulled the SMBAlert line low, the highest priority (lowest address) device will win communication rights via standard arbitration during the slave address transfer. If the device loses this arbitration, the interrupt will not be cleared. The Alert Response Address is 0Ch.

When INTR = 11, the interrupt is generated immediately following the SMBus write operation. Operation then behaves in an SMBAlert mode, and the *software set* interrupt may be cleared by an SMBAlert cycle.

NOTE: Interrupts are based on the value of Channel 0 only.

Table 8. Interrupt Control Register

	7	6	5	4	3	2	1	0	
6h	Resv	Resv	INTR		PERSIST				INTERRUPT
Reset Value:	0	0	0	0	0	0	0	0	

FIELD	BITS	DESCRIPTION
Resv	7:6	Reserved. Write as 0.
INTR	5:4	INTR Control Select. This field determines mode of interrupt logic according to Table 9, below.
PERSIST	3:0	Interrupt persistence. Controls rate of interrupts to the host processor as shown in Table 10, below.

Table 9. Interrupt Control Select

INTR FIELD VALUE	READ VALUE
00	Interrupt output disabled
01	Level Interrupt
10	SMBAlert compliant
11	Test Mode: Sets interrupt and functions as mode 10

NOTE: Field value of 11 may be used to test interrupt connectivity in a system or to assist in debugging interrupt service routine software.

Table 10. Interrupt Persistence Select

PERSIST FIELD VALUE	INTERRUPT PERSIST FUNCTION
0000	Every ADC cycle generates interrupt
0001	Any value outside of threshold range
0010	2 integration time periods out of range
0011	3 integration time periods out of range
0100	4 integration time periods out of range
0101	5 integration time periods out of range
0110	6 integration time periods out of range
0111	7 integration time periods out of range
1000	8 integration time periods out of range
1001	9 integration time periods out of range
1010	10 integration time periods out of range
1011	11 integration time periods out of range
1100	12 integration time periods out of range
1101	13 integration time periods out of range
1110	14 integration time periods out of range
1111	15 integration time periods out of range

ID Register (Ah)

The ID register provides the value for both the part number and silicon revision number for that part number. It is a read-only register, whose value never changes.

Table 11. ID Register

	7	6	5	4	3	2	1	0	
Ah	PARTNO				REVNO				ID
Reset Value:	–	–	–	–	–	–	–	–	

FIELD	BITS	DESCRIPTION
PARTNO	7:4	Part Number Identification: field value 0000 = TSL2560, field value 0001 = TSL2561
REVNO	3:0	Revision number identification

ADC Channel Data Registers (Ch – Fh)

The ADC channel data are expressed as 16-bit values spread across two registers. The ADC channel 0 data registers, DATA0LOW and DATA0HIGH provide the lower and upper bytes, respectively, of the ADC value of channel 0. Registers DATA1LOW and DATA1HIGH provide the lower and upper bytes, respectively, of the ADC value of channel 1. All channel data registers are read-only and default to 00h on power up.

Table 12. ADC Channel Data Registers

REGISTER	ADDRESS	BITS	DESCRIPTION
DATA0LOW	Ch	7:0	ADC channel 0 lower byte
DATA0HIGH	Dh	7:0	ADC channel 0 upper byte
DATA1LOW	Eh	7:0	ADC channel 1 lower byte
DATA1HIGH	Fh	7:0	ADC channel 1 upper byte

The upper byte data registers can only be read following a read to the corresponding lower byte register. When the lower byte register is read, the upper eight bits are strobed into a shadow register, which is read by a subsequent read to the upper byte. The upper register will read the correct value even if additional ADC integration cycles end between the reading of the lower and upper registers.

NOTE: The Read Word protocol can be used to read byte-paired registers. For example, the DATA0LOW and DATA0HIGH registers (as well as the DATA1LOW and DATA1HIGH registers) may be read together to obtain the 16-bit ADC value in a single transaction

APPLICATION INFORMATION: SOFTWARE

Basic Operation

After applying V_{DD}, the device will initially be in the power-down state. To operate the device, issue a command to access the CONTROL register followed by the data value 03h to power up the device. At this point, both ADC channels will begin a conversion at the default integration time of 400 ms. After 400 ms, the conversion results will be available in the DATA0 and DATA1 registers. Use the following pseudo code to read the data registers:

```
// Read ADC Channels Using Read Word Protocol – RECOMMENDED
      Address = 0x39                              //Slave addr – also 0x29 or 0x49

      //Address the Ch0 lower data register and configure for Read Word
      Command = 0xAC                              //Set Command bit and Word bit

      //Reads two bytes from sequential registers 0x0C and 0x0D
      //Results are returned in DataLow and DataHigh variables
      ReadWord (Address, Command, DataLow, DataHigh)
      Channel0 = 256 * DataHigh + DataLow

      //Address the Ch1 lower data register and configure for Read Word
      Command = 0xAE                              //Set bit fields 7 and 5

      //Reads two bytes from sequential registers 0x0E and 0x0F
      //Results are returned in DataLow and DataHigh variables
      ReadWord (Address, Command, DataLow, DataHigh)
      Channel1 = 256 * DataHigh + DataLow         //Shift DataHigh to upper byte


// Read ADC Channels Using Read Byte Protocol
      Address = 0x39                              //Slave addr – also 0x29 or 0x49
      Command = 0x8C                              //Address the Ch0 lower data register
      ReadByte (Address, Command, DataLow)        //Result returned in DataLow
      Command = 0x8D                              //Address the Ch0 upper data register
      ReadByte (Address, Command, DataHigh)       //Result returned in DataHigh
      Channel0 = 256 * DataHigh + DataLow         //Shift DataHigh to upper byte

      Command = 0x8E                              //Address the Ch1 lower data register
      ReadByte (Address, Command, DataLow)        //Result returned in DataLow
      Command = 0x8F                              //Address the Ch1 upper data register
      ReadByte (Address, Command, DataHigh)       //Result returned in DataHigh
      Channel1 = 256 * DataHigh + DataLow         //Shift DataHigh to upper byte
```

APPLICATION INFORMATION: SOFTWARE

Configuring the Timing Register

The command, timing, and control registers are initialized to default values on power up. Setting these registers to the desired values would be part of a normal initialization or setup procedure. In addition, to maximize the performance of the device under various conditions, the integration time and gain may be changed often during operation. The following pseudo code illustrates a procedure for setting up the timing register for various options:

```
// Set up Timing Register
       //Low Gain (1x), integration time of 402ms (default value)
       Address = 0x39
       Command = 0x81
       Data = 0x02
       WriteByte(Address, Command, Data)

       //Low Gain (1x), integration time of 101ms
       Data = 0x01
       WriteByte(Address, Command, Data)

       //Low Gain (1x), integration time of 13.7ms
       Data = 0x00
       WriteByte(Address, Command, Data)

       //High Gain (16x), integration time of 101ms
       Data = 0x11
       WriteByte(Address, Command, Data)

//Read data registers (see Basic Operation example)


//Perform Manual Integration
       //Set up for manual integration with Gain of 1x
       Data = 0x03
       //Set manual integration mode - device stops converting
       WriteByte(Address, Command, Data)

       //Begin integration period
       Data = 0x0B
       WriteByte(Address, Command, Data)

       //Integrate for 50ms
       Sleep (50)                                  //Wait for 50ms

       //Stop integrating
       Data = 0x03
       WriteByte(Address, Command, Data)

//Read data registers (see Basic Operation example)
```

APPLICATION INFORMATION: SOFTWARE

Interrupts

The interrupt feature of the TSL256x device simplifies and improves system efficiency by eliminating the need to poll the sensor for a light intensity value. Interrupt styles are determined by the INTR field in the Interrupt Register. The interrupt feature may be disabled by writing a field value of 00h to the Interrupt Control Register so that polling can be performed.

The versatility of the interrupt feature provides many options for interrupt configuration and usage. The primary purpose of the interrupt function is to provide a meaningful change in light intensity. However, it also be used as an end-of-conversion signal. The concept of a *meaningful change* can be defined by the user both in terms of light intensity and time, or persistence, of that change in intensity. The TSL256x device implements two 16-bit-wide interrupt threshold registers that allow the user to define a threshold above and below the current light level. An interrupt will then be generated when the value of a conversion exceeds either of these limits. For simplicity of programming, the threshold comparison is accomplished only with Channel 0. This simplifies calculation of thresholds that are based, for example, on a percent of the current light level. It is adequate to use only one channel when calculating light intensity differences since, for a given light source, the channel 0 and channel 1 values are linearly proportional to each other and thus both values scale linearly with light intensity.

To further control when an interrupt occurs, the TSL256x device provides an interrupt persistence feature. This feature allows the user to specify a number of conversion cycles for which a light intensity exceeding either interrupt threshold must persist before actually generating an interrupt. This can be used to prevent transient changes in light intensity from generating an unwanted interrupt. With a value of 1, an interrupt occurs immediately whenever either threshold is exceeded. With values of *N*, where *N* can range from 2 to 15, *N* consecutive conversions must result in values outside the interrupt window for an interrupt to be generated. For example, if *N* is equal to 10 and the integration time is 402 ms, then an interrupt will not be generated unless the light level persists for more than 4 seconds outside the threshold.

Two different interrupt styles are available: Level and SMBus Alert. The difference between these two interrupt styles is how they are cleared. Both result in the interrupt line going active low and remaining low until the interrupt is cleared. A level style interrupt is cleared by setting the CLEAR bit (bit 6) in the COMMAND register. The SMBus Alert style interrupt is cleared by an Alert Response as described in the Interrupt Control Register section and SMBus specification.

To configure the interrupt as an end-of-conversion signal, the interrupt PERSIST field is set to 0. Either Level or SMBus Alert style can be used. An interrupt will be generated upon completion of each conversion. The interrupt threshold registers are ignored. The following example illustrates the configuration of a level interrupt:

```
// Set up end-of-conversion interrupt, Level style
      Address = 0x39                          //Slave addr also 0x29 or 0x49
      Command = 0x86                          //Address Interrupt Register
      Data = 0x10                             //Level style, every ADC cycle
      WriteByte(Address, Command, Data)
```

APPLICATION INFORMATION: SOFTWARE

The following example pseudo code illustrates the configuration of an SMB Alert style interrupt when the light intensity changes 20% from the current value, and persists for 3 conversion cycles:

```
// Read current light level
       Address = 0x39                              //Slave addr also 0x29 or 0x49
       Command = 0xAC                              //Set Command bit and Word bit
       ReadWord (Address, Command, DataLow, DataHigh)
       Channel0 = (256 * DataHigh) + DataLow

       //Calculate upper and lower thresholds
       T_Upper = Channel0 + (0.2 * Channel0)
       T_Lower = Channel0 - (0.2 * Channel0)

       //Write the lower threshold register
       Command = 0xA2                              //Addr lower threshold reg, set Word Bit
       WriteWord (Address, Command, T_Lower.LoByte, T_Lower.HiByte)

       //Write the upper threshold register
       Command = 0xA4                              //Addr upper threshold reg, set Word bit
       WriteWord (Address, Command, T_Upper.LoByte, T_Upper.HiByte)

       //Enable interrupt
       Command = 0x86                              //Address interrupt register
       Data = 0x23                                 //SMBAlert style, PERSIST = 3
       WriteByte(Address, Command, Data)
```

In order to generate an interrupt on demand during system test or debug, a test mode (INTR = 11) can be used. The following example illustrates how to generate an interrupt on demand:

```
// Generate an interrupt
       Address = 0x39                              //Slave addr also 0x29 or 0x49
       Command = 0x86                              //Address Interrupt register
       Data = 0x30                                 //Test interrupt
       WriteByte(Address, Command, Data)

       //Interrupt line should now be low
```

APPLICATION INFORMATION: SOFTWARE

Calculating Lux

The TSL256x is intended for use in ambient light detection applications such as display backlight control, where adjustments are made to display brightness or contrast based on the brightness of the ambient light, as perceived by the human eye. Conventional silicon detectors respond strongly to infrared light, which the human eye does not see. This can lead to significant error when the infrared content of the ambient light is high, such as with incandescent lighting, due to the difference between the silicon detector response and the brightness perceived by the human eye.

This problem is overcome in the TSL256x through the use of two photodiodes. One of the photodiodes (channel 0) is sensitive to both visible and infrared light, while the second photodiode (channel 1) is sensitive primarily to infrared light. An integrating ADC converts the photodiode currents to digital outputs. Channel 1 digital output is used to compensate for the effect of the infrared component of light on the channel 0 digital output. The ADC digital outputs from the two channels are used in a formula to obtain a value that approximates the human eye response in the commonly used Illuminance unit of Lux:

CS Package

For $0 < CH1/CH0 \leq 0.52$	$Lux = 0.0315 \times CH0 - 0.0593 \times CH0 \times ((CH1/CH0)^{1.4})$
For $0.52 < CH1/CH0 \leq 0.65$	$Lux = 0.0229 \times CH0 - 0.0291 \times CH1$
For $0.65 < CH1/CH0 \leq 0.80$	$Lux = 0.0157 \times CH0 - 0.0180 \times CH1$
For $0.80 < CH1/CH0 \leq 1.30$	$Lux = 0.00338 \times CH0 - 0.00260 \times CH1$
For $CH1/CH0 > 1.30$	$Lux = 0$

T, FN, and CL Package

For $0 < CH1/CH0 \leq 0.50$	$Lux = 0.0304 \times CH0 - 0.062 \times CH0 \times ((CH1/CH0)^{1.4})$
For $0.50 < CH1/CH0 \leq 0.61$	$Lux = 0.0224 \times CH0 - 0.031 \times CH1$
For $0.61 < CH1/CH0 \leq 0.80$	$Lux = 0.0128 \times CH0 - 0.0153 \times CH1$
For $0.80 < CH1/CH0 \leq 1.30$	$Lux = 0.00146 \times CH0 - 0.00112 \times CH1$
For $CH1/CH0 > 1.30$	$Lux = 0$

The formulas shown above were obtained by optical testing with fluorescent and incandescent light sources, and apply only to open-air applications. Optical apertures (e.g. light pipes) will affect the incident light on the device.

Simplified Lux Calculation

Below is the argument and return value including source code (shown on following page) for calculating lux. The source code is intended for embedded and/or microcontroller applications. Two individual code sets are provided, one for the T, FN, and CL packages, and one for the CS package. All floating point arithmetic operations have been eliminated since embedded controllers and microcontrollers generally do not support these types of operations. Since floating point has been removed, scaling must be performed prior to calculating illuminance if the integration time is not 402 ms and/or if the gain is not 16× as denoted in the source code on the following pages. This sequence scales first to mitigate rounding errors induced by decimal math.

```
extern unsigned int CalculateLux(unsigned int iGain, unsigned int tInt, unsigned int
    ch0, unsigned int ch1, int iType)
```

```
//**************************************************************************
//
//  Copyright © 2004-2005 TAOS, Inc.
//
//  THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
//  KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
//  IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR
//  PURPOSE.
//
//     Module Name:
//           lux.cpp
//
//**************************************************************************

#define LUX_SCALE   14     // scale by 2^14
#define RATIO_SCALE 9      // scale ratio by 2^9

//--------------------------------------------------
// Integration time scaling factors
//--------------------------------------------------

#define CH_SCALE          10     // scale channel values by 2^10
#define CHSCALE_TINT0     0x7517 // 322/11 * 2^CH_SCALE
#define CHSCALE_TINT1     0x0fe7 // 322/81 * 2^CH_SCALE

//--------------------------------------------------
// T, FN, and CL Package coefficients
//--------------------------------------------------
// For Ch1/Ch0=0.00 to 0.50
//           Lux/Ch0=0.0304-0.062*((Ch1/Ch0)^1.4)
//           piecewise approximation
//                 For Ch1/Ch0=0.00 to 0.125:
//                       Lux/Ch0=0.0304-0.0272*(Ch1/Ch0)
//
//                 For Ch1/Ch0=0.125 to 0.250:
//                       Lux/Ch0=0.0325-0.0440*(Ch1/Ch0)
//
//                 For Ch1/Ch0=0.250 to 0.375:
//                       Lux/Ch0=0.0351-0.0544*(Ch1/Ch0)
//
//                 For Ch1/Ch0=0.375 to 0.50:
//                       Lux/Ch0=0.0381-0.0624*(Ch1/Ch0)
//
// For Ch1/Ch0=0.50 to 0.61:
//           Lux/Ch0=0.0224-0.031*(Ch1/Ch0)
//
// For Ch1/Ch0=0.61 to 0.80:
//           Lux/Ch0=0.0128-0.0153*(Ch1/Ch0)
//
// For Ch1/Ch0=0.80 to 1.30:
//           Lux/Ch0=0.00146-0.00112*(Ch1/Ch0)
//
// For Ch1/Ch0>1.3:
//           Lux/Ch0=0
//--------------------------------------------------
#define K1T   0x0040       // 0.125 * 2^RATIO_SCALE
#define B1T   0x01f2       // 0.0304 * 2^LUX_SCALE
#define M1T   0x01be       // 0.0272 * 2^LUX_SCALE

#define K2T   0x0080       // 0.250 * 2^RATIO_SCALE
```

```
#define B2T   0x0214        // 0.0325 * 2^LUX_SCALE
#define M2T   0x02d1        // 0.0440 * 2^LUX_SCALE

#define K3T   0x00c0        // 0.375 * 2^RATIO_SCALE
#define B3T   0x023f        // 0.0351 * 2^LUX_SCALE
#define M3T   0x037b        // 0.0544 * 2^LUX_SCALE

#define K4T   0x0100        // 0.50 * 2^RATIO_SCALE
#define B4T   0x0270        // 0.0381 * 2^LUX_SCALE
#define M4T   0x03fe        // 0.0624 * 2^LUX_SCALE
#define K5T   0x0138        // 0.61 * 2^RATIO_SCALE
#define B5T   0x016f        // 0.0224 * 2^LUX_SCALE
#define M5T   0x01fc        // 0.0310 * 2^LUX_SCALE

#define K6T   0x019a        // 0.80 * 2^RATIO_SCALE
#define B6T   0x00d2        // 0.0128 * 2^LUX_SCALE
#define M6T   0x00fb        // 0.0153 * 2^LUX_SCALE

#define K7T   0x029a        // 1.3 * 2^RATIO_SCALE
#define B7T   0x0018        // 0.00146 * 2^LUX_SCALE
#define M7T   0x0012        // 0.00112 * 2^LUX_SCALE

#define K8T   0x029a        // 1.3 * 2^RATIO_SCALE
#define B8T   0x0000        // 0.000 * 2^LUX_SCALE
#define M8T   0x0000        // 0.000 * 2^LUX_SCALE

//--------------------------------------------------
// CS package coefficients
//--------------------------------------------------
// For 0 <= Ch1/Ch0 <= 0.52
//           Lux/Ch0 = 0.0315-0.0593*((Ch1/Ch0)^1.4)
//           piecewise approximation
//                 For 0 <= Ch1/Ch0 <= 0.13
//                       Lux/Ch0 = 0.0315-0.0262*(Ch1/Ch0)
//                 For 0.13 <= Ch1/Ch0 <= 0.26
//                       Lux/Ch0 = 0.0337-0.0430*(Ch1/Ch0)
//                 For 0.26 <= Ch1/Ch0 <= 0.39
//                       Lux/Ch0 = 0.0363-0.0529*(Ch1/Ch0)
//                 For 0.39 <= Ch1/Ch0 <= 0.52
//                       Lux/Ch0 = 0.0392-0.0605*(Ch1/Ch0)
// For 0.52 < Ch1/Ch0 <= 0.65
//     Lux/Ch0 = 0.0229-0.0291*(Ch1/Ch0)
// For 0.65 < Ch1/Ch0 <= 0.80
//     Lux/Ch0 = 0.00157-0.00180*(Ch1/Ch0)
// For 0.80 < Ch1/Ch0 <= 1.30
//     Lux/Ch0 = 0.00338-0.00260*(Ch1/Ch0)
// For Ch1/Ch0 > 1.30
//     Lux = 0
//--------------------------------------------------
#define K1C   0x0043 // 0.130 * 2^RATIO_SCALE
#define B1C   0x0204 // 0.0315 * 2^LUX_SCALE
#define M1C   0x01ad // 0.0262 * 2^LUX_SCALE

#define K2C   0x0085 // 0.260 * 2^RATIO_SCALE
#define B2C   0x0228 // 0.0337 * 2^LUX_SCALE
#define M2C   0x02c1 // 0.0430 * 2^LUX_SCALE

#define K3C   0x00c8 // 0.390 * 2^RATIO_SCALE
#define B3C   0x0253 // 0.0363 * 2^LUX_SCALE
#define M3C   0x0363 // 0.0529 * 2^LUX_SCALE
```

```
#define K4C   0x010a // 0.520 * 2^RATIO_SCALE
#define B4C   0x0282 // 0.0392 * 2^LUX_SCALE
#define M4C   0x03df // 0.0605 * 2^LUX_SCALE

#define K5C   0x014d // 0.65 * 2^RATIO_SCALE
#define B5C   0x0177 // 0.0229 * 2^LUX_SCALE
#define M5C   0x01dd // 0.0291 * 2^LUX_SCALE

#define K6C   0x019a // 0.80 * 2^RATIO_SCALE
#define B6C   0x0101 // 0.0157 * 2^LUX_SCALE
#define M6C   0x0127 // 0.0180 * 2^LUX_SCALE

#define K7C   0x029a // 1.3 * 2^RATIO_SCALE
#define B7C   0x0037 // 0.00338 * 2^LUX_SCALE
#define M7C   0x002b // 0.00260 * 2^LUX_SCALE

#define K8C   0x029a // 1.3 * 2^RATIO_SCALE
#define B8C   0x0000 // 0.000 * 2^LUX_SCALE
#define M8C   0x0000 // 0.000 * 2^LUX_SCALE

// lux equation approximation without floating point calculations
//////////////////////////////////////////////////////////////////////////////
//     Routine:     unsigned int CalculateLux(unsigned int ch0, unsigned int ch0, int iType)
//
//     Description: Calculate the approximate illuminance (lux) given the raw
//                  channel values of the TSL2560. The equation if implemented
//                  as a piece-wise linear approximation.
//
//     Arguments:   unsigned int iGain – gain, where 0:1X, 1:16X
//                  unsigned int tInt – integration time, where 0:13.7mS, 1:100mS, 2:402mS,
//                   3:Manual
//                  unsigned int ch0 – raw channel value from channel 0 of TSL2560
//                  unsigned int ch1 – raw channel value from channel 1 of TSL2560
//                  unsigned int iType – package type (T or CS)
//
//     Return:      unsigned int – the approximate illuminance (lux)
//
//////////////////////////////////////////////////////////////////////////////
unsigned int CalculateLux(unsigned int iGain, unsigned int tInt, unsigned int ch0,
      unsigned int ch1, int iType)
{

      //---------------------------------------------------------------------
      // first, scale the channel values depending on the gain and integration time
      // 16X, 402mS is nominal.
      // scale if integration time is NOT 402 msec
      unsigned long chScale;
      unsigned long channel1;
      unsigned long channel0;
      switch (tInt)
      {
          case 0:    // 13.7 msec
                chScale = CHSCALE_TINT0;
                break;
          case 1:    // 101 msec
                chScale = CHSCALE_TINT1;
                break;
          default:   // assume no scaling
                chScale = (1 << CH_SCALE);
```

```
            break;
}

// scale if gain is NOT 16X
if (!iGain) chScale = chScale << 4;   // scale 1X to 16X

// scale the channel values
channel0 = (ch0 * chScale) >> CH_SCALE;
channel1 = (ch1 * chScale) >> CH_SCALE;
//--------------------------------------------------------------------------

// find the ratio of the channel values (Channel1/Channel0)
// protect against divide by zero
unsigned long ratio1 = 0;
if (channel0 != 0) ratio1 = (channel1 << (RATIO_SCALE+1)) / channel0;

// round the ratio value
unsigned long ratio = (ratio1 + 1) >> 1;

// is ratio <= eachBreak ?
unsigned int b, m;
switch (iType)
{



    case 0: // T, FN and CL package
          if ((ratio >= 0) && (ratio <= K1T))
                {b=B1T; m=M1T;}
          else if (ratio <= K2T)
                {b=B2T; m=M2T;}
          else if (ratio <= K3T)
                {b=B3T; m=M3T;}
          else if (ratio <= K4T)
                {b=B4T; m=M4T;}
          else if (ratio <= K5T)
                {b=B5T; m=M5T;}
          else if (ratio <= K6T)
                {b=B6T; m=M6T;}
          else if (ratio <= K7T)
                {b=B7T; m=M7T;}
          else if (ratio > K8T)
                {b=B8T; m=M8T;}
          break;

    case 1:// CS package
          if ((ratio >= 0) && (ratio <= K1C))
                {b=B1C; m=M1C;}
          else if (ratio <= K2C)
                {b=B2C; m=M2C;}
          else if (ratio <= K3C)
                {b=B3C; m=M3C;}
          else if (ratio <= K4C)
                {b=B4C; m=M4C;}
          else if (ratio <= K5C)
                {b=B5C; m=M5C;}
          else if (ratio <= K6C)
                {b=B6C; m=M6C;}
          else if (ratio <= K7C)
                {b=B7C; m=M7C;}
```

```
                else if (ratio > K8C)
                      {b=B8C; m=M8C;}
                break;
        }

        unsigned long temp;
        temp = ((channel0 * b) - (channel1 * m));

        // do not allow negative lux value
        if (temp < 0) temp = 0;

        // round lsb (2^(LUX_SCALE-1))
        temp += (1 << (LUX_SCALE-1));

        // strip off fractional portion
        unsigned long lux = temp >> LUX_SCALE;

        return(lux);

}
```

APPLICATION INFORMATION: HARDWARE

Power Supply Decoupling and Application Hardware Circuit

The power supply lines must be decoupled with a 0.1 μF capacitor placed as close to the device package as possible (Figure 18). The bypass capacitor should have low effective series resistance (ESR) and low effective series inductance (ESI), such as the common ceramic types, which provide a low impedance path to ground at high frequencies to handle transient currents caused by internal logic switching.

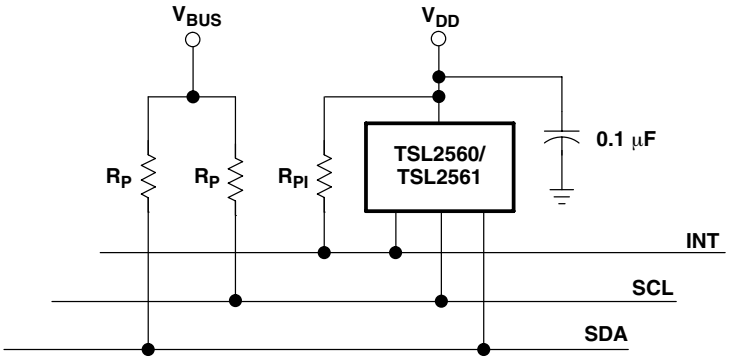


Figure 18. Bus Pull-Up Resistors

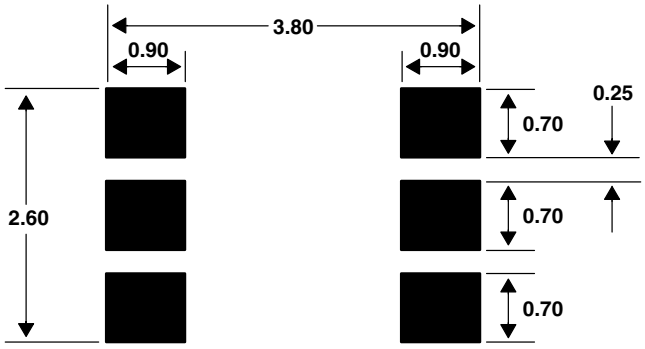
Pull-up resistors (Rp) maintain the SDAH and SCLH lines at a *high* level when the bus is free and ensure the signals are pulled up from a low to a high level within the required rise time. For a complete description of the SMBus maximum and minimum Rp values, please review the SMBus Specification at http://www.smbus.org/specs. For a complete description of I^2C maximum and minimum Rp values, please review the I^2C Specification at http://www.semiconductors.philips.com.

A pull-up resistor (R_{PI}) is also required for the interrupt (INT), which functions as a wired-AND signal in a similar fashion to the SCL and SDA lines. A typical impedance value between 10 kΩ and 100 kΩ can be used. Please note that while Figure 18 shows INT being pulled up to V_{DD}, the interrupt can optionally be pulled up to V_{BUS}.

APPLICATION INFORMATION: HARDWARE

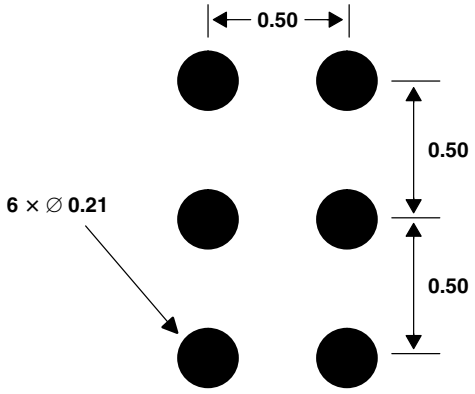
PCB Pad Layout

Suggested PCB pad layout guidelines for the TMB-6 (T) surface mount package, chipscale (CS) package, Dual Flat No-Lead (FN) surface mount package, and ChipLED–6 (CL) surface mount package are shown in Figure 19, Figure 20, Figure 21, and Figure 22.



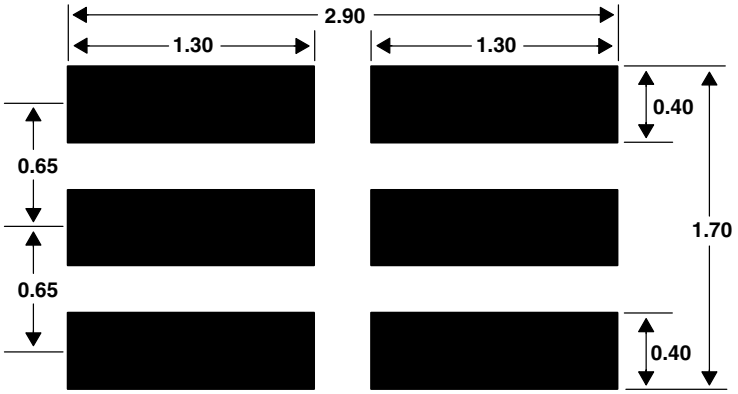
NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.

Figure 19. Suggested T Package PCB Layout



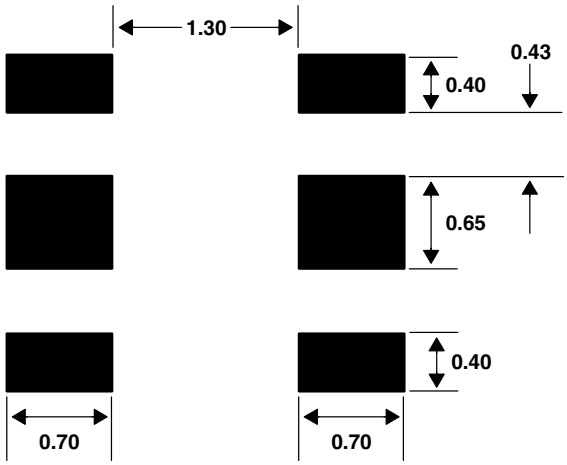
NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.

Figure 20. Suggested CS Package PCB Layout



NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.

Figure 21. Suggested FN Package PCB Layout



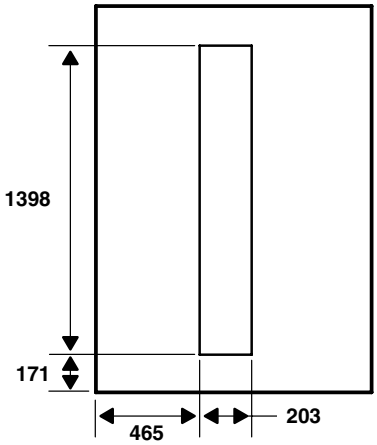
NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.

Figure 22. Suggested CL Package PCB Layout

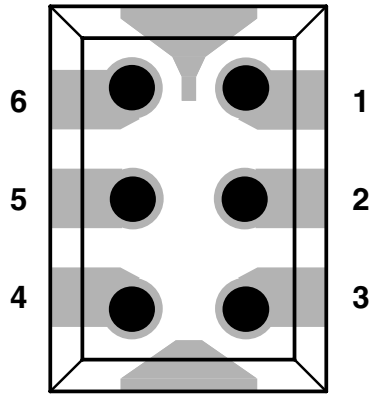
MECHANICAL DATA

PACKAGE CS **Six-Lead Chipscale Device**

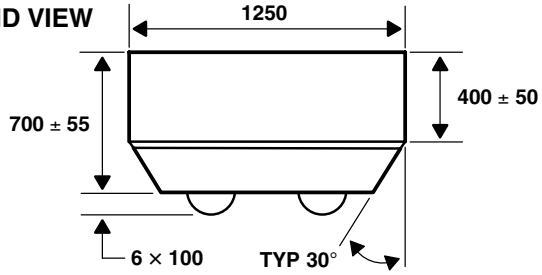
TOP VIEW



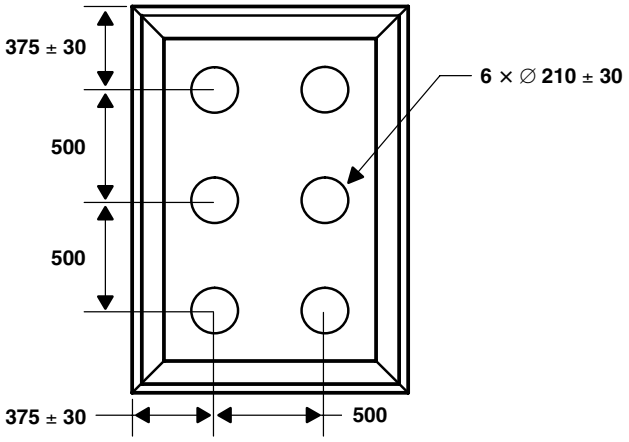
PIN OUT
BOTTOM VIEW



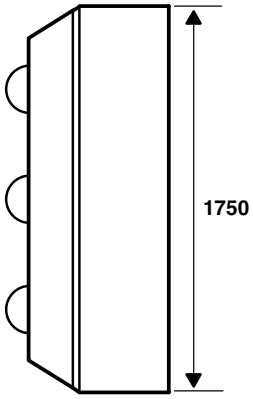
END VIEW



BOTTOM VIEW



SIDE VIEW



Pb
Lead Free

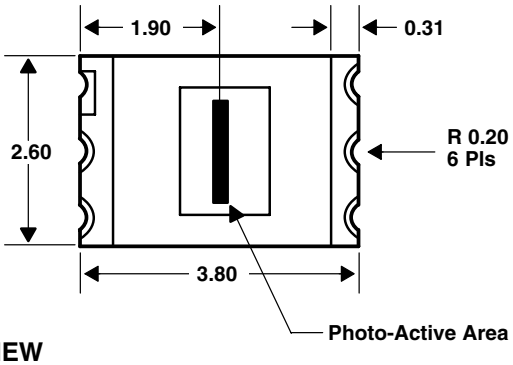
NOTES: A. All linear dimensions are in micrometers. Dimension tolerance is ± 25 µm unless otherwise noted.
B. Solder bumps are formed of Sn (96.5%), Ag (3%), and Cu (0.5%).
C. The top of the photodiode active area is 410 µm below the top surface of the package.
D. The layer above the photodiode is glass and epoxy with an index of refraction of 1.53.
E. This drawing is subject to change without notice.

Figure 23. Package CS — Six-Lead Chipscale Packaging Configuration

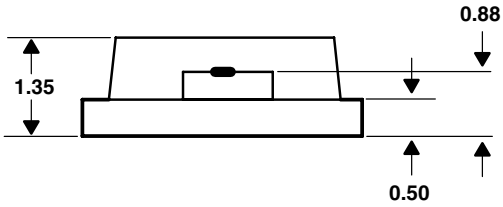
MECHANICAL DATA

PACKAGE TMB-6 **Six-Lead Surface Mount Device**

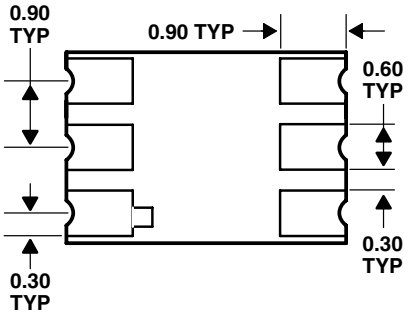
TOP VIEW



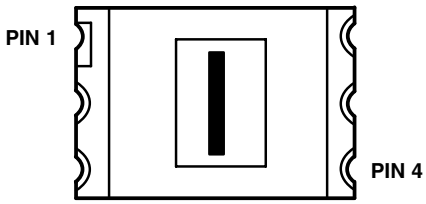
END VIEW



BOTTOM VIEW



TOP VIEW



Pb
Lead Free

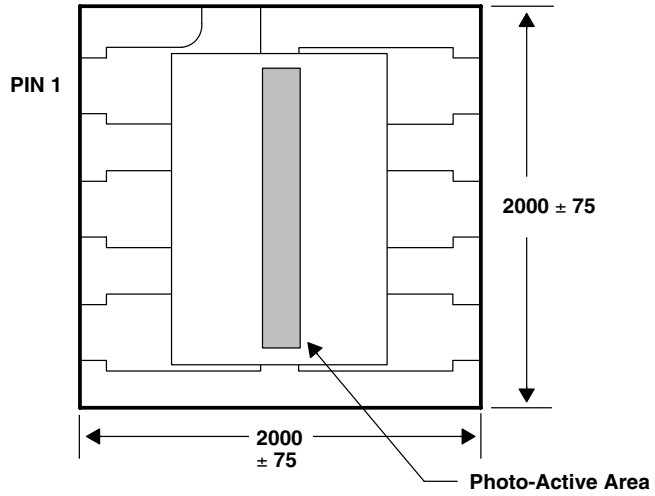
NOTES: A. All linear dimensions are in millimeters. Dimension tolerance is ± 0.20 mm unless otherwise noted.
B. The photo-active area is 1398 μm by 203 μm.
C. Package top surface is molded with an electrically nonconductive clear plastic compound having an index of refraction of 1.55.
D. Contact finish is 0.5 μm minimum of soft gold plated over a 18 μm thick copper foil pattern with a 5 μm to 9 μm nickel barrier.
E. The underside of the package includes copper traces used to connect the pads during package substrate fabrication. Accordingly, exposed traces and vias should not be placed under the footprint of the TMB package in a PCB layout.
F. This package contains no lead (Pb).
G. This drawing is subject to change without notice.

Figure 24. Package T — Six-Lead TMB Plastic Surface Mount Packaging Configuration

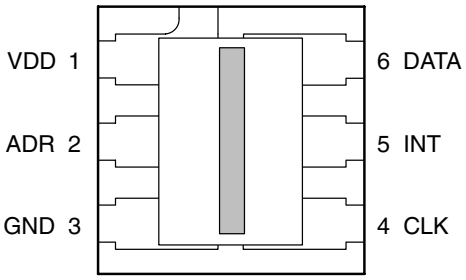
MECHANICAL DATA

PACKAGE FN **Dual Flat No-Lead**

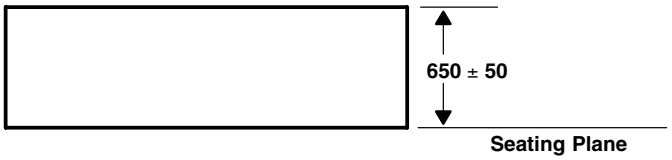
TOP VIEW



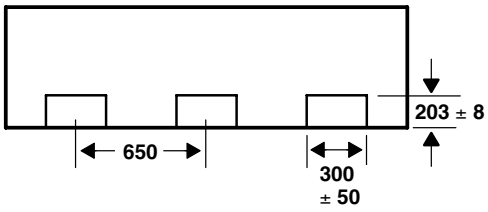
PIN OUT
TOP VIEW



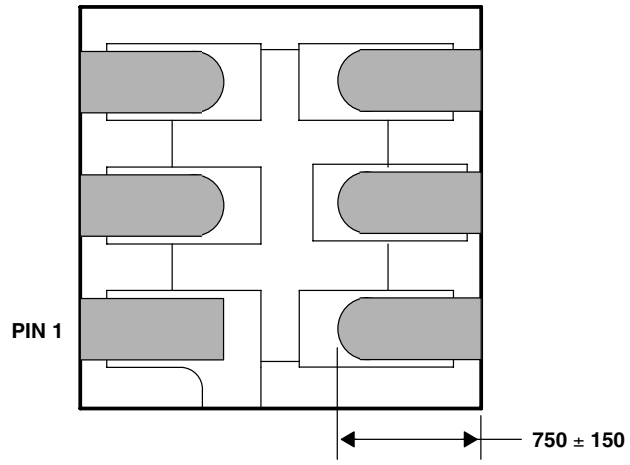
END VIEW



SIDE VIEW



BOTTOM VIEW



Lead Free

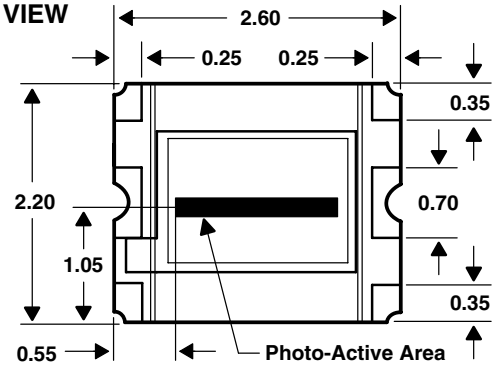
NOTES: A. All linear dimensions are in micrometers. Dimension tolerance is ± 20 µm unless otherwise noted.
B. The photo-active area is 1398 µm by 203 µm.
C. Package top surface is molded with an electrically nonconductive clear plastic compound having an index of refraction of 1.55.
D. Contact finish is copper alloy A194 with pre-plated NiPdAu lead finish.
E. This package contains no lead (Pb).
F. This drawing is subject to change without notice.

Figure 25. Package FN — Dual Flat No-Lead Packaging Configuration

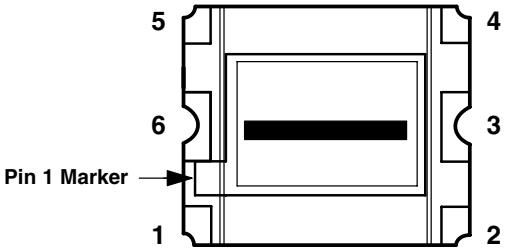
MECHANICAL DATA

PACKAGE CL-6 **Six-Lead Surface Mount Device**

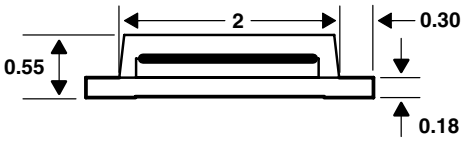
TOP VIEW



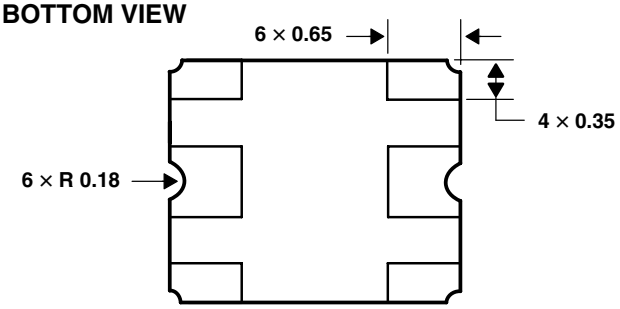
TOP VIEW



SIDE VIEW



BOTTOM VIEW



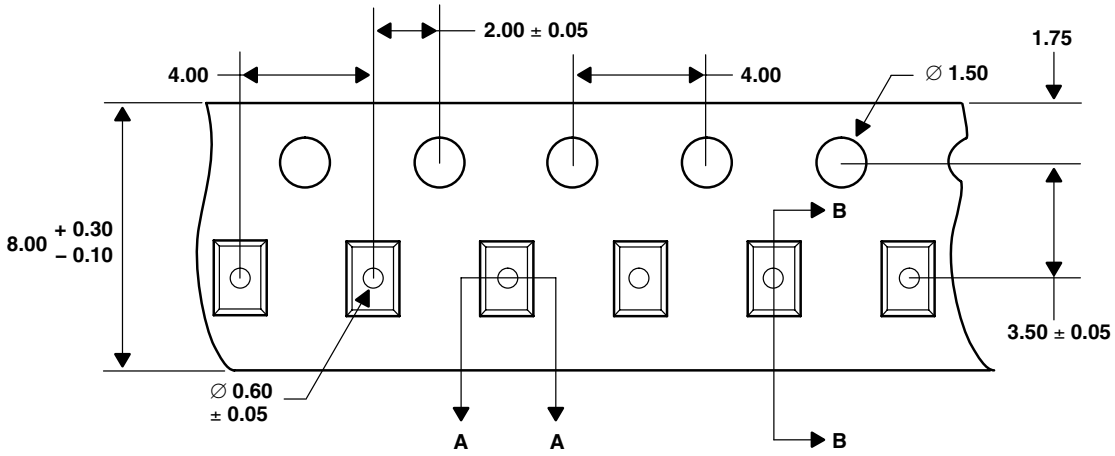
Pb
Lead Free

NOTES: A. All linear dimensions are in millimeters. Dimension tolerance is ± 0.20 mm unless otherwise noted.
B. The photo-active area is 1398 μm by 203 μm.
C. Package top surface is molded with an electrically nonconductive clear plastic compound having an index of refraction of 1.55.
D. Contact finish is 0.1 μm (minimum) to 1.0 μm (maximum) of soft gold plated over a 15 μm (minimum) to 30 μm (maximum) thick copper foil pattern with a 3 μm (minimum) to 15 μm (maximum) nickel barrier.
E. This package contains no lead (Pb).
F. This drawing is subject to change without notice.

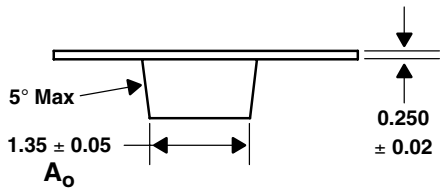
Figure 26. Package CL — Six-Lead ChipLED Plastic Surface Mount Packaging Configuration

MECHANICAL DATA

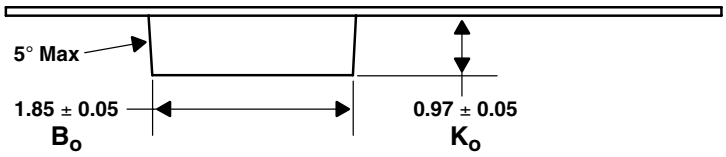
TOP VIEW

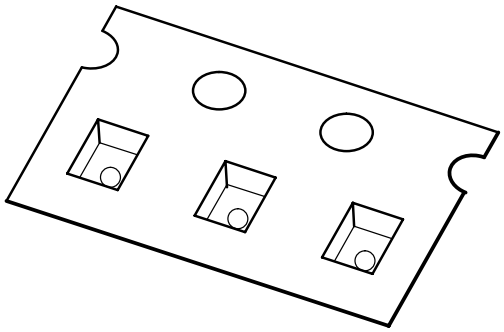


DETAIL A



DETAIL B

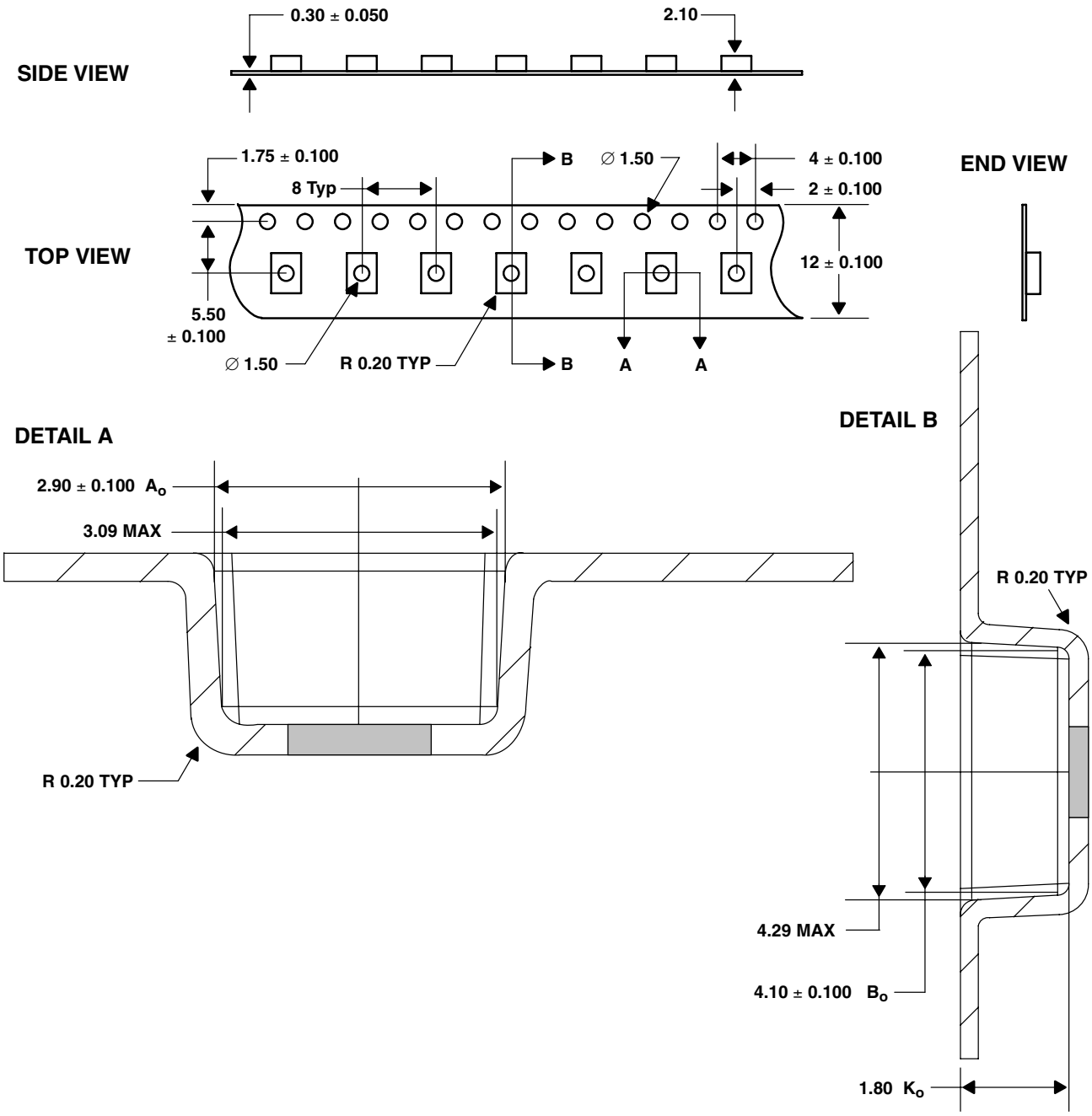




NOTES: A. All linear dimensions are in millimeters. Dimension tolerance is ± 0.10 mm unless otherwise noted.
B. The dimensions on this drawing are for illustrative purposes only. Dimensions of an actual carrier may vary slightly.
C. Symbols on drawing A_o, B_o, and K_o are defined in ANSI EIA Standard 481–B 2001.
D. Each reel is 178 millimeters in diameter and contains 3500 parts.
E. TAOS packaging tape and reel conform to the requirements of EIA Standard 481–B.
F. In accordance with EIA standard, device pin 1 is located next to the sprocket holes in the tape.
G. This drawing is subject to change without notice.

Figure 27. TSL2560/TSL2561 Chipscale Carrier Tape

MECHANICAL DATA

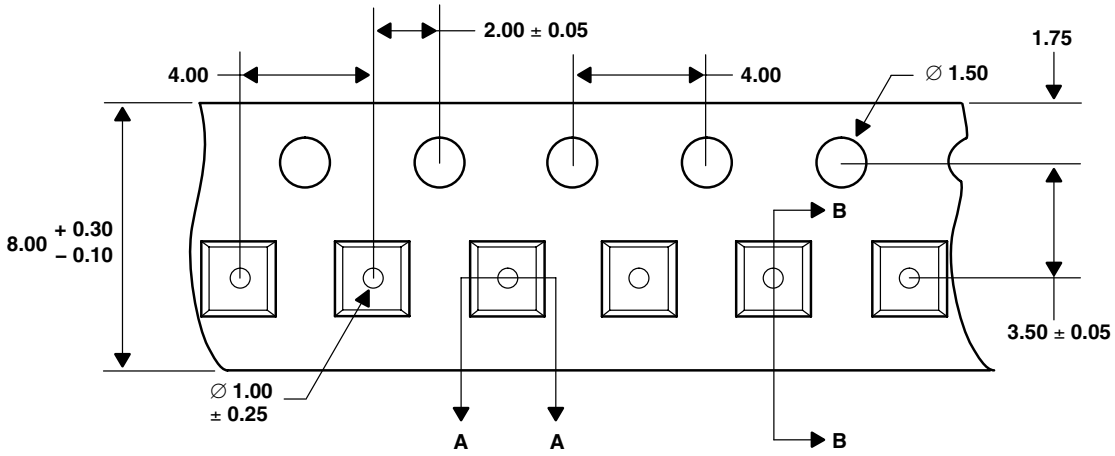


NOTES: A. All linear dimensions are in millimeters.
B. The dimensions on this drawing are for illustrative purposes only. Dimensions of an actual carrier may vary slightly.
C. Symbols on drawing A_o, B_o, and K_o are defined in ANSI EIA Standard 481–B 2001.
D. Each reel is 178 millimeters in diameter and contains 1000 parts.
E. TAOS packaging tape and reel conform to the requirements of EIA Standard 481–B.
F. In accordance with EIA standard, device pin 1 is located next to the sprocket holes in the tape.
G. This drawing is subject to change without notice.

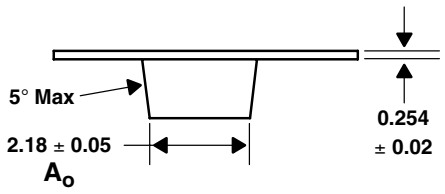
Figure 28. TSL2560/TSL2561 TMB Carrier Tape

MECHANICAL DATA

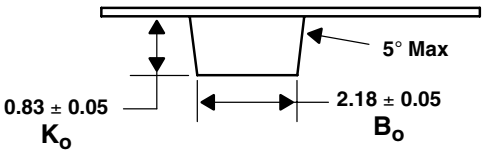
TOP VIEW

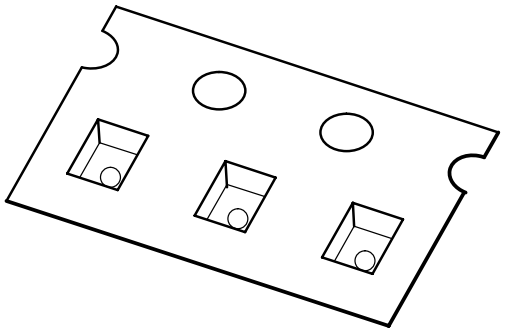


DETAIL A



DETAIL B



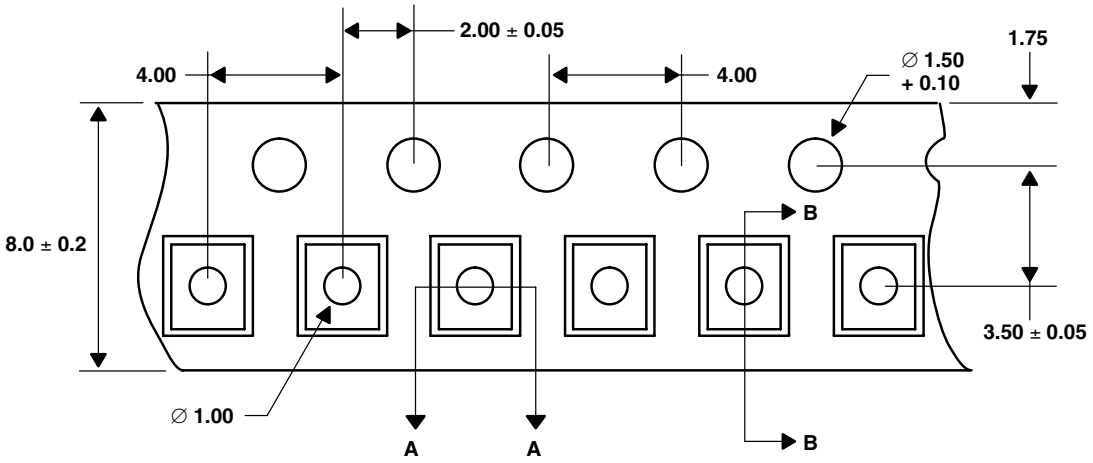


NOTES: H. All linear dimensions are in millimeters. Dimension tolerance is ± 0.10 mm unless otherwise noted.
I. The dimensions on this drawing are for illustrative purposes only. Dimensions of an actual carrier may vary slightly.
J. Symbols on drawing A_o, B_o, and K_o are defined in ANSI EIA Standard 481–B 2001.
K. Each reel is 178 millimeters in diameter and contains 3500 parts.
L. TAOS packaging tape and reel conform to the requirements of EIA Standard 481–B.
M. In accordance with EIA standard, device pin 1 is located next to the sprocket holes in the tape.
N. This drawing is subject to change without notice.

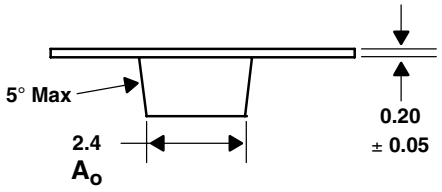
Figure 29. TSL2560/TSL2561 FN Carrier Tape

MECHANICAL DATA

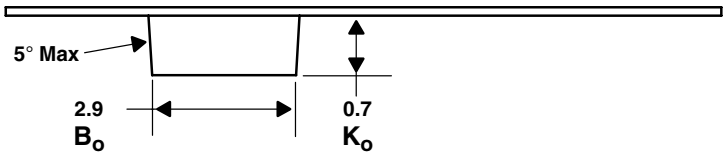
TOP VIEW



DETAIL A



DETAIL B



NOTES: A. All linear dimensions are in millimeters. Dimension tolerance is ± 0.10 mm unless otherwise noted.
B. The dimensions on this drawing are for illustrative purposes only. Dimensions of an actual carrier may vary slightly.
C. Symbols on drawing A_o, B_o, and K_o are defined in ANSI EIA Standard 481–B 2001.
D. Each reel is 178 millimeters in diameter and contains 2500 parts.
E. TAOS packaging tape and reel conform to the requirements of EIA Standard 481–B.
F. In accordance with EIA standard, device pin 1 is located next to the sprocket holes in the tape.
G. This drawing is subject to change without notice.

Figure 30. TSL2560/TSL2561 CL Carrier Tape

MANUFACTURING INFORMATION

The CS, T, FN, and CL packages have been tested and have demonstrated an ability to be reflow soldered to a PCB substrate. The process, equipment, and materials used in these test are detailed below.

The solder reflow profile describes the expected maximum heat exposure of components during the solder reflow process of product on a PCB. Temperature is measured on top of component. The components should be limited to a maximum of three passes through this solder reflow profile.

Table 13. TSL2560/61 Solder Reflow Profile

PARAMETER	REFERENCE	TSL2560/61
Average temperature gradient in preheating		2.5°C/sec
Soak time	t_{soak}	2 to 3 minutes
Time above 217°C	t_1	Max 60 sec
Time above 230°C	t_2	Max 50 sec
Time above T_{peak} –10°C	t_3	Max 10 sec
Peak temperature in reflow	T_{peak}	260° C (–0°C/+5°C)
Temperature gradient in cooling		Max –5°C/sec

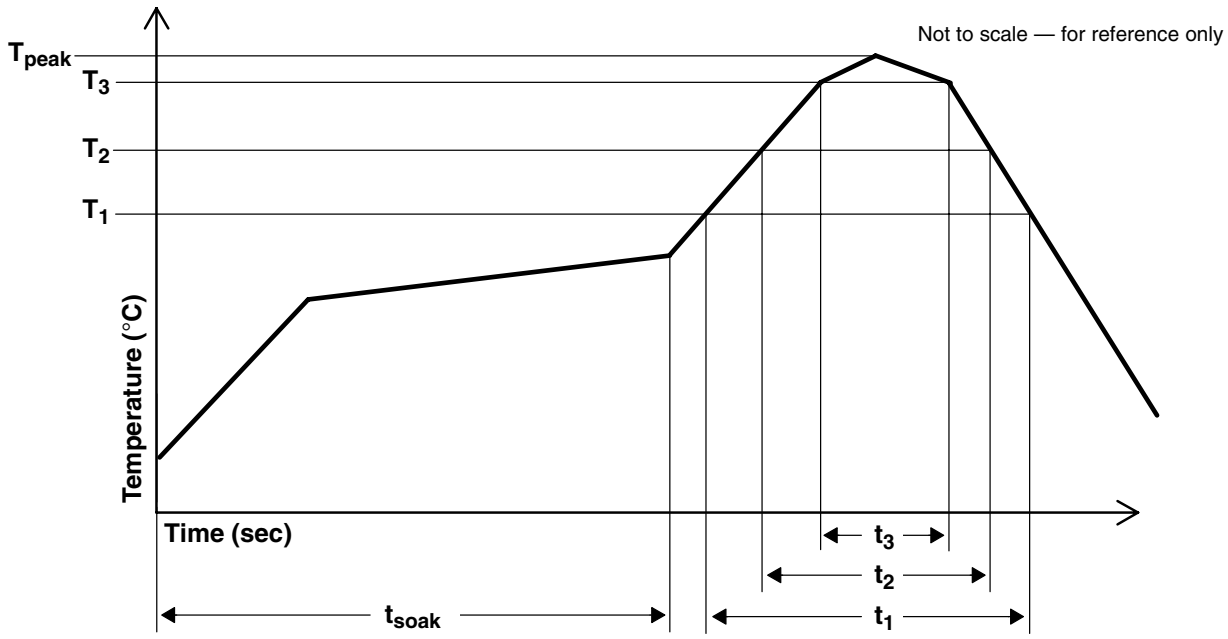


Figure 31. TSL2560/TSL2561 Solder Reflow Profile Graph

MANUFACTURING INFORMATION

Moisture Sensitivity

Optical characteristics of the device can be adversely affected during the soldering process by the release and vaporization of moisture that has been previously absorbed into the package molding compound. To ensure the package molding compound contains the smallest amount of absorbed moisture possible, each device is dry-baked prior to being packed for shipping. Devices are packed in a sealed aluminized envelope with silica gel to protect them from ambient moisture during shipping, handling, and storage before use.

The CS package has been assigned a moisture sensitivity level of MSL 2 and the devices should be stored under the following conditions:

Temperature Range	5°C to 50°C
Relative Humidity	60% maximum
Floor Life	1 year out of bag at ambient < 30°C / 60% RH

Rebaking will be required if the aluminized envelope has been open for more than 1 year. If rebaking is required, it should be done at 90°C for 3 hours.

The T, FN, and CL packages have been assigned a moisture sensitivity level of MSL 3 and the devices should be stored under the following conditions:

Temperature Range	5°C to 50°C
Relative Humidity	60% maximum
Total Time	6 months from the date code on the aluminized envelope — if unopened
Opened Time	168 hours or fewer

Rebaking will be required if the devices have been stored unopened for more than 6 months or if the aluminized envelope has been open for more than 168 hours. If rebaking is required, it should be done at 90°C for 4 hours.

PRODUCTION DATA — information in this document is current at publication date. Products conform to specifications in accordance with the terms of Texas Advanced Optoelectronic Solutions, Inc. standard warranty. Production processing does not necessarily include testing of all parameters.

LEAD-FREE (Pb-FREE) and GREEN STATEMENT

Pb-Free (RoHS) TAOS' terms *Lead-Free* or *Pb-Free* mean semiconductor products that are compatible with the current RoHS requirements for all 6 substances, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, TAOS Pb-Free products are suitable for use in specified lead-free processes.

Green (RoHS & no Sb/Br) TAOS defines *Green* to mean Pb-Free (RoHS compatible), and free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material).

Important Information and Disclaimer The information provided in this statement represents TAOS' knowledge and belief as of the date that it is provided. TAOS bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. TAOS has taken and continues to take reasonable steps to provide representative and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. TAOS and TAOS suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

NOTICE

Texas Advanced Optoelectronic Solutions, Inc. (TAOS) reserves the right to make changes to the products contained in this document to improve performance or for any other purpose, or to discontinue them without notice. Customers are advised to contact TAOS to obtain the latest product information before placing orders or designing TAOS products into systems.

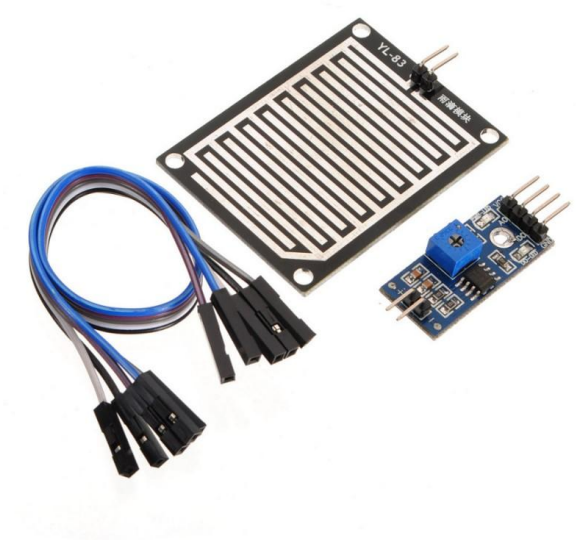
TAOS assumes no responsibility for the use of any products or circuits described in this document or customer product design, conveys no license, either expressed or implied, under any patent or other right, and makes no representation that the circuits are free of patent infringement. TAOS further makes no claim as to the suitability of its products for any particular purpose, nor does TAOS assume any liability arising out of the use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

TEXAS ADVANCED OPTOELECTRONIC SOLUTIONS, INC. PRODUCTS ARE NOT DESIGNED OR INTENDED FOR USE IN CRITICAL APPLICATIONS IN WHICH THE FAILURE OR MALFUNCTION OF THE TAOS PRODUCT MAY RESULT IN PERSONAL INJURY OR DEATH. USE OF TAOS PRODUCTS IN LIFE SUPPORT SYSTEMS IS EXPRESSLY UNAUTHORIZED AND ANY SUCH USE BY A CUSTOMER IS COMPLETELY AT THE CUSTOMER'S RISK.

Sensor de lluvia:

VAISALA www.vaisala.com

Yl-83 Rain Detector



Vaisala YL-83 Rain Detector

Features/Benefits

- Fast and accurate precipitation detection (ON/OFF)
- Rain intensity measurement with processing unit
- Maintenance free
- Heating element for keeping sensor free of snow and condensed moisture, and for quick drying

Rain and snow are quickly and accurately detected with the YL-83 Rain Detector. The YL-83 operates via droplet detection rather than by signal level threshold.

A special delay circuitry allows about two-minute interval between raindrops before assuming an OFF (no rain) position. This enables the sensor to accurately distinguish between rain cessation and light rain.

The YL-83 also features an analog Rain Signal for estimating rain intensity. Since this signal is proportional to the percentage of moist or wet area on the sensor plate, rain intensity has a direct impact on the amplitude and variation of this analog signal.

The YL-83 sensor is positioned at a 30° angle. This design, together with the internal heating element, ensures that the surface dries quickly, an essential factor in calculating intensity. The same heating element also protects the surface from fog and condensed moisture, and is activated at low temperatures in order to melt snow, thus allowing snow detection. Sensor performance is not affected by reasonable amounts of dirt and dust due to droplet detection.

It is intended to be used in areas
with only rain or wet/moist snow precipitation.

Technical Data

Sensor

Capacitive principle, thick layer sensor
RainCap™ with a thin glass shield. Integrated heater element.

Sensitivity of Rain Detection

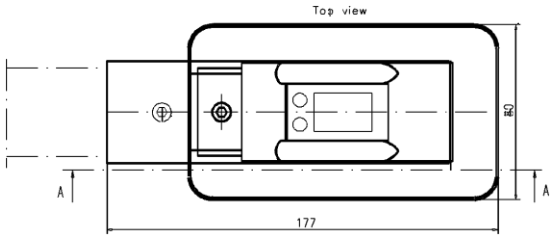
Minimum wet area	0.05 cm²
OFF-delay (active)	< 5 min

Physical

Sensor plate	
Sensing area	7.2 cm²
Angle	30°
Housing material	Polypropylene
Windshield and support bracket	Aluminum
Moisture shield	Polyurethane
Dimensions	(h × w × l)
With wind shield	110 × 80 × 175 mm
Without wind shield	90 × 46 × 157 mm
Weight	500 g
Cable length	4 m

Electrical

Supply voltage	12 VDC ± 10 %
Supply current	
Typical less than	150 mA
Maximum	260 mA
Heater OFF	25 mA
Sensor plate	
Heating power	0.5 ... 2.3 W



Output

Rain ON/OFF	
Open collector, active low signal corresponds to rain	
Maximum voltage	15 V
Maximum current	50 mA
Analog output	1...3 V (wet...dry)
Frequency output	1500...6000 Hz, non-calibrated

Input

Control to switch heater OFF	
Open circuit input enables the heater.	
Connection to GND disables the heater.	
Contact rating min.	15 V, 2 mA

Ground Wiring

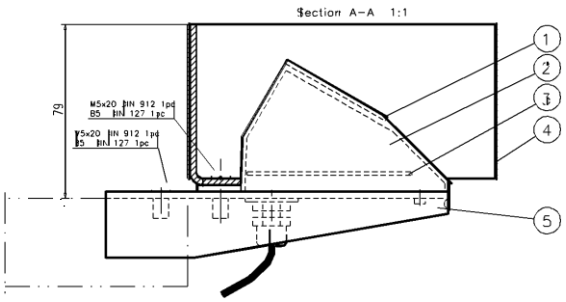
Separate ground wires for signal and heater

Temperature Range

Operating	-15...+55 °C (+5...+131 °F)
Storage	-40...+65 °C (-40...+149 °F)

Mounting

By one screw (M5 x 20 mm) to sensor arm



1. *Sensor, RainCap™*
2. *Polyurethane moisture shield*
3. *Component assembly*
4. *Wind shield*
5. *Mounting plate*

VAISALA

Please contact us at
www.vaisala.com/requestinfo

www.vaisala.com

Scan the code for more information

CE

Mosfet:

N-Channel Enhancement-Mode Vertical DMOS FET

Features

- Low threshold - 1.6V max.
- High input impedance
- Low input capacitance - 130pF typical
- Fast switching speeds
- Low on-resistance guaranteed at V_{GS} = 2, 3, and 5V
- Free from secondary breakdown
- Low input and output leakage

Applications

- Logic level interfaces – ideal for TTL and CMOS
- Solid state relays
- Battery operated systems
- Photo voltaic drives
- Analog switches
- General purpose line drivers
- Telecom switches

General Description

This low threshold, enhancement-mode (normally-off) transistor utilizes a vertical DMOS structure and Supertex's well-proven, silicon-gate manufacturing process. This combination produces a device with the power handling capabilities of bipolar transistors and the high input impedance and positive temperature coefficient inherent in MOS devices. Characteristic of all MOS structures, this device is free from thermal runaway and thermally-induced secondary breakdown.

Supertex's vertical DMOS FETs are ideally suited to a wide range of switching and amplifying applications where very low threshold voltage, high breakdown voltage, high input impedance, low input capacitance, and fast switching speeds are desired.

Ordering Information

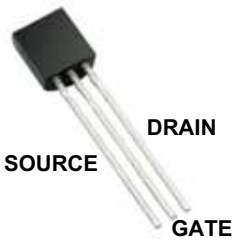
Part Number	Package Option	Packing
TN0702N3-G	TO-92	1000/Bag
TN0702N3-G P002	TO-92	2000/Reel
TN0702N3-G P003		
TN0702N3-G P005		
TN0702N3-G P013		
TN0702N3-G P014		

-G denotes a lead (Pb)-free / RoHS compliant package.
Contact factory for Wafer / Die availablity.
Devices in Wafer / Die form are lead (Pb)-free / RoHS compliant.

Product Summary

BV_{DSS}/BV_{DGS}	$R_{DS(ON)}$ (max)	$I_{D(ON)}$ (min)	$V_{GS(th)}$ (max)
20V	1.3Ω	0.5A	1.0V

Pin Configuration



TO-92

Absolute Maximum Ratings

Parameter	Value
Drain-to-source voltage	BV_{DSS}
Drain-to-gate voltage	BV_{DGS}
Gate-to-source voltage	±20V
Operating and storage temperature	-55°C to +150°C

Absolute Maximum Ratings are those values beyond which damage to the device may occur. Functional operation under these conditions is not implied. Continuous operation of the device at the absolute rating level may affect device reliability. All voltages are referenced to device ground.

Product Marking

YY = Year Sealed
WW = Week Sealed
_____ = "Green" Packaging

Package may or may not include the following marks: Si or

TO-92

Typical Thermal Resistance

Package	θ_{ja}
TO-92	132°C/W

Thermal Characteristics

Package	I_D (continuous)†	I_D (pulsed)	Power Dissipation @T_C = 25°C	I_{DR}†	I_{DRM}
TO-92	530mA	1.0A	1.0W	530mA	1.0A

Notes:

† I_D *(continuous) is limited by max rated* T_j .

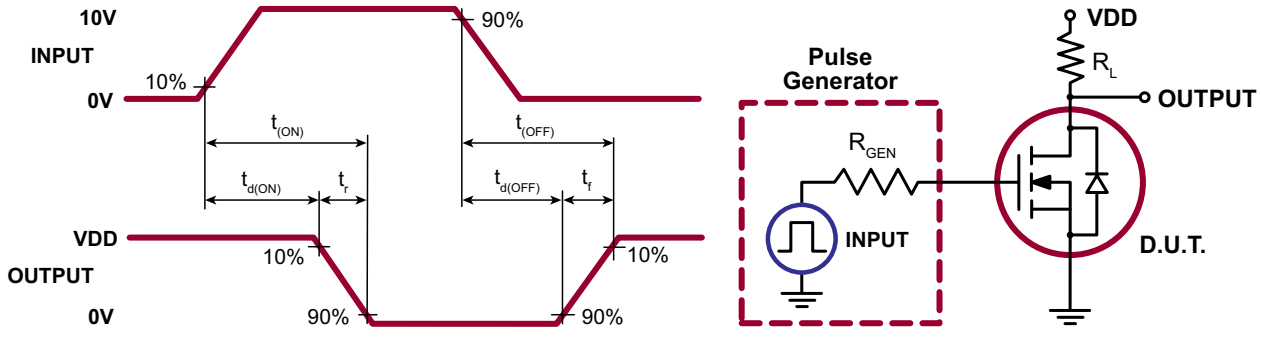
Electrical Characteristics *(T_A = 25°C unless otherwise specified)*

Sym	Parameter	Min	Typ	Max	Units	Conditions
BV_{DSS}	Drain-to-source breakdown voltage	20	-	-	V	V_{GS} = 0V, I_D = 1.0mA
$V_{GS(th)}$	Gate threshold voltage	0.5	0.8	1.0	V	V_{GS} = V_{DS}, I_D= 1.0mA
$\Delta V_{GS(th)}$	Change in $V_{GS(th)}$ with temperature	-	-	-4.0	mV/°C	V_{GS} = V_{DS}, I_D= 1.0mA
I_{GSS}	Gate body leakage	-	-	100	nA	V_{GS} = ± 20V, V_{DS} = 0V
I_{DSS}	Zero gate voltage drain current	-	-	100	nA	V_{GS} = 0V, V_{DS} = Max Rating
		-	-	100	µA	V_{DS} = 0.8 Max Rating, V_{GS} = 0V, T_A = 125°C
$I_{D(ON)}$	On-state drain current	0.5	1.0	-	A	V_{GS} = V_{DS} = 5.0V
$R_{DS(ON)}$	Static drain-to-source on-state resistance	-	4.0	5.0	Ω	V_{GS} = 2.0V, I_D = 50mA
		-	1.9	2.5		V_{GS} = 3.0V, I_D = 200mA
		-	1.0	1.3		V_{GS} = 5.0V, I_D = 500mA
$\Delta R_{DS(ON)}$	Change in $R_{DS(ON)}$ with temperature	-	-	0.75	%/°C	V_{GS} = 5.0V, I_D = 500mA
G_{FS}	Forward transductance	100	500	-	mmho	V_{DS} = 5.0V, I_D = 500mA
C_{ISS}	Input capacitance	-	130	200	pF	V_{GS} = 0V, V_{DS} = 20V, f = 1.0MHz
C_{OSS}	Common source output capacitance	-	70	125		
C_{RSS}	Reverse transfer capacitance	-	30	60		
$t_{d(ON)}$	Turn-on delay time	-	-	20	ns	V_{DD} = 20V, I_D = 0.5A, R_{GEN} = 25Ω
t_r	Rise time	-	-	20		
$t_{d(OFF)}$	Turn-off delay time	-	-	30		
t_f	Fall time	-	-	20		
V_{SD}	Diode forward voltage drop	-	-	1.0	V	V_{GS} = 0V, I_{SD} = 0.5A

Notes:

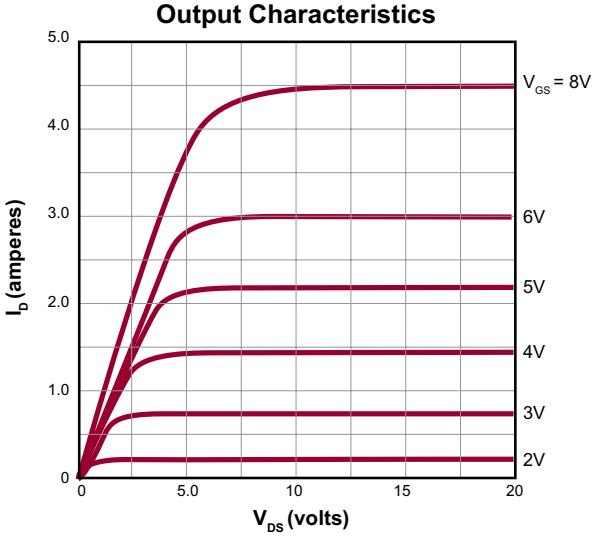
1. *All D.C. parameters 100% tested at 25°C unless otherwise stated. (Pulse test: 300µs pulse, 2% duty cycle.)*
2. *All A.C. parameters sample tested.*

Switching Waveforms and Test Circuit

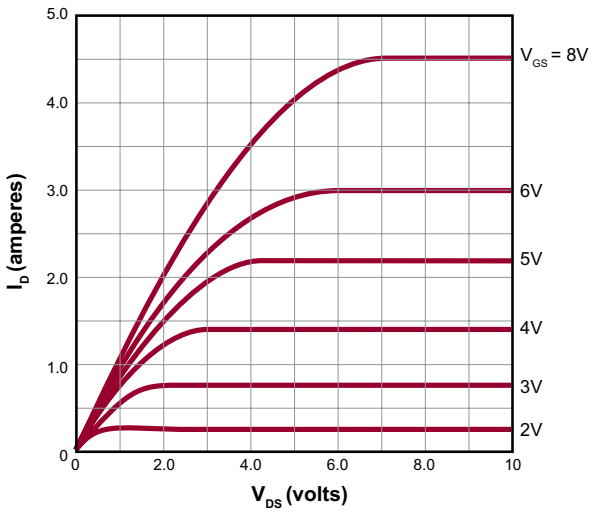


Typical Performance Curves

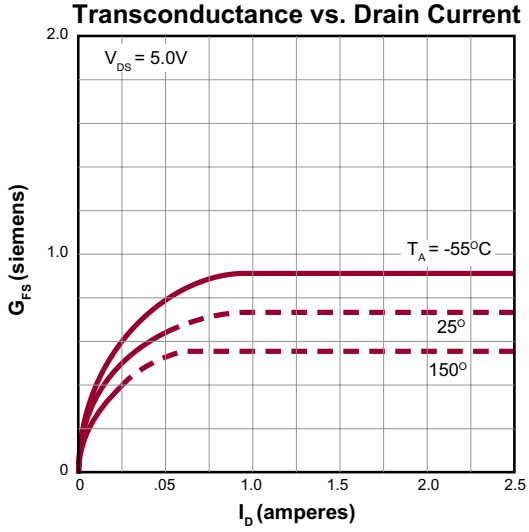
Output Characteristics



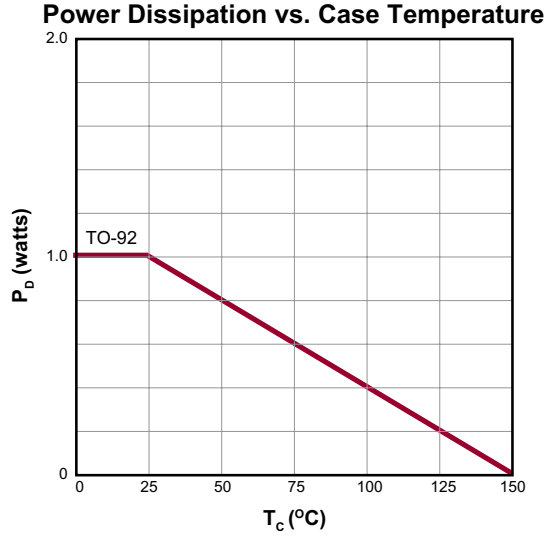
Saturation Characteristics



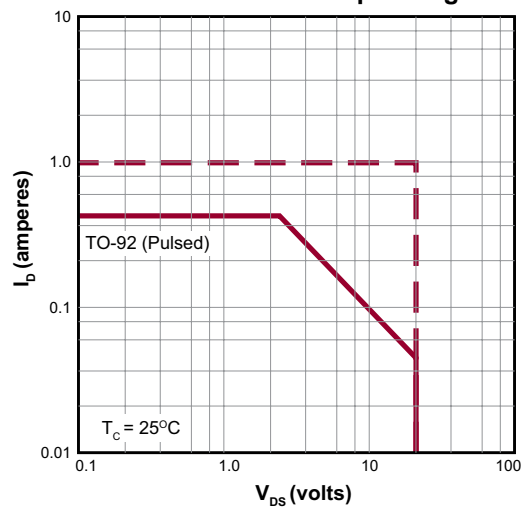
Transconductance vs. Drain Current



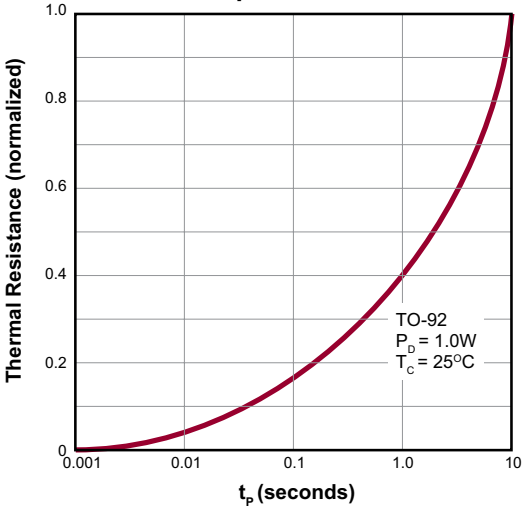
Power Dissipation vs. Case Temperature



Maximum Rated Safe Operating Area

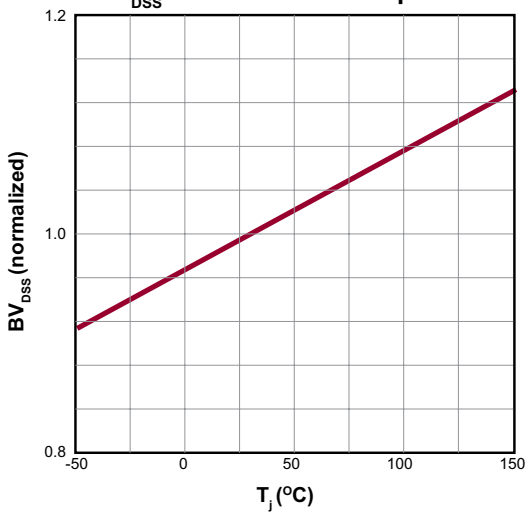


Thermal Response Characteristics

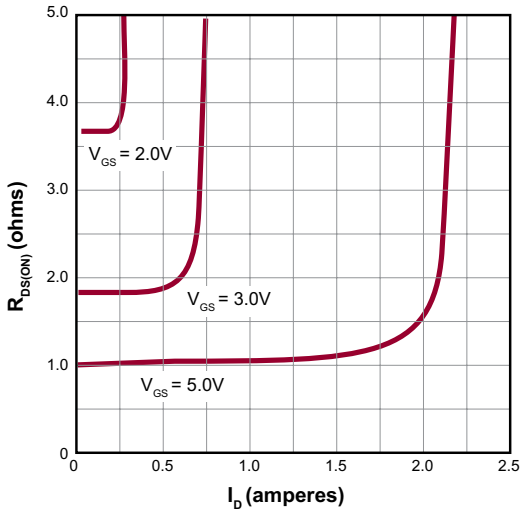


Typical Performance Curves *(cont.)*

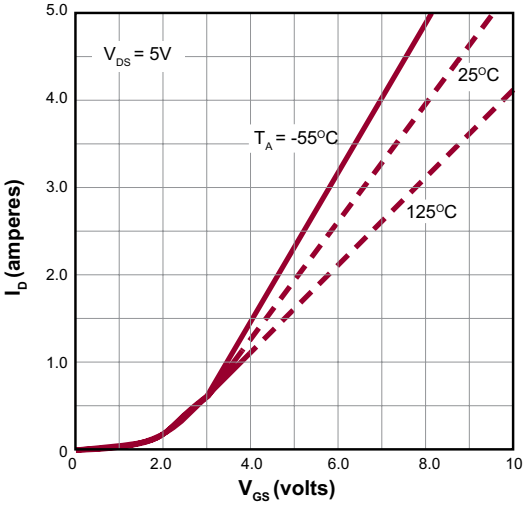
BV_{DSS} Variation with Temperature



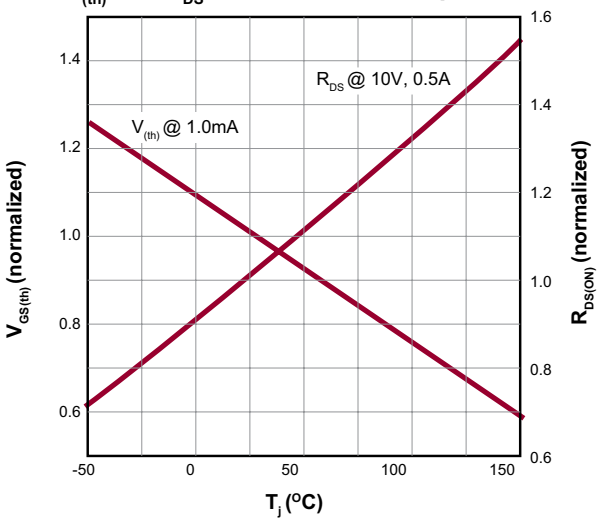
On-Resistance vs. Drain Current



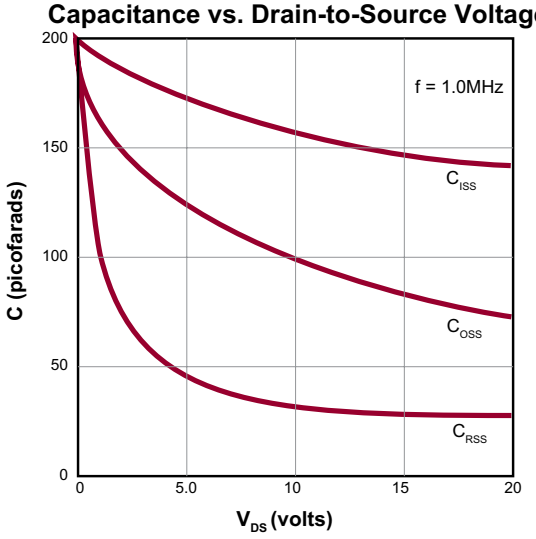
Transfer Characteristics



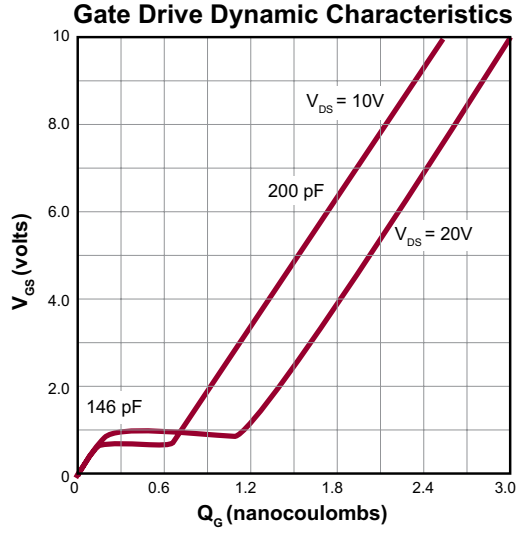
$V_{(th)}$ and R_{DS} Variation with Temperature



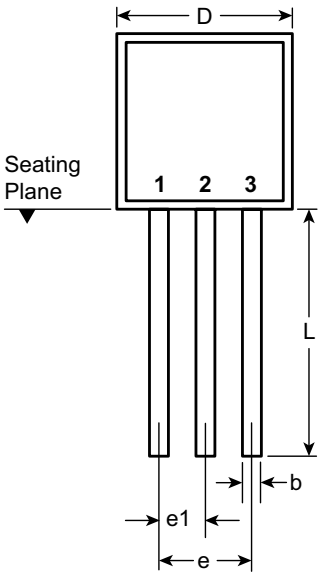
Capacitance vs. Drain-to-Source Voltage



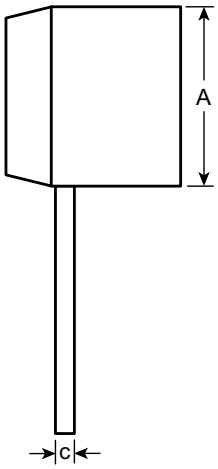
Gate Drive Dynamic Characteristics



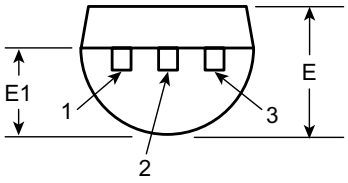
3-Lead TO-92 Package Outline (N3)



Front View



Side View



Bottom View

Symbol		A	b	c	D	E	E1	e	e1	L
Dimensions (inches)	MIN	.170	.014†	.014†	.175	.125	.080	.095	.045	.500
	NOM	-	-	-	-	-	-	-	-	-
	MAX	.210	.022†	.022†	.205	.165	.105	.105	.055	.610*

JEDEC Registration TO-92.
** This dimension is not specified in the JEDEC drawing.*
† This dimension differs from the JEDEC drawing.
Drawings not to scale.
***Supertex Doc.#:** DSPD-3TO92N3, Version E041009.*

(The package drawing(s) in this data sheet may not reflect the most current specifications. For the latest package outline information go to http://www.supertex.com/packaging.html.)

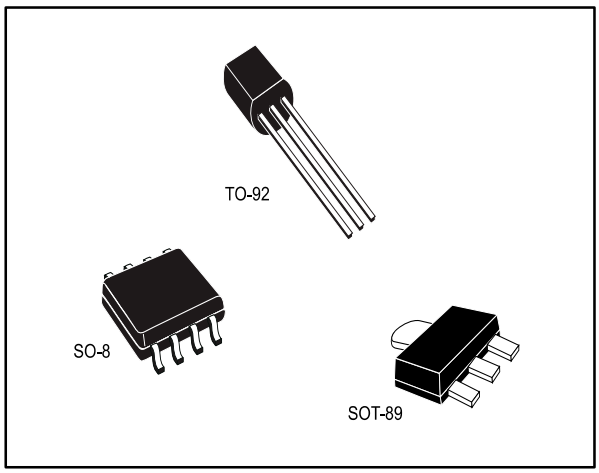
Supertex inc.
1235 Bordeaux Drive, Sunnyvale, CA 94089
Tel: 408-222-8888
www.supertex.com

Regulador de Tensión :

L78L

Positive voltage regulators

Datasheet - production data



Features

- Output current up to 100 mA
- Output voltages of 3.3; 5; 6; 8; 9; 10; 12; 15; 18; 24 V thermal overload protection
- Short-circuit protection
- No external components are required
- Available in either ± 4% (A) or ± 8% (C) selection

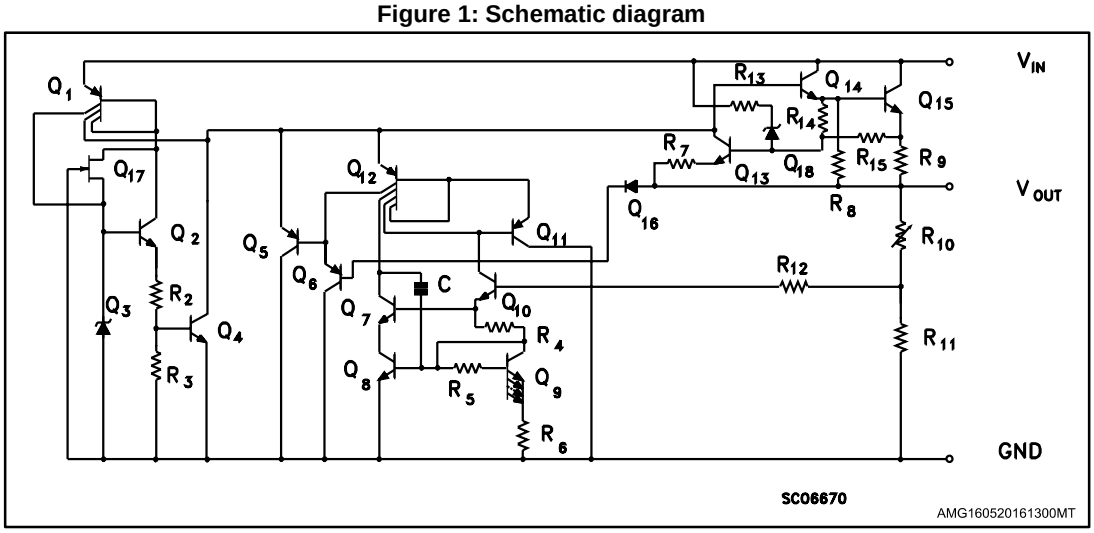
Description

The L78L series of three-terminal positive regulators employ internal current limiting and thermal shutdown, making them essentially indestructible. If adequate heat-sink is provided, they can deliver up to 100 mA output current. They are intended as fixed voltage regulators in a wide range of applications including local or on-card regulation for elimination of noise and distribution problems associated with single-point regulation. In addition, they can be used with power pass elements to make high-current voltage regulators. The L78L series used as Zener diode/resistor combination replacement, offers e improvement along with lower quiescent current and lower noise.

Contents

1 Diagram

Figure 1: Schematic diagram



2 Pin configuration

Figure 2: Pin connection (top view, bottom view for TO-92)

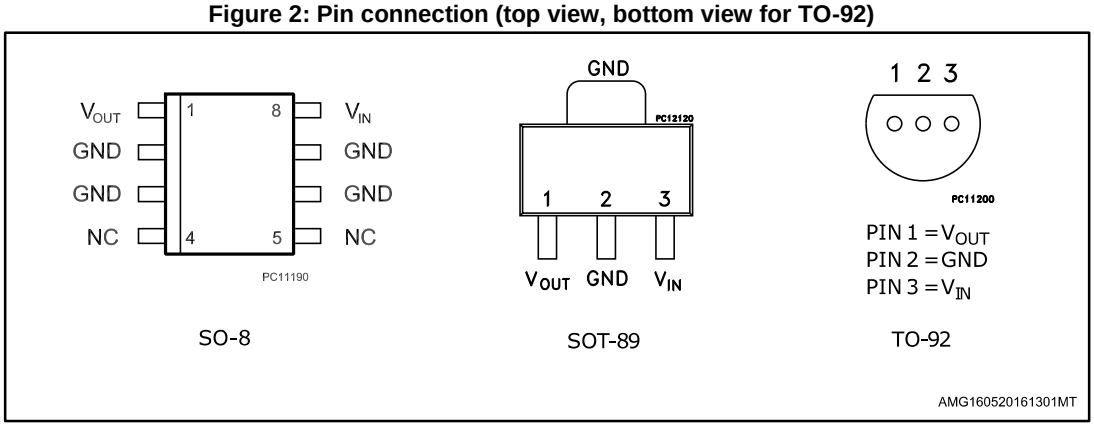
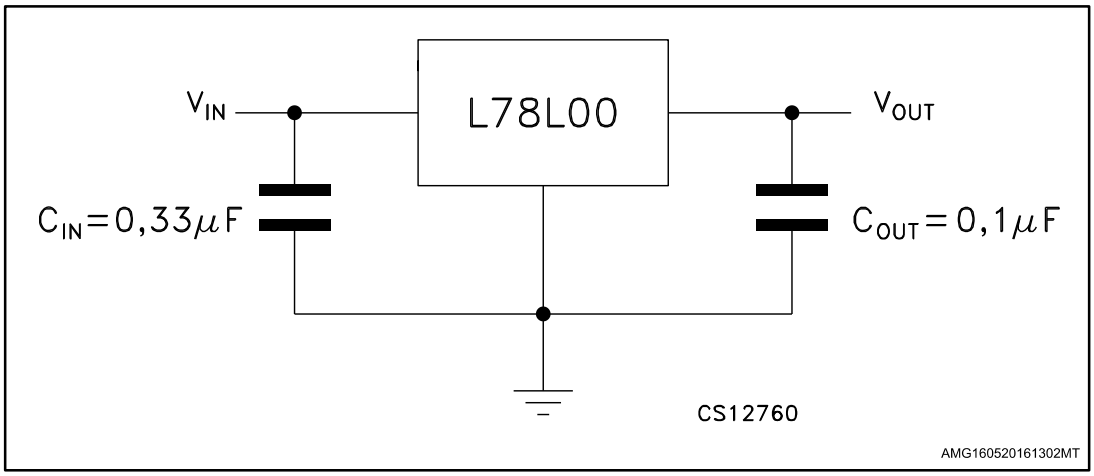


Figure 3: Test circuits



3 Maximum ratings

Table 1: Absolute maximum ratings

Symbol	Parameter		Value	Unit
V_I	DC Input voltage	V_O = 3.3 to 9 V	30	V
		V_O = 12 to 15 V	35	
		V_O = 18 to 24 V	40	
I_O	Output current		100	mA
P_D	Power dissipation		Internally limited (1)	mW
T_{STG}	Storage temperature range		-65 to 150	°C
T_{OP}	Operating junction temperature range	for L78LxxAC / L78LxxC	0 to 125	°C
		for L78LxxAB	-40 to 125	

Notes:

(1)Our SO-8 package used for voltage regulators is modified internally to have pins 2, 3, 6 and 7 electrically communed to the die attach flag. This particular frame decreases the total thermal resistance of the package and increases its ability to dissipate power when an appropriate area of copper on the printed circuit board is available for heat-sinking. The external dimensions are the same as for the standard SO-8.

Table 2: Thermal data

Symbol	Parameter	SO-8	TO-92	SOT-89	Unit
R_{thJC}	Thermal resistance junction-case (max)	20		15	°C/W
R_{thJA}	Thermal resistance junction-ambient (max)	55 (1)	200	55 (1)	°C/W

Notes:

(1)Considering 6 cm² of copper Board heat-sink.

4 Electrical characteristics

Refer to the test circuits, T_J = 0 to 125 °C, V_I = 8.3 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 3: Electrical characteristics of L78L33C

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	3.036	3.3	3.564	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 5.3 to 20 V	2.97		3.63	V
		I_O = 1 to 70 mA, V_I = 8.3 V	2.97		3.63	
ΔV_O	Line regulation	V_I = 5.4 to 20 V, T_J = 25 °C			150	mV
		V_I = 6.3 to 20 V, T_J = 25 °C			100	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			60	mV
		I_O = 1 to 40 mA, T_J = 25 °C			30	
I_d	Quiescent current	T_J = 25 °C			6	mA
		T_J = 125 °C			5.5	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.2	mA
		V_I = 6.3 to 20 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		40		µV
SVR	Supply voltage rejection	V_I = 6.3 to 16.3 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	41	49		dB
V_d	Dropout voltage			2		V

Refer to the test circuits, T_J = 0 to 125 °C, V_I = 10 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 4: Electrical characteristics of L78L05C

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	4.6	5	5.4	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 7 to 20 V	4.5		5.5	V
		I_O = 1 to 70 mA, V_I = 10 V	4.5		5.5	
ΔV_O	Line regulation	V_I = 8.5 to 20 V, T_J = 25 °C			200	mV
		V_I = 9 to 20 V, T_J = 25 °C			150	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			60	mV
		I_O = 1 to 40 mA, T_J = 25 °C			30	
I_d	Quiescent current	T_J = 25 °C			6	mA
		T_J = 125 °C			5.5	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.2	mA
		V_I = 8 to 20 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		40		µV
SVR	Supply voltage rejection	V_I = 9 to 20 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	40	49		dB
V_d	Dropout voltage			2		V

Refer to the test circuits, T_J = 0 to 125 °C,V_I = 14 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 5: Electrical characteristics of L78L08C

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	7.36	8	8.64	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 8.5 to 20 V	7.2		8.8	V
		I_O = 1 to 70 mA, V_I = 12 V	7.2		8.8	
ΔV_O	Line regulation	V_I = 10.5 to 20 V, T_J = 25 °C			200	mV
		V_I = 11 to 20 V, T_J = 25 °C			150	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			80	mV
		I_O = 1 to 40 mA, T_J = 25 °C			40	
I_d	Quiescent current	T_J = 25 °C			6	mA
		T_J = 125 °C			5.5	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.2	mA
		V_I = 8 to 20 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		60		µV
SVR	Supply voltage rejection	V_I = 9 to 20 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	36	45		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C, V_I = 15 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 6: Electrical characteristics of L78L09C

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	8.28	9	9.72	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 11.5 to 23 V	8.1		9.9	V
		I_O = 1 to 70 mA, V_I = 15 V	8.1		9.9	
ΔV_O	Line regulation	V_I = 11.5 to 23 V, T_J = 25 °C			250	mV
		V_I = 12 to 23 V, T_J = 25 °C			200	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			80	mV
		I_O = 1 to 40 mA, T_J = 25 °C			40	
I_d	Quiescent current	T_J = 25 °C			6	mA
		T_J = 125 °C			5.5	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.2	mA
		V_I = 12 to 23 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		70		µV
SVR	Supply voltage rejection	V_I = 12 to 23 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	36	44		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C, V_I = 16 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 7: Electrical characteristics of L78L10C

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	9.2	10	10.8	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 12.5 to 23 V	9		11	V
		I_O = 1 to 70 mA, V_I = 16 V	9		11	
ΔV_O	Line regulation	V_I = 12.5 to 23 V, T_J = 25 °C			230	mV
		V_I = 13 to 23 V, T_J = 25 °C			170	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			80	mV
		I_O = 1 to 40 mA, T_J = 25 °C			40	
I_d	Quiescent current	T_J = 25 °C			6	mA
		T_J = 125 °C			5.5	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.1	mA
		V_I = 13 to 23 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		60		µV
SVR	Supply voltage rejection	V_I = 14 to 23 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	37	45		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C, V_I = 19 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 8: Electrical characteristics of L78L12C

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	11.1	12	12.9	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 14.5 to 27 V	10.8		13.2	V
		I_O = 1 to 70 mA, V_I = 19 V	10.8		13.2	
ΔV_O	Line regulation	V_I = 14.5 to 27 V, T_J = 25 °C			250	mV
		V_I = 16 to 27 V, T_J = 25 °C			200	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			100	mV
		I_O = 1 to 40 mA, T_J = 25 °C			50	
I_d	Quiescent current	T_J = 25 °C			6.5	mA
		T_J = 125 °C			6	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.2	mA
		V_I = 16 to 27 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		80		µV
SVR	Supply voltage rejection	V_I = 15 to 25 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	36	42		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C, V_I = 23 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified

Table 9: Electrical characteristics of L78L15C

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	13.8	15	16.2	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 17.5 to 30 V	13.5		16.5	V
		I_O = 1 to 70 mA, V_I = 23 V	13.5		16.5	
ΔV_O	Line regulation	V_I = 17.5 to 30 V, T_J = 25 °C			300	mV
		V_I = 20 to 30 V, T_J = 25 °C			250	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			150	mV
		I_O = 1 to 40 mA, T_J = 25 °C			75	
I_d	Quiescent current	T_J = 25 °C			6.5	mA
		T_J = 125 °C			6	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.2	mA
		V_I = 20 to 30 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		90		µV
SVR	Supply voltage rejection	V_I = 18.5 to 28.5 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	33	39		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C, V_I = 27 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 10: Electrical characteristics of L78L18C

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	16.6	18	19.4	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 22 to 33 V	16.2		19.8	V
		I_O = 1 to 70 mA, V_I = 27 V	16.2		19.8	
ΔV_O	Line regulation	V_I = 22 to 33 V, T_J = 25 °C			320	mV
		V_I = 22 to 33 V, T_J = 25 °C			270	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			170	mV
		I_O = 1 to 40 mA, T_J = 25 °C			85	
I_d	Quiescent current	T_J = 25 °C			6.5	mA
		T_J = 125 °C			6	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.2	mA
		V_I = 23 to 33 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		120		µV
SVR	Supply voltage rejection	V_I = 23 to 33 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	32	38		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C, V_I = 33 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 11: Electrical characteristics of L78L24C

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	22.1	24	25.9	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 27 to 38 V	21.6		26.4	V
		I_O = 1 to 70 mA, V_I = 33 V	21.6		26.4	
ΔV_O	Line regulation	V_I = 27 to 38 V, T_J = 25 °C			350	mV
		V_I = 28 to 38 V, T_J = 25 °C			300	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			200	mV
		I_O = 1 to 40 mA, T_J = 25 °C			100	
I_d	Quiescent current	T_J = 25 °C			6.5	mA
		T_J = 125 °C			6	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.2	mA
		V_I = 28 to 38 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		200		µV
SVR	Supply voltage rejection	V_I = 29 to 35 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	30	37		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C (AC) T_J = -40 to 125 °C (AB),V_I = 8.3 V, I_O = 40mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 12: Electrical characteristics of L78L33AB and L78L33AC

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	3.168	3.3	3.432	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 5.3 to 20 V	3.135		3.465	V
		I_O = 1 to 70 mA, V_I = 8.3 V	3.135		3.465	
ΔV_O	Line regulation	V_I = 5.4 to 20 V, T_J = 25 °C			150	mV
		V_I = 6.3 to 20 V, T_J = 25 °C			100	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			60	mV
		I_O = 1 to 40 mA, T_J = 25 °C			30	
I_d	Quiescent current	T_J = 25 °C			6	mA
		T_J = 125 °C			5.5	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.1	mA
		V_I = 6.3 to 20 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		40		µV
SVR	Supply voltage rejection	V_I = 6.3 to 16.3 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	41	49		dB
V_d	Dropout voltage			2		V

Refer to the test circuits, T_J = 0 to 125 °C (AC) T_J = -40 to 125 °C (AB), V_I = 10 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 13: Electrical characteristics of L78L05AB and L78L05AC

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	4.8	5	5.2	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 7 to 20 V	4.75		5.25	V
		I_O = 1 to 70 mA, V_I = 10 V	4.75		5.25	
ΔV_O	Line regulation	V_I = 7.3 to 20 V, T_J = 25 °C			150	mV
		V_I = 8 to 20 V, T_J = 25 °C			100	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			60	mV
		I_O = 1 to 40 mA, T_J = 25 °C			30	
I_d	Quiescent current	T_J = 25 °C			6	mA
		T_J = 125 °C			5.5	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.1	mA
		V_I = 8 to 20 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		40		µV
SVR	Supply voltage rejection	V_I = 8 to 18 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	41	49		dB
V_d	Dropout voltage			2		V

Refer to the test circuits, T_J = 0 to 125 °C (AC) T_J = -40 to 125 °C (AB), V_I = 12 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 14: Electrical characteristics of L78L06AB and L78L06AC

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	5.76	6	6.24	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 8.5 to 20 V	5.7		6.3	V
		I_O = 1 to 70 mA, V_I = 12 V	5.7		6.3	
ΔV_O	Line regulation	V_I = 8.5 to 20 V, T_J = 25 °C			150	mV
		V_I = 9 to 20 V, T_J = 25 °C			100	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			60	mV
		I_O = 1 to 40 mA, T_J = 25 °C			30	
I_d	Quiescent current	T_J = 25 °C			6	mA
		T_J = 125 °C			5.5	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.1	mA
		V_I = 9 to 20 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		50		µV
SVR	Supply voltage rejection	V_I = 9 to 20 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	39	46		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C (AC) T_J = -40 to 125 °C (AB), V_I = 14 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 15: Electrical characteristics of L78L08AB and L78L08AC

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	7.68	8	8.32	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 10.5 to 23 V	7.6		8.4	V
		I_O = 1 to 70 mA, V_I = 14 V	7.6		8.4	
ΔV_O	Line regulation	V_I = 10.5 to 23 V, T_J = 25 °C			175	mV
		V_I = 11 to 23 V, T_J = 25 °C			125	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			80	mV
		I_O = 1 to 40 mA, T_J = 25 °C			40	
I_d	Quiescent current	T_J = 25 °C			6	mA
		T_J = 125 °C			5.5	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.1	mA
		V_I = 11 to 23 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		60		µV
SVR	Supply voltage rejection	V_I = 12 to 23 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	37	45		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C (AC) T_J = -40 to 125 °C (AB), V_I =15 V, I_O= 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 16: Electrical characteristics of L78L09AB and L78L09AC

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	8.64	9	9.36	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 11.5 to 23 V	8.55		9.45	V
		I_O = 1 to 70 mA, V_I = 15 V	8.55		9.45	
ΔV_O	Line regulation	V_I = 11.5 to 23 V, T_J = 25 °C			225	mV
		V_I = 12 to 23 V, T_J = 25 °C			150	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			80	mV
		I_O = 1 to 40 mA, T_J = 25 °C			40	
I_d	Quiescent current	T_J = 25 °C			6	mA
		T_J = 125 °C			5.5	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.1	mA
		V_I = 12 to 23 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		70		µV
SVR	Supply voltage rejection	V_I = 12 to 23 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	37	44		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C (AC) T_J = -40 to 125 °C (AB), V_I= 16 V, I_O = 40mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 17: Electrical characteristics of L78L10AC

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	9.6	10	10.4	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 12.5 to 23 V	9.5		10.5	V
		I_O = 1 to 70 mA, V_I = 16 V	9.5		10.5	
ΔV_O	Line regulation	V_I = 12.5 to 23 V, T_J = 25 °C			230	mV
		V_I = 13 to 23 V, T_J = 25 °C			170	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			80	mV
		I_O = 1 to 40 mA, T_J = 25 °C			40	
I_d	Quiescent current	T_J = 25 °C			6	mA
		T_J = 125 °C			5.5	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.1	mA
		V_I = 13 to 23 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		60		µV
SVR	Supply voltage rejection	V_I = 14 to 23 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	37	45		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C (AC) T_J = -40 to 125 °C (AB), V_I =19 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 18: Electrical characteristics of L78L12AB and L78L12AC

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	11.5	12	12.5	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 14.5 to 27 V	11.4		12.6	V
		I_O = 1 to 70 mA, V_I = 19 V	11.4		12.6	
ΔV_O	Line regulation	V_I = 14.5 to 27 V, T_J = 25 °C			250	mV
		V_I = 16 to 27 V, T_J = 25 °C			200	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			100	mV
		I_O = 1 to 40 mA, T_J = 25 °C			50	
I_d	Quiescent current	T_J = 25 °C			6.5	mA
		T_J = 125 °C			6	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.1	mA
		V_I = 16 to 27 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		80		µV
SVR	Supply voltage rejection	V_I = 15 to 25 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	37	42		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C (AC) T_J = -40 to 125 °C (AB),V_I = 23 V, I_O= 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 19: Electrical characteristics of L78L15AB and L78L15AC

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	14.4	15	15.6	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 17.5 to 30 V	14.25		15.75	V
		I_O = 1 to 70 mA, V_I = 23 V	14.25		15.75	
ΔV_O	Line regulation	V_I = 17.5 to 30 V, T_J = 25 °C			300	mV
		V_I = 20 to 30 V, T_J = 25 °C			250	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			150	mV
		I_O = 1 to 40 mA, T_J = 25 °C			75	
I_d	Quiescent current	T_J = 25 °C			6.5	mA
		T_J = 125 °C			6	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.1	mA
		V_I = 20 to 30 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		90		µV
SVR	Supply voltage rejection	V_I = 18.5 to 28.5 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	34	39		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C (AC) T_J = -40 to 125 °C (AB),V_I = 27 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 20: Electrical characteristics of L78L18AC

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	17.3	18	18.7	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 22 to 33 V	17.1		18.9	V
		I_O = 1 to 70 mA, V_I = 27 V	17.1		18.9	
ΔV_O	Line regulation	V_I = 22 to 33 V, T_J = 25 °C			320	mV
		V_I = 22 to 33 V, T_J = 25 °C			270	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			170	mV
		I_O = 1 to 40 mA, T_J = 25 °C			85	
I_d	Quiescent current	T_J = 25 °C			6.5	mA
		T_J = 125 °C			6	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.1	mA
		V_I = 23 to 33 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		120		µV
SVR	Supply voltage rejection	V_I = 23 to 33 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	33	38		dB
V_d	Dropout voltage			1.7		V

Refer to the test circuits, T_J = 0 to 125 °C (AC) T_J = -40 to 125 °C (AB),V_I = 33 V, I_O = 40 mA, C_I = 0.33 µF, C_O = 0.1 µF unless otherwise specified.

Table 21: Electrical characteristics of L78L24AB and L78L24AC

Symbol	Parameter	Test conditions	Min.	Typ.	Max.	Unit
V_O	Output voltage	T_J = 25 °C	23	24	25	V
V_O	Output voltage	I_O = 1 to 40 mA, V_I = 27 to 38 V	22.8		25.2	V
		I_O = 1 to 70 mA, V_I = 33 V	22.8		25.2	
ΔV_O	Line regulation	V_I = 27 to 38 V, T_J = 25 °C			350	mV
		V_I = 28 to 38 V, T_J = 25 °C			300	
ΔV_O	Load regulation	I_O = 1 to 100 mA, T_J = 25 °C			200	mV
		I_O = 1 to 40 mA, T_J = 25 °C			100	
I_d	Quiescent current	T_J = 25 °C			6.5	mA
		T_J = 125 °C			6	mA
ΔI_d	Quiescent current change	I_O = 1 to 40 mA			0.1	mA
		V_I = 28 to 38 V			1.5	
eN	Output noise voltage	B = 10 Hz to 100 kHz, T_J = 25 °C		200		µV5y
SVR	Supply voltage rejection	V_I = 29 to 33 V, f = 120 Hz I_O = 40 mA, T_J = 25 °C	31	37		dB
V_d	Dropout voltage			1.7		V

5 Typical performance

Figure 4: L78L05/12 output voltage vs. ambient temperature

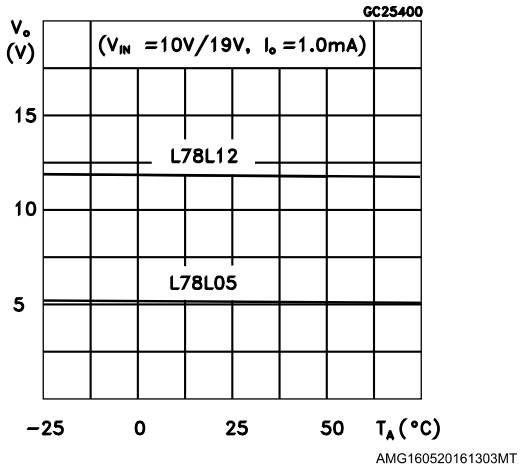


Figure 5: L78L05/12/24 load characteristics

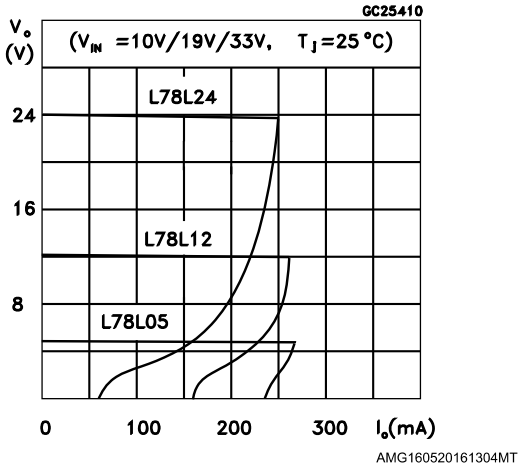


Figure 6: L78L05/12/24 thermal shutdown

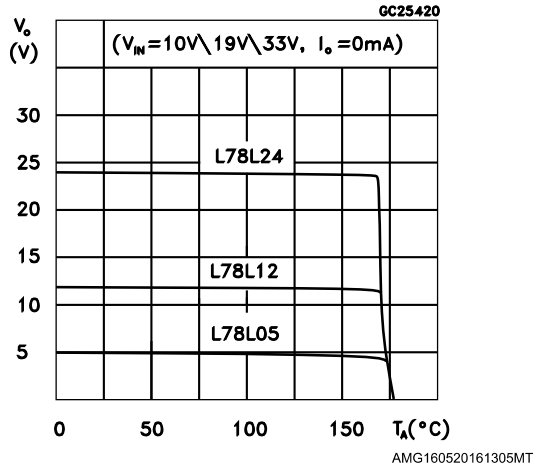


Figure 7: L78L05/12 quiescent current vs. output current

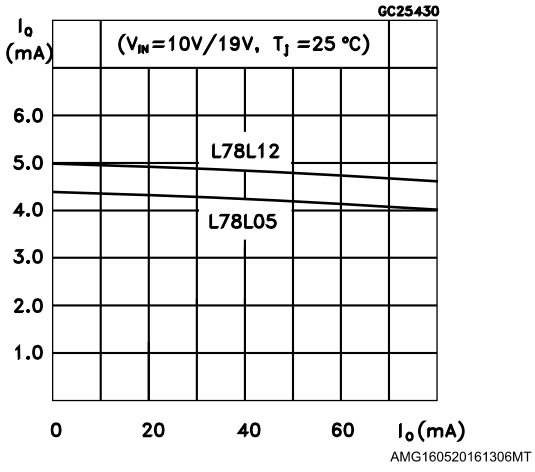


Figure 8: L78L05 quiescent current vs. input voltage

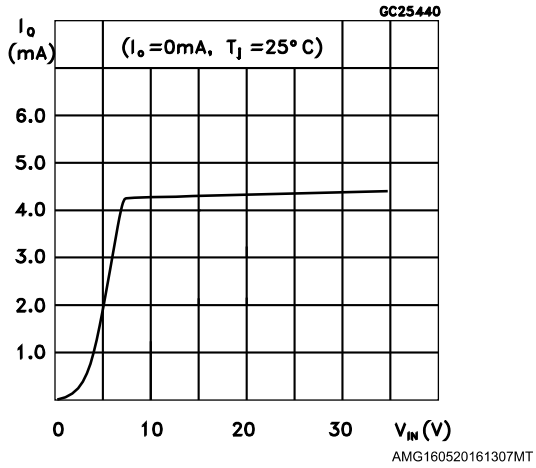


Figure 9: L78L05/12/24 output characteristics

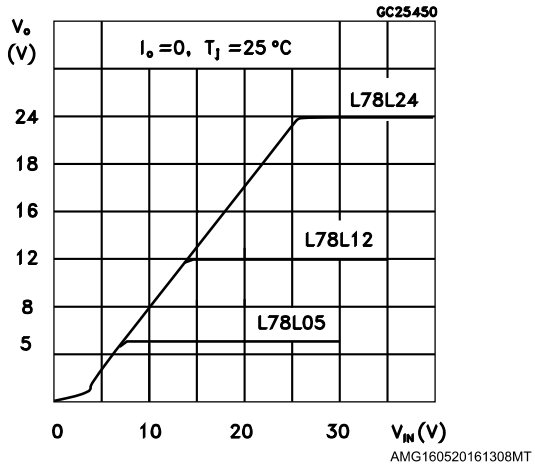


Figure 10: L78L05/12/24 ripple rejection

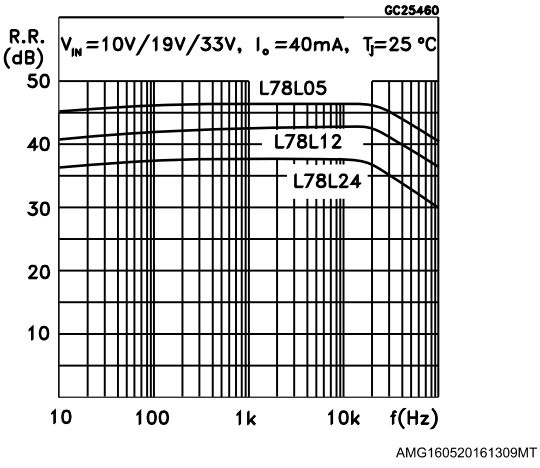


Figure 11: L78L05 dropout characteristics

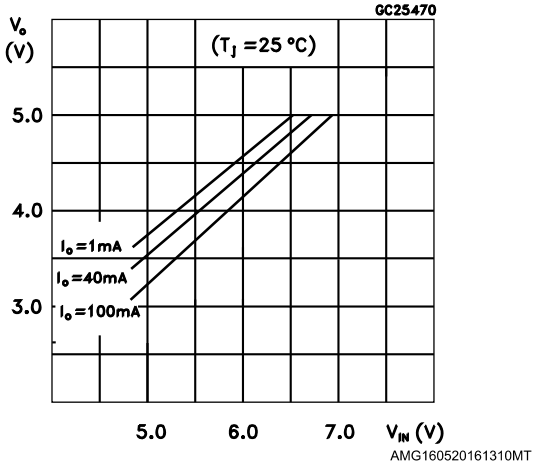
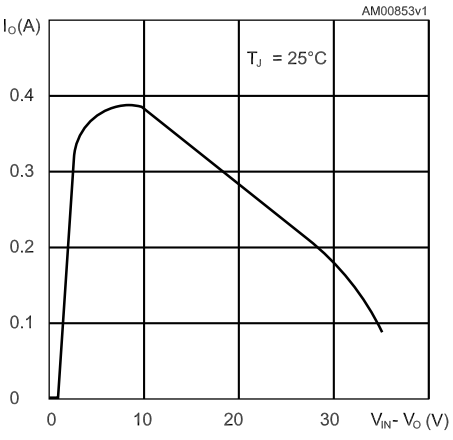


Figure 12: L78L short-circuit output current



6 Typical application

Figure 13: High output current short-circuit protected

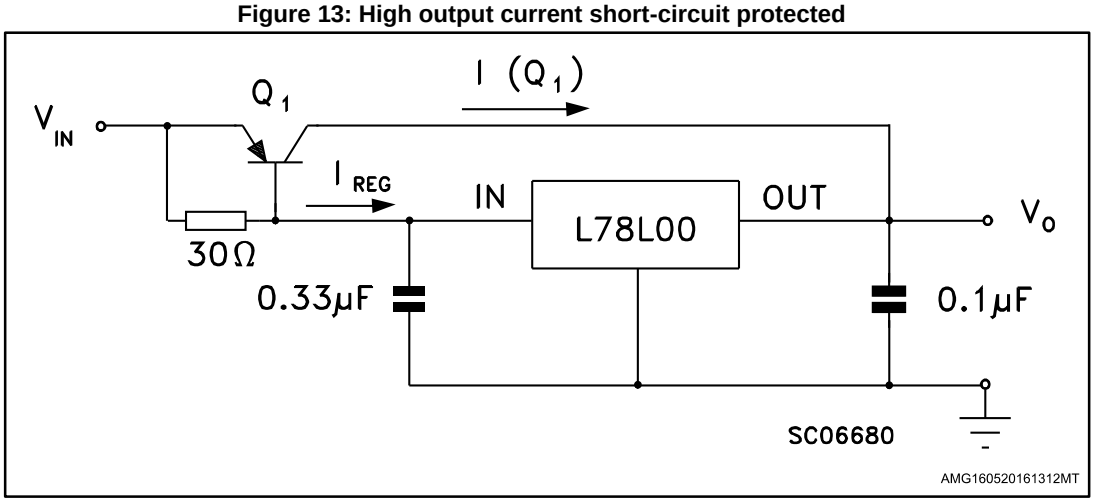


Figure 14: Outuput boost circuit

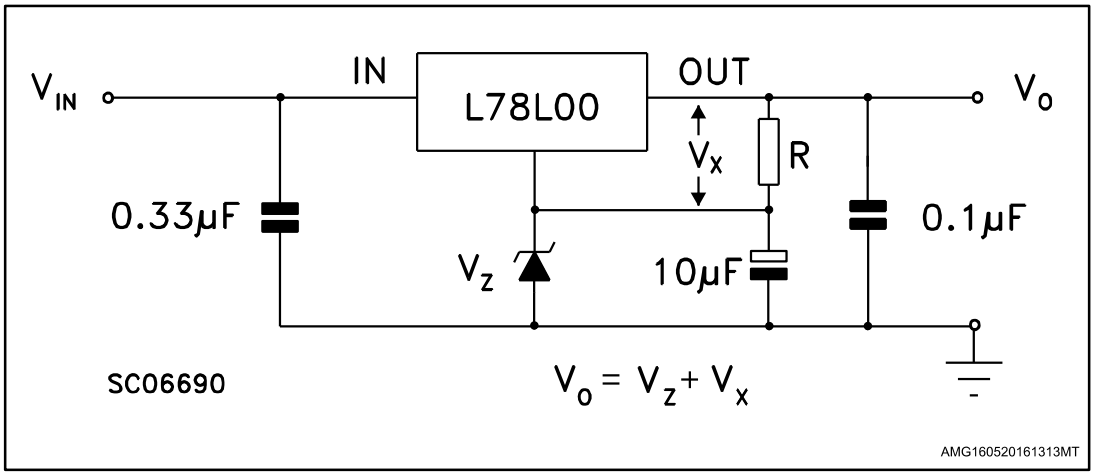


Figure 15: Current regulator

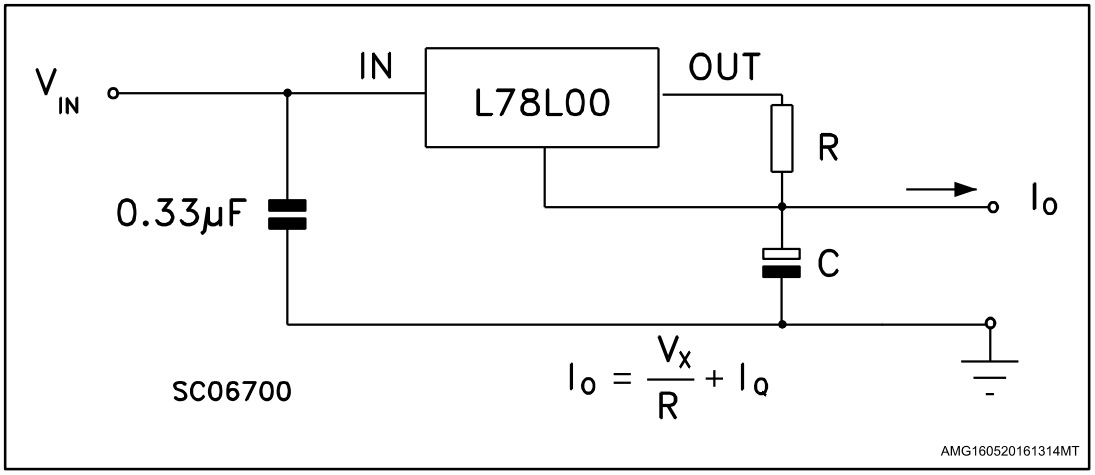
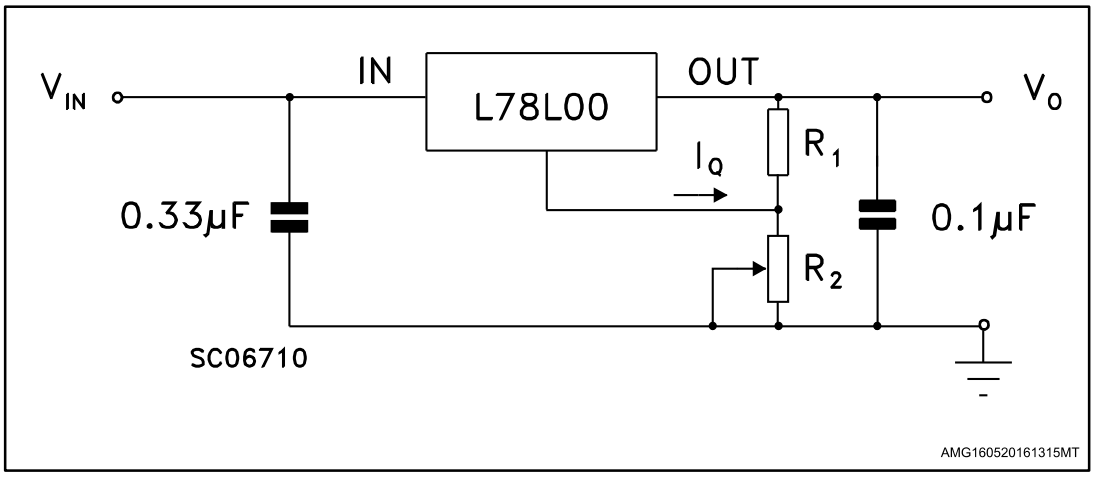


Figure 16: Adjustable output regulator



7 Package information

In order to meet environmental requirements, ST offers these devices in different grades of ECOPACK® packages, depending on their level of environmental compliance. ECOPACK® specifications, grade definitions and product status are available at: ***www.st.com***. ECOPACK® is an ST trademark.

7.1 TO-92 package information

Figure 17: TO-92 package outline

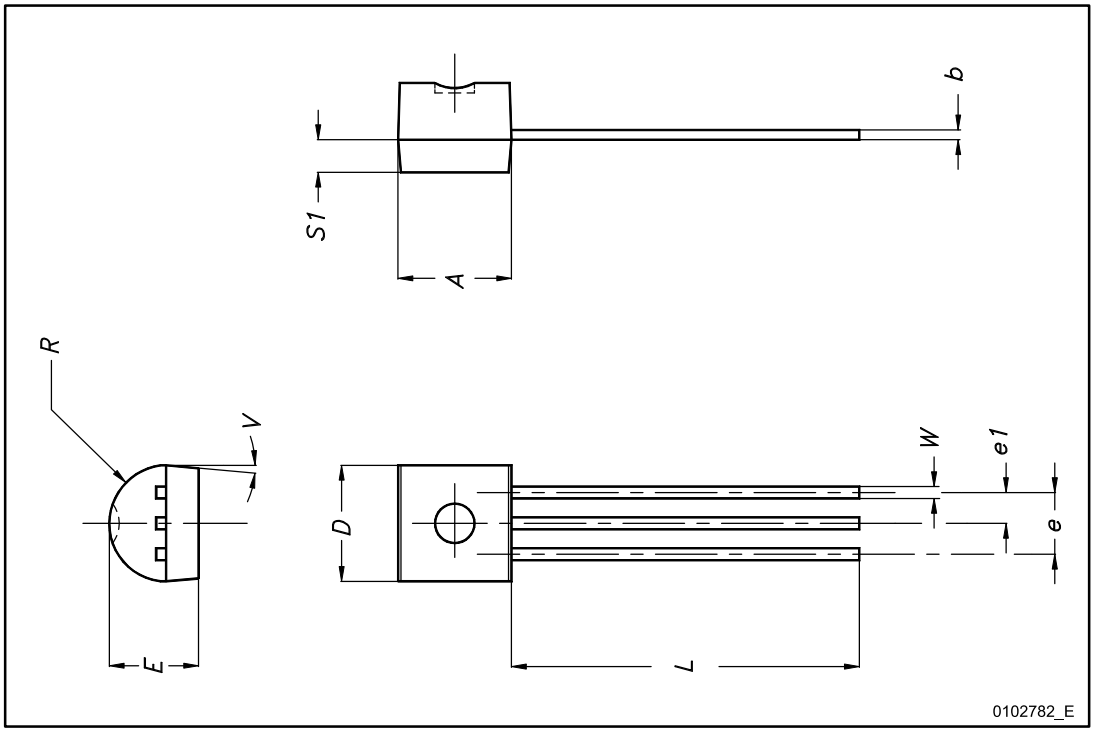


Table 22: TO-92 mechanical data

Dim.	mm		
	Min.	Typ.	Max.
A	4.32		4.95
b	0.36		0.51
D	4.45		4.95
E	3.30		3.94
e	2.41		2.67
e1	1.14		1.40
L	12.70		15.49
R	2.16		2.41
S1	0.92		1.52
W	0.41		0.56
V		5°	

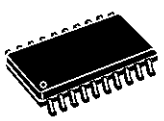
Puente en H :

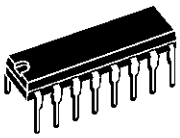
SGS-THOMSON MICROELECTRONICS

L293D
L293DD

PUSH-PULL FOUR CHANNEL DRIVER WITH DIODES

- 600mA OUTPUT CURRENT CAPABILITY PER CHANNEL
- 1.2A PEAK OUTPUT CURRENT (non repetitive) PER CHANNEL
- ENABLE FACILITY
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)
- INTERNAL CLAMP DIODES





SO(12+4+4) **Powerdip (12+2+2)**

ORDERING NUMBERS:

L293DD L293D

DESCRIPTION

The Device is a monolithic integrated high voltage, high current four channel driver designed to accept standard DTL or TTL logic levels and drive inductive loads (such as relays solenoides, DC and stepping motors) and switching power transistors.

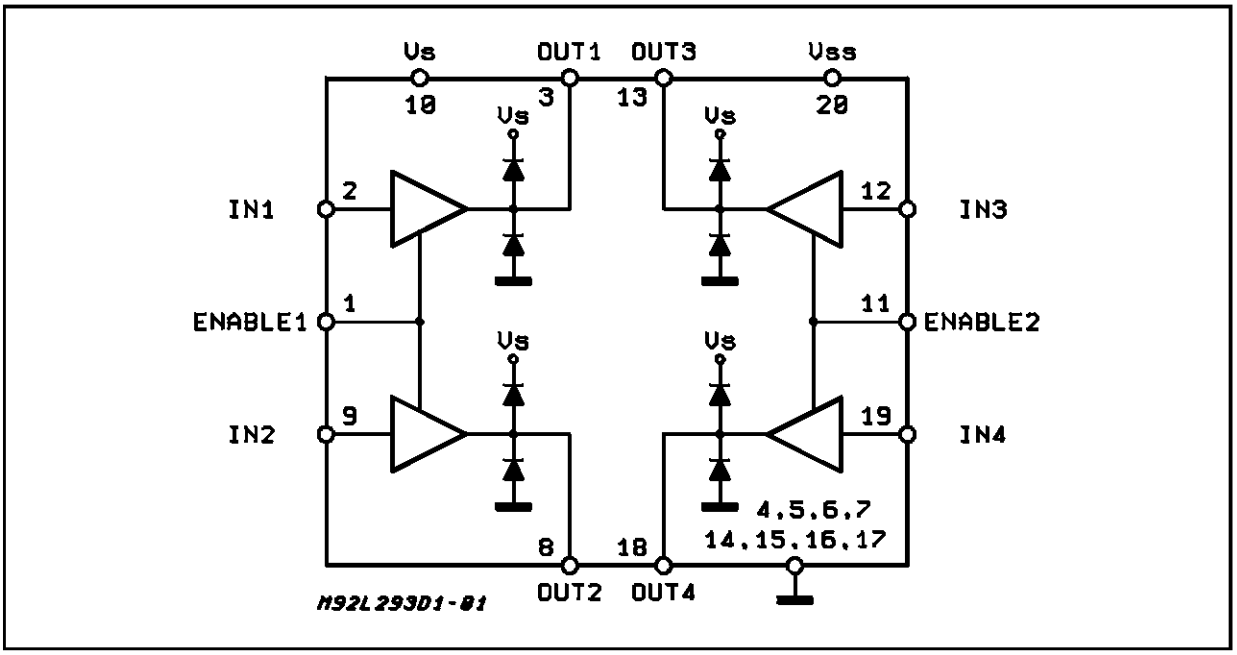
To simplify use as two bridges each pair of channels is equipped with an enable input. A separate supply input is provided for the logic, allowing operation at a lower voltage and internal clamp diodes are included.

This device is suitable for use in switching applications at frequencies up to 5 kHz.

The L293D is assembled in a 16 lead plastic packaage which has 4 center pins connected together and used for heatsinking

The L293DD is assembled in a 20 lead surface mount which has 8 center pins connected together and used for heatsinking.

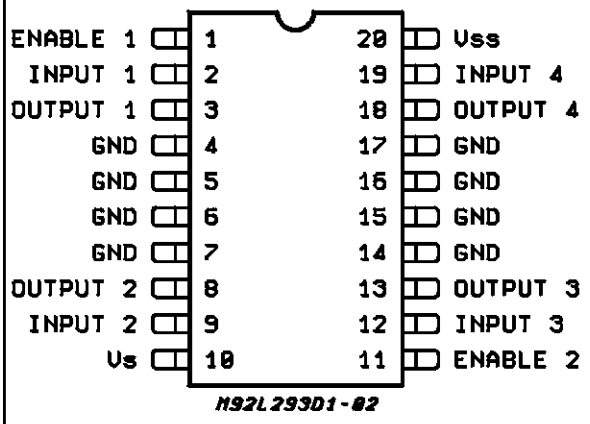
BLOCK DIAGRAM



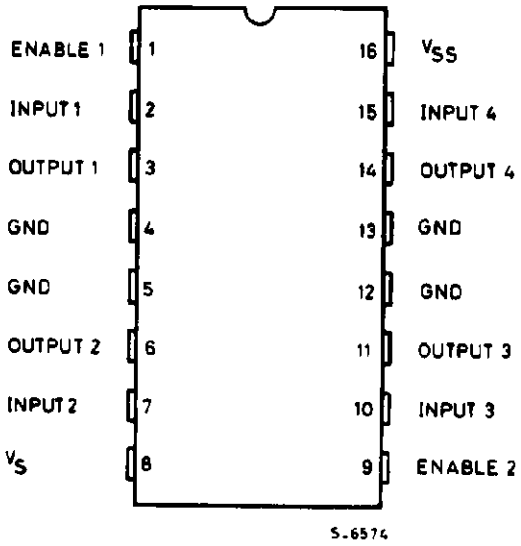
ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Supply Voltage	36	V
V_{SS}	Logic Supply Voltage	36	V
V_i	Input Voltage	7	V
V_{en}	Enable Voltage	7	V
I_o	Peak Output Current (100 µs non repetitive)	1.2	A
P_{tot}	Total Power Dissipation at T_{pins} = 90 °C	4	W
T_{stg}, T_j	Storage and Junction Temperature	– 40 to 150	°C

PIN CONNECTIONS (Top view)



SO(12+4+4)



Powerdip(12+2+2)

THERMAL DATA

Symbol	Decription		DIP	SO	Unit
$R_{th\ j\text{-}pins}$	Thermal Resistance Junction-pins	max.	–	14	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance junction-ambient	max.	80	50 (*)	°C/W
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	max.	14	–	

(*) With 6sq. cm on board heatsink.

ELECTRICAL CHARACTERISTICS (for each channel, V_S = 24 V, V_{SS} = 5 V, T_{amb} = 25 °C, unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 10)		V_{SS}		36	V
V_{SS}	Logic Supply Voltage (pin 20)		4.5		36	V
I_S	Total Quiescent Supply Current (pin 10)	V_i = L ; I_O = 0 ; V_{en} = H		2	6	mA
		V_i = H ; I_O = 0 ; V_{en} = H		16	24	mA
		V_{en} = L			4	mA
I_{SS}	Total Quiescent Logic Supply Current (pin 20)	V_i = L ; I_O = 0 ; V_{en} = H		44	60	mA
		V_i = H ; I_O = 0 ; V_{en} = H		16	22	mA
		V_{en} = L		16	24	mA
V_{IL}	Input Low Voltage (pin 2, 9, 12, 19)		– 0.3		1.5	V
V_{IH}	Input High Voltage (pin 2, 9, 12, 19)	$V_{SS} \leq 7$ V	2.3		V_{SS}	V
		$V_{SS} > 7$ V	2.3		7	V
I_{IL}	Low Voltage Input Current (pin 2, 9, 12, 19)	V_{IL} = 1.5 V			– 10	µA
I_{IH}	High Voltage Input Current (pin 2, 9, 12, 19)	2.3 V ≤ V_{IH} ≤ V_{SS} – 0.6 V		30	100	µA
$V_{en\,L}$	Enable Low Voltage (pin 1, 11)		– 0.3		1.5	V
$V_{en\,H}$	Enable High Voltage (pin 1, 11)	$V_{SS} \leq 7$ V	2.3		V_{SS}	V
		$V_{SS} > 7$ V	2.3		7	V
$I_{en\,L}$	Low Voltage Enable Current (pin 1, 11)	$V_{en\,L}$ = 1.5 V		– 30	– 100	µA
$I_{en\,H}$	High Voltage Enable Current (pin 1, 11)	2.3 V ≤ $V_{en\,H}$ ≤ V_{SS} – 0.6 V			± 10	µA
$V_{CE(sat)H}$	Source Output Saturation Voltage (pins 3, 8, 13, 18)	I_O = – 0.6 A		1.4	1.8	V
$V_{CE(sat)L}$	Sink Output Saturation Voltage (pins 3, 8, 13, 18)	I_O = + 0.6 A		1.2	1.8	V
V_F	Clamp Diode Forward Voltage	I_O = 600nA		1.3		V
t_r	Rise Time (*)	0.1 to 0.9 V_O		250		ns
t_f	Fall Time (*)	0.9 to 0.1 V_O		250		ns
t_{on}	Turn-on Delay (*)	0.5 V_i to 0.5 V_O		750		ns
t_{off}	Turn-off Delay (*)	0.5 V_i to 0.5 V_O		200		ns

(*) See fig. 1.

TRUTH TABLE (one channel)

Input	Enable (*)	Output
H	H	H
L	H	L
H	L	Z
L	L	Z

Z = High output impedance
(*) Relative to the considered channel

Figure 1: Switching Times

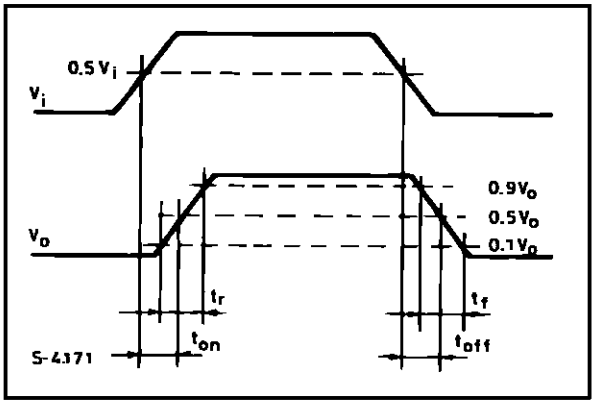
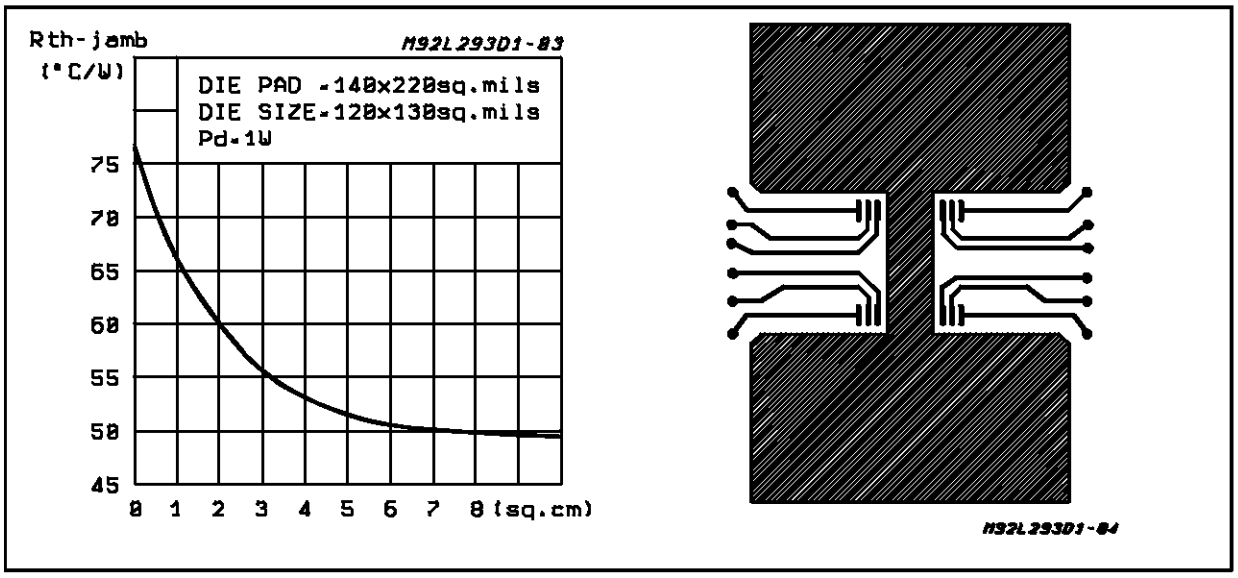
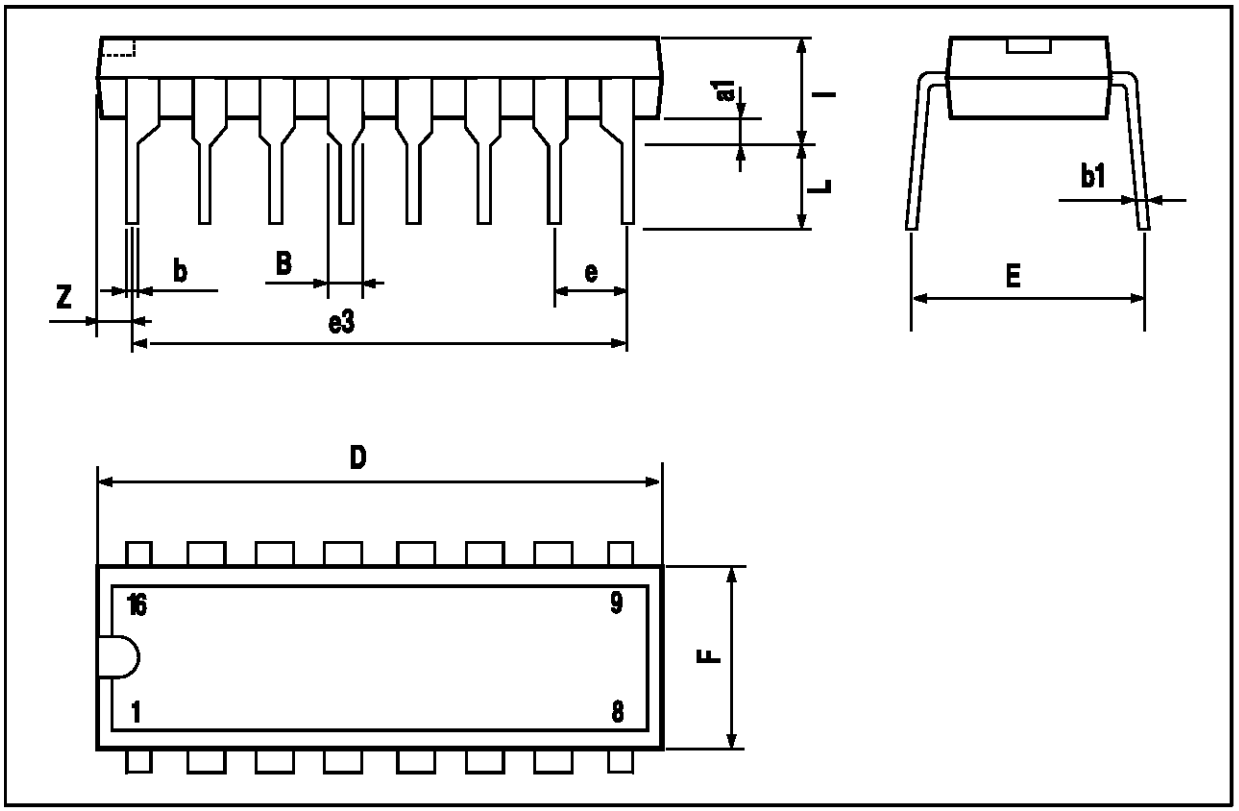


Figure 2: Junction to ambient thermal resistance vs. area on board heatsink (SO12+4+4 package)



POWERDIP16 PACKAGE MECHANICAL DATA

DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
a1	0.51			0.020		
B	0.85		1.40	0.033		0.055
b		0.50			0.020	
b1	0.38		0.50	0.015		0.020
D			20.0			0.787
E		8.80			0.346	
e		2.54			0.100	
e3		17.78			0.700	
F			7.10			0.280
I			5.10			0.201
L		3.30			0.130	
Z			1.27			0.050



SO20 PACKAGE MECHANICAL DATA

DIM.	mm			inch		
	MIN.	TYP.	MAX.	MIN.	TYP.	MAX.
A			2.65			0.104
a1	0.1		0.2	0.004		0.008
a2			2.45			0.096
b	0.35		0.49	0.014		0.019
b1	0.23		0.32	0.009		0.013
C		0.5			0.020	
c1		45			1.772	
D		1	12.6		0.039	0.496
E	10		10.65	0.394		0.419
e		1.27			0.050	
e3		11.43			0.450	
F		1	7.4		0.039	0.291
G	8.8		9.15	0.346		0.360
L	0.5		1.27	0.020		0.050
M			0.75			0.030
S	8° (max.)