UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Departament d'Enginyeria Electrònica

PROYECTO FINAL DE GRADO

SISTEMA DOMÓTICO ABIERTO DE BAJO COSTE PARA LA MEJORA DE LA EFICIENCIA ENERGÉTICA

AUTOR
JAVIER BARRIO RUIZ

DIRECTOR
MANUEL LAMICH AROCAS

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

19 DE MAYO DE 2017

AGRADECIMIENTOS

Con las siguientes palabras me gustaría agradecer a todas aquellas personas que han hecho posible que este trabajo se lleve a cabo.

En el ambito técnico del proyecto, me gustaría agradecer al PDI y PAS del Departamento de Ingeniería Electrónica de la UPC por permitirme hacer uso de los espacios y el material de los laboratorios. Especialmente a Juan Carlos Pineda, y a Manuel Lamich, el director del proyecto, por la ayuda que me han proporcionado durante el transcurso del proyecto.

También quisiera agradecer a David Hurtado, desde el primer curso, mi compañero en la mayoría de laboratorios de la carrera, su apoyo e implicación en el proyecto. Agradecerles también a todos los compañeros de la universidad su compañía todos los años de ésta etapa.

En lo personal, me gustaría agradecer a Aitor Aboy y Berta Porras su colaboración en la construcción de la maqueta. A mis padres y a mis amigos, el apoyo, la paciencia, la compresión y en especial, las facilidades que me han dado siempre.

Dedicación especial para mi hermano Diego, y a mis abuelos Paulino y Francisca. Ellos me han inspirado, y me han dado razones más que suficientes para seguir creciendo como persona.

Gracias a todos.

ÍNDEX

ÍNDEX DE IMÁGENES

ÍNDEX DE TABLAS

ÍNDEX DE GRÁFICOS

1. Introducción

El presente documento es el Proyecto Final de Grado realizado por Javier Barrio Ruiz, alumno de la <<Escola Superior d'Enginyeries Industrial, Aerospacial i Audiovisual de Terrassa>>, en el Grado en Ingeniería Electrónica Industrial y Automática, el cuatrimestre de primavera de 2017.

1.1. Antecedentes

En la actualidad existen entornos totalmente automatizados, generalmente se trata de grandes industrias, aunque también existe la posibilidad de automatizar los procesos más cotidianos. Y es que la integración domótica en el hogar, ya sea cableada o inalámbrica, ofrece numerosas posibilidades tanto al usuario como al diseñador de estos sistemas.

Hace una década, lo habitual era encontrar una única unidad central inteligente capaz de activar y desactivar cargas eléctricas. Esta central podía, además, ofrecer la posibilidad de ser accionada a distancia mediante un módem telefónico.

Con el tiempo, la tendencia ha sido distribuir la inteligencia del sistema domótico entre los distintos elementos que lo conforman. Esta circunstancia ha dado lugar a distintos tipos de arquitecturas o topologías.

Ser capaces de controlar, por ejemplo, el encendido o el apagado de las distintas luminarias, la apertura o el cierre de las persianas, además de la sensorización en el domicilio, tal como medir la cantidad de luz que se encuentra en una habitación, o la detección de movimiento en un espacio concreto, permite optimizar energéticamente nuestra vivienda de manera totalmente automatizada.

Por ejemplo, la vivienda podría ser programada para apagar las luces a una hora establecida o pasado cierto tiempo después de determinada acción. El sistema de control podría ser capaz de detectar que el usuario al salir de la vivienda ha dejado luces encendidas y actuar en consecuencia.

Además de un adelanto en el área energética, también supone un extra en la comodidad del usuario. Asimismo, los avances en telefonía móvil facilitan nuevos conceptos en el mundo de la domótica, pudiendo controlar a través de un smartphone los distintos actuadores que se encuentren distribuidos en el hogar, desde cualquier lugar del planeta.

1.2. Objetivos

Los objetivos de éste proyecto se pueden descifrar según el título del proyecto, <<Sistema domótico abierto, de bajo coste, para la mejora de la eficiencia energética>>.

El principal objetivo es la implementación de un sistema domótico de varios nodos que puedan funcionar como una red, donde, uno de los nodos sea el controlador central.

A través del controlador central, se pretende establecer una comunicación a un dispositivo móvil, que funcionará como interfaz virtual para emitir ordenes, además de la interfaz física del sistema.

El segundo objetivo es crear un sistema totalmente abierto, eso supone utilizar hardware y software libre. Es decir, que tanto el software y el hardware del sistema puedan ser estudiado, modificado o utilizado libremente con cualquier fin y redistribuido con o sin cambios o mejoras.

Otro de los objetivos es realizar un sistema de bajo coste de implementación. La domótica hasta hace poco era un producto de alto coste, pero a medida que la tecnología avanza, el precio va reduciéndose poco a poco.

Por último, y como objetivo común en cualquier sistema domótico, se pretende producir un ahorro energético debido a la implementación del sistema en cualquier hogar.

Además de todas estas condiciones, aunque no esté especificado en el título, se propone crear una red local cableada, evitando conexiones inalámbricas entre los controladores del hogar. Concretamente, estudiar las posibilidades que ofrece el bus CAN (Controller Area Network) para ser aplicado en redes domóticas. Ya que es frecuentemente utilizado en la industria del automóvil, y aplicable a la domótica.

1.3. Motivación

Tras unos años de gran derroche a nivel energético y el exceso en la utilización de los recursos a nivel mundial, la tendencia es optimizar los procesos y los recursos utilizados.

La automatización es uno de los procesos industriales que más ha aportado en la optimización de procesos y recursos en general. Los avances en la tecnología a nivel cotidiano dados en los últimos años, permiten que se puedan controlar y también optimizar los procesos en el hogar, y generar un ahorro económico y energético.

Además de la parte técnica, la motivación del proyecto tiene un componente ético-social. Donde se pretende facilitar algunos procesos cotidianos a personas con discapacidad o movilidad reducida.

2. Domótica

Según la CEDOM (Asociación Española de Domótica e Inmótica), la domótica es el conjunto de tecnologías aplicadas al control y la automatización inteligente de la vivienda, que permite una gestión eficiente del uso de la energía, que aporta seguridad y confort, además de comunicación entre el usuario y el sistema.

El sector de la domótica ha evolucionado considerablemente en la última década, y en la actualidad existe una oferta más consolidada. Hoy en día, la domótica aporta soluciones dirigidas a todo tipo de viviendas, incluyendo viviendas de nueva construcción y viviendas las cuales son automatizables a partir de una obra.

Además, actualmente se ofrecen más funcionalidades por menos dinero, más variedad de producto, que, gracias a la evolución tecnológica, concretamente la evolución de la electrónica y la informática, son más fáciles de usar y de instalar.

La domótica contribuye a mejorar la calidad de vida del usuario. Generalmente, la mejoría se centra en cuatro aspectos: eficiencia energética, accesibilidad, confort y seguridad.

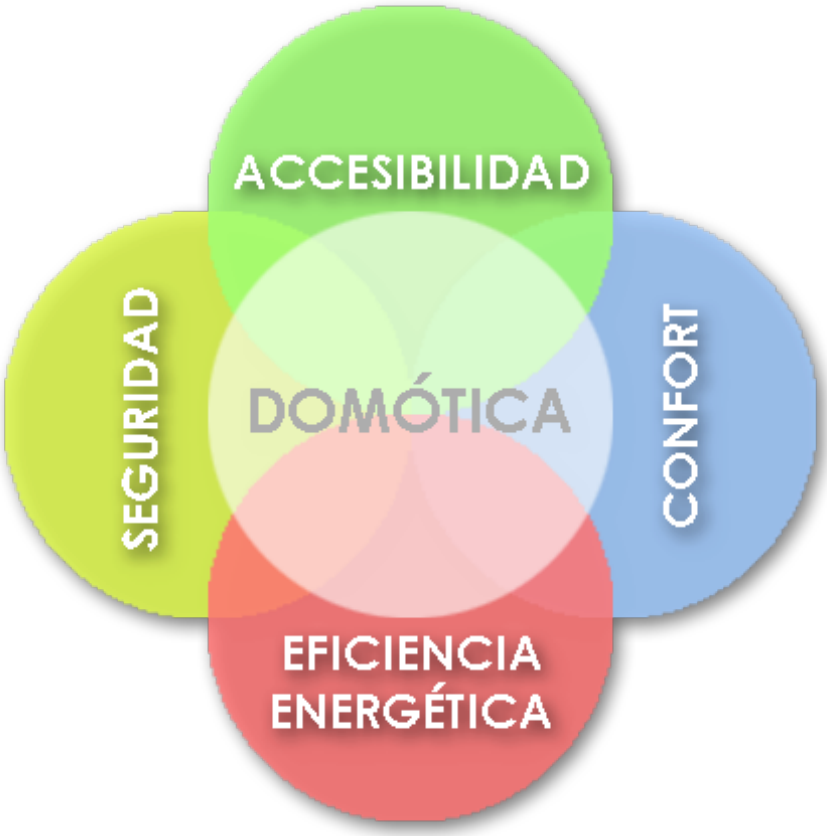


IMAGEN 1. DOMÓTICA

La implantación de un sistema domótico supone una gestión inteligente de la energía consumida en la vivienda. El control de la iluminación, climatización, el caudal de agua y los electrodomésticos entre otros, permite un aprovechamiento mayor de los recursos utilizados y por tanto una reducción de tipo energética y por lo tanto de tipo económica.

También supone un plus en el manejo de los elementos del hogar, facilitando así la accesibilidad al usuario, incluso a personas con ciertas discapacidades, ajustándose a sus necesidades.

Asimismo, la utilización de un sistema domótico en el hogar implica un mayor confort para el usuario, desde el control remoto de los dispositivos a la automatización de encendidos o apagados según la hora del día.

Por último, un sistema domótico aporta seguridad a una vivienda. Mediante subsistemas de vigilancia tales como controles de intrusión, cierres automáticos, simulación de presencia y alarmas, los cuales pueden estar conectados a diversas asistencias.

Los sistemas domóticos, también conocidos como sistemas de automatización, gestión técnica de energía y seguridad para viviendas y edificios, se denominan internacionalmente como HBES (Home and Buildings Electronic Systems). Actualmente la norma que define los requisitos técnicos generales de estos sistemas es la UN-EN 50090-2-2.

2.1. Elementos de un sistema domótico

Un sistema domótico es capaz de recoger información proveniente de unos sensores o entradas, procesarla mediante un controlador y emitir órdenes a unos actuadores o salidas. El sistema puede tener la capacidad de acceder a redes exteriores de comunicación o información o servicios, por ejemplo, la red telefónica conmutada o internet. Estos sistemas se componen de diferentes tipos de elementos:

- **Sensor** – El sensor es el dispositivo que monitoriza el entorno captando información que transmite al sistema (sensores de luz, temperatura, movimiento, humedad, lluvia, agua, gas, humo, viento, etc.).

- **Actuador** – El actuador es un dispositivo capaz de recibir una orden y ejecutarla, así cambiando las características del entorno domótico (encendido/apagado, subida/bajada, apertura/cierre, etc.).

- **Controlador** – Los controladores son los dispositivos que gestionan la información que reciben del sistema y deciden qué hacer en función de la programación previa. Puede haber un solo controlador, o varios distribuidos en función de la arquitectura del sistema.

- **Bus** – Es el medio de comunicación que transporta la información entre los distintos dispositivos, ya sea por una red propia, o por las redes de otros sistemas (red eléctrica, red telefónica, red de datos, etc.). Un sistema puede disponer de diferentes buses.

- **Interfaz** – La interfaz son los dispositivos (pantallas, móvil, Internet, interruptores) en que se muestra la información del sistema para los usuarios y donde elLos mismos pueden interactuar con el sistema.

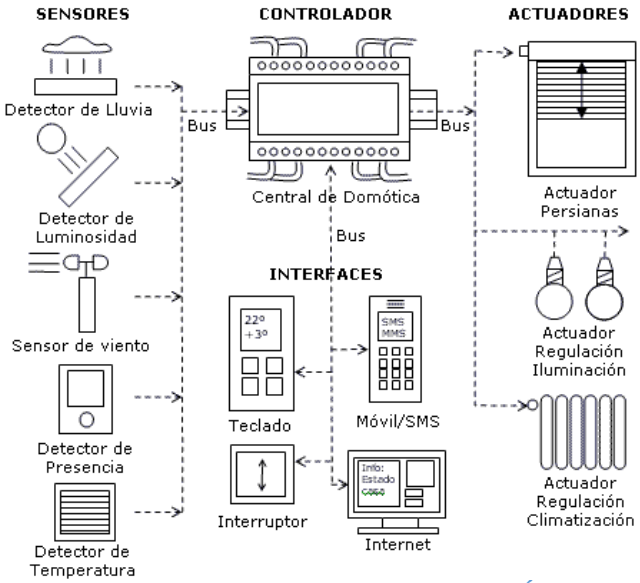


IMAGEN 2. ELEMENTOS DE UN SISTEMA DOMÓTICO

2.2. Modelos de arquitecturas

El tipo de arquitectura de un sistema domótico aporta la información de cómo será la distribución y la ubicación de los elementos del sistema. Los principales tipos de arquitectura son cuatro: arquitectura centralizada, arquitectura descentralizada, arquitectura distribuida y arquitectura híbrida o mixta.

2.2.1. Arquitectura centralizada

En un sistema con arquitectura centralizada, el controlador se dispone en el centro del sistema, ya que ejerce de intermediario entre la información que recibe de los sensores y la orden que envía a los actuadores, mediante las posibles interfaces, según el programa y la configuración establecida por el usuario. El cableado es en estrella cuyo centro es la unidad central de control y no existe comunicación entre sensores y actuadores.

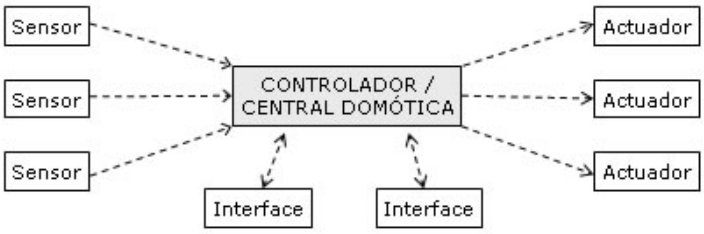


IMAGEN 3. MODELO ARQUITECTURA CENTRALIZADA

2.2.2. Arquitectura descentralizada

Sin embargo, en los sistemas de arquitectura descentralizada coexisten diversos controladores, interconectados por un bus. La inteligencia del sistema se reparte, y cada controlador contiene su propia configuración. El bus permite el envío de información entre los controladores y también a los distintos actuadores e interfaces conectados a los controladores.

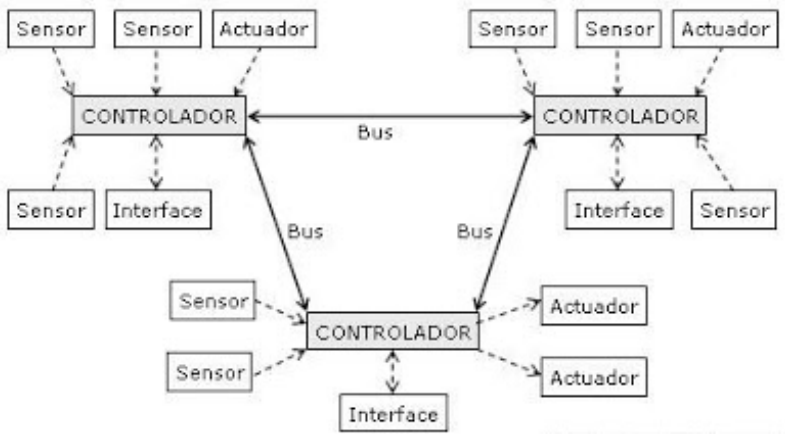


IMAGEN 4. MODELO ARQUITECTURA DESCENTRALIZADA

2.2.3. Arquitectura distribuida

También existe el modelo de arquitectura distribuida, donde cada sensor y actuador ejerce también de controlador capaz de actuar y enviar información al sistema, e interactúa con los distintos elementos del sistema. Todos los elementos disponen de un acoplador al bus con una interfaz de acceso compartido y técnicas de direccionamiento para que la recepción y el envío de información quede definida y el dialogo entre elementos esté asegurado.

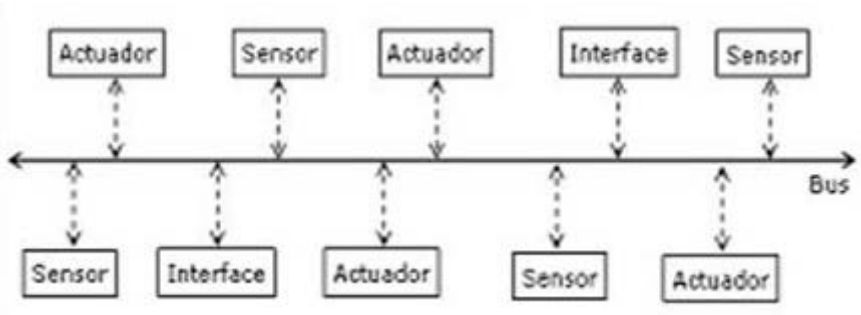


IMAGEN 5. MODELO ARQUITECTURA DISTRIBUIDA

2.2.4. Arquitectura mixta o híbrida

Finalmente, en el modelo de arquitectura mixta se combinan las diversas arquitecturas de los sistemas centralizadas, descentralizadas y distribuidas. A la vez que puede disponer de un controlador central o varios controladores descentralizados, los dispositivos de interfaces, sensores y actuadores pueden también ejercer de controladores, como en un sistema distribuido.

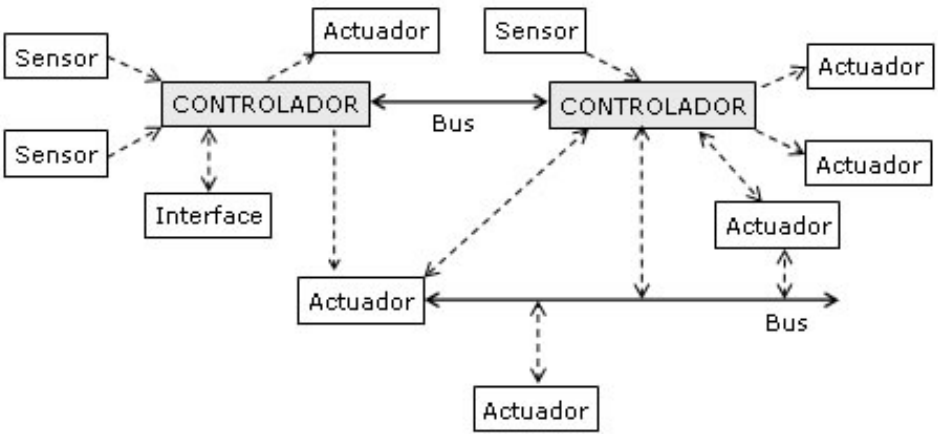


IMAGEN 6. MODELO ARQUITECTURA MIXTA

2.3. Buses de transmisión

Como todo sistema automatizado, siguiendo el modelo OSI (Open Systems Interconection), un sistema domótico precisa de un medio de comunicación entre sus distintos componentes para un correcto funcionamiento. En la industria, éstos medios de comunicación se denominan buses de campo.

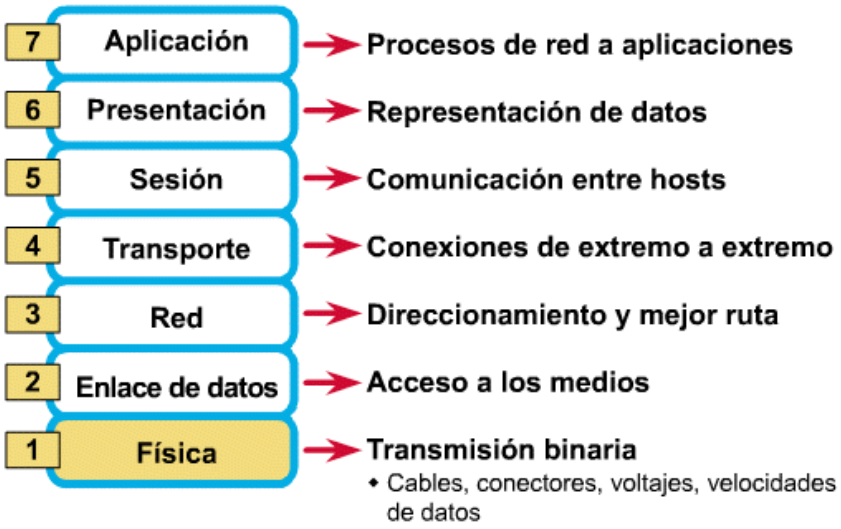


IMAGEN 7. NIVELES DEL MODELO OSI

Un bus de campo es un sistema de trasmisión de información que simplifica la instalación y operación de máquinas y dispositivos. Generalmente son redes digitales, bidireccionales, multipunto, montadas sobre un bus serie, que conectan equipamientos de campo como PLC's, transductores, actuadores y sensores.

Los buses de campo deben ser capaces de hacer frente a los problemas que plantea la industria:

- Control integrado en diferentes niveles de información y diferentes procesos en una misma planta, manteniendo la seguridad digital y la protección física de los datos.

- Cantidad elevada de sensores y actuadores, y una posible ampliación de la cantidad de dispositivos conectados a la red.

- Distancia de control de mando superior a varias decenas de metros, sin perder una velocidad de comunicación eficiente.

En cuanto a un sistema domótico, este no deja de ser una red industrial a pequeña escala, con un menor número de dispositivos conectados entre sí, a la cual se le pueden aplicar los mismos principios que un sistema automatizado.

Por lo que, también se necesitarán buses de transmisión en el sistema domótico a diseñar en el presente proyecto.

A continuación, se encuentras las características principales de los protocolos más frecuentes en el campo de la domótica.

2.3.1. Protocolos de comunicación frecuentes en la domótica

Existen diferentes tipos de buses de transmisión según los protocolos utilizados y el medio físico utilizado para la comunicación del sistema. Los protocolos más utilizados en la domótica son los siguientes.

2.3.1.1. KNX

El estándar de protocolo de comunicaciones de red KNX, basado en OSI, es uno de los más utilizados para sistemas de domótica e inmótica en Europa.

KNX, también conocido como Konnex, es el sucesor y la convergencia de tres estándares previos: el European Home Systems Protocol (EHS), el European Installation Bus (EIB o Instabus) y el BatiBUS pertenecientes, respectivamente, a la EHSA (European Home Systems Association), la EIBA (European Installation Bus Association) y el BCI (BatiBUS Club International).

IMAGEN 8. LOGO DE KNX

El objetivo general de KNX es crear un único estándar europeo para la automatización de las viviendas y oficinas, y de manera concreta los aspectos clave de la convergencia son:

• Crear un único estándar para la domótica e inmótica que cubra todas las necesidades y requisitos de las instalaciones profesionales y residenciales de ámbito europeo.

• Aumentar la presencia de estos buses domóticos en áreas como la climatización o HVAC.

• Mejorar las prestaciones de los diversos medios físicos de comunicación. Sobre todo, en la tecnología de radiofrecuencia.

• Introducir nuevos modos de funcionamiento que permitan aplicar una filosofía Plug and Play (PnP) a muchos de los dispositivos típicos de una vivienda.

El PnP es una arquitectura de software abierta y distribuida que permite a las aplicaciones de los dispositivos conectados a una red intercambiar información y datos de forma sencilla y transparente para el usuario final, sin necesidad de que este tenga que ser un experto en la configuración de redes, dispositivos o sistemas operativos.

• Contactar con empresas proveedoras de servicios, como las de telecomunicaciones y las eléctricas con el objeto de potenciar las instalaciones de telegestión técnica de las viviendas o domótica.

Respecto al nivel físico el estándar puede funcionar sobre: par trenzado (TP1), par trenzado (TP0), ondas portadoras (PL100), ondas portadoras (PL132), Ethernet y radiofrecuencia, aprovechando las normas equivalentes de los protocolos que convergen.

En cuanto a las topologías de conexión admitidas por los sistemas EIB son todas las existentes: centralizadas, descentralizadas y distribuidas.

2.3.1.2. SCP

El Simple Control Protocol (SCP) es un protocolo para redes de control, utilizado en la automatización de edificios y viviendas, generalmente en los Estados Unidos.

El SCP surge en el estado norteamericano como una iniciativa similar a KNX en Europa. Éste protocolo recoge las principales características de otros protocolos como LonWorks, X-10, CEBus, entre otros.

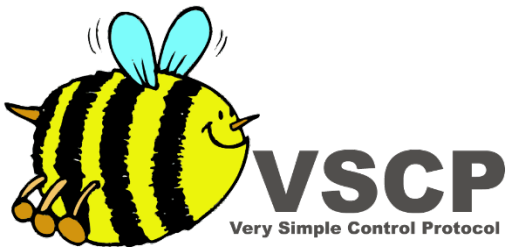


IMAGEN 9. LOGO DE VSCP

El principal objetivo, en el que trabajan conjuntamente con empresas como Microsoft, es el desarrollo del Universal Plug&Play.

A nivel físico el SCP ha escogido una solución basada en la transmisión de datos por las líneas de baja tensión (ondas portadoras) que ya estaba desarrollada en el protocolo CEBus.

El SCP está optimizado para el uso en dispositivos de eléctricos y electrónicos que tienen una memoria y una capacidad de proceso muy limitadas. Igual que otros buses o protocolos de control distribuido, el SCP está diseñado para funcionar sobre redes de control con un ancho de banda muy pequeño (< 10 Kbps) y optimizado para las condiciones de ruido características de las líneas de baja tensión.

2.3.1.3. PROFIBUS

El protocolo de comunicaciones PROFIBUS, es característico en aplicaciones industriales, aunque en ocasiones también se utiliza para sistemas domóticos.

Éste protocolo se caracteriza por su funcionalidad y amplio campo de adaptación dentro de la industria. Este campo abarca desde el nivel de sensores y actuadores hasta el nivel de celdas. Es un protocolo bastante robusto, pero también más complejo que otros de los protocolos mencionados hasta ahora.

IMAGEN 10. LOGO DE PROFIBUS

Utiliza la misma técnica de transmisión y el mismo protocolo de acceso al bus con funciones de aplicación diferenciadas. Esto permite una reducción significativa de esfuerzo en la instalación, mantenimiento y entrenamiento.

Basado en la comunicación controlada entre maestro-esclavo. Definimos de manera particular estos dispositivos como:

- Dispositivos maestros (Master Devices) - Entre estas estaciones activas rota un permiso de acceso y control que les permite enviar mensajes sin necesidad de petición.
- Dispositivos secundarios (Slaves Devices) - Son los periféricos asignados a los maestros, una serie de dispositivos lo suficientemente inteligentes como para seguir las normas del protocolo. Su papel es pasivo, pudiendo sólo transmitir cuando se les ha realizado una petición previa.

Los medios físicos de transmisión de datos más frecuentes para PROFIBUS son el RS-485, IEC 1158-2 y la fibra óptica.

2.3.1.4. Modbus

El protocolo Modbus se utiliza principalmente en aplicaciones de control industrial. Modbus utiliza la disposición maestro-esclavo donde cada dispositivo inicia toda la actividad de comunicación.

Actualmente existen dos sistemas ModBus principales: ModBus RS-485 y ModBus TCP/IP. Ambos se basan en define un bus de transmisión serie multipunto, donde, en un instante, puede haber un equipo transmitiendo y varios recibiendo. La comunicación es semiduplex, de forma un equipo puede enviar y recibir, pero no realizar ambas acciones a la vez.

El medio físico básico consiste en un par de hilos de cobre trenzados sobre el que se transmite una señal diferencial para enviar los bits de datos, que es bastante inmune a las interferencias y admite largas distancias.

Además del par trenzado para datos, pueden usarse líneas de 0V y 5V para alimentar dispositivos del bus. Los bits se transmiten mediante una trama asíncrona.

IMAGEN 11. LOGO DE MODBUS

Algunas de las ventajas del ModBus son: bajo coste de implementación y estandarización, la ampliación de nodos esclavos es sencilla, el fallo de un nodo no afecta al funcionamiento del resto, trabaja con gran variedad de formatos de datos.

2.3.1.5. BACnet

El BACnet (Building Automation and Control Networks) es un protocolo de comunicación de datos norteamericano, diseñado para comunicar entre sí los diferentes aparatos electrónicos presentes en los edificios actuales.

En sus inicios, se definió como un protocolo que implementaba la arquitectura de niveles OSI y que utilizaba la tecnología RS-485 como nivel físico.

IMAGEN 12. LOGO DE BACNET

El aporte más característico de BACnet es la definición de un conjunto de reglas, de hardware y de software, que permiten comunicar dos dispositivos independientemente de si éstos utilizan protocolos diferentes tales como los antiguos EIB, EHS, BatiBus, LonWorks, TCP/IP, etc.

Como particularidad, utiliza la denominación “objeto” para las entradas y salidas, analógicas y digitales, lazos de control y los dispositivos del sistema.

2.3.2. Niveles físicos frecuentes en la domótica

Existen diferentes niveles físicos utilizados para conexionar los elementos que conforman el sistema automatizado de una vivienda, independiente del lenguaje o protocolo que estén utilizando. A continuación, se mencionan los medios físicos más comunes en un sistema domótico.

2.3.2.1. Par trenzado

El par trenzado se basa en un bus de comunicación cuya instalación puede hacerse en diversas topologías: bus, estrella, anillo, árbol o cualquier combinación de estas. Siempre respetando la asignación de direcciones por cada dispositivo de la instalación.

IMAGEN 13. PAR TRENZADO

Consiste en dos alambres de cobre, o a veces de aluminio, aislados. Los alambres se trenzan con el propósito de reducir la interferencia eléctrica de pares similares cercanos. Los pares trenzados se agrupan bajo una cubierta común de PVC.

Posee un bus de control independiente y está pensado para nuevas instalaciones y grandes renovaciones. Además, también posee un nivel máximo de fiabilidad en la transmisión y una gran velocidad de transmisión.

En su contra, se necesita una instalación adicional a la red eléctrica del hogar.

2.3.2.2. Corrientes portadoras

Este medio de transmisión de corrientes portadoras, es también conocido como Power Line Carrier (PLC).

El sistema hace posible la transmisión de telegramas a través de la red eléctrica del hogar. La distancia máxima que se puede lograr sin repetidor es de 600 metros.

IMAGEN 14. CORRIENTES PORTADORAS

De este modo, no es necesaria una línea de bus independiente. La transmisión de telegramas tiene lugar a través de los conductores de fase y neutro, los cuales deben estar conectados a cada uno de los aparatos.

Debido a la indefinición de las condiciones de la red, que pueden variar constantemente, la transmisión de telegramas puede verse interrumpida, producir errores en el sistema o interferir con otros sistemas.

Si bien no es el medio más adecuado para la transmisión de datos, sí es una alternativa a tener en cuenta para las comunicaciones domésticas, dado el bajo coste que implica su uso al tratarse de una instalación ya existente.

2.3.2.3. Cable Coaxial

El cable coaxial fue el primer cable empleado, aparte de cable eléctrico convencional en la transmisión de información entre dispositivos y existen diferentes tipos según su uso y utilización. El cable coaxial tenía una gran utilidad en sus inicios por su propiedad idónea de transmisión de voz, audio y video.

Los factores a tener en cuenta a la hora de elegir un cable coaxial son su ancho de banda, su resistencia o impedancia característica, su capacidad y su velocidad de propagación.

IMAGEN 15. CABLE COAXIAL

El ancho de banda del cable coaxial está entre los 500 MHz, esto hace que el cable coaxial sea ideal para transmisión de televisión por cable por múltiples canales. La resistencia o la impedancia característica depende del grosor del conductor central o malla; si varía éste, también varía la impedancia característica.

Es mucho más rígido que el par trenzado, por lo que al realizar las conexiones entre redes la labor será más difícil.

2.3.2.4. Fibra óptica

En la última década la fibra óptica ha pasado a ser una de las tecnologías más avanzadas que se utilizan como medio de transmisión. Los logros con este material fueron más que satisfactorios, desde lograr una mayor velocidad y disminuir casi en su totalidad ruidos e interferencias, hasta multiplicar las formas de envío en comunicaciones y recepción por vía telefónica.

La fibra óptica está compuesta por filamentos de vidrio de alta pureza muy compactos. El grosor de una fibra es como la de un cabello humano aproximadamente. Fabricadas a alta temperatura con base en silicio, su proceso de elaboración es controlado por medio de computadoras, para permitir que el índice de refracción de su núcleo, que es la guía de la onda luminosa, sea uniforme y evite las desviaciones.

IMAGEN 16. FIBRA ÓPTICA

Como características de la fibra podemos destacar que son compactas, ligeras, con bajas pérdidas de señal, amplia capacidad de transmisión y un alto grado de fiabilidad ya que son inmunes a las interferencias electromagnéticas de radio-frecuencia.

Las fibras ópticas no conducen señales eléctricas, conducen rayos luminosos, por lo tanto, son ideales para incorporarse en cables sin ningún componente conductivo y pueden usarse en condiciones peligrosas de alta tensión.

En comparación con el sistema convencional de cables de cobre, donde la atenuación de sus señales es de tal magnitud que requieren de repetidores cada dos kilómetros para regenerar la transmisión, en el sistema de fibra óptica se pueden instalar tramos de hasta 70 km sin que haya necesidad de recurrir a repetidores.

2.3.2.5. Radiofrecuencia

Además de conexionado por diferentes tipos de cables, también existen transmisiones de datos de forma inalámbrica.

En la transmisión por radiofrecuencia se emplean señales de radio para transmitir telegramas. Permite implementaciones unidireccionales y bidireccionales, además, se caracteriza por su bajo nivel de consumo energético.

La introducción de las radiofrecuencias como soporte de transmisión en la vivienda, ha venido precedida por la proliferación de los teléfonos inalámbricos y sencillos telemandos.

Este medio de transmisión puede parecer, en principio, idóneo para el control a distancia de los sistemas domóticos, dada la gran flexibilidad que supone su uso.

Sin embargo, resulta particularmente sensible a grandes distancias y a perturbaciones electromagnéticas producidas, tanto por los medios de transmisión, como por los equipos domésticos. No obstante, no cabe ninguna duda que las tecnologías inalámbricas en radiofrecuencia son las que más se van a desarrollar en los próximos años.

Actualmente los medios de transmisión de radiofrecuencia más utilizados en los sistemas domóticos son Bluetooth, WiFi (IEEE 802.11b, g), y ZigBee (IEEE 802.15.4).

IMAGEN 17. LOGOS DE BLUETOOTH, WI-FI, ZIGBEE

Bluetooth es un enlace radio de corto alcance que aparece asociado a las Redes de Área Personal Inalámbricas, o sus siglas en inglés WPAN (Wireless Personal Area Network). Este concepto hace referencia a una red sin cables que se extiende a un espacio de funcionamiento personal con un radio de 10 a 100 metros según la clase de Bluetooth utilizado.

En cuanto al medio de transmisión WiFi (Wireless Fidelity), estipulado en la norma del IEEE (Institute of Electrical and Electronic Engineers) 802.11, representa el primer estándar de una organización independiente reconocida a nivel internacional, que además ha definido las principales normas en redes LAN cableadas.

Las ventajas que se pueden notar para este nivel físico en el caso de un edificio, es la ausencia de cableado para crear una red de datos, dentro de las limitaciones del sistema. Como inconvenientes aportaremos dos fundamentales, una es la seguridad y otro es el coste de los dispositivos que incorporaran WiFi.

Finalmente, en la norma IEEE 802.15.4 se estipula el medio de transmisión ZigBee. Es una tecnología inalámbrica de baja velocidad y bajo consumo, con velocidades comprendidas entre 20kB/s y 250kB/s y rangos de 10 a 75 metros.

ZigBee puede utilizar las bandas libres ISM de 2,4 GHz, 868 MHz (Europa) y 915 MHz (EEUU). Una red ZigBee puede estar formada por hasta 255 nodos.

2.4. Eficiencia energética

Desde un punto de vista social, el ahorro y la eficiencia energética no sólo aseguran el abastecimiento energético y mejoran el medio ambiente, sino que también ayudan a incrementar la competitividad del sector industrial, beneficiando el aumento del Producto Interior Bruto del país.

Según los datos que aparecen en la Guía práctica de la energía. Consumo eficiente y responsable, publicada en el 2007 por el IDAE (Instituto para la Diversificación y el Ahorro de la Energía), los españoles cada vez consumimos más energía. A nivel mundial, al ritmo actual, sólo se tardarán 35 años en duplicar el consumo de energía y menos de 55 años en triplicarlo.

El consumo de energía de las familias españolas supone ya un 30% del consumo total de energía del país, el 18% corresponde al consumo doméstico. Cada hogar es responsable de producir hasta 5 toneladas de CO2 anuales.

Las familias españolas, con sus pautas de comportamiento, son decisivas para conseguir que los recursos energéticos se utilicen eficientemente. En cuanto al consumo eléctrico, un hogar medio consume unos 4.000 kWh al año.

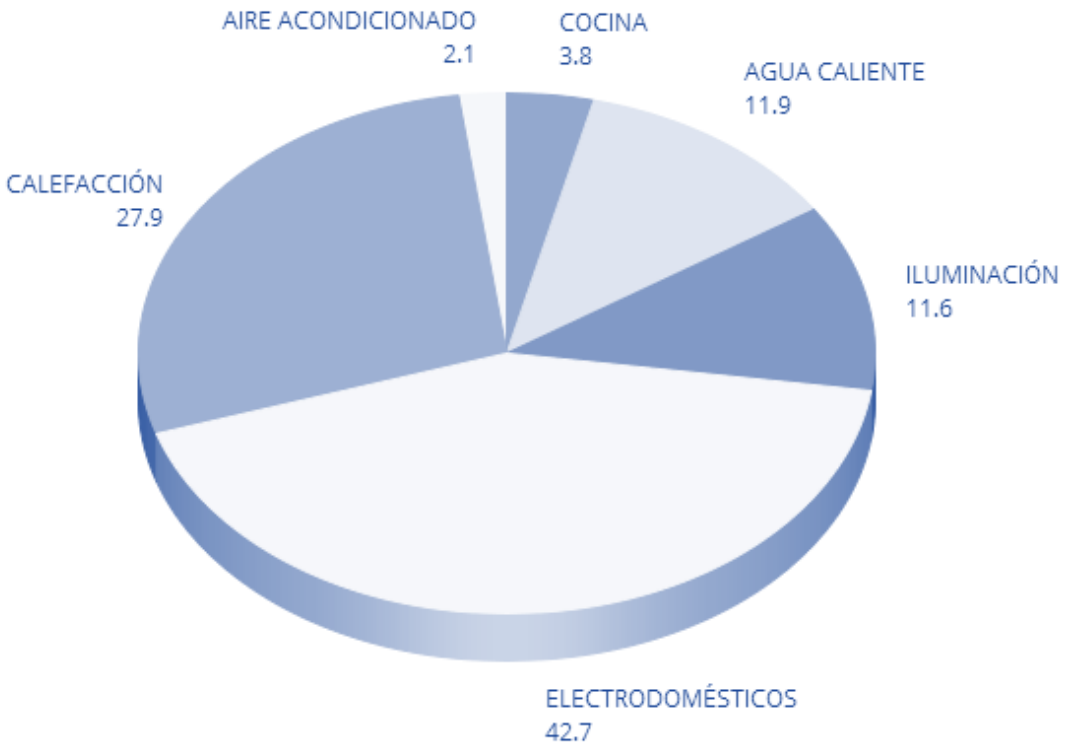


GRÁFICO 1. PORCENTAJE DE CONSUMO ELÉCTRICO MEDIO EN EL HOGAR EN ESPAÑA

Además de la necesidad de reducir el consumo de energía para contribuir a la disminución de la contaminación, se debe tener en cuenta el factor económico. El coste del consumo energético de los hogares españoles para una familia supone al año unos 900 €, distribuidos según se explica a continuación.

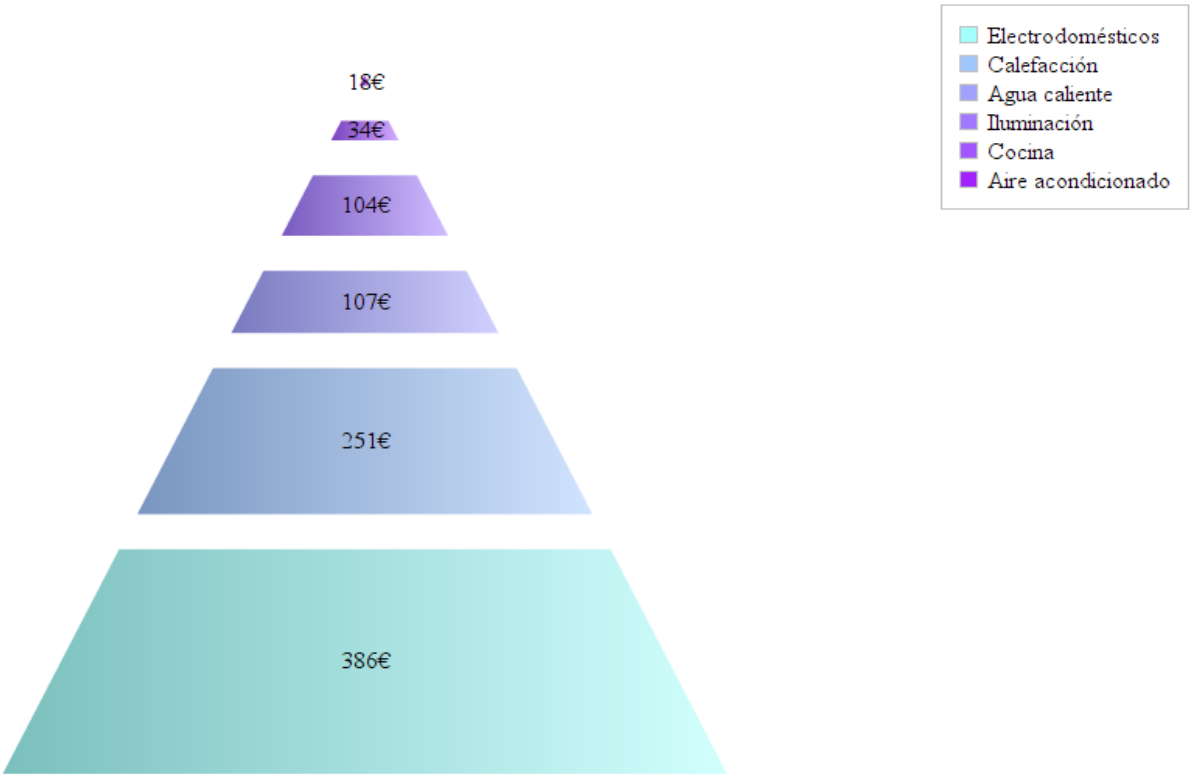


GRÁFICO 2. COSTE MEDIO ANUAL DEL GASTO ENERGÉTICO EN EL HOGAR EN ESPAÑA

España importa el 75% de la energía primaria que consume, frente al 50% de media de la UE. Un menor consumo implicaría reducir la dependencia energética de España respecto al exterior.

Los precios de la electricidad, el agua, y los combustibles como el gas natural evolucionan con una tendencia alcista como consecuencia del carácter perecedero de las energías no renovables y el imparable incremento de la intensidad energética (indicador que relaciona el consumo de energía y el Producto Interior Bruto). En los últimos 5 años el precio del gas y la electricidad han aumentado en torno a un 15%.

El creciente consumo de energía y la limitación de los recursos energéticos generan efectos negativos en el medio ambiente que se reflejan en dos aspectos:

- Económico: los precios de la energía tienden a subir, por lo que un control del consumo energético incrementa significativamente el ahorro para el usuario.
- Ecológico: el usuario puede disminuir el impacto negativo sobre su entorno si disminuye su consumo de energía.

La domótica gestiona elementos de control que contribuyen al ahorro de agua, electricidad y combustibles, notándose sus efectos tanto en el aspecto económico, menos coste, como en el ecológico, menos consumo de energía.

Por lo tanto, existen varias maneras de ahorrar energía. Para ahorrar electricidad, se utilizan:

- Sistemas de iluminación eficientes: adaptan el nivel de iluminación en función de la variación de la luz solar, la zona de la casa o la presencia de personas, ajustándola a las necesidades de cada momento.

  Por ejemplo, detectan la presencia de personas en zonas de paso, como los pasillos de la vivienda o de las zonas comunes de un edificio, y las iluminan sólo cuando es necesario.

- Control automático inteligente de toldos, persianas y cortinas de la vivienda: permite que se aproveche al máximo la luz solar.

- Control automático del encendido y apagado de todas las luces de la vivienda: permite evitar el dejarse luces encendidas al salir de casa.

- Control de forma automática del encendido y apagado de las luces exteriores en función de la luz solar.

- Sistemas de regulación de la calefacción: adaptan la temperatura de la vivienda en función de la variación de la temperatura exterior, la hora del día, la zona de la casa o la presencia de personas.

- Control o secuenciado de la puesta en marcha de electrodomésticos: programando su funcionamiento en horarios en los que el precio de la energía es menor.

- Detección y gestión del consumo en espera de los electrodomésticos.

- Programación de la desconexión de circuitos eléctricos no prioritarios, como por ejemplo, el del aire acondicionado, antes de alcanzar la potencia contratada.

También se pueden utilizar mecanismos para ahorrar en el uso de combustibles:

- Detección de la apertura y cierre de ventanas: avisan al usuario de si hay ventanas abiertas cuando está activada la climatización.

- Implantación de sistemas de control y regulación de fugas de fas: permiten detectar y avisar en caso de averías como, por ejemplo, una fuga de gas, provocando un corte del suministro que evite los peligros que pudieran ocasionarse. Además, la domótica facilita una buena gestión del mantenimiento de las instalaciones, con el consecuente ahorro económico que esto supone.

Finalmente, también se puede ahorrar en el consumo de agua:

- Sistemas de control y regulación de fugas de agua: detectan si se produce una inundación, dan señal de aviso, y provocan un corte del suministro.

- Control inteligente de riego: a través de un sensor de humedad o de lluvia, detecta la humedad del suelo y de forma autónoma riega sólo cuando es necesario.

- Sistemas de medición de la calidad del agua: facilitan la gestión del reciclaje de aguas grises.

- Grifos inteligentes: gestionan el caudal y la temperatura del agua.

Además, cualquier tipo de ahorro de agua, aunque no se trate de agua caliente, conlleva un ahorro energético, ya que el agua es impulsada hacia nuestras viviendas mediante bombas eléctricas que consumen energía.

En la actualidad, los sistemas domóticos ofrecen una gran variedad de funcionalidades orientadas a monitorizar el consumo de agua, de combustibles y el consumo eléctrico de todos los sistemas de la vivienda.

Esto permite hacer una gestión personalizada del consumo (consumo por franjas horarias, diario, mensual, etc.), así como detectar malos funcionamientos de los equipos del hogar. La información obtenida permite optimizar el ahorro energético en el futuro y corregir las pautas de comportamiento.

IMAGEN 18. MONITORIZACIÓN DE UN SISTEMA DOMÓTICO

Monitorizar la calidad del suministro eléctrico permite, además, notificar remotamente la información al suministrador de electricidad, mejorando así el funcionamiento global del sistema de distribución eléctrica para ajustar con más exactitud los patrones de producción a los hábitos de consumo.

3. Bus CAN

Controller Area Network (CAN) es un protocolo de comunicación serie que soporta eficientemente el control distribuido en tiempo real con un alto nivel de seguridad.

En el capítulo anterior no se menciona el bus CAN entre los protocolos más usuales en el sector domótico. Para averiguar si CAN era un bus aplicable a una red domótica, ha sido necesario llevar a cabo un estudio sobre este protocolo.

Para agilizar la lectura, en este capítulo sólo se presentan los conceptos básicos necesarios para poder abordar el resto de secciones. El resto de información recopilada se presenta en los anexos para ofrecer una documentación más completa.

3.1. Características principales

Las características de este bus podrían justificar su uso en este proyecto, tal y como se verá en el siguiente capítulo, donde se describe el sistema diseñado. Algunas de las características principales de este bus de transmisión son las siguientes:

• Económico y sencillo: Dos de las razones que motivaron su desarrollo, por parte del sector automovilístico fueron precisamente la necesidad de economizar el coste y el de minimizar la complejidad del cableado.

• Estandarizado: Se trata de un estándar definido en las normas ISO (Internacional Organization for Standardization), concretamente la ISO 11898, que se divide a su vez en varias partes, cada una de las cuales aborda diferentes aspectos de CAN.

• Medio de transmisión adaptable: El cableado, como ya hemos dicho, es muy reducido a comparación de otros sistemas. Además, a pesar de que por diversas razones el estándar de hardware de transmisión sea un par trenzado de cables, el sistema de bus CAN también es capaz de trabajar con un solo cable.

• Estructura definida: La información que circula entre las unidades a través del bus, son paquetes de bits (0's y 1's) con una longitud limitada y con una estructura definida de campos que conforman el mensaje.

• Programación sencilla.

• Número de nodos: Es posible conectar hasta 64 dispositivos en un único bus CAN.

• Garantía de tiempos de latencia: CAN aporta la seguridad de que se transmitirá cierta cantidad de datos en un tiempo concreto, es decir, que la latencia de extremo a extremo no excederá un nivel específico de tiempo. Además, la transmisión siempre será en tiempo real, siempre que no se superen el número de nodos expuesto anteriormente.

• Optimización del ancho de banda: Los métodos utilizados para distribuir los mensajes en la red, como el envío de estos según su prioridad, contribuyen a un mejor empleo del ancho de banda disponible.

• Desconexión autónoma de nodos defectuosos: Si un nodo de red cae, sea cual sea la causa, la red puede seguir funcionado, ya que es capaz de desconectarlo o aislarlo del resto. De forma contraria, también se pueden añadir nodos al bus sin afectar al resto del sistema, y sin necesidad de reprogramación.

• Velocidad flexible: ISO define dos tipos de redes CAN: una red de alta velocidad (de hasta 1 Mbps) definida por la ISO 11898-2, y una red de baja velocidad tolerante a fallos (menor o igual a 125 Kbps) definida por la ISO 11898-3.

• Relación velocidad-distancia: Al punto anterior habría que añadir que la velocidad también depende de la distancia hasta un máximo de 1000 metros sin bridges o repetidores.

Además, podemos hablar de otras características técnicas en las que profundizaremos más adelante:

• Orientado a mensajes: Se trata de un protocolo orientado a mensajes, y no a direcciones, es decir, la información que se va a intercambiar se descompone en mensajes, a los cuales se les asigna un identificador y son encapsulados en tramas para su transmisión. Cada mensaje tiene un identificador único dentro de la red, a partir del cual los nodos deciden aceptar o no dicho mensaje. Además, están priorizados.

• Multidifusión (multicast): Permite que todos los nodos puedan acceder al bus de forma simultánea con sincronización de tiempos.

• Medio compartido (broadcasting): La información es enviada en la red a todos los destinos de forma simultánea. Así que los destinos habrán de saber si la información les concierne o deben rechazarla.

• Detección y señalización de errores: CAN posee una gran capacidad de detección de errores, tanto temporales, como permanentes, lograda a través de cinco mecanismos de detección, 3 al nivel de mensaje y 2 a nivel de bit. Los errores además pueden ser señalizados.

• Retransmisión automática de tramas erróneas: Junto a la detección y señalización de errores la retransmisión automática de tramas erróneas aporta la integridad de los datos. Además, ambos procesos son transparentes al usuario.

• Jerarquía multimaestro: CAN es un sistema multimaestro en el cual puede haber más de un maestro (o master) al mismo tiempo y sobre la misma red, es decir, todos los nodos son capaces de transmitir, hecho que permite construir sistemas inteligentes y redundantes.

3.2. Especificaciones

El aumento de la complejidad de los sistemas electrónicos para automoción y la exigencia de mayor seguridad y confort por parte de los usuarios motivaron a la multinacional Bosch a diseñar un bus de campo que diera solución a estas necesidades.

IMAGEN 19. LOGO DE BUS CAN

El ejemplo más habitual de aplicación del bus CAN es el sistema ABS de los automóviles, que requiere la actuación conjunta de las revoluciones del motor y del carburador para reducir el par cuando una rueda motriz patina.

Aunque inicialmente, el bus CAN fue utilizado para la automoción, sus características le permiten adaptarse a un amplio rango de aplicaciones, desde redes de alta velocidad hasta cableado de bajo coste para múltiples elementos, con velocidades de hasta 1 Mbit/s, pudiendo controlar máquinas, sensores, etc.

3.2.1. Protocolo CAN

CAN ha sido estandarizado internacionalmente de manera que numerosos fabricantes de semiconductores han desarrollado circuitos integrados basados en este estándar.

De acuerdo con el modelo OSI, CAN se subdivide en capas, de las cuales el estándar ISO 11898 define las dos primeras, capa física y capa de enlace. No existen los niveles del 3 al 6 puesto que se pasa directamente a la capa de aplicación desde la de enlace. Una peculiaridad de CAN es que las capas de enlace y aplicación no están totalmente separadas, sino que guardan un cierto vínculo.

Existen dos partes dentro de la especificación CAN 2.0, que es la especificación utilizada actualmente. Básicamente la única diferencia que existe entre CAN 2.0 A y CAN 2.0 B, además de utilizar nombres diferentes para las subcapas del nivel de enlace, son los diferentes formatos de tramas que utilizan.

La especificación CAN 2.0A utiliza un formato de trama estándar, con identificadores de 11 bits, compatible con anteriores versiones de CAN.

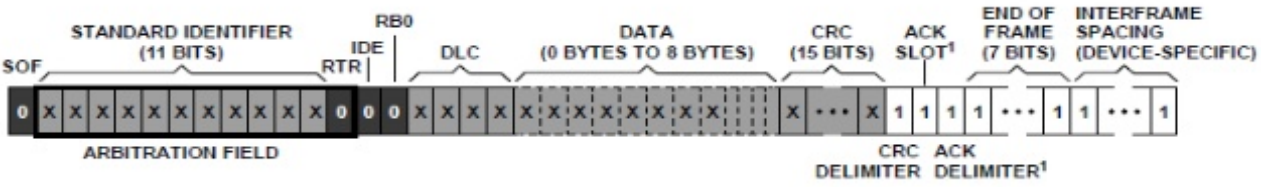


IMAGEN 20. ESPECIFICACIÓN BUS CAN 2.0A

Mientras que la especificación CAN 2.0B utiliza formatos de trama extendida, con identificadores de 29 bits.

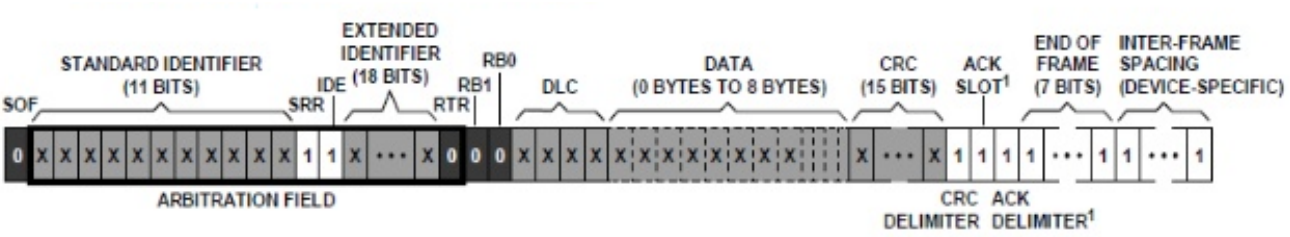


IMAGEN 21. ESPECIFICACIÓN BUS CAN 2.0B

Antes de entrar en detalle con los diferentes formatos de las tramas de datos, es conveniente citar las dos características más importantes de CAN, clave esencial para comprender las enormes ventajas que trae consigo utilizar este bus:

• Los identificadores no son direcciones de nodos concretos, sino que expresan el contenido del mensaje. Por ejemplo: podríamos asignar un identificador para los datos de temperatura, otro para los de velocidad, etc. De esta manera se pueden establecer comunicaciones punto a multipunto ya que sólo los nodos a los que interese el mensaje lo recibirán.

• Las colisiones de datos se resuelven mediante un arbitraje de bits en el que gana el nodo con más prioridad, sin que esto suponga la modificación de ninguno de los bits que este nodo ha transmitido. Esto evita grandes pérdidas de tiempo innecesarias.

Respecto a la capa de enlace de datos, es la capa responsable del acceso al medio y el control lógico y está dividida a su vez en dos niveles:

• El subnivel LLC (Logical Link Control): Gestiona el filtrado de los mensajes, las notificaciones de sobrecarga y la administración de la recuperación.

• El subnivel MAC (Medium Acces Control): Es el núcleo del protocolo CAN y gestiona el tramado y desentramado de los mensajes, el arbitraje a la hora de acceder al bus y el reconocimiento de los mensajes, así como el chequeo de posibles errores y su señalización, el aislamiento de fallos en unidades de control y la identificación del estado libre del bus para iniciar una transmisión o recepción de un nuevo mensaje.

El protocolo CAN proporciona un acceso multimaestro al bus con una resolución determinista de las colisiones. La capa de enlace de datos define el método de acceso al medio, así como los tipos de tramas para el envío de mensajes.

Existe una variedad de tipos de tramas en un sistema que utilice bus CAN. Además de la trama de datos (Data Frame), existen la trama remota (Remote Frame), trama de error (Error Frame), trama de sobrecarga (Overload Frame). Aunque la trama a destacar en éste proyecto es la trama de datos, para poder entender el envío de datos del sistema domótico.

Data Frame es la trama utilizada por un nodo normalmente para transmitir mensajes de datos otro nodo. Puede incluir entre 0 y 8 bytes de información útil.

Los mensajes de datos consisten en celdas que envían datos y añaden información definida por las especificaciones CAN:

- **Inicio de trama (Start Of Frame):** El inicio de trama es una celda de 1 bit siempre dominante que indica el inicio del mensaje, sirve para la sincronización con otros nodos.

- **Celda de arbitraje (Arbitration Field):** Es la celda que concede prioridad a unos mensajes o a otros:

  En formato estándar tendrá 11 bits seguidos del bit RTR (Remote Transmision Request) que en este caso será dominante.

  En formato extendido serán 11 bits de identificador base y 18 de extendido. Tras los 11 primeros bits del identificador, se encuentra el bit SRR (Substitute Remote Request) y el bit IDE y tras el último bit del identificador se encontraría el bit RTR, que en este caso será recesivo.

- **Celda de control (Control Field):** El campo de control está formado por, un bit indicador, uno o dos bits reservados para uso futuro y cuatro bits adicionales que indican el número de bytes de datos.

  El primero de estos bits, el bit de extensión de identificador (IDE), como su propio nombre indica, se utiliza para indicar si la trama es de CAN Estándar (IDE dominante) o Extendido (IDE recesivo).

  El segundo bit, es el bit reservado RB0, es siempre recesivo, en el formato extendido también se encuentra otro bit reservado RB1, siempre recesivo, en la posición anterior a RB0.

  Los cuatro bits de código de longitud (DLC) indican en binario el número de bytes de datos en el mensaje (0 a 8).

- **Celda de datos (Data Field):** Es el campo de datos, que puede tener una longitud de 0 a 8 bytes.

- **Código de redundancia cíclica (CRC):** Tras comprobar este código se podrá comprobar si se han producido errores en la transmisión.

- **Celda de reconocimiento (Acknowledgement):** es un campo de 2 bits que indica si el mensaje ha sido recibido correctamente. El nodo transmisor pone este bit como recesivo y cualquier nodo que reciba el mensaje lo pone como dominante para indicar que el mensaje ha sido recibido. Los bits ACK sirven como acuse de recibo.
- **Fin de trama (End Of Frame):** Consiste en 7 bits recesivos sucesivos e indica el final de la trama.

- **Espaciado entre tramas (IFS):** Consta de un mínimo de 3 bits recesivos.

3.2.2. Nivel físico

En el presente apartado, se analizará con más detalle la capa de nivel físico del bus de transmisión CAN. El nivel físico del medio de transmisión CAN está estandarizado por el ISO en la norma ISO 11898.

La transmisión puede efectuarse de dos formas, la primera es a través de una sola línea, siempre que todos los nodos tengan una referencia de tierra común y los niveles de tensión estarían referidos a tierra.

La segunda es a través de dos hilos en modo diferencial. Nos centraremos en esta última forma puesto que es la que regula el estándar ISO 11898.

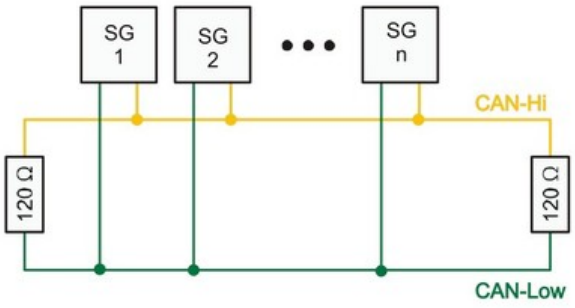


IMAGEN 22. EJEMPLO DE CONEXIÓN FÍSICA BUS CAN

El bus está formado por dos líneas, CAN-H (High) y CAN-L (Low), dentro de un cable, que puede estar apantallado, por lo tanto, podemos utilizar cable de par trenzado y/o blindado según los requerimientos electromagnéticos.

Es muy importante reseñar que el bus debe estar terminado en los extremos con resistencias de terminación de 120Ω. Además, no es conveniente incluir las resistencias de terminación en los nodos que están colocados en los extremos puesto que, si estos nodos son retirados, el bus se quedará sin terminación y se pueden dar reflexiones que imposibiliten una correcta comunicación.

Además del cableado y de las resistencias de cierre de bus, también existen otros elementos en la transmisión con bus CAN.

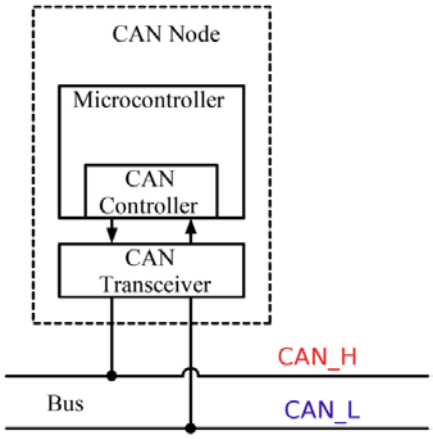


IMAGEN 23. CAPA FÍSICA DE BUS CAN

El controlador, que debe ser capaz de acondicionar y gestionar la información que entra y sale del nodo. Y el transceptor o receptor, que es el elemento que une el controlador, por los terminales Rx y Tx, con el bus, por medio de los terminales CAN-H y CAN-L, y que tiene la misión de recibir y de trasmitir los datos para que pueda ser utilizada por los controladores.

La especificación ISO 11898 no define el tipo de conectores y cables a usar, pero sí define los parámetros eléctricos mínimos que deben cumplir los materiales. En el caso del cable se recomienda que tengan una impedancia característica en torno a los 120 Ω, una resistencia de menos de 70mΩ/m y un retraso de línea específico de menos de 5ns/m.

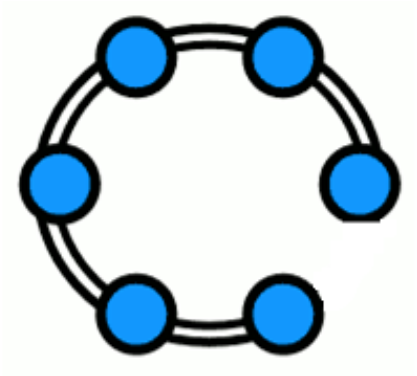


IMAGEN 24. CONEXIÓN FÍSICA EN ANILLO ABIERTO

En cuanto a la topología hay que procurar que las conexiones entre nodos se parezcan lo más posible a una línea recta para evitar reflexiones. Además, la conexión física entre nodos se realiza en forma de bus, que, en la realidad queda conectado como una especie de red en anillo abierto. Los parámetros que debe cumplir la topología, para especificaciones de 1Mbit/s son:

Parámetro	Notación	Mínimo	Máximo
Longitud del bus (Metros)	L	0	40
Longitud latiguillo (Metros	I	0	0,3
Distancia entre nodos	D	0,1	40

TABLA 1. PARÁMETROS FÍSICOS DE BUS CAN

En la realización de la transmisión, el bus está en estado recesivo cuando todos los transmisores están desactivados. La tensión de las líneas del bus en este caso es generada por las resistencias de terminación y los circuitos de recepción de los nodos, que muestran una impedancia alta entre las líneas del bus.

Un bit dominante es enviado al bus cuando al menos uno de los nodos tiene habilitado su transmisor y quiere escribir un bit dominante. Esto provoca un flujo de corriente a través de las resistencias de terminación y consecuentemente una tensión diferencial entre ambas líneas del bus. El bus puede estar en uno de los dos estados: recesivo o dominante.

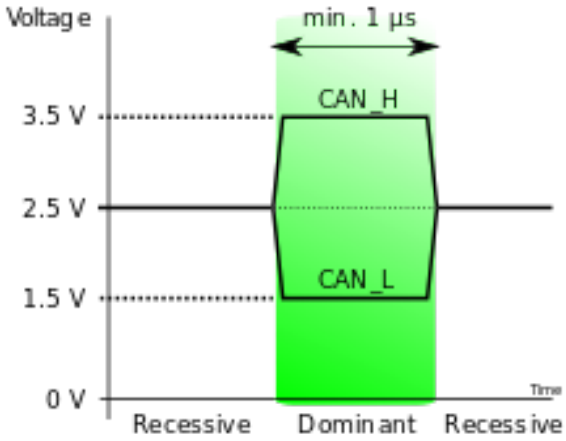


IMAGEN 25. LÍNEAS CAN-H Y CAN-L

En el estado recesivo, las tensiones en CAN-H y CAN-L son fijadas al nivel de tensión de modo común, y se considerarán como un '1' lógico siempre que la tensión diferencial no supere un cierto umbral máximo. El bus está en estado recesivo cuando se quiere transmitir un '1' o cuando el bus está en reposo.

RECESIVO				
Parámetro	Notación	Mínimo	Típico	Máximo
Tensión del bus en modo común (V)	V_{CANH}		2,5	7,0
	V_{CANL}	-2,0	2,5	
Tensión diferencial (mV)	V_{DIFF}	-120,0	0	12,0

TABLA 2. PARÁMETROS BIT RECESIVO

En el estado dominante la tensión diferencial es mayor que un umbral mínimo. Un bit dominante sobreescribe a un bit dominante y ocurre cuando uno o más nodos quieren transmitir un '0' lógico.

DOMINANTE				
Parámetro	Notación	Mínimo	Típico	Máximo
Tensión del bus en modo común (V)	V_{CANH}		3,5	7,00
	V_{CANL}	-2,0	1,5	
Tensión diferencial (mV)	V_{DIFF}	1,2	2,0	3,0

TABLA 3. PARÁMETROS BIT DOMINANTE

Las limitaciones del bus dependen de varios factores como por ejemplo la velocidad de transmisión y la longitud del bus, que son inversamente proporcionales.

En la siguiente tabla se muestra una comparativa entre la longitud máxima del bus y la velocidad:

Velocidad (Kbps)	Tiempo de Bit (µS)	Longitud Máxima (Metros)
10	100	5000
20	50	2500
50	20	1000
125	8	500
250	4	250
500	2	100
800	1,25	50
1000	1	40

TABLA 4. VELOCIDAD, TIEMPO Y LONGITUD MÁXIMA DE BUS CAN

Éstas velocidades y longitudes, son más que suficientes para cualquier sistema domótico, pues el tiempo de respuesta no es necesariamente alto, ni urge tanta urgencia como, por ejemplo, en el sistema ABS de un automóvil.

Otra de las limitaciones que hay que tener en cuenta es el número de nodos que se pueden insertar en la red. Aunque la norma no especifica ningún límite, lo cierto es que el bus está limitado por carga y dependiendo del tipo de transceiver que usemos podremos poner más o menos nodos. Como valor de referencia se puede establecer un máximo de 64 nodos.

La capa física se encarga también de otros aspectos tales como la codificación de bits, los tiempos de bit y la sincronización.

3.3. Detección de errores

Si por alguna razón destaca el bus CAN como medio de transmisión, es sobretodo unos métodos implacables de detección de errores en el envío y recepción de tramas.

Existen cinco tipos de errores que se pueden dar en un bus CAN:

- **Error de bit.** Al tiempo que un nodo transmite un bit al bus, también monitoriza el nivel real en el bus, y cuando el valor detectado es diferente del valor enviado se genera un error de bit.

  Excepto cuando se está transmitiendo un bit que pertenece al campo de arbitraje o a la ranura de asentimiento. Tampoco se interpreta como un error de bit cuando un nodo envía una señal de error pasivo mientras otros nodos ponen en el bus un nivel dominante debido a sus señales de error activo.

- **Error en bits de relleno.** Cuando se detectan 6 bits consecutivos con el mismo valor en alguno de los campos codificados con el método de bits de relleno, se genera este tipo de error.

- **Error en CRC.** Los receptores recalculan el CRC de los mensajes que le están llegando y si detectan discrepancia entre el CRC que han obtenido y el que reciben en la trama, lo señalan como error en el CRC.

- **Error de formato.** Este tipo de error ocurre cuando se detectan bits ilegales en campos con un formato fijo. Si se detecta un bit dominante durante el último bit del campo Final de trama, no se considera como error de formato.

- **Error de asentimiento.** Si un nodo transmisor no detecta un nivel dominante durante la ranura de asentimiento de la trama que está transmitiendo, genera este error.

En cuanto se detecta alguno de estos errores se transmite inmediatamente a partir del siguiente bit una trama de error utilizando una señal de error activo o pasivo según el estado del nodo, a excepción de los errores en CRC. Ante errores en el CRC, la trama de error se transmite justo después del delimitador de asentimiento siempre que no se haya producido alguna de las otras condiciones de error antes.

Para evitar algunos de estos errores, existen diferentes métodos para perfeccionar la transmisión en el bus:

- Monitorización de la información escrita al bus.
- Código de redundancia cíclico.
- Comprobación de la estructura de la trama.
- Bits de relleno.

Los tres primeros dependen de la propia estructura del mensaje, pero el método por bits de relleno requiere una mención especial y una breve explicación.

Los bits de relleno, o también conocido como bit stuffing, no es más que la inserción de un bit de polaridad opuesta después de cinco bits consecutivos de la misma polaridad.

Este procedimiento se utiliza para asegurar que hay suficientes transiciones recesivo-dominante y garantizar así la sincronización. Esta práctica es necesaria debido a la codificación sin vuelta a cero del protocolo CAN. Los bits insertados son eliminados por el receptor.

Todos los campos de la trama son rellenados a excepción del delimitador CRC, el acuse de recibo ACK, y el fin de trama. Cuando un nodo detecta seis bits consecutivos iguales en un campo susceptible de ser rellenado lo considera un error y emite un error activo. Un error activo consiste en seis bits consecutivos dominantes y viola la regla de relleno de bits.

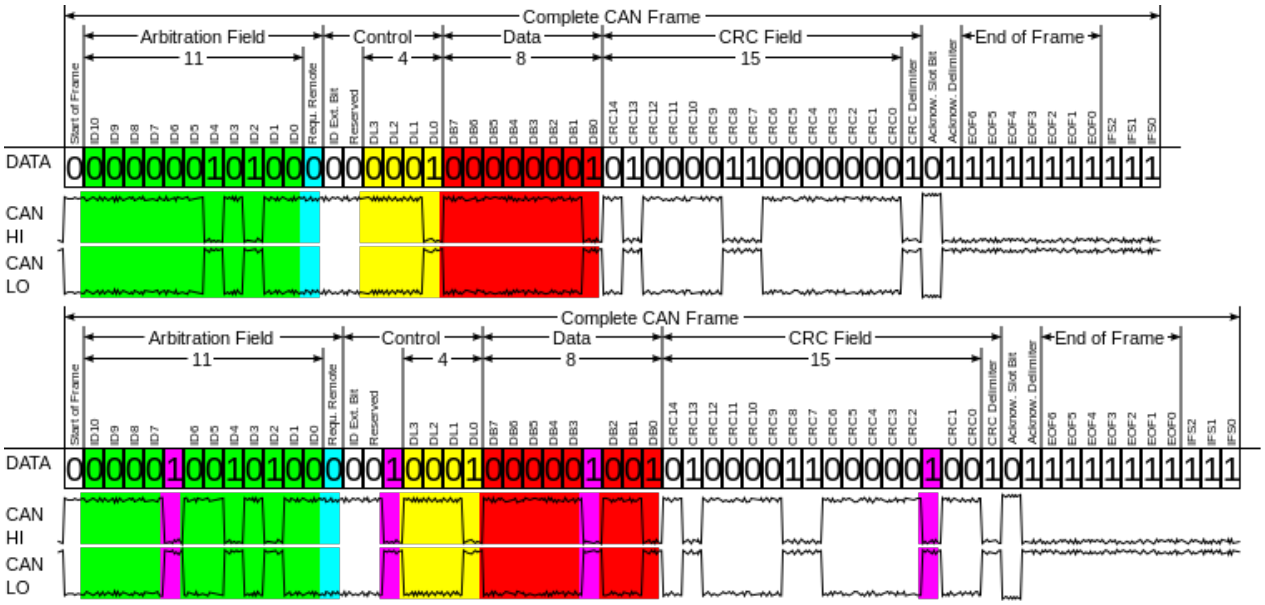


IMAGEN 26. BITS DE RELLENO BUS CAN

La regla de los bits de relleno implica que una trama puede ser más larga de lo esperado si se suman los bits teóricos de cada campo de la trama.

El flujo de bits en un mensaje es codificado de acuerdo con el método de no retorno a cero (NRZ). Esto significa que, durante todo el tiempo de bit, el nivel generado se mantiene.

Finalmente, existen otros procedimientos para asegurar la correcta funcionalidad del bus CAN:

- **Fault confinement.** También conocido como confinamiento de nodos defectuosos. Los nodos CAN distinguen entre errores temporales y permanentes, siendo capaces de desactivar los nodos defectuosos para que no perturben el funcionamiento de la red. En el cual, se diferencia tres estados de un nodo:

  Un nodo en error activo puede tomar parte en las comunicaciones del bus con normalidad y enviar señales de error activo cuando detecte un error.

  Un nodo en error pasivo también puede tomar parte en las comunicaciones, pero a la hora de señalar un error debe hacerlo con la señal de error pasivo. Además, después de una transmisión, los nodos en error pasivo deben esperar un tiempo antes de iniciar una nueva transmisión.
  Un nodo en bus off, está desactivado y no puede ejercer influencia alguna sobre el bus. Para hacer efectivo el mecanismo de confinamiento de nodos defectuosos, se implementan dos contadores en cada nodo, el contador de errores de transmisión y el contador de errores de recepción.

- **Asentimientos.** Todos los receptores comprueban la consistencia de un mensaje que está siendo recibido y lo asentirán si el mensaje es consistente. En caso contrario lo señalarán con la pertinente trama de error.

- **Sleep mode.** Para reducir el consumo los nodos CAN pueden entrar en este modo si no van a ser utilizados. El nodo saldrá de este modo bien porque detecte actividad en el bus o bien porque el sistema requiera que el nodo CAN vuelva a estar operativo.

3.4. Máscaras y filtros

Aunque la especificación CAN no regula del todo la forma de intercambiar información entre el sistema microprocesador y el controlador CAN, lo cierto es que la mayoría de dispositivos establecen para ello una serie de buzones (mailboxes).

En el caso de los buzones de transmisión, cada uno de ellos lleva un campo con el identificador que incluirán en sus mensajes y en el caso de los buzones de recepción, cada uno puede llevar uno o más filtros.

Los filtros se componen de un identificador y una máscara. De esta manera en un buzón de recepción pueden entrar un grupo de mensajes siempre que el identificador del mensaje entrante coincida con el identificador del filtro en los bits especificados en la máscara. A continuación, se muestra un ejemplo de funcionamiento

	BITS											HEXA
ID 1	1	0	0	0	1	0	1	1	0	1	0	0x45A
ID 2	0	1	0	1	1	0	0	0	1	1	0	0x2C6
ID 3	1	0	1	1	0	1	1	0	0	1	1	0x5B3
ID FILTRO	1	0	0	0	0	0	0	0	0	0	0	0x400
MÁSCARA	1	1	1	0	0	0	0	0	0	0	0	0x700

TABLA 5. EJEMPLO DE FILTRO BUS CAN

La máscara fija con un '1' los valores que deben ser iguales en la ID entrante respecto a la ID del filtro. Por descontado, el formato de la trama de datos es muy importante, antes de realizar cualquier filtro debemos saber si es formato estándar o formato extendido

En el formato estándar las ID de los mensajes no podrán superar el valor 0x7FF debido a su longitud de 11 bits, mientras que, en el formato extendido, las ID de los mensajes comprenderán valores formados por una composición de 29 bits.

En el caso del ejemplo, el filtro con ID = 0x400 y la máscara con ID = 0x700, sólo permitirá pasar mensajes de datos, al mailbox especificado, con una identificación de valores hexadecimales comprendidos entre 400÷4FF.

Según la tabla anterior, el primer ejemplo identificador (ID1) sí entraría al mailbox que especifiquemos por código con éste filtro, en cambio, el segundo y el tercer ejemplo identificador (ID2, ID3) no entrarían a dicho mailbox.

En el caso de los buzones configurados para recibir tramas remotas, el controlador no guarda los datos de éstas puesto que no contienen datos, y en su lugar inicia automáticamente una transmisión de una trama de datos con la información que estuviera guardada en dicho buzón por el sistema microprocesador.

4. Descripción de los elementos del sistema

Una vez explicados los diferentes elementos de un sistema domótico, los modelos de arquitectura, y los distintos buses de transmisión, procedemos al planteamiento del proyecto en singular, y a determinar todos los componentes de éste proyecto.

Éste capítulo se dividirá en dos secciones, el software y el hardware utilizado. Como se comentó en la introducción, el objetivo de éste proyecto, tal y como su propio nombre indica, es el diseño de un sistema domótico, de bajo coste y abierto.

Cumpliendo esas condiciones, y habiendo tomado la decisión de utilizar bus CAN como elemento de transmisión del sistema, sólo queda determinar que supone un sistema abierto.

El concepto de sistema abierto proviene del término anglosajón Open Source. El término abierto o libre, se refiere al hardware (dispositivos), cuyas especificaciones y diagramas esquemáticos son de acceso público, ya sea bajo algún tipo de pago, o de forma gratuita; y al software (programa informático) que puede ser copiado, estudiado, modificado, utilizado libremente con cualquier fin y redistribuido con o sin cambios o mejoras.

Teniendo en cuenta las características hasta ahora mencionadas, a continuación, se muestra un esquema sencillo del proyecto final para poder entender a qué se referencia cada uno de los elementos que serán explicados en éste capítulo.

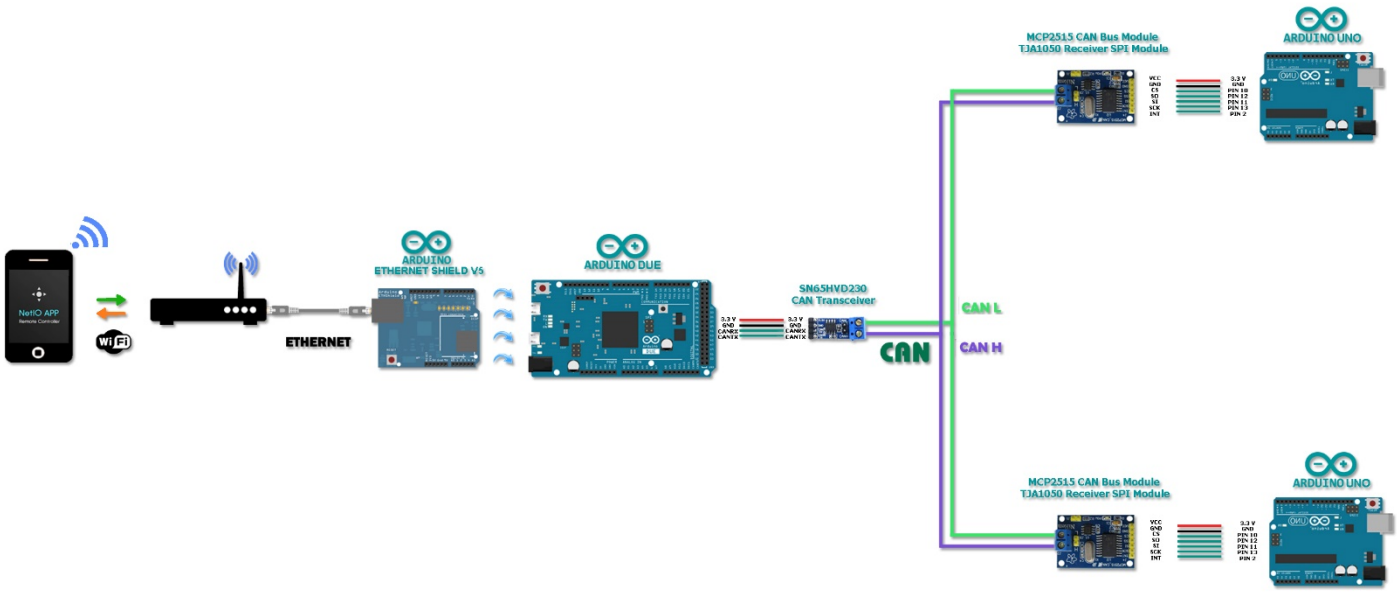


IMAGEN 27. ESQUEMA GENERAL DEL SISTEMA DOMÓTICO DISEÑADO

4.1. Software utilizado

En uno de los capítulos anteriores del proyecto se describían los diferentes elementos de un sistema domótico, entre los cuales, se encontraban los controladores y las diferentes interfaces de un sistema.

En éste sistema debemos diferenciar dos tipos de software utilizados. En primer lugar, el software utilizado para la programación de los controladores del sistema, Arduino. En segundo lugar, la aplicación para smartphone NetIO, que ha permitido crear una interfaz virtual a través de un móvil, además de una interfaz física con pulsadores.

Finalmente, también se han utilizado librerías específicas de algunos de los componentes del sistema, para poder anexarlos al sistema de control.

4.1.1. Arduino

Arduino es una plataforma Open Source (código abierto) con un entorno de desarrollo integrado (IDE), basado en el lenguaje de programación Processing/Wiring, y que utiliza un lenguaje de programación con grandes semejanzas a C y C++. El microcontrolador de la placa se programa mediante un computador, usando una comunicación serial.

Arduino nació en Italia en el año 2005 como un proyecto para los estudiantes del Instituto de diseño interactivo Ivrea. Su nombre proviene del Bar di Re Arduino (Bar del Rey Arduino), establecimiento que rinde homenaje a un antiguo monarca italiano y en el que uno de los fundadores del proyecto (Massimo Banzi) pasaba algunas horas.

Arduino es una plataforma abierta que facilita la programación de un microcontrolador. Los microcontroladores nos rodean en nuestra vida diaria, usan los sensores para escuchar el mundo físico y los actuadores para interactuar con el mundo físico. Los microcontroladores leen sobre los sensores y escriben sobre los actuadores.

Arduino se enfoca en acercar y facilitar el uso de la electrónica y programación de sistemas embebidos en proyectos multidisciplinarios. Toda la plataforma, tanto para sus componentes de hardware como de software, son liberados con licencia de código abierto que permite libertad de acceso a ellos.

4.1.2. NetIO App

La otra gran aplicación que se ha utilizado en éste proyecto es NetIO App. Esta aplicación universal permite controlar todos los dispositivos conectados a su red local mediante un controlador remoto genérico.

Desde cualquier smartphone, permite enviar y leer información en formato String, cadenas de texto, a través de un socket o canal TCP/UDP, o la consulta de interfaces HTTP.

Permite una fácil comunicación con los microcontroladores conectados a su LAN (por ejemplo, AVR-Board, Arduino, HomeVision, Raspberry Pi, Beaglebone, TI Launchpad, microcontrolador casero, entre otros), o con otro software de ordenador, por ejemplo, software multimedia como XBMC.

Por lo que permite, indirectamente, el control de dispositivos infrarrojos o leer datos de sensores, como la temperatura. También se utiliza para muchos proyectos de internet de las cosas (IoT), y posibilita en gran medida la automatización del hogar.

En éste proyecto, se utilizará ésta aplicación para smartphones como interface, además de los posibles pulsadores físicos.

La aplicación, tras una descarga con un coste adicional, permite la realización de varios proyectos en un mismo dispositivo, o en más de uno a la vez.

4.1.3. Librerías

Además de un software concreto, también se necesitarán librerías para poder realizar las pertinentes comunicaciones entre los diferentes dispositivos del sistema. Vulgarmente, se puede decir que las librerías se utilizan para que los distintos dispositivos se entiendan entre ellos.

Las librerías son un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.

A diferencia de un programa ejecutable, el comportamiento que implementa una biblioteca no espera ser utilizada de forma autónoma, sino que su fin es ser utilizada por otros programas, independientes y de forma simultánea.

El uso de éstas facilita mucho la programación y hace que el programa sea más sencillo de hacer y de entender.

En éste proyecto se han utilizado diferentes librerías, para la transmisión de datos, y la conexión de diferentes sensores y actuadores.

LIBRERÍA	CREADOR	VERSIÓN	UTILIDAD
due_can	Collin Kidder (collin80)	2.0.1	Librería orientada a objetos, específica para placas Arduino Due que tengan que conectarse a bus CAN.
Ethernet	Arduino LLC	1.1.2	Librería que permite a una placa Arduino conectarse a Internet. La placa puedehacer de servidor o de cliente.
CAN_BUS_Shield	Seeed Studio	1.0.0	Librería que permite a una placa Arduino conectarse a bus CAN mediante un controlador de bus CAN MCP2515 con interfaz SPI y el transceptor CAN MCP2551.
SPI	Arduino LLC	1.0.0	Librería que permite a una placa arduino comunicarse por SPI (Serial Peripheral Interface) con uno o más dispositivos periféricos rápidamente en distancias cortas.
DHT	Adafruit	1.2.3	Librería que permite a una placa Arduino conectarse a un sensor de temperatura y humedad de modelo DHT.
RTClib	Adafruit	1.2.0	Librería que permite a una placa Arduino conectarse a un reloj de modelo RTC.
Wire	Arduino LLC	1.0.0	Librería que permite unir dos dispositivos o sensores conectados mediante la tecnología I2C.

TABLA 6. LIBRERÍAS UTILIZADAS EN EL PROYECTO

Entre las cuales, se pueden distinguir las librerías utilizadas para el bus de transmisión CAN. Entre las que se encuentran *due_can*, *CAN_BUS_Shield* y *SPI*, ya que según el hardware utilizado también utilizaremos el protocolo de comunicación SPI para transmitir datos por bus CAN.

En la actualidad, la información sobre las librerías de bus CAN en internet es reducida en comparación a otras librerías, debido a un menor uso por parte de los usuarios. Lo que implica una mayor labor a la hora de programar con dicha librería.

También se ha utilizado la librería *Ethernet* para poder comunicar el sistema con internet, conjuntamente con la librería *SPI* ya mencionada. Es decir, a través de un router y un dispositivo móvil con WiFi, podemos conectarnos al sistema a través de la red de internet, mediante una shield Ethernet de Arduino.

Finalmente, las librerías restantes han sido utilizadas para el resto de componentes del sistema. La librería *DHT* ha sido utilizada para el manejo del sensor de temperatura, y las librerías *RTClib* y *Wire* han sido utilizadas para el control de un reloj de tiempo real (Real Time Clock).

4.2. Hardware utilizado

En éste apartado, se pueden diferenciar tres sectores claros: controladores, mecanismos de transmisión, por último, sensores y actuadores.

Los controladores utilizados en éste sistema han sido placas de Arduino. La interconexión de ésta placas ha sido realizada mediante un bus de transmisión CAN, por lo que también se han utilizado módulos adaptadores de CAN específicos para cada tipo de controlador.

También se ha utilizado un router que ha permitido crear una red local, sin conexión a internet, a la cual se ha conectado el controlador central del sistema mediante una shield de Ethernet y un cable de red, con un dispositivo móvil, en éste caso un Android con la aplicación NetIO.

Dicha conexión, podría ser perfectamente a la propia red de internet si habilitamos los puertos del router en modo abierto, entre otras opciones, aunque pudiera contraer algunos problemas en la seguridad del sistema.

Finalmente, se explican con detalle los diferentes sensores y actuadores utilizados en el sistema domótico diseñado.

A continuación, se describen las especificaciones de todo el hardware utilizado, que ha sido programado para un control automático y manual mediante el software y las librerías anteriormente mencionadas.

4.2.1. Arduino

Arduino, además de ser un software de código abierto, también es una plataforma que se dedica a crear hardware libre. En concreto, se trata de una familia de placas electrónicas con microcontrolador, capaces de ser programadas y de ejecutar las órdenes grabadas en su memoria.

El conjunto de entradas y salidas analógicas y digitales del dispositivo, permite controlar todo tipo de actuadores y sensores, acoplar tarjetas de expansión (comunicación, etc.) e interaccionar con otros circuitos.

Además, las placas cuentan con interfaces de comunicación serie, USB en la mayoría de modelos, que permite cargar los programas desde un ordenador personal sin dificultad alguna.

Existe una gran variedad de placas con diferentes características como la tensión utilizada (3,3 ó 5 V dependiendo del microcontrolador empleado), el número de entradas y salidas, el procesador instalado, la memoria o la posibilidad de alimentar otros elementos desde la placa, entre otras.

En éste proyecto se han utilizado los modelos Arduino Uno y Arduino Due como controladores para el sistema domótico a diseñar, debido a una fácil programación y la estandarización de éstos dispositivos en la realización de proyectos de tipo industrial y automático.

La gran ventaja es que ambos modelos nos permiten conectar shields para trabajar con el bus de transmisión que hemos escogido, en éste caso CAN.

4.2.1.1. Arduino Uno

Un Arduino UNO dispone de 14 pines digitales que pueden configurarse como entrada o salida, de las cuales 6 pueden actuar como PWM. Además, también dispone de 6 entradas analógicas. Entre algunas de sus características principales se encuentran:

- Microcontrolador ATmega328
- Voltaje de funcionamiento: 5V
- Voltaje de entrada (recomendado): 7-12V
- Voltaje de entrada (límites): 6-20V
- Pines digitales: 14 (6 con opción PWM)
- Pines analógicos: 6
- Corriente en pines de entrada-salida: 40mA
- Corriente en pin de 3.3V: 50mA
- Memoria Flash: 32KB
- SRAM: 2KB
- EEPROM: 1KB
- Velocidad reloj: 16MHz
- Dimensiones: 68.6x53.4mm

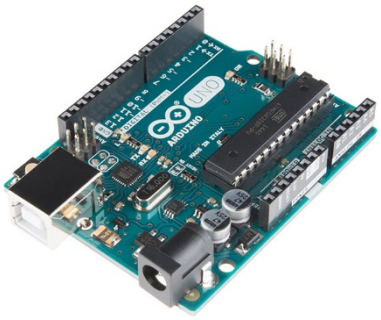


IMAGEN 28. ARDUINO UNO

Se ha optado por su utilización debido a que la universidad posee estás placas para su uso en prácticas y prototipos. Se podría haber utilizado otro modelo de Arduino o incluso otro tipo de microcontrolador. Su función en el sistema es de controlador secundario.

4.2.1.2. Arduino Due

El Arduino Due, de la misma familia de placas que Arduino UNO, dispone de unas características diferentes al anterior. No es tan comúnmente utilizado, aunque tiene unas prestaciones que lo hacen realmente atractivo para ser el controlador central de éste proyecto.

El voltaje de funcionamiento es de 3,3V, menos común, y eso puede dar ciertos problemas con algunos de los pines de la placa a la hora de conectarlos con otros dispositivos.

Pero la principal ventaja de éste microcontrolador en el presente proyecto, es que contiene el controlador de CAN, con dos canales diferenciados, integrado en su electrónica.

Además, dispone de más pines de conexión que Arduino UNO, entre algunas de sus características principales:

- Microcontrolador: AT91SAM3X8E
- Voltaje de operación: 3.3V
- Voltaje de entrada recomendado: 7-12V
- Voltaje de entrada min/max: 6-20V
- Digital I/O Pins: 54 (12 proveen PWM)
- Analog Input Pins: 12
- Analog Outputs Pins: 2
- Corriente total de salida DC I/O: 130 mA
- Corriente DC para el Pin de 3.3V: 800 mA
- Corriente DC para el Pin de 5V: 800 mA
- Memoria Flash: 512 KB
- SRAM: 96 KB (two banks: 64KB and 32KB)
- Clock Speed: 84 MHz

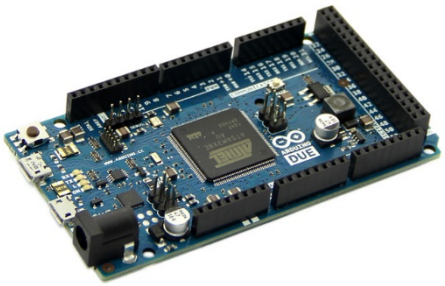


IMAGEN 29. ARDUINO DUE

Como se puede observar, es una placa algo más potente que Arduino UNO. La versión utilizada, también dispone de la electrónica que permitiría conectar Arduino Due a Ethernet, aunque en la placa, faltan las pistas al microprocesador que nos garantizarían la conexión. Por lo que también se ha utilizado una shield Ethernet de Arduino.

4.2.1.3. Arduino Ethernet Shield

El Arduino Ethernet Shield permite a una placa Arduino conectarse a internet. Contiene un chip que es capaz de conectarse vía TCP y UDP. Soporta hasta cuatro conexiones de sockets (canales) simultáneas. En éste proyecto se ha utilizado la versión Ethernet Shield V5 o 05. Aunque la función es la misma, hay pequeñas variaciones entre las diferentes versiones. La versión V5 de la shield, utiliza el Wiznet W5100 e incluye una ranura para tarjetas microSD.

IMAGEN 30. ARDUINO ETHERNET SHIELD

La placa utiliza la cabecera ICSP de seis pines para las señales SPI, lo que la hace compatible con los modelos Arduino Uno, Mega y Due, entre otros.

La ranura de la tarjeta microSD está conectada los pines analógicos A0 y A1 de Arduino con resistencias pull-up, evitando el uso de estas entradas analógicas para otros fines sin modificar la conexión a Ethernet.

4.2.2. Módulos CAN

El bus de transmisión que se utilizará para interconectar los controladores del sistema, es bus CAN. Tal y como se ha comentado en anteriores apartados del proyecto, Arduino Due contiene el controlador CAN en su electrónica, pero el otro tipo de controlador del sistema, Arduino UNO necesita de una electrónica extra.

Además de un controlador CAN, ambos necesitan un CAN Transceiver para efectuar la comunicación mediante el bus de transmisión.

Actualmente, no existe una gran variedad de dispositivos en el mercado, capaces de realizar una transmisión mediante el protocolo CAN. Según el estudio realizado, los conectores más comunes para dispositivos Arduino son RS-232 y par trenzado.

A continuación, se detallan algunas de las características de los módulos que se han utilizado para realizar la transmisión por bus CAN en el sistema domótico diseñado. Si se precisan más detalles, se han añadido los datasheets de los módulos en los anexos.

4.2.2.1. SN65HVD230 CAN Transceiver

En redes de computadoras y telecomunicación, un transceiver o transceptor es un dispositivo que se encarga de realizar funciones de recepción de una comunicación, contando con un circuito eléctrico que permite un procesamiento para también realizar la transmisión de esta información, sin importar su diseño o formato. Lo que significa que pueden enviarse señales entre dos terminales en ambos sentidos, pero no simultáneamente.

Como se ha comentado anteriormente, Arduino Due ya dispone de un controlador CAN, por lo que tan solo necesitaría un transceptor para lograr enviar y recibir información con el protocolo estipulado.

El transceiver SN65HVD230 es compatible con las especificaciones de la capa física estándar de la norma ISO 11898-2 de alta velocidad CAN. Estos dispositivos están diseñados para velocidades de datos de hasta 1 Mbps (Megabit por segundo), e incluyen algunas características de protección que proporcionan robustez de la red del dispositivo y bus CAN.

La característica principal de este transceptor es que opera a una tensión de 3.3V, adecuada para la tensión de funcionamiento de Arduino Due.

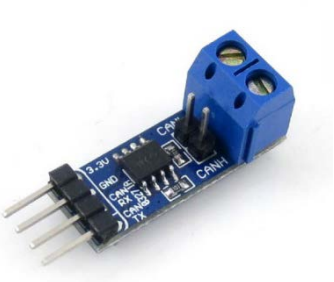


IMAGEN 31. SN65HVD230 CAN TRANSCEIVER

La ventaja o el hándicap, según se mire, es que incluye una resistencia terminal de 120Ω conectada al bus, como tramo inicial o final el bus CAN.

Además, incluye los pines de alimentación (VCC y GND), los pines de conexión con Arduino (CAN Rx y CAN Tx) y los terminales CAN-H y CAN-L.

4.2.2.2. SPI - CAN Bus Module

El otro módulo utilizado para la realización de la transmisión mediante bus CAN es un módulo que tiene integrado un chip MCP2515 con las funciones de controlador CAN, y el chip TJA1050, que realiza las funciones de transceiver.

El MCP2515 es un circuito integrado fabricado por Microchip el cual realiza las funciones de controlador. Implementa CAN V2.0B y CAN V2.0A con una velocidad de hasta 1Mbps.

El circuito integrado TJA1050, el cual podría ser sustituido por un chip MCP2551, funciona como transceptor, es decir, hace de interfaz entre el controlador de CAN bus y el bus físico.

Además, éste módulo contiene SPI para la comunicación entre el microcontrolador de la placa Arduino y el módulo CAN. En concreto Arduino UNO, utiliza los pines 13, 12, 11 y 10, que corresponden con los terminales SCK, MISO, MOSI y CS respectivamente, incluyendo el pin INT, para generar una interrupción, que generalmente suele ser el pin 2.

IMAGEN 32. SPI - CAN BUS MODULE

También es muy importante, el cristal de cuarzo de 8MHz del módulo, que permite el correcto funcionamiento de la tarjeta y establece la unidad mínima de tiempo para la sincronización de los bits durante la comunicación.

Finalmente, el módulo también dispone de una resistencia terminal de 120Ω, característica de bus CAN, la cual se puede conectar al bus mediante un jumper, para indicar inicio o final de línea.

4.2.3. Sensores

Los sensores son dispositivos formados por células sensibles que detectan variaciones en una magnitud física y las convierten en señales útiles para un sistema de medida o control. Son los elementos físicos que transmiten una señal al sistema cuando hay una variación de algún parámetro.

En una casa domótica se necesitan sensores para poder medir las magnitudes físicas del entorno, tales como la temperatura, la humedad, la luminosidad, entre otras, y poder actuar en función del valor medido.

IMAGEN 33. SENSORIZACIÓN DEL HOGAR

Existen muchos tipos de sensores, pero en éste proyecto se han utilizado un sensor de temperatura, un sensor de luz y un sensor de infrarrojos. Se podrían haber utilizado otro tipo de sensores como un sensor de lluvia, sensor de humos o llamas o caudalímetros para medir el caudal y el consumo de agua.

Éstos otros sensores también serían típicos en un sistema totalmente domotizado. Se desestimó esta opción por el simple hecho de que la simulación de un entorno con esas características, sería difícil de representar en el mismo entorno de la presentación del proyecto.

Aunque, una vez determinado el sistema de transmisión y siguiendo el modelo de los otros sensores, no sería laboriosa la implementación de cualquiera de los sensores no utilizados. El sistema domótico diseñado es tan extensible como desee el cliente.

4.2.3.1. Sensor de temperatura

Un sensor de temperatura, como su nombre indica, mide la temperatura de una zona concreta. Es necesario para realizar la automatización del sistema de climatización de una casa.

En éste proyecto se ha utilizado un sensor de temperatura DHT11, que también proporciona la humedad en el ambiente.

Éste tipo de sensores de temperatura, dispone de un procesador interno que realiza el proceso de medición, proporcionando la medición mediante una señal digital, por lo que resulta muy sencillo obtener la medición desde un microprocesador como Arduino.

Las características del DHT11 son realmente escasas, especialmente en rango de medición y precisión, aunque más que suficientes para éste proyecto:

- Medición de temperatura entre 0 a 50ºC
- Precisión de temperatura de 2ºC.
- Medición de humedad entre 20 a 80%.
- Precisión de humedad del 5%.
- Una muestra por segundo.
- Voltaje de funcionamiento: 3,3 a 5 V.

IMAGEN 34. DHT11 SHIELD

Se ha utilizado una shield para Arduino que facilitaba el conexionado del sensor al sistema.

4.2.3.2. Sensor de luz

En éste proyecto se ha utilizado un fotoresistor como sensor de luz. Un fotoresistor, o LDR (light-dependent resistor) es un dispositivo cuya resistencia varía en función de la luz recibida. Podemos usar esta variación para medir, a través de las entradas analógicas, una estimación del nivel de la luz.

Un fotoresistor está formado por un semiconductor, típicamente sulfuro de cadmio. Al incidir la luz sobre él algunos de los fotones son absorbidos, provocando que electrones pasen a la banda de conducción y, por tanto, disminuyendo la resistencia del componente.

Por tanto, un fotoresistor disminuye su resistencia a medida que aumenta la luz sobre él. Los valores típicos son de 1 MΩ en total oscuridad, a 50-100 Ω bajo luz brillante.

Como en el sensor de temperatura, comentado anteriormente, también se ha optado por utilizar una shield para Arduino, que facilite su instalación en el sistema. Algunas de sus características son:

- Potenciómetro para ajustar el umbral de brillo para la detección de luz
- Basado en el chip comparador LM393
- Dimensiones placa: 3.2cm x 1.4cm
- Pines:
    - VCC - 3.3V a 5V
    - GND - 0V
    - DO - Salida de señal digital
    - AO - Salida de señal analógica

IMAGEN 35. LDR SHIELD

4.2.3.3. Sensor de presencia

También se ha utilizado un sensor de presencia, para como su propio nombre indica detectar el paso por cierta zona de la vivienda. Juntamente con el sensor de luz, serán utilizados para automatizar el alumbrado del hogar.

La primera opción fue utilizar un sensor PIR o de movimiento, pero se descartó debido a que el modelo utilizado no era muy preciso.

Como segunda opción, se tomó la iniciativa de utilizar un sensor de infrarrojos, y el resultado fue satisfactorio. Un sensor de infrarrojos es un dispositivo que detecta la presencia de un objeto mediante la reflexión que produce en la luz. El uso de luz infrarroja (IR) es simplemente para que esta no sea visible para los humanos.

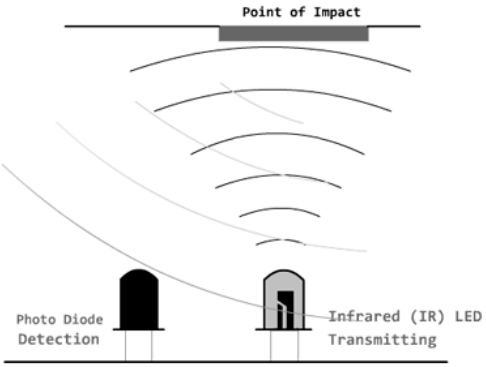


IMAGEN 36. FUNCIONAMIENTO SENSOR INFRARROJOS

Constitutivamente son sensores sencillos. Se dispone de un LED emisor de luz infrarroja y de un fotodiodo que recibe la luz reflejada por un posible obstáculo.

Este tipo de sensores actúan a distancias cortas, típicamente de 5 a 20mm. Además, la cantidad de luz infrarroja recibida depende del color, material, forma y posición del obstáculo, por lo que no disponen de una precisión suficiente para proporcionar una estimación de la distancia al obstáculo.

IMAGEN 37. SENSOR DE INFRARROJOS

Pese a esta limitación son ampliamente utilizados para la detección de obstáculos, y es adecuado para su uso en éste proyecto. Tal y como sucede con los dos sensores anteriormente mencionados, se ha utilizado una shield de Arduino por su fácil implantación en el sistema.

4.2.3.4. Reloj RTC

Finalmente, también se ha utilizado un reloj de tipo RTC, que también se incluye en la categoría de sensores. Un reloj de tiempo real (RTC) es un dispositivo electrónico que permite obtener mediciones de tiempo en las unidades temporales que empleamos de forma cotidiana. Es decir, es capaz de obtener la hora y la fecha, una vez programado.

El término RTC se creó para diferenciar este tipo de relojes de los relojes electrónicos habituales, que simplemente miden el tiempo contabilizando pulsos de una señal, sin existir relación directa con unidades temporales.

Por el contrario, los RTC son más parecidos a los relojes y calendarios que usamos habitualmente, y que funcionan con segundos, minutos, horas, días, semanas, meses y años.

Los RTC normalmente están formados por un resonador de cristal integrado con la electrónica necesaria para contabilizar de forma correcta el paso del tiempo. La electrónica de los RTC tiene en cuenta las peculiaridades de nuestra forma de medir el tiempo, como por ejemplo el sistema sexagesimal, los meses con diferentes días, o los años bisiestos.

Los RTC aportan la ventaja de reducir el consumo de energía, aportar mayor precisión y liberar a Arduino de tener que realizar la contabilización del tiempo. Además, frecuentemente los RTC incorporan algún tipo de batería que permite mantener el valor del tiempo en caso de pérdida de alimentación.

IMAGEN 38. RELOJ RTC DS3231

En éste proyecto, se ha utilizado un modelo DS3231, que incorpora medición y compensación de la temperatura garantizando una precisión de al menos 2ppm, lo que equivale a un desfase máximo 172ms/día o un segundo cada 6 días. En el mundo real normalmente consiguen precisiones superiores, equivalente a desfases de 1-2 segundos al mes.

4.2.4. Actuadores

Los actuadores son los dispositivos utilizados por el sistema de control para modificar el estado de ciertos equipos o instalaciones (el aumento o la disminución de la calefacción o el aire acondicionado, el corte del suministro de gas o agua, el envío de una alarma a una centralita de seguridad, etc.). Estos dispositivos suelen estar distribuidos por toda la vivienda y, según el modelo, pueden admitir baterías. En algunos casos, el sensor y el actuador son integrados en el mismo dispositivo.

Entre los más comúnmente utilizados están: los contactores (o relés de actuación) de carril DIN, los contactores para base de enchufe, las electroválvulas de corte de suministro (gas y agua), las válvulas para la zonificación de la calefacción por agua caliente, y sirenas o elementos zumbadores para el aviso de alarmas en curso.

Tal y como pasa con los sensores, en éste proyecto se simula una casa domótica, por lo que los actuadores son elementos de un tamaño minimizado. En lugar de luces se han utilizado diodos LEDs de tipo DIP, las persianas han sido reemplazadas por motores paso a paso o el sistema de climatización por un ventilador de pequeñas dimensiones.

También se podría extender el número y los tipos de actuadores controlados por el sistema, pero el entorno y el dimensionado de la maqueta limita las posibilidades.

4.2.4.1. LEDS

Un LED es un diodo emisor de luz, es decir, un tipo particular de diodo que emite luz al ser atravesado por una corriente eléctrica.

Un diodo es una unión de dos materiales semiconductores con dopados distintos. Sin entrar en detalles, esta diferencia de dopado hace que genere una barrera de potencial, que como primera consecuencia hace que el paso de corriente en uno de los sentidos no sea posible.

IMAGEN 39. LED

Los diodos tienen polaridad, es decir, solo dejan pasar la corriente en un sentido. Por tanto, se tiene que conectar correctamente la tensión al dispositivo. La patilla larga debe ser conectada al voltaje positivo (ánodo), y la corta al voltaje negativo (cátodo).

En éste proyecto se han utilizado los diodos LEDs como alumbrado de la vivienda. La caída de tensión que provoca un LED depende de su color en gran medida, además, según la tensión a la que se alimente, se deberá conectar una resistencia para un correcto funcionamiento.

Color	Vdd	Resistencia (Ohmios)			
		3.3V	5V	9V	12V
Infrarrojo	1.4V	150	270	510	680
Rojo	1.8V	100	220	470	680
Naranja	2.1V	100	200	470	680
Amarillo	2.2V	100	200	470	680
Verde	3.2V	10	150	330	560
Azul	3.5V	–	100	330	560
Violeta	3.6V	–	100	330	560
Blanco	3.8V	–	100	330	560

TABLA 7. VALORES DE CAÍDA DE TENSIÓN PARA LEDS

4.2.4.2. Motores paso a paso

El motor paso a paso es un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares discretos, lo que significa que es capaz de girar una cantidad de grados (paso o medio paso) dependiendo de sus entradas de control.

En éste proyecto han sido utilizados para la simulación de las persianas del hogar. Se ha utilizado el modelo 28BYJ-48 y el driver controlador basado en el integrado ULN2003.

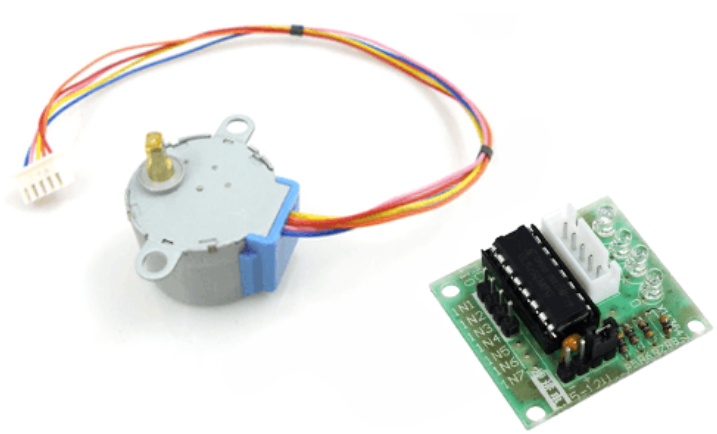


IMAGEN 40. MOTOR PASO A PASO 28BYJ-48 CON CONTROLADORA ULN2003

El 28BYJ-48 es un pequeño motor paso a paso bipolar de bajo precio. Las características eléctricas del 28BYJ-48 son modestas, pero incorpora un reductor integrado que lo convierte en un componente mucho más útil e interesante.

El 28BYJ-48 tiene un paso de 5.625 grados (64 pasos por vuelta). El reductor interno tiene una relación de 1/64. Combinados, la precisión total es de 4096 pasos por vuelta, equivalente a un paso de 0.088º, que es una precisión muy elevada. En realidad, la relación del reductor no es exactamente 1/64 por lo que el número de pasos es 4076 por vuelta (equivalente a un reductor de 1/63.6875)

La alimentación del motor es de 5V o 12V según modelo. El 28BYJ-48 tiene un par máximo tras el reductor de 3N·cm. La frecuencia máxima es de 100Hz, lo que supone unos 40 segundos por vuelta, o equivalentemente una velocidad de giro máxima en torno a 1.5 rpm.

4.2.4.3. Ventilador

Un ventilador es una máquina hidráulica que incrementa la energía cinética de un gas, normalmente aire. Dispone de un rotor con alabes (o aspas) accionado, habitualmente, por un motor eléctrico. Al rotar, los alabes impulsan las moléculas de aire aumentando su energía cinética sin apenas variación de volumen.

Industrialmente los ventiladores se emplean en todo tipo de aplicaciones, como control de temperatura, instalaciones de climatización, homogenización de gases, evacuación de humos, refrescamiento de maquinaria, o disipación de calor en sistemas de refrigeración.

IMAGEN 41. VENTILADOR DE 5V

En éste proyecto, el ventilador simula un sistema de climatización HVAC (Heating, ventilation and air conditioning), que sería manejado por el controlador, y dependiente la temperatura del entorno.

Pueden tener alimentaciones muy diferentes según el modelo. En éste caso, el ventilador tiene una alimentación de 5V.

4.2.4.4. Módulo de relés

Para actuar los LEDs y hacer más real el proceso de iluminación del proyecto, se ha utilizado un módulo de relés de 4 entradas. Un relé es un dispositivo electromecánico que permite a un procesador como Arduino controlar cargas a un nivel tensión o intensidad muy superior a las que su electrónica puede soportar.

Por ejemplo, con una salida por relé podemos encender o apagar cargas de corriente alterna a 220V e intensidades de 10A, lo cual cubre la mayoría de dispositivos domésticos que conectamos en casa a la red eléctrica.

Las salidas por relé son muy frecuentes en el campo de la automatización de procesos, y casi todos los autómatas incluyen salidas por relé para accionar cargas como motores, bombas, climatizadores, iluminación, o cualquier otro tipo de instalación o maquinaria.

IMAGEN 42. MÓDULO DE 4 RELÉS

Físicamente un relé se comporta como un interruptor "convencional" pero que, en lugar de accionarse manualmente, es activado de forma electrónica. Los relés son aptos para accionar cargas tanto de corriente alterna como continua.

Un relé dispone de dos circuitos. El circuito primario se conecta con la electrónica de baja tensión, en nuestro caso Arduino, y recibe la señal de encendido y apagado. Y el circuito secundario es el interruptor encargado de encender o apagar la carga.

La utilización de un módulo de relés da veracidad al proyecto, ya que se podría sustituir perfectamente el LED utilizado en el esquema, por una bombilla, siempre y cuando se cambie la alimentación de los 5V que se le aplican al LED por 220V que proporcionaría la red eléctrica.

Aunque los relés permitirían el control de casi cualquier dispositivo eléctrico o electrónico que se encuentre en el hogar.

4.2.5. Otros

Además de los elementos principales del sistema, también se han utilizado otros componentes electrónicos adicionales que permiten la realización del proyecto.

4.2.5.1. Resistencias

Todo elemento que, conectado en un circuito, presenta una oposición constante a la corriente, independientemente de la tensión aplicada entre sus extremos, es un resistor. Existen diferentes tipos de resistores, según el uso que se les dé, la potencia que sean capaces de consumir, etc.

IMAGEN 43. RESISTENCIA

Los resistores utilizados en electrónica, los de baja potencia, están identificados según un código de colores, en el que cada color va a tener un significado según como se ubique en el resistor.

COLOR	1ª | 2ª	3ª | 4ª	FACTOR	TOLERANCIA
NEGRO	0	0	1Ω	
CAFÉ	1	1	10Ω	±1%
ROJO	2	2	100Ω	±2%
NARANJA	3	3	1KΩ	
AMARILLO	4	4	10KΩ	
VERDE	5	5	100KΩ	±0.5%
AZUL	6	6	1MΩ	±0.25%
VIOLETA	7	7	10MΩ	±0.10%
GRIS	8	8		±0.05%
BLANCO	9	9		
ORO			0.1Ω	±5%
PLATA			0.01Ω	±10%

TABLA 8. VALORES DE RESISTENCIA POR COLOR

Los dos primeros colores pintados en el resistor expresan las cifras más significativas del valor de la resistencia; el tercer color es el exponente de una potencia de 10, que será multiplicada por el valor expresado por los dos primeros colores.

4.2.5.2. Transistor

Los transistores son dispositivos electrónicos que, de forma muy resumida, sirven para dos funciones distintas, amplificar una señal eléctrica o actuar como interruptor controlado de forma eléctrica.

El modelo utilizado en el proyecto es un BC549B, aunque podría ser también un 2N2222 perfectamente. Su función es la de hacer de interruptor del ventilador del sistema.

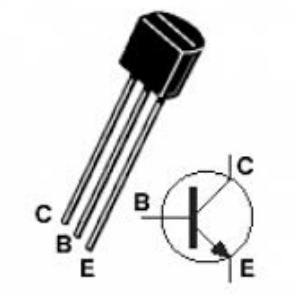


IMAGEN 44. TRANSISTOR

Las tres partes diferenciadas de un transistor son la base, el emisor y el colector. Existe más de una forma de realizar la conexión del transistor, ya que existen dos sub familias de transistores, el tipo PNP y el tipo NPN.

En éste caso el BC549 es un transistor NPN y la corriente entra por el colector y sale por el emisor cuando llega una corriente de entrada por la base. La principal diferencia con el PNP es que la corriente entre el emisor y colector entra por el emisor y sale por el colector.

4.2.5.1. Cableado

Finalmente, para conexionar todos los elementos del sistema se han utilizado cables dupont, con todas las combinaciones posibles (macho-hembra, macho-macho, hembra-hembra), y cables jumper de diferentes dimensiones.

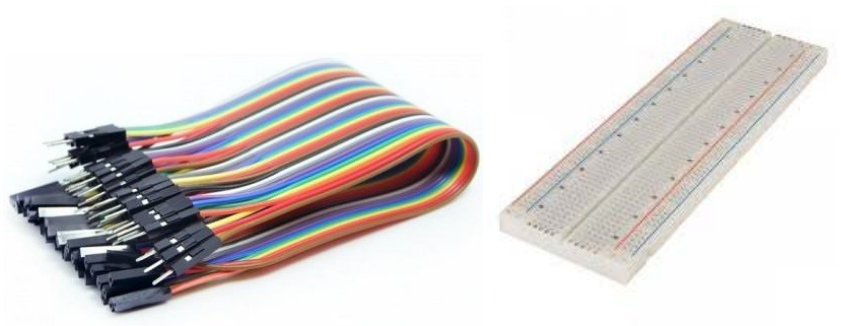


IMAGEN 45. CABLES DUPONT Y PROTOBOARD

Éstos facilitan la conexión al sistema, aunque los componentes no quedan soldados, permiten la reubicación de los componentes con facilidad.

Esto es una particularidad del prototipo, en un sistema real, éste tipo de cableado no añadiría seguridad ni fiabilidad al sistema. También se han utilizado varias placas protoboards de distintos tamaños para la unión del cableado.

5. Desarrollo del proyecto

Una vez explicados todos los elementos que configuran el sistema, se procede a explicar el funcionamiento del conjunto global. En éste capítulo se detalla el diseño del sistema, desde el modelo de arquitectura seleccionada, al tipo de comunicaciones que hacen posible la realización del sistema.

Además, se incluyen los resultados obtenidos según el diseño especificado y algunas de las posibles mejoras del sistema.

5.1. Diseño y especificaciones del sistema

El primer paso en la realización de un proyecto, tras un estudio profundo de las múltiples posibilidades que existen, es el diseño del sistema. En éste apartado se detallan las especificaciones del sistema, las cuales son determinadas según los objetivos mencionados al principio de éste documento.

5.1.1. Arquitectura del sistema

El sistema domótico que se pretende diseñar está dotado de una arquitectura mixta. Tal y como se comenta en el segundo capítulo de éste proyecto, en una arquitectura domótica mixta se combinan los diversos modelos de arquitectura, centralizadas, descentralizadas y distribuidas.

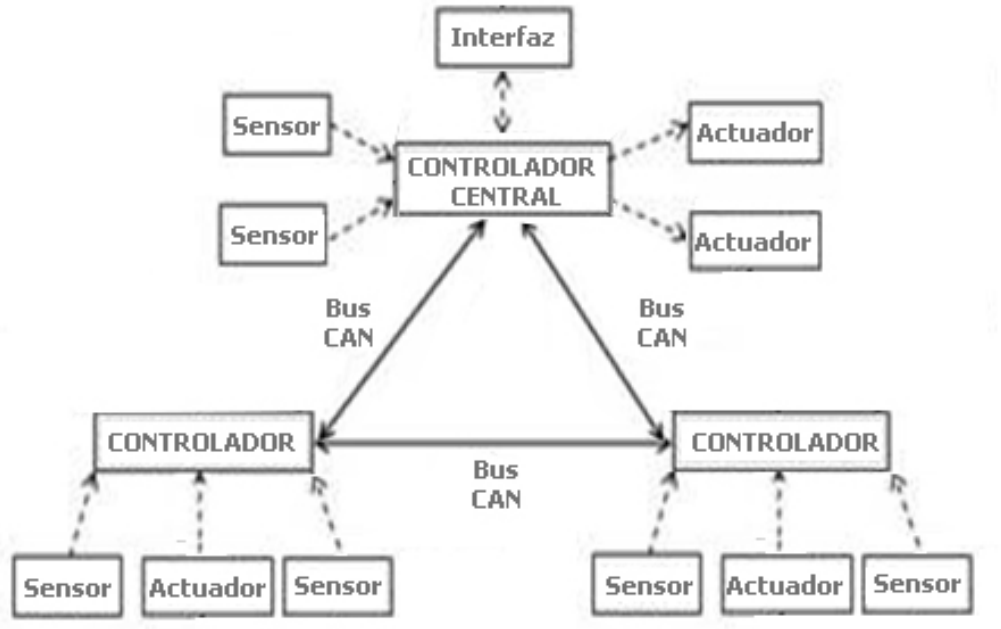


IMAGEN 46. MODELO DE ARQUITECTURA MIXTA DEL SISTEMA DISEÑADO

En éste caso, el sistema dispone de tres controladores, aunque el número de controladores sería fácilmente ampliable.

Uno de los controladores, ejerce de controlador central (Arduino Due), permitirá el acceso al sistema a través de una interfaz virtual.

Ésta interfaz virtual está diseñada a partir de la aplicación NetIO para smartphone, la cual nos permite controlar el sistema remotamente desde un dispositivo móvil.

Además de la interfaz virtual que se consigue gracias a la aplicación, también existe una interfaz física, construida a partir de pulsadores, que permite controlar parte del sistema de manera física sin necesidad de estar conectado a la red local que genera el router, concretamente, tiene acceso al alumbrado de la vivienda.

Mientras, los otros dos controladores (Arduino UNO), responden a las órdenes comandadas por las distintas interfaces, que previamente son enviadas al controlador central.

Cada uno de los controladores tiene a su vez conectados distintos sensores y actuadores que hacen posible la automatización del sistema.

5.1.2. Comunicaciones del sistema

La disposición de cada uno de los elementos del sistema es muy importante, ya que repercutirá en cómo se realizará la comunicación entre los distintos componentes del sistema.

Tal y como se ha establecido en el apartado anterior, el sistema diseñado dispone de tres controladores, que están configurados en un modelo de arquitectura mixta, y uno de ellos ejerce de controlador central.

La comunicación entre los distintos controladores se realiza mediante bus CAN, ya que permite realizar la comunicación entre los controladores con tan solo dos cables, CAN-H y CAN-L, que terminan en los módulos CAN respectivos de cada controlador, que, a su vez están interconectados con los propios controladores.

El controlador central (Arduino Due), además de integrar un controlador CAN, incluye una shield Ethernet V5 de Arduino, que permite conectar el sistema a Internet mediante un router y un cable de red Ethernet que se conecta a uno de los puertos del router.

Éste router genera una red local a la cual se puede conectar un smartphone mediante WiFi, el cual permitirá el acceso al sistema a través de una interfaz virtual.

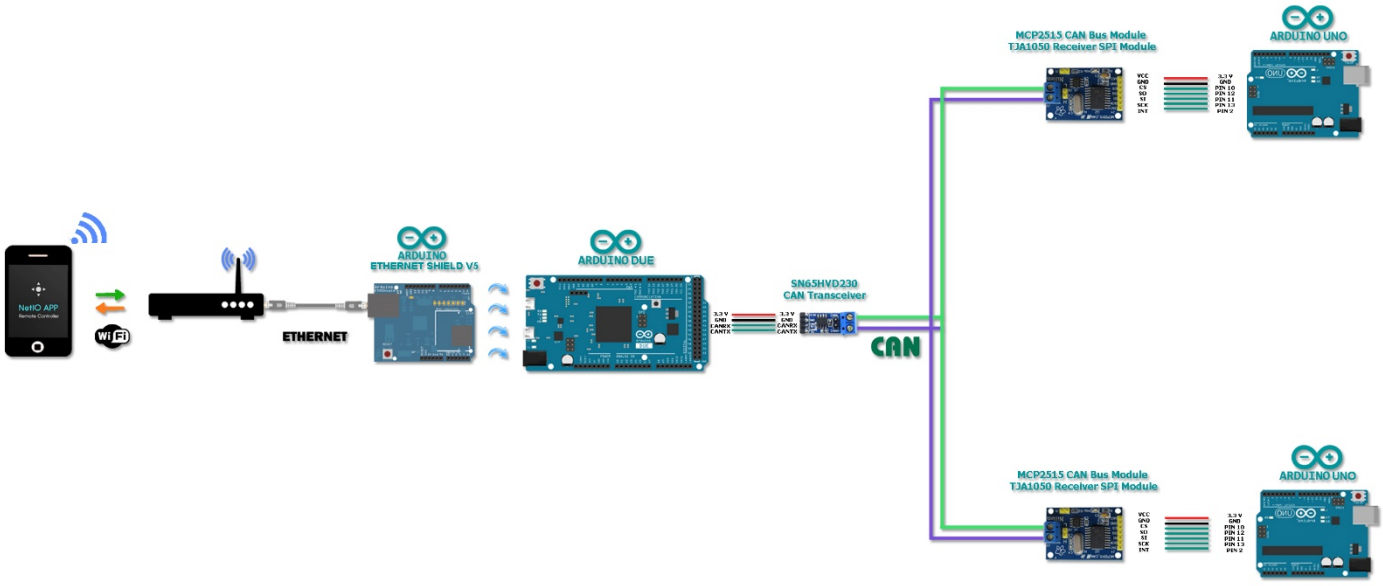


IMAGEN 47. CONEXIÓN GENERAL DEL SISTEMA

Finalmente, las conexiones entre los controladores y sensores o actuadores se realizan mediante cables puente. Éstos cables transmiten señales digitales o analógicas en función del sensor o actuador a controlar, que el controlador interpreta.

5.2. Funcionamiento del sistema

Después de haber diseñado la columna vertebral del proyecto, estableciendo la disposición de cada dispositivo y la relación entre éstos, se procede a detallar cómo se producirá el intercambio de información entre los distintos elementos.

El punto más importante en éste apartado es la comunicación de los controladores mediante bus CAN. Como se ha comentado en el capítulo anterior, se utilizan dos tipos de módulos CAN según el modelo de controlador que se utilice.

El controlador central, Arduino Due, utiliza un SN65HVD230 CAN Transceiver, ya que lleva integrado el controlador de CAN. En éste caso, se utiliza la librería *due_can* ya comentada.

SEÑAL	PIN
VCC	3.3 V
GND	GND
CAN Rx	CAN Rx
CAN Tx	CAN Tx

TABLA 9. CONEXIÓN A BUS CAN DE ARDUINO DUE

Por otro lado, los otros dos controladores, Arduino UNO, no contienen ningún elemento de la capa física de CAN por defecto, así que utiliza el módulo CAN-SPI descrito en el capítulo anterior. Éste módulo utiliza una librería distinta, *CAN_BUS_Shield*, aunque con una estructura similar a la utilizada para el controlador Arduino Due.

SEÑAL	PIN
VCC	5V
GND	GND
CS	10
SO	12
SI	11
SCK	13
INT	2

TABLA 10. CONEXIÓN A BUS CAN DE ARDUINO UNO

La diferencia entre ambos módulos es que el transceiver que utiliza el Arduino Due, envía la información directamente a los pines CAN-Rx y CAN-Tx del controlador, mientras que el módulo CAN que utilizan los Arduino UNO, realizan una conversión mediante el protocolo de comunicación SPI, y utiliza la librería *SPI* adicionalmente. A continuación, se detalla en un esquema simple el resultado de el sistema de transmisión de información mediante bus CAN:

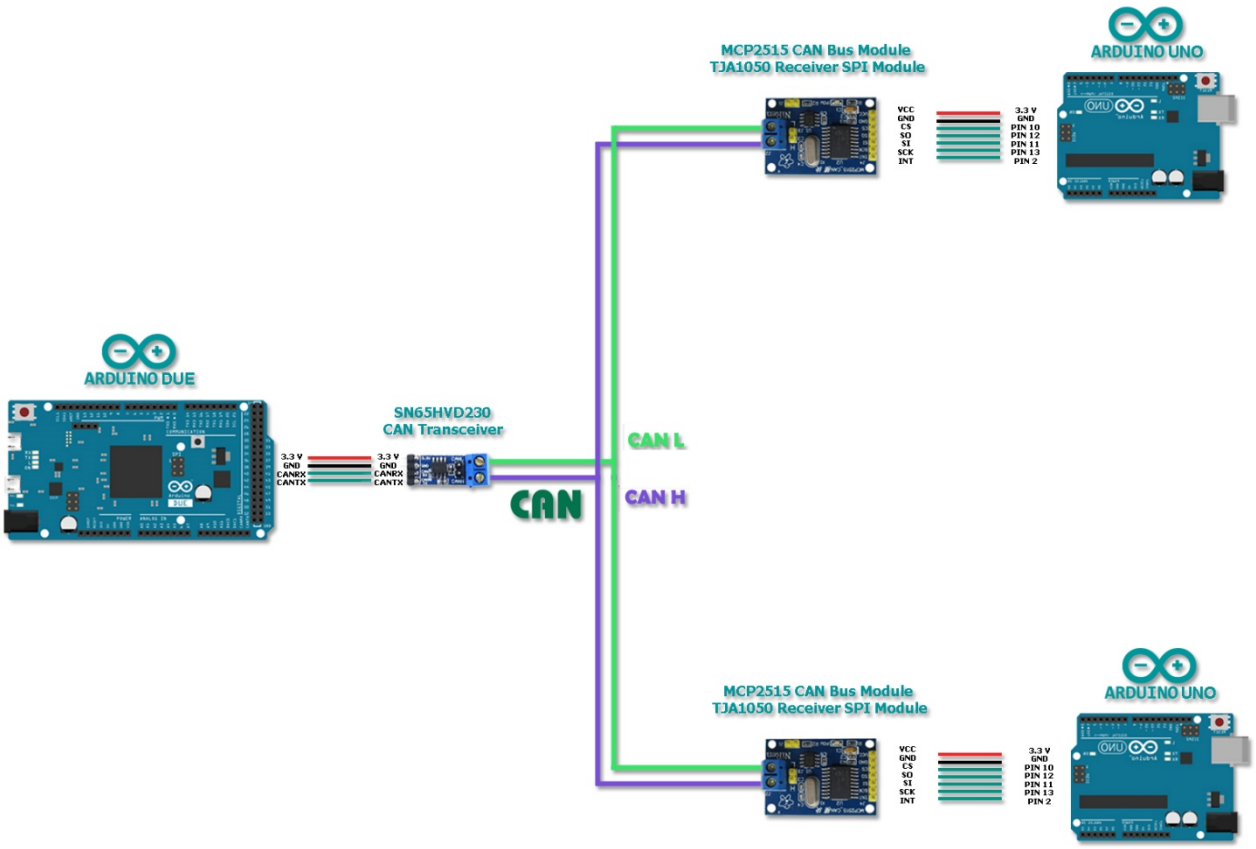


IMAGEN 48. TRANSMISIÓN DE DATOS BUS CAN DEL SISTEMA

Las principales funciones empleadas de ambas librerías son básicamente funciones utilizadas en la transmisión de información. Cabe decir, que Arduino Due tiene dos canales disponibles para la comunicación por bus CAN, y solo se ha utilizado el canal 0. En el caso de la librería *due_can*, se han utilizado las siguientes funciones:

- `Can0.begin();` - Inicializa el canal 0 de CAN bus señalado con la velocidad, en baudrates, determinada entre paréntesis. Ejemplo: Can0.begin(CAN_BPS_125K);

- `Can0.watchFor();` - Es una manera fácil de permitir todo el tráfico de datos por el canal señalado. Ejemplo: Can0.watchFor();

- `Can0.setRXFilter(mailbox, id, mask, extended);` Configura un filtro para las tramas entrantes. Ejemplo: Can0.setRXFilter (0, 0x7E0, 0x7E0, false);

- `Can0.sendFrame(Can0_FRAME&);` - Envía una trama de datos mediante el canal 0 del bus CAN. Las tramas de datos tienen cinco parámetros modificables (.id, .data, .extended, .lenght, .priority).

  A partir de ésta función, se han creado varias funciones con el prefijo *envía* que contienen el rellenado de los parámetros según la necesidad, y el envío de la trama. Ejemplo: Can0.sendFrame(enviaLuz);

Y las funciones creadas a partir de la estructura de la librería, para adaptarlo al proyecto:

- `printFrame(CAN_FRAME &frame);` - Imprime el mensaje enviado por pantalla. El tipo de mensaje recibido es una trama de tipo CAN_FRAME. Existe la variante `printFrameIN`, que se utiliza para imprimir los mensajes recibidos por pantalla.

  Mediante otra parte de código, conseguimos mostrar por la interfaz virtual el dato del mensaje, distinguiendo entre mensajes recibidos y mensajes enviados.

```
void printFrame (CAN_FRAME &frame){
   Serial.print("ID: 0x" ); Serial.print(frame.id, HEX); Serial.print("\t");
   Serial.print ("DATO:"); Serial.print("\t");

 for(int i = 0; i<frame.length; i++){ // imprime los datos
          Serial.print(frame.data.byte[i], HEX);
          Serial.print("  ");
          b[i] = frame.data.byte[i];
 }
 Serial.println();
}
```

IMAGEN 49. FUNCIÓN PRINTFRAME

Finalmente, en cuanto a la transmisión de datos mediante el bus CAN, se han utilizado interrupciones para evitar el colapso de información del controlador central a la hora de leer datos. Para las cuales, se ha utilizado la siguiente función:

- `Can0.attachCANInterrupt(mailbox, función);` - Crea una interrupción en el controlador de bus CAN del controlador central, para leer los datos recibidos en ése mismo instante. Deposita el mensaje en el mailbox señalado, y llama a la función señalada entre paréntesis. Ejemplo: Can0.attachCANInterrupt(1, leePersiana);

Una vez explicada la función de interrupción de CAN para el controlador central, se debe definir el filtrado de mensajes realizados en el proyecto. En total se han realizado dos filtrados de mensajes, ya que solamente se envían mensajes al controlador central desde otros controladores para dos tipos de objetos.

Los dos tipos de objetos que necesitan el filtrado de mensajes son el reloj RTC (con IDs de 0x100 ÷ 0x1FF) y las persianas (con IDs entre 0x700 ÷ 0x7FF). Tal y como se explica en el capítulo 3, en el cuarto apartado *Máscaras y filtros*, a continuación, se muestra el procedimiento seguido para obtener el valor de las máscaras y los filtros.

	BITS											HEXA
ID 1	0	0	1	0	1	0	0	0	1	0	1	0x145
ID 2	0	1	0	0	0	1	1	0	1	1	1	0x237
FILTRO	0	0	1	0	0	0	0	0	0	0	0	0x100
MÁSCARA	1	1	1	0	0	0	0	0	0	0	0	0x700

TABLA 11. PRIMER FILTRO DE CAN DEL SISTEMA

Con la función *Can0.setRXFilter(0, 0x100, 0x700, false);* se enviarían los datos con IDs, en formato estándar, de valores comprendidos entre 0x100 ÷ 0x1FF, al mailbox 0. Debido al filtro 0x100 y la máscara 0x700 aplicadas a los mensajes específicos del clock RTC.

	BITS											HEXA
ID 1	1	1	1	1	0	1	0	1	0	1	0	0x7AA
ID 2	1	0	0	0	1	0	1	1	0	1	1	0x45B
FILTRO	1	1	1	0	0	0	0	0	0	0	0	0x700
MÁSCARA	1	1	1	0	0	0	0	0	0	0	0	0x700

TABLA 12. SEGUNDO FILTRO DE CAN DEL SISTEMA

Con la función *Can0.setRXFilter(1, 0x700, 0x700, false);* se enviarían los datos con IDs, en formato estándar, de valores comprendidos entre 0x700 ÷ 0x7FF, al mailbox 0. Debido al filtro 0x100 y la máscara 0x700 aplicadas a los mensajes específicos de persianas.

En el caso de los otros controladores, Arduino UNO, las funciones de las librerías utilizadas (*CAN_BUS_Shield* y *SPI*), son similares a las anteriores descritas, aunque con algunas variantes y cambios en las nomenclaturas.

- `CAN_OK` y `CAN.begin();` - Inicializa el canal de transmisión de bus CAN a la velocidad seleccionada, y comprueba que el módulo CAN ha iniciado correctamente.

```
while (CAN_OK != CAN.begin(CAN_250KBPS)){
    Serial.println("CAN BUS Shield init fail");
    Serial.println(" Init CAN BUS Shield again");
    delay(100);
}
Serial.println("CAN BUS Shield init ok!");
```

IMAGEN 50. FUNCIÓN CAN.BEGIN

NOTA: Hay que tener muy en cuenta que al llamar la función que regula la velocidad de transmisión de los controladores Arduino UNO, la velocidad debe duplicar velocidad que se le asigna al módulo CAN del controlador Arduino Due.

Eso es debido a que el módulo CAN que utilizamos Arduino UNO, utiliza un cristal de cuarzo de 8MHz cuando la librería ha sido diseñada para un cristal de cuarzo de 16MHz. Por lo que la velocidad de transmisión será la misma en ambos módulos, pero no su codificación.

- `MCP_CAN CAN(SPI_CS_PIN);` - Selección del pin CS al que estará conectado el módulo. Según el modelo del módulo, el pin puede variar entre los pines digitales 9 y 10. Ejemplo: MCP_CAN CAN(10);

- `leeDato();` - La función ha sido creada para éste proyecto, contiene otras funciones exclusivas de la librería *CAN_BUS_Shield*, que se utilizan para poder leer los datos recibidos en el buffer del módulo CAN.

- `CAN_MSGAVAIL` y `CAN.checkReceive();` - Permite saber si se ha recibido un mensaje en el canal del bus.

- `CAN.readMsgBuf();` - Lee el último dato que ha recibido el buffer del módulo receptor. Ejemplo: CAN.readMsgBuf (&len, buf);

- `CAN.getCanId();` - Obtiene la identificación de un mensaje.

```
void leeDato(){

    if(CAN_MSGAVAIL == CAN.checkReceive()){ // check if data coming

      CAN.readMsgBuf(&len, buf); // lee datos, len: longitude datos, buf: data buf

      canId = CAN.getCanId();

      Serial.print("ID: 0x" ); Serial.print(canId, HEX); Serial.print("\t");
      Serial.print ("DATO:"); Serial.print("\t");

      for(int i = 0; i<len; i++){ // imprime los datos
          Serial.print(buf[i], HEX);
          Serial.print("  ");
      }
      Serial.println();
  }
}
```

IMAGEN 51. FUNCIÓN LEEDATO

- `CAN.sendMsgBuf();` - Envía mensajes a través de bus CAN. Tal como pasa en su función equivalente para la librería anterior. El mensaje tiene una estructura y unos parámetros que podemos modificar (id, extended, longitud, dato). Ejemplo: CAN.sendMsgBuf (id, 0, sizeof(datoTEMP), datoTEMP);

Para entender la transmisión de mensajes entre los diferentes controladores, y las funciones que se explicarán a continuación, previamente se debe explicar en profundidad el sistema de transmisión a nivel de byte.

Según el valor del byte y la posición en la que se encuentra, además de su identificador tendrá un significado en la gestión de los mensajes entre los diferentes controladores del sistema.

La siguiente tabla, ayuda a descifrar el significado de cada mensaje según su composición a nivel de byte.

ID	Nº PLACA	TIPO OBJETO	Nº OBJETO	ESTADO	DESCRIPCIÓN	DATOS ADICIONALES				ACCIÓN
11-BIT	BYTE [3]	BYTE [2]	BYTE [1]	[BYTE [0]		BYTE [7]	BYTE [6]	BYTE [5]	BYTE [4]	
0x100 ÷ 0x1FF	0x01	0x06	0x00	0x00	CLOCK RTC	0xAA	0xAA	0xAA	0xAA	Desactiva Alarma
				0x01		Hora	Minuto	Día	Mes	Activa Alarma 1
				0x02		Hora	Minuto	Día	Mes	Activa Alarma 2
0x200 ÷ 0x2FF	0x02	0x05	0x00	0x00	VENTILADOR	0xAA	0xAA	0xAA	0xAA	Encender Ventilador
				0x01		0xAA	0xAA	0xAA	0xAA	Apagar Ventilador
0x300 ÷ 0x3FF	0x01	0x00	0x01	0x00	LUZ	0xAA	0xAA	0xAA	0xAA	Encender Luz 1
				0x01		0xAA	0xAA	0xAA	0xAA	Apagar Luz 1
	0x01		0x02	0x00		0xAA	0xAA	0xAA	0xAA	Encender Luz 2
				0x01		0xAA	0xAA	0xAA	0xAA	Apagar Luz 2
	0x02		0x03	0x00		0xAA	0xAA	0xAA	0xAA	Encender Luz 3
				0x01		0xAA	0xAA	0xAA	0xAA	Apagar Luz 3
	0x02		0x04	0x00		0xAA	0xAA	0xAA	0xAA	Encender Luz 4
				0x01		0xAA	0xAA	0xAA	0xAA	Apagar Luz 4
	0x00		0x05	0x00		0xAA	0xAA	0xAA	0xAA	Encender Luz 5
				0x01		0xAA	0xAA	0xAA	0xAA	Apagar Luz 5
	0x00		0x06	0x00		0xAA	0xAA	0xAA	0xAA	Encender Luz 5
				0x01		0xAA	0xAA	0xAA	0xAA	Apagar Luz 5
0x400 ÷ 0x4FF	0x00	0x02	0x00	0x00	SENSOR DE LUZ	0xAA	0xAA	0xAA	0xAA	No detecta luz
				0x01		0xAA	0xAA	0xAA	0xAA	Detecta Luz
0x500 ÷ 0x5FF	0x00	0x01	0x00	0x00	SENSOR TEMPERATURA	0xAA	0xAA	Temperatura	Humedad	Envía datos
				0x01		0xAA	0xAA	Temperatura	Humedad	Envía datos
0x600 ÷ 0x6FF	0x00	0x03	0x00	0x00	SENSOR INFRARROJOS	0xAA	0xAA	0xAA	0xAA	No detecta movimiento
				0x01		0xAA	0xAA	0xAA	0xAA	Detecta movimiento
0x700 ÷ 0x7FF	0x01	0x04	0x01	0x00	PERSIANA	0xAA	0xAA	Posición	Estado	Parar Persiana 1
				0x01		0xAA	0xAA	0xAA	0xAA	Subir Persiana 1
				0x02		0xAA	0xAA	0xAA	0xAA	Bajar Persiana 1
	0x02		0x02	0x00		0xAA	0xAA	Posición	Estado	Parar Persiana 2
				0x01		0xAA	0xAA	0xAA	0xAA	Subir Persiana 2
				0x02		0xAA	0xAA	0xAA	0xAA	Bajar Persiana 2

TABLA 13. TABLA DE BYTES ESPECÍFICA DEL SISTEMA

Algunos de los parámetros que se pueden obtener del propio dato, además del identificador, son: el número de placa a la que pertenece el mensaje (byte 3), el tipo de objeto del que proviene el mensaje (byte 2), el número de objeto (byte 1) y el estado o acción que se desea transmitir (byte 0).

Un breve resumen de alguno de los datos proporcionados por la tabla ayudará al entendimiento del funcionamiento del sistema.

BYTE [3]	Nº PLACA
0x00	ARDUINO DUE
0x01	ARDUINO UNO - 1
0x02	ARDUINO UNO - 2

TABLA 14. BYTE 3 - Nº DE PLACA

Según el byte 3 del mensaje, que corresponde al número de placa que envía el mensaje, obtenemos que el valor 0x00 corresponde al controlador central, Arduino Due, y el valor 0x01 y 0x02 corresponden a los dos Arduino UNO respectivamente.

También se extrae una segunda clasificación de mensajes, según el tipo de objeto al que se le envía, o recibe la orden.

BYTE [2]	TIPO DE OBJETO
0x00	LUZ
0x01	SENSOR TEMPERATURA
0x02	SENSOR DE LUZ
0x03	SENSOR INFRARROJOS
0x04	PERSIANA
0x05	VENTILADOR
0x06	CLOCK RTC

TABLA 15. BYTE 2 - TIPO DE OBJETO

Además, también se utilizan los bytes 4 y 5 para la transmisión de algunos datos informativos del sistema, como son la temperatura, la humedad, la posición y el estado de las persianas.

Exceptuando el sensor de luz y el sensor infrarrojo, que trabajan en modo local en el controlador central Arduino Due, activando una de las luces conectadas al mismo controlador, todos los tipos de objetos tienen un mensaje específico que informa a todo el sistema de su estado o directamente, envía información de utilidad para el sistema.

Es por eso, que a lo largo del código no se encuentra ningún mensaje con una ID comprendida entre 0x400 ÷ 0x4FF y 0x600 ÷ 0x6FF. Aunque a ambos se les reserva un espacio en la tabla de identificadores, por si se utilizaran para dar información a todo el sistema.

Todos los mensajes enviados por bus CAN son recibidos por todo el sistema, y en el mismo mensaje encontramos algunos parámetros importantes del mensaje que nos permitirán una identificación del mensaje más sencilla.

Los bytes no utilizados se rellenan mediante una trama de 0xAA, que bit a bit corresponde a una trama de 1010 1010, con la que se pretende minimizar los posibles errores en la transmisión.

Cada tipo de objeto tiene una franja de 0xFF identificadores, que puede ser utilizada para ampliar el número de objetos, y por tanto ampliar el número de mensajes diferentes que se enviarían.

Lo que corresponde a 256 mensajes diferentes según el tipo de objeto, que es equivalente a 85 o 128 (según la cantidad de mensajes que utilice) números de objeto por tipo de objeto. Situación que es casi improbable en un hogar.

Si se quisiera ampliar el tipo de objetos, sólo habría que redistribuir las IDs, reduciendo el número de objeto por tipo de objeto. O también, se podría utilizar la versión extended de bus CAN que nos permitiría utilizar IDs con un valor superior a 0x7FF y así poder ampliar los diferentes tipos de mensajes que se podrían enviar.

Una vez detallada toda la información de cómo interpretar los mensajes según su formato, se procede a explicar las funciones creadas específicamente para éste proyecto.

- `enviaLuz();` - Función que envía la orden del controlador central a la placa correspondiente, de encender o apagar la luz seleccionada mediante la interfaz.

- `enviaTemp();` - Función que envía los datos de temperatura y humedad desde el controlador central al resto del sistema.

- `enviaPersiana();` - Función que envía la orden del controlador central a la placa correspondiente, de subir, bajar o parar la persiana seleccionada mediante la interfaz.

- `leePersiana();` - Función que ejerce el controlador central para leer el dato recibido del controlador al que pertenece la persiana a la cual se le había enviado la orden, para obtener los datos de estado y posición de la persiana en cuestión.

- `persaDue();` - Función que envía los datos de estado y posición de la persiana correspondiente al controlador central desde el controlador al que pertenece la persiana la cual se le había enviado la orden previamente.

- `muevePersiana();` - Función que envía el controlador a la persiana en cuestión, transmitiendo la orden de subir, bajar o parar, en función de los datos obtenidos del controlador central.

- `sentidoReloj();` - Función perteneciente a `muevePersiana()`, que permite mover el eje del motor en sentido horario.

- `sentidoAntiReloj();` - Función incluida en `muevePersiana()`, que permite mover el eje del motor en sentido horario.

- `setOutput();` - Función perteneciente a `muevePersiana()`, que transmite la posición al eje del motor y permite el movimiento de la persiana.

- `enviaVent();` - Función que envía la orden del controlador central a la placa correspondiente, de encender o apagar el ventilador seleccionado mediante la interfaz.

- `clockaDue();` - Función que envía información desde el controlador al que pertenece el reloj RTC, al controlador central, sobre si el estados de las diferentes alarmas programadas.

- `leeClock();` - Función que utiliza el controlador central para leer los datos pertenecientes al tipo de objetos clock RTC. Y que, informa sobre el estado de las alarmas programas.

- `Alarma1ON();` y `Alarma2ON();` - Funciones que detectan si se debe activar el número de alarma, respectivamente, según una condición horaria.

Tras haber explicado el funcionamiento de las funciones internas del sistema, para interpretar las ordenes del controlador, mediante los distintos actuadores, en función de la información recibida por los sensores, se pretende explicar el funcionamiento de las distintas interfaces utilizadas, mediante las cuales se ejerce el control del sistema.

Existen dos interfaces en éste proyecto, una interfaz física, muy sencilla, en la que únicamente se controla el alumbrado del hogar mediante dos pulsadores.

Cada pulsador controla el alumbrado general de dos zonas diferenciadas en la maqueta, que se nombrarán Planta 0 y Planta 1.

Lo único que se pretende demostrar con ésta interfaz simple es que sería posible un control físico de cualquier elemento del sistema, además de un control remoto vía smartphone, como se propone en la segunda interfaz diseñada.

La interfaz virtual diseñada, se ha realizado a partir de la aplicación para smartphone NetIO App. Ésta aplicación permite la creación de botoneras sofisticadas, que ejercen como control remoto para cualquier aplicación.

Para ello, se debe conectar el sistema a una red Ethernet que permita conectar el smartphone a la red del sistema, que ejercerá de control remoto.

Esto se consigue mediante la inclusión de un router al sistema, que permita conectar el controlador central Arduino Due, con una shield de Ethernet mediante un cable de red Ethernet, y por otra banda, permita conectar el smartphone vía WiFi.

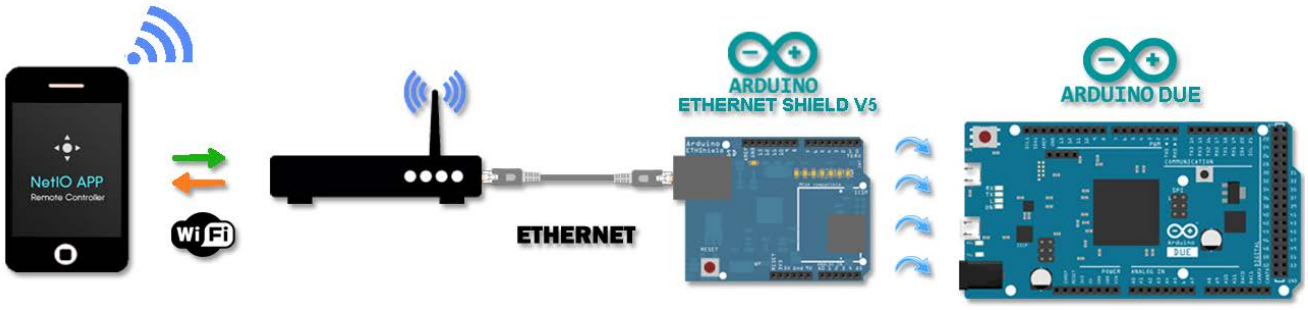


IMAGEN 52. TRANSMISIÓN DE DATOS ETHERNET DEL SISTEMA

También se debe incluir el fragmento de código, que se muestra a continuación, en el software del controlador central, para realizar la conexión a la red de internet, incluyendo las librerías *Ethernet* y *SPI* al encabezado del programa.

```
#include <SPI.h>
#include <Ethernet.h>
```

IMAGEN 53. INCLUSIÓN DE LIBRERÍAS ETHERNET Y SPI

Declarando las variables que se utilizarán para la conexión a Ethernet por parte del controlador central. Forzando el valor de la MAC y los valores correspondientes a la IP que se le asigna al dispositivo. Y determinando el tamaño del buffer y la variable donde se realizará la lectura de los datos.

```
//VARIABLES ETHERNET
byte mac[] = { 0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5 };
// change network settings to yours
IPAddress ip( 192, 168, 1, 6 );
IPAddress gateway( 192, 168, 1, 1 );
IPAddress subnet( 255, 255, 255, 0 );
EthernetClient client;
#define BUFFER 30
char comando[BUFFER];
EthernetServer servidorArduino(80); // nombreServidor(puerto)
```

IMAGEN 54. VARIABLES ETHERNET

Inicializando el servidor declarado, en el setup del programa.

```
Ethernet.begin(mac, ip);
servidorArduino.begin();
```

IMAGEN 55. INICIALIZACIÓN DEL SERVIDOR

Finalmente, iniciando la conexión con el servidor y obteniendo el dato mediante la lectura de éste, y la posterior comparación con la orden a seleccionar, mediante la función *strstr* que compara dos Strings, carácter a carácter.

```
void loop() {

  // put your main code here, to run repeatedly:
  int index = 0;

  EthernetClient clienteApp = servidorArduino.available();
  if (clienteApp) {          //Establim connexió i llegim buffer
   if (clienteApp.connected()) {
          while (clienteApp.available()) {
          char caracter = clienteApp.read();
          if (caracter != '\n') {
            comando[index] = caracter;
            index++;
            if (index >= BUFFER) index = BUFFER -1;
            continue;
          }
            comando[index] = '\0';
          }
```

IMAGEN 56. INICIO DE CONEXIÓN AL SERVIDOR

Además, hace falta conectar la aplicación a la IP destinada al controlador central, que habilita la conexión a la interfaz virtual. En éste caso, la IP es 192.168.1.6 y puerto 80, y se añade en la pestaña de conexiones que se encuentra en el editor de la aplicación.

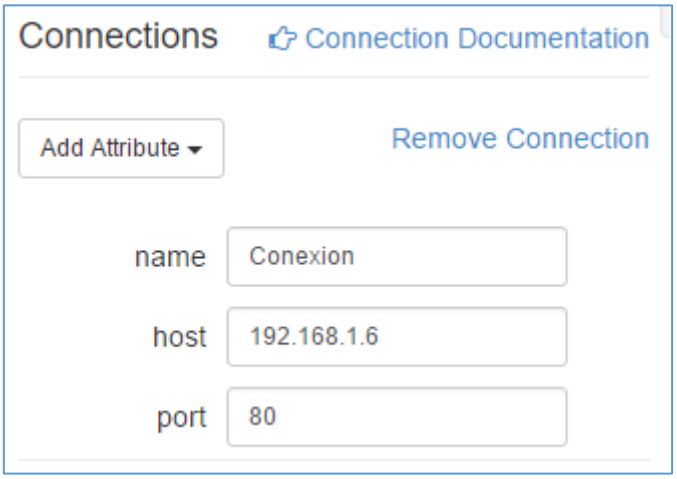


IMAGEN 57. CONEXIÓN EN NETIO APP

La aplicación NetIO App solamente envía un dato de tipo String, o cadena de caracteres, por cada botón, que es interpretado por el sistema como una orden.

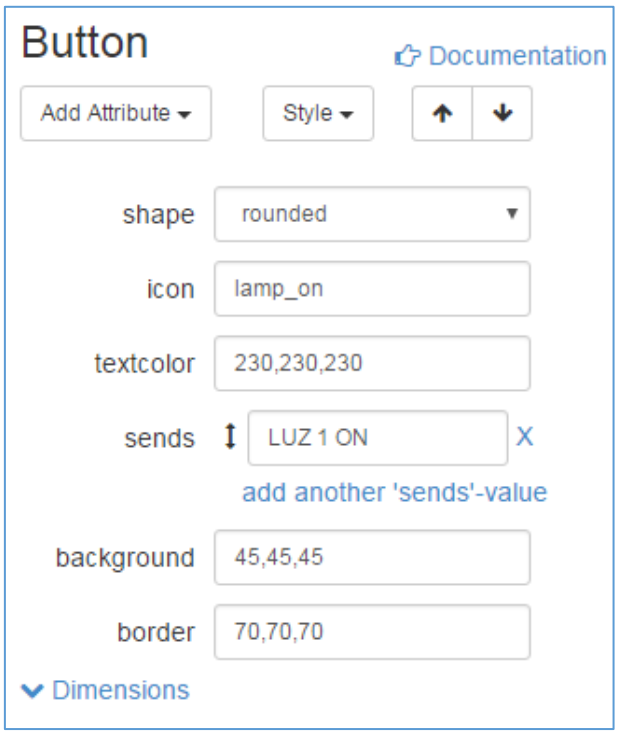


IMAGEN 58. MENÚ BUTTON EN NETIO APP

Los botones tienen muchos parámetros modificables en cuanto al aspecto visual, pero se destaca el parámetro *sends*, mediante el cual se emite el String que se desea enviar cuando se pulse el botón seleccionado.

Además de botones, también permite crear elementos de tipo label o etiqueta, que permite realizar lecturas del sistema mediante la función *reads*.

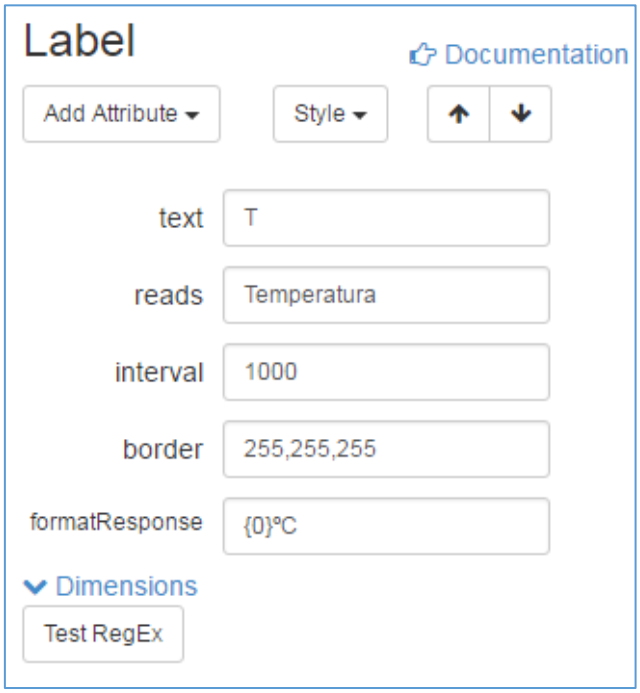


IMAGEN 59. MENÚ LABEL EN NETIO APP

Lo único que diferencia un botón de una etiqueta en el código, además del formato, es que al final de cada botón, el servidor debe enviar un String <<OK>> para su correcto funcionamiento. Como se muestra en el siguiente ejemplo:

```
clienteApp.println("OK");
```

IMAGEN 60. COMANDO OK

Además de la utilización de interfaces para el control de los distintos dispositivos, el sistema también contiene un grado de automatización.

El alumbrado correspondiente a la luz número cinco, depende de el sensor infrarrojos y el sensor de luz. Pretendiendo así, que el LED se ilumine cuando se simule movimiento y se detecte que no haya luz suficiente en el entorno.

También se incluye la automatización del ventilador, el cual se activa automáticamente cuando la temperatura iguala o supera los 25ºC, y se desactiva automáticamente cuando la temperatura es inferior a 25ºC.

Finalmente, el sistema incluye un reloj RTC que pretende automatizar distintos procesos según una condición horaria. Muy útil para poder apagar las luces a cierta hora tardía del día, o subir las persianas en función de la hora para aprovechar la luz solar.

Por lo que se han configurado dos alarmas, la Alarma 1 en horario de mañana que enciende la Luz 3 y sube la Persiana 1, simulando un despertador automático. Y la Alarma 2, en horario de noche, que simula un apagado de los actuadores del sistema, mediante el apagado de la Luz 4 y bajando la Persiana 2.

5.3. Resultados

El resultado del proyecto ha sido plasmado en una maqueta miniatura que simula un hogar de dos plantas. Aunque exceptuando algunos de los sensores y actuadores, que se han adaptado al tamaño de la maqueta, el sistema de control, las interfaces y los buses de transmisión son útiles para un sistema domótico a escala real.

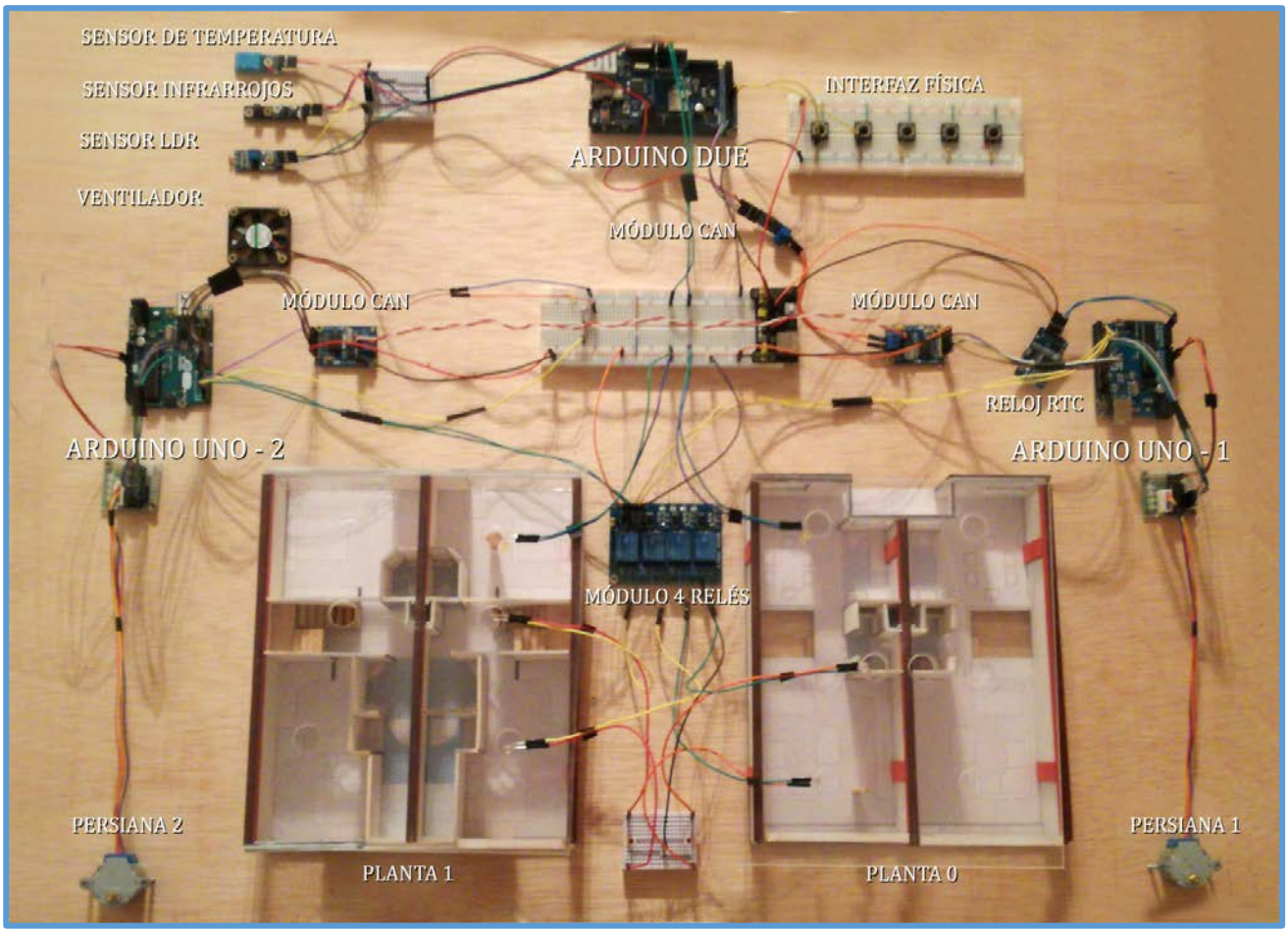


IMAGEN 61. MAQUETA DEL SISTEMA

El conexionado de los distintos elementos del sistema, de forma individual, a sus respectivos controladores, está incluido en los anexos.

Finalmente, tal como se ha ido detallando en los apartados de diseño y especificaciones del sistema, como en el funcionamiento de éste, se ha conseguido realizar un sistema domótico que cumple con los requisitos expuestos en los objetivos del proyecto.

El análisis de las características principales del sistema se divide en tres partes: la parte técnica, un análisis económico y el impacto en la gestión de la energía.

5.3.1. Características del sistema

El sistema domótico realizado sigue todos los parámetros escogidos en el diseño de éste. Es un sistema domótico con un modelo de arquitectura mixta, que utiliza bus CAN como medio de transmisión de información entre el controlador central, Arduino Due, y los otros controladores, Arduino UNO.

El otro medio de transmisión es Ethernet, desde el controlador central se comunica con un smartphone a la aplicación NetIO, mediante un router, que se utiliza como interfaz virtual para controlar el sistema.

Desde la interfaz virtual, se pueden controlar el alumbrado la vivienda, los motores de las persianas, la climatización del hogar y la automatización del conjunto de procesos según una condición horaria o la información que aportan los diferentes sensores.

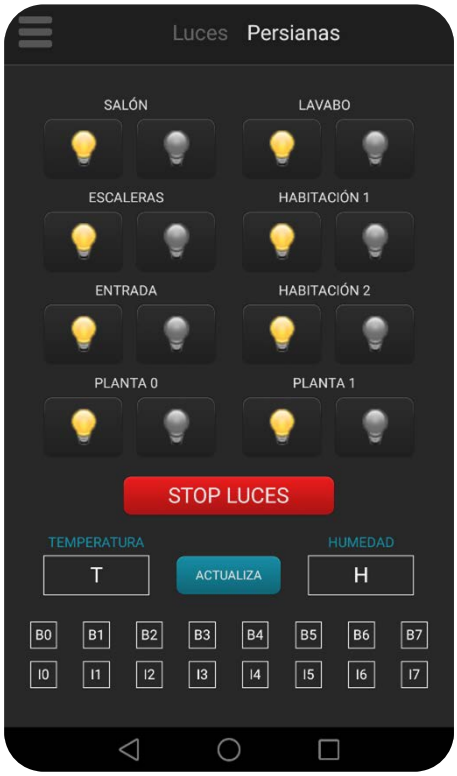


IMAGEN 62. INTERFAZ VIRTUAL - PARTE 1

El controlador central, *Arduino DUE*, además de establecer la comunicación con la interfaz virtual del sistema, también controla parte del alumbrado de la casa. La *Entrada* y la *Habitación 2* son gobernadas y actuadas por el mismo controlador. Además, también recibe información directa de los sensores de temperatura y humedad, el sensor de infrarrojos y el sensor de luz.

El sensor de temperatura, además de obtener los datos de temperatura en grados centígrados, también obtiene la humedad en tanto por ciento. Éstos valores se representan en la interfaz en las casillas de *Temperatura* y *Humedad*.

El botón *Actualiza*, envía los datos de temperatura y humedad, del controlador central, donde está conectado el sensor, al resto del sistema, que podría requerir de éstos valores para activar o desactivar alguno de los elementos del sistema.

El sensor de temperatura influye en la automatización del sistema. Según la temperatura del entorno, el controlador central ejecuta automáticamente la orden de activar o desactivar el *Ventilador*.

El sensor de infrarrojos y el sensor de luz, tal y como pasa en el caso anterior, también influyen en la automatización del sistema. Ya que, si el sensor de infrarrojos detecta un obstáculo, y el sensor de luz detecta poca luz, el controlador central enciende la luz de la *Entrada* de forma automática.

En la interfaz virtual, también podemos observar las casillas *B0-B7* y *I0-I7*. Éstas casillas muestran byte a byte, el último dato enviado y el último dato recibido, respectivamente, del controlador central

El controlador central también incluye el control de una interfaz simple. Ésta interfaz simple, creada a partir de dos pulsadores, controla el alumbrado general de la *Planta 0* y de la *Planta 1* del hogar, respectivamente. El control de estos alumbrados también se encuentra en la interfaz virtual en forma de botones, además de un botón *Stop Luces*, que apaga todas las luces de la vivienda.

ZONA	LUZ	CONTROLADA POR
SALÓN	LED 1	ARDUINO UNO - 1
LAVABO	LED 2	
ESCALERAS	LED 3	ARDUINO UNO - 2
HABITACIÓN 1	LED 4	
ENTRADA	LED 5	ARDUINO DUE
HABITACIÓN 2	LED 6	

TABLA 16. UBICACIÓN DE LEDS

Además del controlador central, los otros controladores, a partir de ahora, *Arduino UNO – 1* y *Arduino UNO – 2*, también controlan parte del alumbrado. El primero, controla la luz del *Salón* y la luz del *Lavabo*, y el segundo, controla la iluminación de las *Escaleras* y la *Habitación 2*.

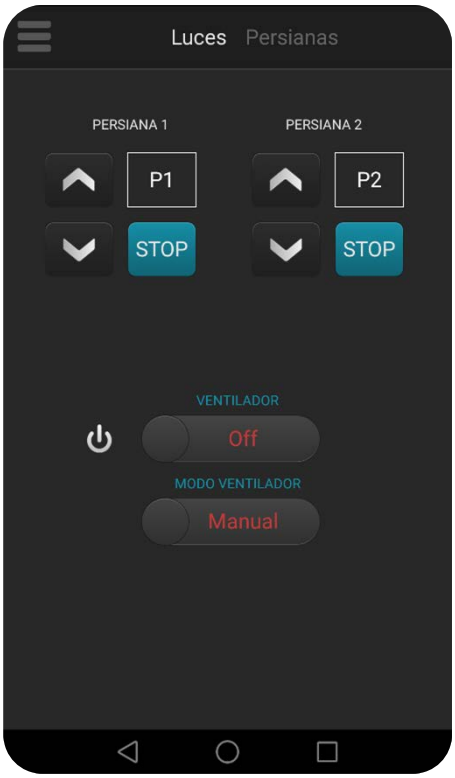


IMAGEN 63. INTERFAZ VIRTUAL - PARTE 2

El *Arduino UNO – 1* también controla la *Persiana 1*, representada con un motor paso a paso en el exterior de la maqueta de la casa. En la interfaz virtual se muestra su posición mediante la casilla *P1*.

El sistema, también incluye un reloj RTC, que controla el mismo controlador. Asimismo, realiza la gestión de las alarmas del sistema, *Alarma 1* y *Alarma 2*, que provocan el encendido o apagado de algunos de los elementos según una condición horaria configurada en el código.

Finalmente, el *Arduino UNO – 2*, además del alumbrado mencionado, también controla la *Persiana 2*. También dirige el encendido del sistema de climatización del conjunto, representado por un *Ventilador*.

En la interfaz virtual, existe un selector para los modos de funcionamiento del *Ventilador*, modo *Automático* o modo *Manual*. Si el modo es *Automático*, el *Ventilador* dependerá de la temperatura ambiente proporcionada por el sensor de temperatura. Por el contrario, si el modo es *Manual*, el *Ventilador* se podrá controlar mediante un switch ON|OFF incluido en la interfaz virtual.

5.3.2. Análisis económico

En cuanto al aspecto económico, hay que diferenciar entre tres posibles costes del proyecto. Del proyecto realizado, se podría extraer el sistema de control domótico, el bus de transmisión y la interfaz virtual, y serviría para cualquier proyecto a gran escala real.

Sin embargo, los sensores y actuadores utilizados, simulan un entorno, y deberían ser substituidos por sensores y actuadores a tamaño real.

En otro de los apartados, también se incluye el presupuesto de la suma de un modulo controlador adicional al sistema, que incluiría el controlador y el respectivo módulo transmisor CAN.

Finalmente, se determina el coste total de la maqueta, que simula el funcionamiento de un sistema domótico real. Aunque hay algunos aspectos a tener en cuenta en los tres presupuestos.

Se incluye el precio de la aplicación NetIO App, que una vez descargada ya no tiene ningún coste adicional. Se puede usar para otros proyectos simultáneamente. Se excluye el precio de un smartphone, ya que, en la actualidad, en la mayoría de hogares hay al menos una persona poseedora de un smartphone.

Las longitudes de los cables son estimadas para el sistema diseñado, según las longitudes utilizadas puede haber variaciones en el precio.

En la mayoría de casos, los componentes han sido proporcionados por proveedores españoles por cuestiones de tiempo, en otros países se pueden encontrar los mismos componentes con un precio menor, y podría provocar pequeñas variaciones en el presupuesto final.

También se podría disminuir el precio del sistema, si en los sensores utilizados no se hubieran utilizado los modelos tipo shield, que facilitan la inclusión a la maqueta.

5.3.2.1. Presupuesto del sistema domótico

En éste apartado, se entiende por sistema domótico el conjunto de controladores, la interfaz virtual diseñada, el bus de transmisión CAN y todos los elementos que permiten la comunicación del sistema, tales como los transceiver CAN o el router para la conexión a Internet. Excluyendo así, los actuadores y sensores del presupuesto global del sistema diseñado.

PRODUCTO	CANTIDAD	PRECIO UNIDAD	PRECIO TOTAL
Arduino Due (Compatible)	1	8,76 €	8,76 €
SN65HVD230 CAN Transceiver	1	4,47 €	4,47 €
Arduino UNO (Compatible)	2	5,80 €	11,60 €
SPI - CAN Bus Module	2	3,38 €	6,76 €
Arduino Ethernet Shield	1	7,54 €	7,54 €
Router Sagem F@st 2064	1	10,00 €	10,00 €
Cable Dupont Par Trenzado 40 cm	2	0,75 €	1,50 €
Cable Ethernet 1m 3ft CAT5E	1	1,10 €	1,10 €
NetIO App	1	7,54 €	7,54 €
TOTAL			59,27 €

TABLA 17. PRESUPUESTO CONTROL DEL SISTEMA E INTERFAZ VIRTUAL

Incluir un módulo controlador al sistema, que sea capaz de ser integrado a bus CAN, y extienda la red creada, no supone un gran desembolso.

PRODUCTO	CANTIDAD	PRECIO UNIDAD	PRECIO TOTAL
Arduino UNO (Compatible)	1	5,80 €	5,80 €
SPI - CAN Bus Module	1	3,38 €	3,38 €
TOTAL			9,18 €

TABLA 18. PRESUPUESTO DE INCLUSIÓN MÓDULO CAN ARDUINO UNO

También existe la posibilidad de incluir un controlador Arduino Due, como un controlador adicional, con su respectivo medio de transmisión de CAN. O simplemente, la posibilidad de tener un controlador central de repuesto con la mismo programa ya configurado.

PRODUCTO	CANTIDAD	PRECIO UNIDAD	PRECIO TOTAL
Arduino Due (Compatible)	1	8,76 €	8,76 €
SN65HVD230 CAN Transceiver	1	4,47 €	4,47 €
TOTAL			13,23 €

TABLA 19. PRESUPUESTO DE INCLUSIÓN MODULO CAN ARDUINO DUE

NOTA: Siempre teniendo en cuenta, que, al añadir un nuevo módulo, la resistencia terminal característica de bus CAN de 120Ω debe situarse tanto en el tramo inicial, como el tramo final del bus.

Algo que caracteriza los módulos CAN utilizados es que tienen dos terminales por cada señal (CAN-H y CAN-L), y permite la posibilidad de conectar cada módulo a otros dos módulos en forma de anillo abierto, como se explicaba en el tercer capítulo de éste documento.

5.3.2.2. Presupuesto de la maqueta diseñada

Finalmente, se incluye el presupuesto del coste total de la maqueta y del sistema domótico en general, incluyendo los sensores y actuadores utilizados en ésta simulación de una vivienda.

CATEGORÍA	PRODUCTO	CANTIDAD	PRECIO UNIDAD	PRECIO TOTAL
CONTROL E INTERFAZ VIRTUAL	Arduino Due (Compatible)	1	8,76 €	8,76 €
	SN65HVD230 CAN Transceiver	1	4,47 €	4,47 €
	Arduino UNO (Compatible)	2	5,80 €	11,60 €
	SPI - CAN Bus Module	2	3,38 €	6,76 €
	Arduino Ethernet Shield	1	7,54 €	7,54 €
	Router Sagem F@st 2064	1	10,00 €	10,00 €
	Cable Dupont Par Trenzado 40 cm	2	0,75 €	1,50 €
	Cable Ethernet 1m 3ft CAT5E	1	1,10 €	1,10 €
	NetIO App	1	7,54 €	7,54 €
SENSORES	Sensor de temperatura - DTH11 Shield	1	2,34 €	2,34 €
	Sensor de luz - LDR Shield	1	2,75 €	2,75 €
	Sensor de presencia - IR Shield	1	1,49 €	1,49 €
	Reloj RTC DS3231	1	2,30 €	2,30 €
ACTUADORES	Diodos LED de colores (20 uds)	1	2,00 €	2,00 €
	Motor paso a paso - 28BYJ-48	2	3,70 €	7,40 €
	Ventilador 5V	1	5,15 €	5,15 €
	Módulo de 4 relés	1	4,95 €	4,95 €
OTROS	Cables Dupont 20cm (40 uds)	3	2,85 €	8,55 €
	Cables Jumper Wires (Kit)	1	1,72 €	1,72 €
	Pulsadores (Kit de 5 uds)	1	1,20 €	1,20 €
	Mini Protoboard	2	1,19 €	2,38 €
	Protoboard	2	2,00 €	4,00 €
	Módulo alimentación de Protoboard	1	1,84 €	1,84 €
	Material de la maqueta	1	50,00 €	50,00 €
	Acondicionamiento del material	1	25,00 €	25,00 €
	Otros (resistencias, condensadores, etc.)	1	1,00 €	1,00 €
TOTAL				183,34 €

TABLA 20. PRESUPUESTO GENERAL DE LA MAQUETA DISEÑADA

5.3.3. Impacto energético

Una de los puntos más importantes del sistema domótico diseñado es su impacto en la gestión de la energía de la vivienda, ya que se propuso como objetivo al principio del proyecto.

Como objetivo de evaluar su impacto en el hogar, para comenzar, se debe estimar el gasto del propio sistema, en éste caso, el sistema de control y transmisión CAN. Dado que se supone que el uso del sistema debe ser constante, sin apenas desconexiones a lo largo de un año, se calculará el gasto energético estimado en un año.

MODELO	ALIMENTACIÓN (V)	CONSUMO (mA·h)	DURACIÓN BATERÍA 1200 mA·h
ARDUINO UNO	5	46	26 horas
ARDUINO DUE	3,3	75	16 horas
ARDUINO MEGA	5	93	13 horas
ARDUINO NANO	5	15	80 horas

TABLA 21.CONSUMO DE ARDUINO

Según la tabla anterior, un Arduino UNO, está alimentado a 5V y consume 46mA·h, y un Arduino Due está alimentado a 3,3V y consume 75mA·h. Los módulos CAN están alimentados a partir de los propios Arduinos, así que, para realizar la estimación de gasto energético anual, se engloban en el consumo del propio controlador.

$$\text{GASTO ANUAL DEL SISTEMA} = [(3{,}3\text{V} \cdot 75\text{mAh}) + 2 \cdot (5\text{V} \cdot 46\text{mAh})] \cdot 24\text{h} \cdot 365\text{d}$$

Teniendo en cuenta que sistema actualmente dispone de un Arduino Due y dos Arduino UNO, el gasto anual del sistema domótico diseñado sería equivalente a 6,2kWh al año.

Un hogar medio en España consume unos 4.000 kWh al año. Por lo que el gasto energético del sistema domótico se corresponde a un 0,1% del consumo energético anual de un hogar.

Pero teniendo en cuenta las ventajas de un sistema domótico, tal y como se ha detallado en el segundo capítulo de éste documento, la instalación de un sistema domótico en el hogar es más que aprovechable.

Mientras que el consumo energético anual del sistema domótico es del 0,1% del total del consumo energético del hogar, el mismo sistema provoca ahorros energéticos en torno a un 80% en el consumo energético de iluminación, del 15% al 25% de ahorro en sistemas de climatización, y entre un 5% y un 10% en otros sectores como el agua caliente.

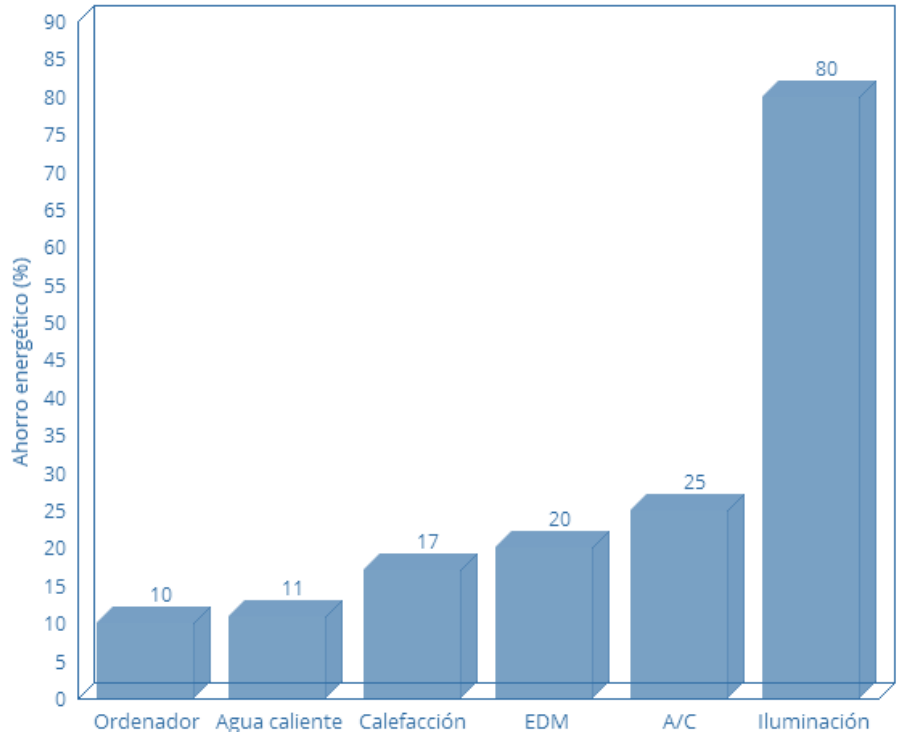


GRÁFICO 3.PORCENTAJE DE AHORRO ENERGÉTICO POR DOMÓTICA

En éste proyecto, se ha centrado especial atención al control de la iluminación, control de persianas y sistema de climatización, además de la automatización y el control de los procesos del hogar. Lo que supone un ahorro energético en los sectores donde puede haber un ahorro energético y económico mayor.

6. Conclusiones

Finalmente, se ha logrado crear un sistema domótico sin ningún conocimiento previo sobre domótica ni los tipos de controladores utilizados. El sistema cumple las expectativas propuestas. Las conclusiones obtenidas, fruto del estudio previo al diseño, del desarrollo del sistema y una póstuma evaluación del proyecto en general son las siguientes:

- El sistema es abierto, distribuir el trabajo realizado bajo una licencia libre, sin duda es la mejor forma de apoyar que el proyecto tenga una continuidad y un posible desarrollo y/o utilización por parte de otros usuarios. En definitiva, generar conocimiento y material reutilizable.

- Cumple el objetivo de bajo coste, la domótica se caracteriza por ser una tecnología cara. Se han obtenido resultados muy llamativos, ya que, el sistema diseñado tiene un precio en torno a 60€, y cada módulo adicional tiene un coste entre 10€ y 15€.

  Éste precio es mucho menor que el de cualquier sistema que se pueda encontrar en el mercado, y accesible para la mayoría de familias, independientemente de su situación económica.

- Es un sistema rentable, ya que produce un ahorro energético y un ahorro económico. Además de ser un sistema respetuoso con el factor medioambiental, la inversión en el sistema es rápidamente recuperada, con la eficiencia energética que produce en el hogar.

- La interfaz virtual diseñada, gobernada por un smartphone, facilita algunas situaciones cotidianas a las personas con discapacidades o movimiento reducido.

- La utilización de bus CAN como medio transmisor es un acierto. Crea una complejidad en el sistema, pero sus mecanismos de detección de errores, reduce en gran cantidad el número de fallos en el sistema.

  Otra de las ventajas, es la creación del bus de transmisión CAN con un presupuesto reducido debido al hardware empleado, y su compatibilidad con el protocolo.

  Además, garantiza una fiabilidad y robustez al sistema, debido a que, la comunicación entre los controladores del sistema descarta cualquier conexión inalámbrica. Por lo tanto, se incrementa también la seguridad.

- Otra de las conclusiones extraídas a raíz de realizar la parte práctica del proyecto es la importancia de las fuentes de alimentación de cualquier sistema.

Durante el transcurso del proyecto, han surgido diferentes problemas en cuanto a las conexiones de los elementos del sistema, que no se habían tenido en cuenta anteriormente.

Por ejemplo, la transmisión de datos del bus CAN era errónea debido a la tensión de alimentación de los módulos CAN. Finalmente, se han podido solucionar algunos de los distintos problemas surgidos.

Una de las soluciones ha sido la utilización de un módulo de alimentación externo, además de la alimentación proporcionada por los propios controladores. O la utilización de una masa común, resistencias de pull-up y pull-down, y filtros RC, para los distintos dispositivos conectados.

- La solución que se propone, es distinta para cada hogar, dependiendo de su configuración y arquitectura. Si bien, es cierto que el sistema se puede implementar en viviendas de obra nueva o ya construidas.

  Esto hace que el sistema sea flexible a cambios y totalmente modular. Ya que, por ejemplo, existen viviendas donde las persianas no están motorizadas, y sería imposible su control.

  La flexibilidad y extensibilidad de la que dispone el sistema domótico diseñado, permite satisfacer las necesidades del usuario y del hogar, en cada caso.

Como conclusión final, se han resuelto satisfactoriamente los distintos objetivos propuestos al inicio del proyecto. Aunque, el sistema tiene aspectos a mejorar todavía, algunas de las ideas para la mejora del sistema son comentadas en el siguiente apartado.

6.1. Futuras mejoras del sistema

Una vez dado por concluido el desarrollo del proyecto y su posterior evaluación, se exponen algunos de los aspectos a mejorar del sistema domótico diseñado.

- El uso de un sistema de monitorización de consumos energéticos, para tomar consciencia del consumo energético del hogar. Esta funcionalidad de la domótica aportará la información necesaria para modificar los hábitos e incrementar el ahorro y la eficiencia.

  Además de crear un sistema inteligente capaz de actuar en función de cada situación, en función del entorno. Esto conlleva al estudio de la zona en el mapa donde se ubicará el sistema, y de sus características, tales como la climatología del lugar.

- Ampliar la instalación domótica . Crear más nodos e incorporarlos a la red para probar nuevas situaciones, con más tráfico de datos y posibilidades de fallo.

  También se podría ampliar utilizando los mismos nodos existentes, añadiendo nuevos sensores y actuadores a cada nodo, hasta que la capacidad física lo permita. Aunque esto supondría una sobrecarga de los controladores innecesaria.

  Esta mejora dispondría de una base ya creada: el protocolo está especificado y las funciones genéricas de los nodos ya están creadas, aunque podría hacer falta la creación de más funciones.

  Además, el propio código del programa está adaptado para añadir nuevos objetos de los tipos de objetos ya creados.

  Simplemente habría que configurar el nuevo nodo, respecto a los sensores y actuadores que tendría conectados.

- Incorporar más servicios a los nodos, es decir, controlar más tipos de objetos. Añadir sensores y actuadores que permitan el control del consumo de agua, o prevenga fugas de gases, que permitan incrementar el ahorro energético

  Otro de los aspectos a introducir al sistema es la creación de un sistema de vigilancia que aporte seguridad al sistema domótico diseñado.

- Comprobar otro tipo de controladores que ejerzan de elemento principal de cada nodo. Las placas Arduino Mega y Arduino Nano, serían algunas de las soluciones propuestas, aunque se debería realizar un estudio en profundidad previamente, en el que se incluiría su compatibilidad con bus CAN y los módulos disponibles en el mercado.

  El Arduino Mega podría realizar la función actual de Arduino Due, de controlador central. Tiene un procesador más potente y un mayor número de pines disponibles. Además se alimenta con 5V, y solucionaría el problema de utilizar dos alimentaciones distintas (3,3V y 5V) para los controladores.

  Hay que decir que Arduino Mega tiene un mayor consumo que un Arduino Due, pero eso se podría compensar de alguna manera utilizando Arduino Nano en el lugar de las placas Arduino Uno utilizadas.

  El Arduino Nano dispone del mismo número de pines digitales y entradas analógicas que un Arduino UNO, además su consumo es tres veces menor.

Finalmente, también se ha valorado la utilización de Raspberry Pi 3 como intermediario del sistema con la interfaz virtual. La Raspberry Pi 3 permite un mayor número de soluciones en torno a la programación y creación de entornos virtuales.

Habría que retomar los estudios iniciales, ya que su compatibilidad con bus CAN es posible, pero no se llegó a conseguir debido a la dificultad de programación, y por eso se descartó como controlador central y se optó por utilizar Arduino Due en su lugar.

- Mejorar la interfaz gráfica. Crear una nueva aplicación adaptable al sistema creado, siguiendo el modelo de la aplicación utilizada. Que permita también monitorizar los consumos energéticos.

  Una posible continuación respecto a este proyecto podría ser la creación de una interfaz gráfica con prestaciones distintivas entre diseñador y usuario, en el propio dispositivo móvil.

- Gestión de la memoria de los controladores. Realizar una gestión óptima de la memoria de programa es vital, ya que se tiene que procurar proporcionar al usuario avanzado, espacio suficiente en la memoria para que pueda elaborar su programa personalizado.

- Salvado de información del sistema. Dado que en muchas ocasiones el flujo de información a supervisar puede llegar a ser elevado para realizarlo en tiempo real, se debería añadir la funcionalidad de poder guardar en un fichero el resultado de todas las peticiones e incidencias ocurridas.

- Determinar un modo optimizado de adaptar la instalación eléctrica del hogar al sistema diseñado. Teniendo en cuenta las alimentaciones de los controladores.

  Además de promover el uso de energías renovables en la instalación eléctrica, para asegurar un hogar totalmente limpio en aspecto energético.

- Implementar un diseño inalámbrico alternativo, para otros clientes que deseen utilizar ésta tecnología para controlar los dispositivos de su vivienda.

- Medir y controlar diferentes dispositivos del hogar, tales como pequeños electrodomésticos, para rentabilizar su uso y consumo.

7. Bibliografía y webgrafía

Durante la realización del proyecto se han utilizado algunas referencias para la obtención de la información, posteriormente procesada y adaptada a la necesidad del proyecto. A continuación, se encuentran la bibliografía y las webs consultadas:

Bibliografía:

- Domótica. Edificios Inteligentes. Huidobro, J.M; Millán, R.J. Creaciones Copyright. ISBN 84-933336-9-7.
- Domótica e Inmótica. Viviendas y edificios inteligentes. Romero,C ; Vázquez, F; De Castro, C. Editorial Ra-Ma. ISBN 84-7897-653-1.
- Domótica y Hogar Digital. Junestrand, S; Passaret, X; Vázquez, D. Thomson Paraninfo ISBN 84-283-2891-9.
- Instalaciones automatizadas en Viviendas y edificios . Molina, L; Ruiz, J. Mc-GrawHill. ISBN 84-481-9946-4.
- El Hogar Digital. Fernández, V; Ruz E. Creaciones Copyright. ISBN 84-96300- 07-2.
- Implementación de aspectos de eficiencia energética en el hogar digital: un caso práctico". Gil Pascual, Miriam. Valencia, 2008.
- Sistemas De Control Para Viviendas Y Edificios Domóticos, José María Quinteiro González, Javier Lamas Graziani, Juan D. Sandoval González.
- Guía técnica de aplicación de instalaciones de sistemas de automatización gestión técnica de energía y seguridad para viviendas y edificios, Ministerio de Industria, Turismo y Comercio. Guía BT-5. Edición Feb. 07. Revisión 1.
- Build Your Own: Smart Home, Robert C. Elsenpeter, Toby J. Velte, ed. McGraw-Hill.

Webs consultadas:

[1] http://www.casadomo.com

[2] http://www.domoticaviva.com

[3] http://www.lonmark.es

[4] http://www.bioingenieria.es

[5] http://www.konnex.org

[6] http://www.knx.org/es

[7] http://www.eiba.org

[8] http://www.ehsa.com

[9] http://www.batibus.com

[10] http://www.lonmark.com

[11] http://www.eiba-es.com

[12] http://www.mundomotica.es

[13] http://www.homesystems.es

[14] http://www.ldingenieria.com/domotica.html

[15] http://www.ni.com/white-paper/2732/es/

[16] http://es.wikipedia.org/wiki/Modelo_OSI#/media/File:Pila-osi-es.svg

[17] http://electronics.stackexchange.com/questions/73303/for-the-can-bus-is-it-ok-to-swap-canh- and-canl-lines

[18] http://www.seeedstudio.com/wiki/CAN-BUS_Shield

[19] https://www.sparkfun.com/products/10039

[20] http://www.microchip.com/wwwproducts/devices.aspx?dDocName=en010406

[21] http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010405

[22] https://www.adafruit.com/products/714

[23] http://img.dxcdn.com/productimages/sku_326450_1.jpg

[24] http://www.seeedstudio.com/wiki/CAN-BUS_Shield ;

[25] https://cdn.sparkfun.com/assets/learn_tutorials/8/3/cableG.jpg

[26] http://store.arduino.cc/product/M000006

[27] https://github.com/Seeed-Studio/CAN_BUS_Shield

[28] http://forum.arduino.cc/index.php?topic=131801.0

[29] http://forum.jeelabs.net/node/1188.html

[30] http://www.can-cia.org/index.php?id=166

[31] http://www.can-cia.org/fileadmin/cia/pdfs/technology/hartwich1.pdf

[32] http://www.sciencedirect.com/science/article/pii/S014193310100148X

[33] http://papers.sae.org/2001-01-0073/

[34] https://github.com/adafruit/Adafruit-RGB-LCD-Shield-Library

[35] http://gfsapivitoria.blogspot.com.es/p/servicios-y-funciones-usuarios_25.html

[36] http://server-die.alc.upv.es/asignaturas/PAEEES/2005-06/A05-A06%20-%20Controlador%20CAN%20-%20Trabajo.pdf

[37] http://bibing.us.es/proyectos/abreproy/11047/fichero/VOLUMEN_I%252F4_CAP2.pdf

[38] http://upcommons.upc.edu/bitstream/handle/2099.1/4691/memoria.pdf

[39] http://onlycarspictures.com/uploads/skoda/skoda-fabia-combi/skoda-fabia-combi-08.jpg

[40] http://en.wikipedia.org/wiki/OBD-II_PIDs

[41] http://upload.wikimedia.org/wikipedia/commons/a/ac/VW_Touran_1.9_TDI_BlueMotion_Technology_Highline_Deep_Black.JPG

[42] http://www.atmel.com/images/doc8161.pdf

[43] http://www.ti.com/lit/an/sloa101a/sloa101a.pdf

[44] http://www.seeedstudio.com/wiki/CAN-BUS_Shield

[45] http://arduino.cc/en/pmwiki.php?n=Hacking/LibraryTutorial

[46] http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/cia99paper.pdf

[47] http://www.volkspage.net/technik/ssp/ssp/SSP_238.pdf

[48] http://www.outilsobdfacile.com/location-plug-connector-obd/Skoda-octavia-3

[49] http://www.can-cia.org/index.php?id=166

[50] http://hackaday.com/2013/10/29/can-hacking-protocols/

[51] http://en.wikipedia.org/wiki/CAN_bus

[52] http://www.kvaser.com/can-protocol-tutorial/

[53] http://www.adafruit.com/products/715

[54] http://www.canbushack.com/blog/index.php

[55] http://www.cedom.es/sobre-domotica/que-es-domotica

[56] http://sdm3.blogspot.com.es/2009/09/domotica.html

[57] https://felixmaocho.wordpress.com/2013/01/19/arduino-como-funciona-u-se-utiliza-un-pulsador/

[58] http://rufianenlared.com/pulsadores-arduino/

[59] https://arduino.stackexchange.com/questions/21324/arduino-is-not-responding-to-a-tiny-rtc-i2c-modules-how-to-fix

[60] http://modelrail.otenko.com/arduino/arduino-controller-area-network-can

[61] http://smallshire.org.uk/sufficientlysmall/2012/01/01/an-atlas-of-arduino-ethernet-shields/

[62] https://soloarduino.blogspot.com.es/2016/04/mb102-alimentacion-para-breadboards.html

Todas las referencias relacionadas con páginas webs (links) están comprobadas a fecha de 19 de Mayo del 2017.

8. Anexos

8.1. Código del programa

Los programas correspondientes a cada placa son los siguientes:

PROGRAMA	PLACA
PFG_DUE.ino	Programa de la placa Arduino Due
PFG_UNO-1.ino	Programa de la placa Arduino UNO - 1
PFG_UNO-2.ino	Programa de la placa Arduino UNO - 2

TABLA 22. PROGRAMA DE CADA PLACA ARDUINO

8.1.1. Arduino Due

```
//ARDUINO DUE

#include <SPI.h>
#include <Ethernet.h>
#include "variant.h"
#include <due_can.h>
#include <Wire.h>
#include "RTClib.h"
#include <DHT.h>

//Leave defined if you use native port, comment if using programming
port
//#define Serial SerialUSB

#define DHTTYPE DHT11   // DHT 11

//VARIABLES FUNCIONES
boolean Luz;
boolean bLuz[8];               //Control del estado de las luces
  //num total luces =>[i]
boolean Temperatura;
boolean sentidoPersiana;
boolean persianaMov;
boolean Ventilador;
boolean autoVent = false;
boolean Alarma1;
boolean Alarma2;
int valorH;
int valorT;
int posPersiana1;
int posPersiana2;

String Temp = "0";
String Hum = "0";
int b[8] = {0, 1, 2, 3, 4, 5, 6, 7};
int bIN[8] = {0, 1, 2, 3, 4, 5, 6, 7};

//VARIABLE RELOJ
RTC_DS3231 rtc; //Reloj RTC modelo DS3231
String daysOfTheWeek[7] = {"Domingo", "Lunes", "Martes", "Miercoles",
"Jueves", "Viernes", "Sabado" };
String monthsNames[12] = {"Enero", "Febrero", "Marzo", "Abril",
"Mayo",  "Junio",
"Julio","Agosto","Septiembre","Octubre","Noviembre","Diciembre" };


//VARIABLES ETHERNET
byte mac[] = { 0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5 };
// change network settings to yours
IPAddress ip( 192, 168, 1, 6 );
IPAddress gateway( 192, 168, 1, 1 );
IPAddress subnet( 255, 255, 255, 0 );
EthernetClient client;
#define BUFFER 30
char comando[BUFFER];
EthernetServer servidorArduino(80); // nombreServidor(puerto)
```

```
    //DECLARACIÓN DE PINES
const int pinLED5 = 4;   //LED5
const int pinLED6 = 5;   //LED6
const int pinDHT = 6;    //Sensor de Temperatura y Humedad
const int pinIR = 7;     //Sensor Infrarrojo
const int pinLDR = 8;    //Sensor de Luz
const int pinPULS1 = 41; //Pulsador1
const int pinPULS2 = 43; //Pulsador2

    //PULSADORES
   int vPULS1 = 0;
   int vPULS2 = 0;
   int valorIF = 0;  //sensor IR
   int valorLDR = 0; //sensor LDR

    //SENSOR TEMPERATURA
  DHT dht(pinDHT, DHTTYPE);
  int h = dht.readHumidity(); // Reading temperature or humidity takes
about 250 milliseconds!
  int t = dht.readTemperature();


void setup() {

  Serial.begin(115200);

  //INICIALIZA CONEXIÓN A ETHERNET
  Ethernet.begin(mac, ip);
  servidorArduino.begin();

  //INICIALIZA CONEXIÓN A BUS CAN
  Can0.begin(CAN_BPS_125K); // Inicializa el canal Can0, a la
velocidad indicada entre paréntesis
  Can0.watchFor();          //Habilita el tráfico por el bus

  //FILTROS PARA MAILBOXES DE BUS CAN
  Can0.setRXFilter(0, 0x100, 0x700, false); //Filtrar al mailbox 0
datos de CLOCK RTC con ID: (0x100 : 0x1FF) - modo estandard: false
  Can0.setRXFilter(1, 0x700, 0x700, false); //Filtrar al mailbox 2
datos de PERSIANAS con ID: (0x700 : 0x7FF) - modo estandard: false

  //INICIALIZA SENSOR DE TEMPERATURA
  dht.begin();

  //INICIALIZA PINES UTILIZADOS DE ARDUINO DUE
  pinMode(pinLED5, OUTPUT);  //LED5
  pinMode(pinLED6, OUTPUT);  //LED6
  pinMode(pinPULS1, INPUT);  //Pulsador 1
  pinMode(pinPULS2, INPUT);  //Pulsador 2
  pinMode(pinIR, INPUT);     //Sensor Infrarrojo

}
```

```
void loop() {

  //CONEXIÓN A SERVIDOR DE ETHERNET
  int index = 0;
  EthernetClient clienteApp = servidorArduino.available();
  if (clienteApp) {          //Establim connexió i llegim buffer
   if (clienteApp.connected()) {
          while (clienteApp.available()) {
          char caracter = clienteApp.read();
          if (caracter != '\n') {
            comando[index] = caracter;
            index++;
            if (index >= BUFFER) index = BUFFER -1;
            continue;
          }
            comando[index] = '\0';
          }

//INTERFAZ FÍSICA CUANDO ESTÁ CONECTADO EL SMARTPHONE
   vPULS1 = digitalRead(pinPULS1); //PULSADOR 1
   vPULS2 = digitalRead(pinPULS2); //PULSADOR 2

   //SENSORES INFRARROJOS Y LUZ
   valorIF = digitalRead(pinIR);  //lectura sensor IR
   valorLDR = digitalRead(pinLDR); //lectura sensor LDR

   //PULSADOR PLANTA 0
   if (vPULS1 == HIGH) {
    Serial.println("Pulsador Planta 0");
    if(bLuz[1]== false){ Luz = true; enviaLuz(1);}
    else if(bLuz[1]== true){ Luz = false; enviaLuz(1);}
    delay(500);
    if(bLuz[2]== false){ Luz = true; enviaLuz(2);}
    else if(bLuz[2]== true){ Luz = false; enviaLuz(2);}
    delay(500);
    if(bLuz[5]== false){ Luz = true; enviaLuz(5);}
    else if(bLuz[5]== true){ Luz = false; enviaLuz(5);}
    delay(500);
   }

   //PULSADOR PLANTA 1
   if (vPULS2 == HIGH) { //PULSADOR PLANTA 1
    Serial.println("Pulsador Planta 1");
    if(bLuz[3]== false){ Luz = true; enviaLuz(3);}
    else if(bLuz[3]== true){ Luz = false; enviaLuz(3);}
    delay(500);
    if(bLuz[4]== false){ Luz = true; enviaLuz(4);}
    else if(bLuz[4]== true){ Luz = false; enviaLuz(4);}
    delay(500);
    if(bLuz[6]== false){ Luz = true; enviaLuz(6);}
    else if(bLuz[6]== true){ Luz = false; enviaLuz(6);}
    delay(500);
   }

  //DETECTOR INFRARROJO - LUZ 5
  if (valorIF == LOW && bLuz[6] == false) {
      Serial.println("Detectado obstaculo");
      Luz = true;
      enviaLuz(5);
      delay(1000);
  }
```

```
  //VENTILADOR AUTOMÁTICO
   h = dht.readHumidity(); // LECTURA DE HUMEDAD
   t = dht.readTemperature(); //LECTURA DE TEMPERATURA


   if (autoVent == true){

   if (t>=25 && Ventilador == false){          // switch Ventilador
turned ON
   Ventilador = true;
   Serial.println("AUTO - Ventilador encendido debido a temperatura >=
25ºC");
   enviaVentilador(1);
   }
   else if (t<23 && Ventilador == true){         // switch Ventilador
turned OFF
   Ventilador = false;
   Serial.println("AUTO - Ventilador apagado debido a temperatura <
25ºC");
   enviaVentilador(1);
   }

   }



//NetIO APP - INTERFAZ VIRTUAL

             //SENSOR TEMPERATURA

                if (strstr(comando, "Temperatura")) { //LABEL
                Temp = String (t); //valorT
                clienteApp.println(Temp);
                }

                if (strstr(comando, "Humedad")) {
                Hum = String (h); //valorH
                clienteApp.println(Hum);
                }

                if (strstr(comando, "Posicion Persiana 1")) {
                String Pos1 = String (posPersiana1);
                clienteApp.println(Pos1);
                }

                if (strstr(comando, "Posicion Persiana 2")) {
                String Pos2 = String (posPersiana2);
                clienteApp.println(Pos2);
                }

                //MENSAJES ENVIADOS - printFrame
                if (strstr(comando, "B0")) {
               String bit0 = String (b[0]); clienteApp.println(bit0); }
                if (strstr(comando, "B1")) {
               String bit1 = String (b[1]); clienteApp.println(bit1); }
                if (strstr(comando, "B2")) {
                String B2 = String (b[2]); clienteApp.println(B2); }
                if (strstr(comando, "B3")) {
                String B3 = String (b[3]); clienteApp.println(B3); }
                if (strstr(comando, "B4")) {
                String B4 = String (b[4]); clienteApp.println(B4); }
```

```
                if (strstr(comando, "B5")) {
                String B5 = String (b[5]); clienteApp.println(B5); }
                if (strstr(comando, "B6")) {
                String B6 = String (b[6]); clienteApp.println(B6); }
                if (strstr(comando, "B7")) {
                String B7 = String (b[7]); clienteApp.println(B7); }

                //MENSAJES RECIBIDOS - printFrameIN
                if (strstr(comando, "I0")) {
                String I0 = String (bIN[0]); clienteApp.println(I0); }
                if (strstr(comando, "I1")) {
                String I1 = String (bIN[1]); clienteApp.println(I1); }
                if (strstr(comando, "I2")) {
                String I2 = String (bIN[2]); clienteApp.println(I2); }
                if (strstr(comando, "I3")) {
                String I3 = String (bIN[3]); clienteApp.println(I3); }
                if (strstr(comando, "I4")) {
                String I4 = String (bIN[4]); clienteApp.println(I4); }
                if (strstr(comando, "I5")) {
                String I5 = String (bIN[5]); clienteApp.println(I5); }
                if (strstr(comando, "I6")) {
                String I6 = String (bIN[6]); clienteApp.println(I6); }
                if (strstr(comando, "I7")) {
                String I7 = String (bIN[7]); clienteApp.println(I7); }


    if (
          !strstr(comando, "Temperatura")
       && !strstr(comando, "Humedad")
       && !strstr(comando, "Dia")
       && !strstr(comando, "Mes")
       && !strstr(comando, "Año")
       && !strstr(comando, "Hora")
       && !strstr(comando, "Minuto")
       && !strstr(comando, "Segundo")
       && !strstr(comando, "Posicion Persiana 1")
       && !strstr(comando, "Posicion Persiana 2")
       && !strstr(comando, "B0")
       && !strstr(comando, "B1")
       && !strstr(comando, "B2")
       && !strstr(comando, "B3")
       && !strstr(comando, "B4")
       && !strstr(comando, "B5")
       && !strstr(comando, "B6")
       && !strstr(comando, "B7")
       && !strstr(comando, "I0")
       && !strstr(comando, "I1")
       && !strstr(comando, "I2")
       && !strstr(comando, "I3")
       && !strstr(comando, "I4")
       && !strstr(comando, "I5")
       && !strstr(comando, "I6")
       && !strstr(comando, "I7")
       ){
//---------------- do below -------------------------
```

```
//ORDENES
     //LUCES
                    if (strstr(comando,"LUZ 1 ON")) {      //LED1 ON
                    Luz = true;
                    enviaLuz(1);
                    }

                    if (strstr(comando,"LUZ 1 OFF")) {    //LED1 OFF
                    Luz = false;
                    enviaLuz(1);
                    }

                    if (strstr(comando,"LUZ 2 ON")) {      //LED2 ON
                    Luz = true;
                    enviaLuz(2);
                    }

                    if (strstr(comando,"LUZ 2 OFF")) {    //LED2 OFF
                    Luz = false;
                    enviaLuz(2);
                    }

                    if (strstr(comando,"LUZ 3 ON")) {      //LED3 ON
                    Luz = true;
                    enviaLuz(3);
                    }

                    if (strstr(comando,"LUZ 3 OFF")) {    //LED3 OFF
                    Luz = false;
                    enviaLuz(3);
                    }

                    if (strstr(comando,"LUZ 4 ON")) {      //LED4 ON
                    Luz = true;
                    enviaLuz(4);
                    }

                    if (strstr(comando,"LUZ 4 OFF")) {    //LED4 OFF
                    Luz = false;
                    enviaLuz(4);
                    }

                    if (strstr(comando,"LUZ 5 ON")) {      //LED5 ON
                    Luz = true;
                    enviaLuz(5);
                    }

                    if (strstr(comando,"LUZ 5 OFF")) {    //LED5 OFF
                    Luz = false;
                    enviaLuz(5);
                    }

                    if (strstr(comando,"LUZ 6 ON")) {      //LED6 ON
                    Luz = true;
                    enviaLuz(6);
                    }

                    if (strstr(comando,"LUZ 6 OFF")) {    //LED6 OFF
                    Luz = false;
                    enviaLuz(6);
                    }
```

```
                if (strstr(comando,"PLANTA 0 ON")) {    //PLANTA 0 ON
                  Luz = true;
                  enviaLuz(1);
                  delay (500);
                  enviaLuz(2);
                  delay (500);
                  enviaLuz(5);
                  delay (500);
                }

                if (strstr(comando,"PLANTA 0 OFF")) {   //PLANTA 0 OFF
                  Luz = false;
                  enviaLuz(1);
                  delay (500);
                  enviaLuz(2);
                  delay (500);
                  enviaLuz(5);
                  delay (500);
                }

                if (strstr(comando,"PLANTA 1 ON")) {     //PLANTA 1 ON
                  Luz = true;
                  enviaLuz(3);
                  delay (500);
                  enviaLuz(4);
                  delay (500);
                  enviaLuz(6);
                  delay (500);
                }

                if (strstr(comando,"PLANTA 1 OFF")) {   //PLANTA 1 OFF
                  Luz = false;
                  enviaLuz(3);
                  delay (500);
                  enviaLuz(4);
                  delay (500);
                  enviaLuz(6);
                  delay (500);
                }

                if (strstr(comando,"STOP LUCES")) {    //LED8 OFF
                for (int i=0; i<8; i++){
                  Luz = false;
                  enviaLuz(i);
                  delay (100);
                }
                }


//PERSIANAS

   //PERSIANA 1

                 //PERSIANA 1 ARRIBA
                if (strstr(comando,"Persiana 1 Arriba")) {
                persianaMov = true;
                sentidoPersiana = true;
                enviaPersiana(1);
                Can0.attachCANInterrupt(1, leePersiana);
                }
```

```
            //PERSIANA 1 ABAJO
            if (strstr(comando,"Persiana 1 Abajo")) {
            persianaMov = true;
            sentidoPersiana = false;
            enviaPersiana(1);
            Can0.attachCANInterrupt(1, leePersiana);
            }

            //PERSIANA 1 STOP
            if (strstr(comando,"Persiana 1 Stop")) {
            persianaMov = false;
            enviaPersiana(1);
            Can0.attachCANInterrupt(1, leePersiana);
            }

  //PERSIANA 2
            //PERSIANA 2 ARRIBA
            if (strstr(comando,"Persiana 2 Arriba")) {
            persianaMov = true;
            sentidoPersiana = true;
            enviaPersiana(2);
            Can0.attachCANInterrupt(1, leePersiana);
            }

           //PERSIANA 2 ABAJO
            if (strstr(comando,"Persiana 2 Abajo")) {
            persianaMov = true;
            sentidoPersiana = false;
            enviaPersiana(2);
            Can0.attachCANInterrupt(1, leePersiana);
            }

            //PERSIANA 2 STOP
            if (strstr(comando,"Persiana 2 Stop")) {
            persianaMov = false;
            enviaPersiana(2);
            Can0.attachCANInterrupt(1, leePersiana);
            }

//SENSOR TEMPERATURA
            if (strstr(comando, "Actualiza")) { //BOTÓN ACTUALIZA
            Temperatura=!Temperatura;
            enviaTemp();
            Temp = String (t);
            Hum = String (h);
            }

//VENTILADOR
           // switch Ventilador turned ON
            if ( strstr(comando, "Ventilador ON")){
             if(autoVent == false){ //Si está en modo manual
              Ventilador = true;
              enviaVentilador(1);
              }
              else if (autoVent == true){
    Serial.println("Desactive el modo automatico primero");}
            }
```

```
                // switch Ventilador turned OFF
                 if ( strstr(comando, "Ventilador OFF")){
                  if(autoVent == false){
                   Ventilador = false;
                   enviaVentilador(1);
                   }
                   else if (autoVent == true){
Serial.println("Desactive el modo automatico primero");}
                 }

                 //Modo Automático ON
                 if ( strstr(comando, "AUTO ON")){
                 autoVent = true;
                 Serial.println("Modo Ventilador Automatico");
                 }

                 //Modo Manual ON
                 if ( strstr(comando, "AUTO OFF")){
                 autoVent = false;
                 Ventilador = false;
                 enviaVentilador(1);
                 Serial.println("Modo Ventilador Manual");
                 }



  clienteApp.println("OK"); //Muy importante! "OK" al final de cada
  Button, en los Labels no se necesita

  }
}
}



//INTERFAZ FÍSICA CUANDO NO ESTÁ CONECTADO EL SMARTPHONE
else if (!clienteApp){

//ALARMA 1 ó 2 ON?
Can0.attachCANInterrupt(0, leeClock);

//PULSADOR PLANTA 0
    vPULS1 = digitalRead(pinPULS1);
   if (vPULS1 == HIGH) {
    Serial.println("Pulsador Planta 0");
    if(bLuz[1]== false){ Luz = true; enviaLuz(1);}
    else if(bLuz[1]== true){ Luz = false; enviaLuz(1);}
    delay(500);
    if(bLuz[2]== false){ Luz = true; enviaLuz(2);}
    else if(bLuz[2]== true){ Luz = false; enviaLuz(2);}
    delay(500);
    if(bLuz[5]== false){ Luz = true; enviaLuz(5);}
    else if(bLuz[5]== true){ Luz = false; enviaLuz(5);}
    delay(500);
   }
```

```
//PULSADOR PLANTA 1
   vPULS2 = digitalRead(pinPULS2);
   if (vPULS2 == HIGH) { //PULSADOR PLANTA 1
    Serial.println("Pulsador Planta 1");
    if(bLuz[3]== false){ Luz = true; enviaLuz(3);}
    else if(bLuz[3]== true){ Luz = false; enviaLuz(3);}
    delay(500);
    if(bLuz[4]== false){ Luz = true; enviaLuz(4);}
    else if(bLuz[4]== true){ Luz = false; enviaLuz(4);}
    delay(500);
    if(bLuz[6]== false){ Luz = true; enviaLuz(6);}
    else if(bLuz[6]== true){ Luz = false; enviaLuz(6);}
    delay(500);
   }

//VENTILADOR AUTOMÁTICO
   h = dht.readHumidity(); // LECTURA DE HUMEDAD
   t = dht.readTemperature(); //LECTURA DE TEMPERATURA

   if (autoVent == true){

      if (t>=25 && Ventilador == false){          // switch Ventilador
turned ON cuando Temp>=25
      Ventilador = true;
      Serial.println("AUTO - Ventilador encendido debido a temperatura
>= 25ºC");
      enviaVentilador(1);
      }

      else if (t<=23 && Ventilador == true){       // switch
Ventilador turned OFF cuando Temp<25
      Ventilador = false;
      Serial.println("AUTO - Ventilador apagado debido a temperatura <
25ºC");
      enviaVentilador(1);
      }
   }


}
}



/************************ACABA LOOP************************/


//MENSAJES ENVIADOS
void printFrame (CAN_FRAME &frame){
   Serial.print("ID: 0x" ); Serial.print(frame.id, HEX);
Serial.print("\t");
   Serial.print ("DATO:"); Serial.print("\t");

 for(int i = 0; i<frame.length; i++){ // imprime los datos
          Serial.print(frame.data.byte[i], HEX);
          Serial.print("  ");
          b[i] = frame.data.byte[i];
 }
 Serial.println();
}
```

```
//Si el mensaje es recibido, el código de lectura sería el siguiente,
pero se puede solucionar llamando a la función con un *. Ex:
printFrameIN(*tempCAN);
        /*Serial.print("ID: 0x" ); Serial.print(tempCAN->id, HEX);
Serial.print("\t"); Serial.print("DATO:"); Serial.print("\t");

        for(int i = 0; i<sizeof(tempCAN->data); i++){
                        //Imprime los datos recibidos, byte a byte
                Serial.print(tempCAN->data.byte[i], HEX);
                Serial.print("  ");
        }
        Serial.println();*/


//MENSAJES RECIBIDOS
void printFrameIN (CAN_FRAME &frame){
   Serial.print("ID: 0x" ); Serial.print(frame.id, HEX);
Serial.print("\t"); Serial.print ("DATO:"); Serial.print("\t");

 for(int i = 0; i<frame.length; i++){ // imprime los datos
          Serial.print(frame.data.byte[i], HEX);
          Serial.print("  ");
          bIN[i] = frame.data.byte[i];
 }
 Serial.println();
}

//FUNCIONES LUZ
void enviaLuz(int numLuz){

  CAN_FRAME enviaLuz;
  enviaLuz.extended = false;
  enviaLuz.priority = 4; //0-15 lower is higher priority
  enviaLuz.length =  8;

  //Rellenamos la parte alta del dato, que de momento no usaremos
byte[7-4]
  enviaLuz.data.high = 0xAAAAAAAA; //TRAMA DE RELLENO de "1010..10"
succesivamente
  //BYTE [3] Nº PLACA
  enviaLuz.data.byte[2] = 0x00; //BYTE [2] TIPO OBJETO (LUZ - 0x00)
  //BYTE [1] Nº OBJETO
  //BYTE [0] ACCIÓN


if (Luz == true){
  enviaLuz.data.byte[0] = 0x01; //ENCENDER LUZ

    if (numLuz==1) { enviaLuz.data.byte[1] = 0x01;
        enviaLuz.data.byte[3] = 0x01; enviaLuz.id = 0x300;}
    if (numLuz==2) { enviaLuz.data.byte[1] = 0x02;
        enviaLuz.data.byte[3] = 0x01; enviaLuz.id = 0x302;}
    if (numLuz==3) { enviaLuz.data.byte[1] = 0x03;
        enviaLuz.data.byte[3] = 0x02; enviaLuz.id = 0x304;}
    if (numLuz==4) { enviaLuz.data.byte[1] = 0x04;
        enviaLuz.data.byte[3] = 0x02; enviaLuz.id = 0x306;}



    if (numLuz==5 && bLuz[5]== false) { enviaLuz.data.byte[1] = 0x05;
enviaLuz.data.byte[3] = 0x00; enviaLuz.id = 0x308;
digitalWrite(pinLED5, HIGH); }
```

```
    if (numLuz==6 && bLuz[6]== false) { enviaLuz.data.byte[1] = 0x06;
enviaLuz.data.byte[3] = 0x00; enviaLuz.id = 0x30A;
digitalWrite(pinLED6,HIGH); }

    bLuz[numLuz] = true;
}

else if (Luz == false){
  enviaLuz.data.byte[0] = 0x00; //APAGAR LUZ

    if (numLuz==1) { enviaLuz.data.byte[1] = 0x01;
enviaLuz.data.byte[3] = 0x01; enviaLuz.id = 0x301;}
    if (numLuz==2) { enviaLuz.data.byte[1] = 0x02;
enviaLuz.data.byte[3] = 0x01; enviaLuz.id = 0x303;}
    if (numLuz==3) { enviaLuz.data.byte[1] = 0x03;
enviaLuz.data.byte[3] = 0x02; enviaLuz.id = 0x305;}
    if (numLuz==4) { enviaLuz.data.byte[1] = 0x04;
enviaLuz.data.byte[3] = 0x02; enviaLuz.id = 0x307;}
    if (numLuz==5 && bLuz[5]== true) { enviaLuz.data.byte[1] = 0x05;
enviaLuz.data.byte[3] = 0x00; enviaLuz.id = 0x309;
digitalWrite(pinLED5,LOW); }
    if (numLuz==6 && bLuz[6]== true) { enviaLuz.data.byte[1] = 0x06;
enviaLuz.data.byte[3] = 0x00; enviaLuz.id = 0x30B;
digitalWrite(pinLED6,LOW); }

    bLuz[numLuz] = false;
}



 Can0.sendFrame(enviaLuz);
 printFrame(enviaLuz);
 Serial.print("LED "); Serial.print(numLuz);
 if (Luz == true){ Serial.println(" ENCENCIDO");}
 else if (Luz == false){ Serial.println(" APAGADO");}
}

//FUNCIONES TEMPERATURA

void enviaTemp(){

  CAN_FRAME enviaTemp;  //creamos una variable tipo CAN_FRAME

  enviaTemp.id = 0x500;
  enviaTemp.extended = false;
  enviaTemp.priority = 4; //0-15 lower is higher priority
  enviaTemp.length =  8;

  h = dht.readHumidity(); // Reading temperature or humidity takes
about 250 milliseconds!
  t = dht.readTemperature();
```

```
  //Rellenamos la parte alta del dato, que de momento no usaremos
byte[7-4]
  enviaTemp.data.high = 0xAAAAAAAA; //trama de "1010..10"
succesivamente
  enviaTemp.data.byte[2] = 0x01; //BYTE [2] TIPO OBJETO (SENSOR
TEMPERATURA - 0x01)
  enviaTemp.data.byte[3] = 0x00; //BYTE [3] NºPLACA - 0x00
  enviaTemp.data.byte[4] = h; //BYTE [4] HUMEDAD
  enviaTemp.data.byte[5] = t; //BYTE [5] TEMPERATURA

                if(Temperatura == true){
                enviaTemp.data.byte[0] = 0x01;
                enviaTemp.data.byte[1] = 0x00;
                }

                if(Temperatura == false){
                enviaTemp.data.byte[0] = 0x00;
                enviaTemp.data.byte[1] = 0x00;
                }

 Can0.sendFrame(enviaTemp);
 printFrame(enviaTemp);

  Serial.print("TEMPERATURA: "); Serial.print(t); Serial.print(" *C
"); Serial.print("\t");
  Serial.print("HUMEDAD: "); Serial.print(h); Serial.print(" %\t");
Serial.println();
}

void dht_OK (){ //Comprueba si el sensor está OK

  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
}

//FUNCIONES PERSIANAS

void enviaPersiana(int numPersiana){

  CAN_FRAME enviaPersiana;

  enviaPersiana.extended = false;
  enviaPersiana.priority = 4; //0-15 lower is higher priority
  enviaPersiana.length =  8;


  //Rellenamos la parte alta del dato, que de momento no usaremos
byte[7-4]
  enviaPersiana.data.high = 0xAAAAAAAA; //TRAMA DE RELLENO de
"1010..10" succesivamente
  //BYTE [3] Nº PLACA
  enviaPersiana.data.byte[2] = 0x04; //BYTE [2] TIPO OBJETO (PERSIANA
- 0x04)
  //BYTE [1] Nº OBJETO
  //BYTE [0] ACCIÓN
```

```
if (persianaMov == false){ //PARAR PERSIANA
  enviaPersiana.data.byte[0] = 0x00;
      if (numPersiana==1) { enviaPersiana.data.byte[1] = 0x01;
enviaPersiana.data.byte[3] = 0x01;  enviaPersiana.id = 0x701;}
      if (numPersiana==2) { enviaPersiana.data.byte[1] = 0x02;
enviaPersiana.data.byte[3] = 0x02;  enviaPersiana.id = 0x704;}

}

else if (persianaMov == true){
  if (sentidoPersiana == true){ //SUBIR PERSIANA
    enviaPersiana.data.byte[0] = 0x01;
      if (numPersiana==1) { enviaPersiana.data.byte[1] = 0x01;
enviaPersiana.data.byte[3] = 0x01;  enviaPersiana.id = 0x702;}
      if (numPersiana==2) { enviaPersiana.data.byte[1] = 0x02;
enviaPersiana.data.byte[3] = 0x02;  enviaPersiana.id = 0x705;}
  }


  else if (sentidoPersiana == false){ //BAJAR PERSIANA
    enviaPersiana.data.byte[0] = 0x02;
      if (numPersiana==1) { enviaPersiana.data.byte[1] = 0x01;
enviaPersiana.data.byte[3] = 0x01;  enviaPersiana.id = 0x703;}
      if (numPersiana==2) { enviaPersiana.data.byte[1] = 0x02;
enviaPersiana.data.byte[3] = 0x02;  enviaPersiana.id = 0x706;}
  }
}



 Can0.sendFrame(enviaPersiana);
 printFrame(enviaPersiana);
}


void leePersiana(CAN_FRAME *persianaCAN){

        int nP; //número persiana
        int estadoPersiana;
        int posP;



        if(persianaCAN->data.byte[2] == 0x04 && persianaCAN-
>data.byte[0] == 0x00){

                  nP = persianaCAN->data.byte[1];
                  estadoPersiana = persianaCAN->data.byte[4];
                  posP = persianaCAN->data.byte[5];

                  if(nP == 1){posPersiana1 = posP;}
                  else if(nP == 2){posPersiana2 = posP;}
```

```
                    Serial.print("PERSIANA " ); Serial.print(nP);
if (estadoPersiana==0 || posP==10 || posP==0){Serial.print(":
PARADA");}
                    else if (estadoPersiana == 1){Serial.print(":
SUBIENDO");}
                    else if (estadoPersiana == 2){Serial.print(":
BAJANDO");}
                    Serial.print(" en POSICION "); Serial.println(posP);



        Serial.print("Mensaje de persiana recibido:");
Serial.print("\t");

        printFrameIN(*persianaCAN);

    }

}




//FUNCIONES VENTILADOR
void enviaVentilador(int numVent){

  CAN_FRAME enviaVent;


  enviaVent.extended = false;
  enviaVent.priority = 4; //0-15 lower is higher priority
  enviaVent.length =  8;


  //Rellenamos la parte alta del dato, que de momento no usaremos
byte[7-4]
  enviaVent.data.high = 0xAAAAAAAA; //TRAMA DE RELLENO de "1010..10"
succesivamente
  enviaVent.data.byte[3] = 0x02; //BYTE [3] Nº PLACA - 0x00
  enviaVent.data.byte[2] = 0x05; //BYTE [2] TIPO OBJETO (VENTILADOR -
0x05)
  //BYTE [1] Nº OBJETO
  //BYTE [0] ACCIÓN


if (Ventilador == true){
  enviaVent.data.byte[0] = 0x01; //ENCENDER VENTILADOR
    if (numVent==1) { enviaVent.data.byte[1] = 0x00; enviaVent.id =
0x200;}
}

else if (Ventilador == false){
  enviaVent.data.byte[0] = 0x00; //APAGAR VENTILADOR
    if (numVent==1) { enviaVent.data.byte[1] = 0x00; enviaVent.id =
0x201;}
}
```

```
 Can0.sendFrame(enviaVent);
 printFrame(enviaVent);
 Serial.print("VENTILADOR "); Serial.print(numVent);
 if (Ventilador == true){ Serial.println(" ENCENCIDO");}
 else if (Ventilador == false){ Serial.println(" APAGADO");}
}

//FUNCIONES CLOCK RTC


void leeClock(CAN_FRAME *clockCAN){

        int numAlarm; //estado alarma

        if(clockCAN->data.byte[2] == 0x06 && clockCAN->data.byte[1] ==
0x00){

                numAlarm = clockCAN->data.byte[0];


         if(numAlarm == 1 || numAlarm == 2){
                int Mes = clockCAN->data.byte[4];
                int Dia = clockCAN->data.byte[5];
                int Minuto = clockCAN->data.byte[6];
                int Hora = clockCAN->data.byte[7];
                Serial.println("Mensaje de reloj RTC recibido:");
                printFrameIN(*clockCAN);
           }


         //ALARMA DESACTIVADA
         if (numAlarm == 0 && (Alarma1 == true || Alarma2 == true)){
            if(Alarma1 == true){Serial.println("ALARMA 1
DESACTIVADA");}
            if(Alarma2 == true){Serial.println("ALARMA 2
DESACTIVADA");}
         Alarma1 = false;
         Alarma2 = false;
         }

         //ALARMA 1 ACTIVADA
         if (numAlarm == 1 && Alarma1 == false){
         Serial.println("ALARMA 1 ACTIVADA");
            //ENCIENDE LUZ 3
            if(bLuz[3]== false){ Luz = true; enviaLuz(3);}
            //SUBE PERSIANA 1
            persianaMov = true;
            sentidoPersiana = true;
            enviaPersiana(1);
            Can0.attachCANInterrupt(1, leePersiana);
         Alarma1 = true;
         }
```

```
            //ALARMA 2 ACTIVADA
            if (numAlarm == 2 && Alarma2 == false){
            Serial.println("ALARMA 2 ACTIVADA");
               //APAGA LUZ 4
               if(bLuz[4]== false){ Luz = true; enviaLuz(4);} //FALSE
               //BAJA PERSIANA 2
               persianaMov = true;
               sentidoPersiana = true; //FALSE
               enviaPersiana(2);
               Can0.attachCANInterrupt(1, leePersiana);
            Alarma2 = true;
            }

         }

}
```

8.1.2. Arduino UNO - 1

```
//ARDUINO UNO - 1

#include <mcp_can.h>
#include <SPI.h>
#include <Wire.h>
#include "RTClib.h"

const int SPI_CS_PIN = 10; //the cs pin of the version after v1.1 is
default to D9, v0.9b and v1.0 is default to D10

  unsigned char buf[8]; //declaramos el buffer para poder leer datos
  unsigned char len;    //longitud del buffer
  unsigned int canId;   //ID del buffer

boolean Luz[8];               //Control del estado de las luces
  //num total luces =>[i]
boolean Ventilador[4];
boolean Temperatura;          //Variable para poder actualizar la
Temperatura
boolean PIR;
boolean persianaParada;
boolean estadoAlarma1 = false;
boolean estadoAlarma2 = false;


//VARIABLE RELOJ RTC
RTC_DS3231 rtc; //Reloj RTC modelo DS3231
String daysOfTheWeek[7] = {"Domingo", "Lunes", "Martes", "Miercoles",
"Jueves", "Viernes", "Sabado" };
String monthsNames[12] = {"Enero", "Febrero", "Marzo", "Abril",
"Mayo",  "Junio",
"Julio","Agosto","Septiembre","Octubre","Noviembre","Diciembre" };


//TEMPERATURA Y HUMEDAD
int valorT = 0;
int valorH = 0;

//PERSIANAS
const int pos[8] = {0,0,0,0,0,0,0,0}; //Control de la posición de las
persianas //num total persianas => [i]
int posicion = 0;
int posMax = 10; //Las persianas tienen (0-10) posiciones
int posMin = 0;
int estadoPers; //Parada, Subiendo, Bajando (0,1,2)

//definicion variables
int motorSpeed = 1200;   // variable para fijar la velocidad
int stepCounter = 0;     // contador para los pasos
int stepsPerRev = 4076;  // pasos para una vuelta completa

const int numSteps = 8; //secuencia media fase
const int stepsLookup[8] = { B1000, B1100, B0100, B0110, B0010, B0011,
B0001, B1001 }; //secuencia media fase
```

```
//PINES

//LEDS
const int pinLED1 = 4; //LED1
const int pinLED2 = 5; //LED2

//PERSIANA 1
const int pinIN1 = 6;    // 28BYJ48 In1
const int pinIN2 = 7;    // 28BYJ48 In2
const int pinIN3 = 8;    // 28BYJ48 In3
const int pinIN4 = 9;   // 28BYJ48 In4

const int pinPersiana1[4] = { pinIN1, pinIN2, pinIN3, pinIN4 };


//INICIAMOS COMPONENTES
MCP_CAN CAN(SPI_CS_PIN);                                    // Set CS
pin


void setup(){
    Serial.begin(115200);


    while (CAN_OK != CAN.begin(CAN_250KBPS)){            // DOBLE
VELOCIDAD AL ARDUINO DUE!!!!  init can bus : baudrate = 500k
        Serial.println("CAN BUS Shield init fail");
        Serial.println(" Init CAN BUS Shield again");
        delay(100);
    }
    Serial.println("CAN BUS Shield init ok!");

    //ACTIVAMOS RELOJ
    Wire.begin();
    rtc.begin();
    clock_OK(); //Reloj RTC

    //ACTIVAMOS CONTROL LUCES
    pinMode(pinLED1, OUTPUT); //LED1
    pinMode(pinLED2, OUTPUT); //LED2


    //ACTIVAMOS PERSIANA 1
    pinMode(pinIN1, OUTPUT);  //28BYJ48 In1 - Persiana 1
    pinMode(pinIN2, OUTPUT);  //28BYJ48 In2 - Persiana 1
    pinMode(pinIN3, OUTPUT);  //28BYJ48 In3 - Persiana 1
    pinMode(pinIN4, OUTPUT);  //28BYJ48 In4 - Persiana 1

}

void loop(){

  //ALARMA 1: 7:15 DE LUNES A VIERNES
  if (estadoAlarma1 == false && Alarma1ON()){ // Apagado y debería
estar encendido
      //ENVIA MENSAJE
      clockaDue(1);
      fechaActual();
      estadoAlarma1 = true;
      //Serial.println("Alarma 1 activada");
  }
```

```
 else if (estadoAlarma1 == true && !Alarma1ON()){  // Encendido y
deberia estar apagado
      clockaDue(0);
      estadoAlarma1 = false;
      //Serial.println("Alarma 1 desactivada");
  }

  //ALARMA 2: 23:15 DE LUNES A VIERNES
  if (estadoAlarma2 == false && Alarma2ON()){ // Apagado y debería
estar encendido
      //ENVIA MENSAJE
      clockaDue(2);
      fechaActual();
      estadoAlarma2 = true;
      //Serial.println("Alarma 2 activada");
  }
 else if (estadoAlarma2 == true && !Alarma2ON()){  // Encendido y
deberia estar apagado
      clockaDue(0);
      estadoAlarma2 = false;
      //Serial.println("Alarma 2 desactivada");
  }

  //LECTURA DE BUFFER CAN
  leeDato();

  //LUCES
  if (buf[2] == 0x00){

    //LUZ 1
    if (buf[1] == 0x01){
      if (buf[0] == 0x01 && Luz[1] == false){ //LED1 ON
        Luz[1] = true;
        digitalWrite(pinLED1,HIGH);
        Serial.println("LED1 encendido");
      }

      if (buf[0] == 0x00 && Luz[1] == true){ //LED1 OFF
        Luz[1] = false;
        digitalWrite(pinLED1,LOW);
        Serial.println("LED1 apagado");
      }
    }

    //LUZ 2
    if (buf[1] == 0x02){
      if (buf[0] == 0x01 && Luz[2] == false){ //LED2 ON
        Luz[2] = true;
        digitalWrite(pinLED2,HIGH);
        Serial.println("LED2 encendido");
      }

      if (buf[0] == 0x00 && Luz[2] == true){ //LED2 OFF
        Luz[2] = false;
        digitalWrite(pinLED2,LOW);
        Serial.println("LED2 apagado");
      }
    }
```

```
  //LUZ 3
  if (buf[1] == 0x03){
    if (buf[0] == 0x01 && Luz[3] == false){ //LED3 ON
      Luz[3] = true;
      Serial.println("LED3 encendido");
    }

    if (buf[0] == 0x00 && Luz[3] == true){ //LED3 OFF
      Luz[3] = false;
      Serial.println("LED3 apagado");
    }
  }

  //LUZ 4
  if (buf[1] == 0x04){
    if (buf[0] == 0x01 && Luz[4] == false){ //LED4 ON
      Luz[4] = true;
      Serial.println("LED4 encendido");
    }

    if (buf[0] == 0x00 && Luz[4] == true){ //LED4 OFF
      Luz[4] = false;
      Serial.println("LED4 apagado");
    }
  }

  //LUZ 5
  if (buf[1] == 0x05){
    if (buf[0] == 0x01 && Luz[5] == false){ //LED5 ON
      Luz[5] = true;
      Serial.println("LED5 encendido");
    }

    if (buf[0] == 0x00 && Luz[5] == true){ //LED5 OFF
      Luz[5] = false;
      Serial.println("LED5 apagado");
    }
  }

  //LUZ 6
  if (buf[1] == 0x06){
    if (buf[0] == 0x01 && Luz[6] == false){ //LED6 ON
      Luz[6] = true;
      Serial.println("LED6 encendido");
    }

    if (buf[0] == 0x00 && Luz[6] == true){ //LED6 OFF
      Luz[6] = false;
      Serial.println("LED6 apagado");
    }
  }

}
```

```
 //SENSOR TEMPERATURA
 else if (buf[2] == 0x01){

     if (buf[1] == 0x00 && buf[0] == 0x01 && Temperatura == false){
     valorT = buf[5];
     valorH = buf[4];
     Serial.print("Temperatura: "); Serial.print(valorT, DEC);
Serial.print(" *C ");
     Serial.print("Humedad: "); Serial.print(valorH, DEC);
Serial.print(" %\t");
     Serial.println();
     Temperatura = true;
     }

     else if (buf[1] == 0x00 && buf[0] == 0x00 && Temperatura ==
true){
     valorT = buf[5];
     valorH = buf[4];
     Serial.print("Temperatura: "); Serial.print(valorT, DEC);
Serial.print(" *C ");
     Serial.print("Humedad: "); Serial.print(valorH, DEC);
Serial.print(" %\t");
     Serial.println();
     Temperatura = false;
     }

   }



  //PERSIANAS
   else if (buf[2] == 0x04){

    if (buf[1] == 0x01){ muevePersiana(1);} //PERSIANA 1
    //else if (buf[1] == 0x02){ Serial.println("Orden para persiana
2");} //PERSIANA 2
   }

 //VENTILADOR
  else if (buf[2] == 0x05){

    //LUZ 1
    if (buf[1] == 0x00){
      if (buf[0] == 0x01 && Ventilador[1] == false){ //Ventilador ON
        Ventilador[1] = true;
        Serial.println("Ventilador encendido");
      }

      if (buf[0] == 0x00 && Ventilador[1] == true){ //Ventilador OFF
        Ventilador[1] = false;
        Serial.println("Ventilador apagado");
      }
    }
  }

}
/**************ACABA LOOP**************/
```

```
void leeDato(){

    if(CAN_MSGAVAIL == CAN.checkReceive()){ // check if data coming

      CAN.readMsgBuf(&len, buf); // lee datos, len: longitude datos,
buf: data buf

      canId = CAN.getCanId();

      Serial.print("ID: 0x" ); Serial.print(canId, HEX);
Serial.print("\t");
      Serial.print ("DATO:"); Serial.print("\t");

      for(int i = 0; i<len; i++){ // imprime los datos
          Serial.print(buf[i], HEX);
          Serial.print("  ");
      }
      Serial.print("\t");

      fechaActual();
  }
}


//FUNCIONES PERSIANAS

void persaDue(int numPers){

      byte datoPERS[8];
      int id = 0x700;
      int pos;

      pos = posicion;

      datoPERS[0] = 0x00;         //BYTE [0] ACCIÓN
      datoPERS[1] = numPers;      //BYTE [1] Nº OBJETO
      datoPERS[2] = 0x04;         //BYTE [2] TIPO DE OBJETO (0x04 -
PERSIANA)
      datoPERS[3] = 0x01;         //BYTE [3] NºPLACA
      datoPERS[4] = estadoPers;   //ESTADO DE LA PERSIANA
      datoPERS[5] = pos;          //POSICIÓN DE LA PERSIANA
      datoPERS[6] = 0xAA;
      datoPERS[7] = 0xAA;

   CAN.sendMsgBuf(id, 0, sizeof(datoPERS), datoPERS); // send data:
 id = 0x00, standard frame (0 estandard, 1 extended), data len = 8,
stmp: data buf

   Serial.print("PERSIANA " ); Serial.print(numPers);
   if (estadoPers == 0){Serial.print(" PARADA");}
   if (estadoPers == 1){Serial.print(" SUBIENDO");}
   if (estadoPers == 2){Serial.print(" BAJANDO");}

   Serial.print(" en POSICION "); Serial.println(pos);

   Serial.print("Mensaje enviado:  ");

   Serial.print("ID: 0x" ); Serial.print(id, HEX); Serial.print("\t");
Serial.print ("DATO:"); Serial.print("\t");
```

```
   for(int i = 0; i<(sizeof(datoPERS)); i++){ // imprime los datos
      Serial.print(datoPERS[i], HEX);
      Serial.print("  ");
   }
   Serial.println();
}


void muevePersiana (int numPers){

  while (posMax >= posicion >= posMin && buf[3] == 0x01 && buf[2] ==
0x04 ){

    leeDato();

    if (buf[3] == 0x01 && buf[2] == 0x04 && buf[1] == 0x01){

          if(buf[0] == 0x00 && persianaParada==false){ //OK MANUAL
            Serial.println("Persiana Parada");
            estadoPers=0;
            break;
            }
          else if (buf[0] == 0x01 && posicion != posMax){ //PERSIANA
SUBE
            sentidoReloj(numPers);
            Serial.println("Persiana Subiendo");
            estadoPers=1;
            persaDue(numPers);
            persianaParada=false;
            }
          else if (buf[0] == 0x02 && posicion != posMin){ //PERSIANA
BAJA
            sentidoAntiReloj(numPers);
            Serial.println("Persiana Bajando");
            estadoPers=2;
            persaDue(numPers);
            persianaParada=false;
            }

          else if (buf[0] == 0x01 && posicion == posMax &&
persianaParada==false){ //PERSIANA NO PUEDE SUBIR MÁS
            Serial.println("Persiana Arriba");
            estadoPers=0;
            break;
            }
          else if (buf[0] == 0x02 && posicion == posMin &&
persianaParada==false){ //PERSIANA NO PUEDE BAJAR MÁS
            Serial.println("Persiana Abajo");
            estadoPers=0;
            break;
            }
    }
    else if (buf[3] != 0x01 && buf[2] != 0x04 && buf[1] != 0x01){
          //Serial.println("Cambio de Orden");
          estadoPers=0;
          break; //OK AUTOMÁTICO
    }

  }
```

```
  if (estadoPers == 0 && persianaParada==false){ persaDue(numPers);}
//Envia mensaje estado persiana parada
  persianaParada=true;

}

void sentidoReloj(int numPers){ //Da una vuelta arriba

  for (int i = 0; i < stepsPerRev ; i++){
    stepCounter++;
    if (stepCounter >= numSteps) stepCounter = 0;

    if (numPers == 1) { setOutput(pinPersiana1, stepCounter);}
    //else if (numPers == 2) { setOutput(pinPersiana2, stepCounter);}

    delayMicroseconds(motorSpeed);
  }

  posicion++;

}

void sentidoAntiReloj(int numPers){ //Da una vuelta abajo

  for (int i = 0; i < stepsPerRev ; i++){
    stepCounter--;
    if (stepCounter < 0) stepCounter = numSteps - 1;

    if (numPers == 1) { setOutput(pinPersiana1, stepCounter);}
    //else if (numPers == 2) { setOutput(pinPersiana2, stepCounter);}

    delayMicroseconds(motorSpeed);
  }

  posicion--;

}

void setOutput(const int pinPersiana[4], int step){
  digitalWrite(pinPersiana[0], bitRead(stepsLookup[step], 0));
  digitalWrite(pinPersiana[1], bitRead(stepsLookup[step], 1));
  digitalWrite(pinPersiana[2], bitRead(stepsLookup[step], 2));
  digitalWrite(pinPersiana[3], bitRead(stepsLookup[step], 3));
}

//FUNCIONES RELOJ
void clock_OK (){

 if (!rtc.begin()){
   Serial.println(F("Couldn't find RTC"));
 }

 if(rtc.begin()){
   rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
   //rtc.adjust(DateTime(2016, 1, 21, 3, 0, 0));
   Serial.println("Reloj Ajustado");
 }
```

```
if (rtc.lostPower()) { // Si se ha perdido la corriente, fijar fecha y
hora
 rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));  // Fijar a fecha y
hora de compilacion. Si queremos fijar a fecha y hora específica. Ej:
21 de Enero de 2016 a las 03:00:00 // rtc.adjust(DateTime(2016, 1, 21,
3, 0, 0));
 }
}

void imprimeFecha (DateTime date){

 //Imprimir Fecha
 Serial.print(daysOfTheWeek[date.dayOfTheWeek()]); Serial.print(" ");
Serial.print(date.day(), DEC); Serial.print('/');
Serial.print(date.month(), DEC); Serial.print('/');
Serial.print(date.year(), DEC); Serial.print(" ");
 //Imprimir Hora
 Serial.print(date.hour(), DEC); Serial.print(':');
Serial.print(date.minute(), DEC); Serial.print(':');
Serial.print(date.second(), DEC); Serial.println();
}

void fechaActual(){
   DateTime fActual = rtc.now();
   imprimeFecha(fActual);
}

void clockaDue(int numAlarm){

      byte datoCLOCK[8];
      int id = 0x100;
      DateTime fActual = rtc.now(); //Enviamos (Hora:Minuto) (Dia/Mes)
de cuando se ha producido la alarma
      int Hora = fActual.hour();
      int Minuto = fActual.minute();
      int Dia = fActual.day();
      int Mes = fActual.month();

      datoCLOCK[0] = numAlarm; //BYTE[0] ACCIÓN
      datoCLOCK[1] = 0x00; //BYTE [1] Nº DE OBJETO
      datoCLOCK[2] = 0x06; //BYTE [2] TIPO DE OBJETO (0x06 - RTC
CLOCK)
      datoCLOCK[3] = 0x01; //BYTE [3] Nº DE PLACA
      datoCLOCK[4] = Mes;
      datoCLOCK[5] = Dia;
      datoCLOCK[6] = Minuto;
      datoCLOCK[7] = Hora;

   Serial.print("ALARMA " );
   if (numAlarm == 0){
      Serial.println(" DESACTIVADA");
      for(int i = 4; i<8; i++){ // Rellena los bytes 4-7 con 0xAA
      datoCLOCK[i] = 0xAA;
      }
   }
   if (numAlarm == 1){Serial.println(" 1 ACTIVADA");}
   if (numAlarm == 2){Serial.println(" 2 ACTIVADA");}

   CAN.sendMsgBuf(id, 0, sizeof(datoCLOCK), datoCLOCK); // send data:
 id = 0x00, standard frame (0 estandard, 1 extended), data len = 8,
stmp: data buf
```

```
   Serial.print("Mensaje enviado:  ");

   Serial.print("ID: 0x" ); Serial.print(id, HEX); Serial.print("\t");
Serial.print ("DATO:"); Serial.print("\t");

   for(int i = 0; i<(sizeof(datoCLOCK)); i++){ // imprime los datos
      Serial.print(datoCLOCK[i], HEX);
      Serial.print("  ");
   }
   Serial.println();

}

//ALARMA 1
 bool Alarma1ON(){

    DateTime date;
    bool hourCondition;
    int hourAlarm = 7;
    int minutesAlarm = 15;
    int secondsAlarm = 0;
    date = rtc.now();

        if (hourAlarm == date.hour()) {
          if (minutesAlarm == date.minute()) {
            if (secondsAlarm == date.second()) {
            hourCondition = true;
            }
            else{
            hourCondition = false;
            }
          }
        }

  if (hourCondition){ return true;}
  return false;
}

//ALARMA 2
 bool Alarma2ON(){

    DateTime date;
    bool hourCondition;
    int hourAlarm = 23;
    int minutesAlarm = 15;
    int secondsAlarm = 0;
    date = rtc.now();

        if (hourAlarm == date.hour()) {
          if (minutesAlarm == date.minute()) {
            if (secondsAlarm == date.second()) {
            hourCondition = true;
            }
            else{
            hourCondition = false;
            }
          }
        }

  if (hourCondition){ return true;}
  return false;
}
```

8.1.3. Arduino UNO - 2

```
//ARDUINO UNO - 2

#include <mcp_can.h>
#include <SPI.h>


const int SPI_CS_PIN = 10; //the cs pin of the version after v1.1 is
default to D9, v0.9b and v1.0 is default to D10

  unsigned char buf[8]; //declaramos el buffer para poder leer datos
  unsigned char len;    //longitud del buffer
  unsigned int canId;   //ID del buffer

boolean Luz[8];               //Control del estado de las luces
  //num total luces =>[i]
boolean Ventilador[4];
boolean Temperatura;          //Variable para poder actualizar la
Temperatura
boolean persianaParada;
boolean Alarma1;
boolean Alarma2;

//TEMPERATURA Y HUMEDAD
int valorT = 0;
int valorH = 0;

//PERSIANAS
const int pos[8] = {0,0,0,0,0,0,0,0}; //Control de la posición de las
persianas //num total persianas => [i]
int posicion = 0;
int posMax = 10; //Las persianas tienen (0-10) posiciones
int posMin = 0;
int estadoPers; //Parada, Subiendo, Bajando (0,1,2)

//definicion variables
int motorSpeed = 1200;   // variable para fijar la velocidad
int stepCounter = 0;     // contador para los pasos
int stepsPerRev = 4076;  // pasos para una vuelta completa

const int numSteps = 8; //secuencia media fase
const int stepsLookup[8] = { B1000, B1100, B0100, B0110, B0010, B0011,
B0001, B1001 }; //secuencia media fase

//PINES
//VENTILADOR
const int pinVENT = 3; //Ventilador

//LEDS
const int pinLED3 = 4; //LED1
const int pinLED4 = 5; //LED2
//PERSIANA 2
const int pinIN1 = 6;    // 28BYJ48 In1
const int pinIN2 = 7;    // 28BYJ48 In2
const int pinIN3 = 8;    // 28BYJ48 In3
const int pinIN4 = 9;   // 28BYJ48 In4

const int pinPersiana2[4] = { pinIN1, pinIN2, pinIN3, pinIN4 };
```

```
//INICIAMOS COMPONENTES
MCP_CAN CAN(SPI_CS_PIN);                                    // Set CS pin

void setup(){
    Serial.begin(115200);

    while (CAN_OK != CAN.begin(CAN_250KBPS)){           // DOBLE
VELOCIDAD AL ARDUINO DUE!!!!  init can bus : baudrate = 500k
        Serial.println("CAN BUS Shield init fail");
        Serial.println(" Init CAN BUS Shield again");
        delay(100);
    }
    Serial.println("CAN BUS Shield init ok!");

    //ACTIVAMOS CONTROL LUCES
    pinMode(pinVENT, OUTPUT); //VENTILADOR
    pinMode(pinLED3, OUTPUT); //LED3
    pinMode(pinLED4, OUTPUT); //LED4

    //ACTIVAMOS PERSIANA 1
    pinMode(pinIN1, OUTPUT);  //28BYJ48 In1 - Persiana 1
    pinMode(pinIN2, OUTPUT);  //28BYJ48 In2 - Persiana 1
    pinMode(pinIN3, OUTPUT);  //28BYJ48 In3 - Persiana 1
    pinMode(pinIN4, OUTPUT);  //28BYJ48 In4 - Persiana 1

}


void loop(){

  leeDato();

  //LUCES
  if (buf[2] == 0x00){

    //LUZ 1
    if (buf[1] == 0x01){
      if (buf[0] == 0x01 && Luz[1] == false){ //LED1 ON
        Luz[1] = true;
        Serial.println("LED1 encendido");
      }

      if (buf[0] == 0x00 && Luz[1] == true){ //LED1 OFF
        Luz[1] = false;
        Serial.println("LED1 apagado");
      }
    }

    //LUZ 2
    if (buf[1] == 0x02){
      if (buf[0] == 0x01 && Luz[2] == false){ //LED2 ON
        Luz[2] = true;
        Serial.println("LED2 encendido");
      }

      if (buf[0] == 0x00 && Luz[2] == true){ //LED2 OFF
        Luz[2] = false;
        Serial.println("LED2 apagado");
      }
    }
```

```
  //LUZ 3
  if (buf[1] == 0x03){
    if (buf[0] == 0x01 && Luz[3] == false){ //LED3 ON
      Luz[3] = true;
      digitalWrite(pinLED3,HIGH);
      Serial.println("LED3 encendido");
    }

    if (buf[0] == 0x00 && Luz[3] == true){ //LED3 OFF
      Luz[3] = false;
      digitalWrite(pinLED3,LOW);
      Serial.println("LED3 apagado");
    }
  }

  //LUZ 4
  if (buf[1] == 0x04){
    if (buf[0] == 0x01 && Luz[4] == false){ //LED4 ON
      Luz[4] = true;
      digitalWrite(pinLED4,HIGH);
      Serial.println("LED4 encendido");
    }

    if (buf[0] == 0x00 && Luz[4] == true){ //LED4 OFF
      Luz[4] = false;
      digitalWrite(pinLED4,LOW);
      Serial.println("LED4 apagado");
    }
  }

  //LUZ 5
  if (buf[1] == 0x05){
    if (buf[0] == 0x01 && Luz[5] == false){ //LED5 ON
      Luz[5] = true;
      Serial.println("LED5 encendido");
    }

    if (buf[0] == 0x00 && Luz[5] == true){ //LED5 OFF
      Luz[5] = false;
      Serial.println("LED5 apagado");
    }
  }

  //LUZ 6
  if (buf[1] == 0x06){
    if (buf[0] == 0x01 && Luz[6] == false){ //LED6 ON
      Luz[6] = true;
      Serial.println("LED6 encendido");
    }

    if (buf[0] == 0x00 && Luz[6] == true){ //LED6 OFF
      Luz[6] = false;
      Serial.println("LED6 apagado");
    }
  }
}
```

```
//SENSOR TEMPERATURA
 else if (buf[2] == 0x01){

     if (buf[1] == 0x00 && buf[0] == 0x01 && Temperatura == false){
     valorT = buf[5];
     valorH = buf[4];
     Serial.print("Temperatura: "); Serial.print(valorT, DEC);
Serial.print(" *C ");
     Serial.print("Humedad: "); Serial.print(valorH, DEC);
Serial.print(" %\t");
     Serial.println();
     Temperatura = true;
     }

     else if (buf[1] == 0x00 && buf[0] == 0x00 && Temperatura ==
true){
     valorT = buf[5];
     valorH = buf[4];
     Serial.print("Temperatura: "); Serial.print(valorT, DEC);
Serial.print(" *C ");
     Serial.print("Humedad: "); Serial.print(valorH, DEC);
Serial.print(" %\t");
     Serial.println();
     Temperatura = false;
     }
   }


  //PERSIANAS
   else if (buf[2] == 0x04){

    //if (buf[1] == 0x01){ Serial.println("Orden para persiana 1");}
//PERSIANA 1
    if (buf[1] == 0x02){ muevePersiana(2);} //PERSIANA 2
   }

  //VENTILADOR
  else if (buf[2] == 0x05){

    //LUZ 1
    if (buf[1] == 0x00){
      if (buf[0] == 0x01 && Ventilador[1] == false){ //Ventilador ON
        Ventilador[1] = true;
        digitalWrite(pinVENT,HIGH);
        Serial.println("Ventilador encendido");
      }

      if (buf[0] == 0x00 && Ventilador[1] == true){ //Ventilador OFF
        Ventilador[1] = false;
        digitalWrite(pinVENT,LOW);
        Serial.println("Ventilador apagado");
      }
    }
  }


  //CLOCK RTC
  else if (buf[2] == 0x06){

    //ALARMA DESACTIVADA
         if (buf[0] == 0 && (Alarma1 == true || Alarma2 == true)){
            if(Alarma1 == true){Serial.println("ALARMA 1
DESACTIVADA");}
```

```
            if(Alarma2 == true){Serial.println("ALARMA 2
DESACTIVADA");}
          Alarma1 = false;
          Alarma2 = false;
          }

          //ALARMA 1 ACTIVADA
          if (buf[0] == 1 && Alarma1 == false){
          Serial.println("ALARMA 1 ACTIVADA");
          Alarma1 = true;
          }

          //ALARMA 2 ACTIVADA
          if (buf[0] == 2 && Alarma2 == false){
          Serial.println("ALARMA 2 ACTIVADA");
          Alarma2 = true;
          }
    }

}
/**************ACABA LOOP**************/



void leeDato(){

    if(CAN_MSGAVAIL == CAN.checkReceive()){ // check if data coming

      CAN.readMsgBuf(&len, buf); // lee datos, len: longitude datos,
buf: data buf

      canId = CAN.getCanId();

      Serial.print("ID: 0x" ); Serial.print(canId, HEX);
Serial.print("\t"); Serial.print ("DATO:"); Serial.print("\t");

      for(int i = 0; i<len; i++){ // imprime los datos
          Serial.print(buf[i], HEX);
          Serial.print("  ");
      }
      Serial.println();
  }
}
```

```
//FUNCIONES PERSIANAS

void persaDue(int numPers){

      byte datoPERS[8];
      int id = 0x700;
      int pos;

      pos = posicion;

      datoPERS[0] = 0x00;         //BYTE [0] ACCIÓN
      datoPERS[1] = numPers;      //BYTE [1] Nº OBJETO
      datoPERS[2] = 0x04;         //BYTE [2] TIPO DE OBJETO (0x04 -
PERSIANA)
      datoPERS[3] = 0x02;         //BYTE [3] NºPLACA
      datoPERS[4] = estadoPers;   //ESTADO DE LA PERSIANA
      datoPERS[5] = pos;          //POSICIÓN DE LA PERSIANA
      datoPERS[6] = 0xAA;
      datoPERS[7] = 0xAA;

   CAN.sendMsgBuf(id, 0, sizeof(datoPERS), datoPERS); // send data:
 id = 0x00, standard frame (0 estandard, 1 extended), data len = 8,
stmp: data buf

   Serial.print("PERSIANA " ); Serial.print(numPers);
   if (estadoPers == 0){Serial.print(" PARADA");}
   if (estadoPers == 1){Serial.print(" SUBIENDO");}
   if (estadoPers == 2){Serial.print(" BAJANDO");}

   Serial.print(" en POSICION "); Serial.println(pos);

   Serial.print("Mensaje enviado:  ");

   Serial.print("ID: 0x" ); Serial.print(id, HEX); Serial.print("\t");
Serial.print ("DATO:"); Serial.print("\t");

   for(int i = 0; i<(sizeof(datoPERS)); i++){ // imprime los datos
      Serial.print(datoPERS[i], HEX);
      Serial.print("  ");
   }
   Serial.println();
}


void muevePersiana (int numPers){

  while (posMax >= posicion >= posMin && buf[3] == 0x02 && buf[2] ==
0x04 ){

    leeDato();

    if (buf[3] == 0x02 && buf[2] == 0x04 && buf[1] == 0x02){

          if(buf[0] == 0x00 && persianaParada==false){ //OK MANUAL
            Serial.println("Persiana Parada");
            estadoPers=0;
            break;
            }
```

```
          else if (buf[0] == 0x01 && posicion != posMax){ //PERSIANA
SUBE
            sentidoReloj(numPers);
            Serial.println("Persiana Subiendo");
            estadoPers=1;
            persaDue(numPers);
            persianaParada=false;
            }
          else if (buf[0] == 0x02 && posicion != posMin){ //PERSIANA
BAJA
            sentidoAntiReloj(numPers);
            Serial.println("Persiana Bajando");
            estadoPers=2;
            persaDue(numPers);
            persianaParada=false;
            }

          else if (buf[0] == 0x01 && posicion == posMax &&
persianaParada==false){ //PERSIANA NO PUEDE SUBIR MÁS
            Serial.println("Persiana Arriba");
            estadoPers=0;
            break;
            }
          else if (buf[0] == 0x02 && posicion == posMin &&
persianaParada==false){ //PERSIANA NO PUEDE BAJAR MÁS
            Serial.println("Persiana Abajo");
            estadoPers=0;
            break;
            }
    }
    else if (buf[3] != 0x02 && buf[2] != 0x04 && buf[1] != 0x02){
          //Serial.println("Cambio de Orden");
          estadoPers=0;
          break; //OK AUTOMÁTICO
    }

  }

  if (estadoPers == 0 && persianaParada==false){ persaDue(numPers);}
//Envia mensaje estado persiana parada
  persianaParada=true;


}


void sentidoReloj(int numPers){ //Da una vuelta arriba

  for (int i = 0; i < stepsPerRev ; i++){
    stepCounter++;
    if (stepCounter >= numSteps) stepCounter = 0;

    //if (numPers == 1) { setOutput(pinPersiana1, stepCounter);}
     if (numPers == 2) { setOutput(pinPersiana2, stepCounter);}


    delayMicroseconds(motorSpeed);
  }
  posicion++;
}
```

```
void sentidoAntiReloj(int numPers){ //Da una vuelta abajo

  for (int i = 0; i < stepsPerRev ; i++){
    stepCounter--;
    if (stepCounter < 0) stepCounter = numSteps - 1;

    //if (numPers == 1) { setOutput(pinPersiana1, stepCounter);}
    if (numPers == 2) { setOutput(pinPersiana2, stepCounter);}

    delayMicroseconds(motorSpeed);
  }

  posicion--;

}

void setOutput(const int pinPersiana[4], int step){
  digitalWrite(pinPersiana[0], bitRead(stepsLookup[step], 0));
  digitalWrite(pinPersiana[1], bitRead(stepsLookup[step], 1));
  digitalWrite(pinPersiana[2], bitRead(stepsLookup[step], 2));
  digitalWrite(pinPersiana[3], bitRead(stepsLookup[step], 3));
}
```

8.2. Conexionado de del sistema

Éste apartado pretende especificar las distintas conexiones del sistema de modo gráfico, se han utilizado los programas Adobe Photoshop y Fritzing, un editor de circuitos para Arduino. Se incluyen las conexiones realizadas a los distintos actuadores y sensores en función del controlador.

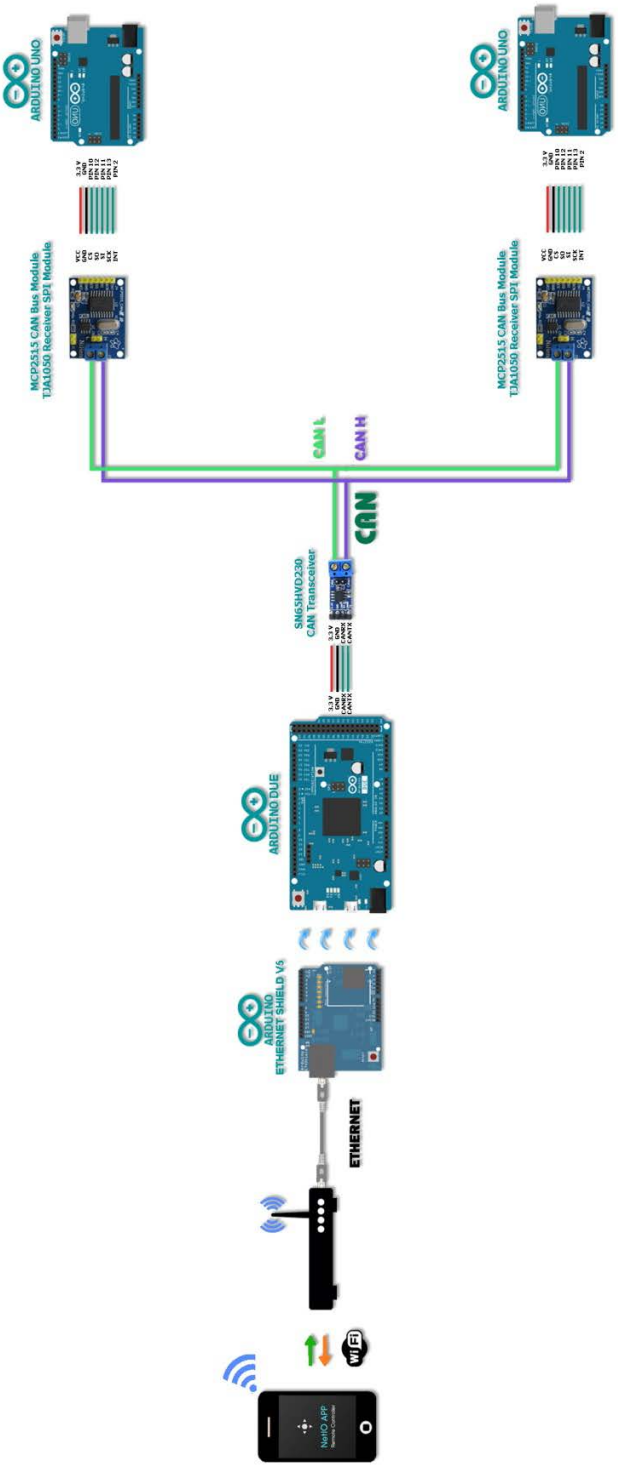


IMAGEN 64. ESQUEMA PRINCIPAL DEL SISTEMA DOMÓTICO

8.2.1. Arduino Due

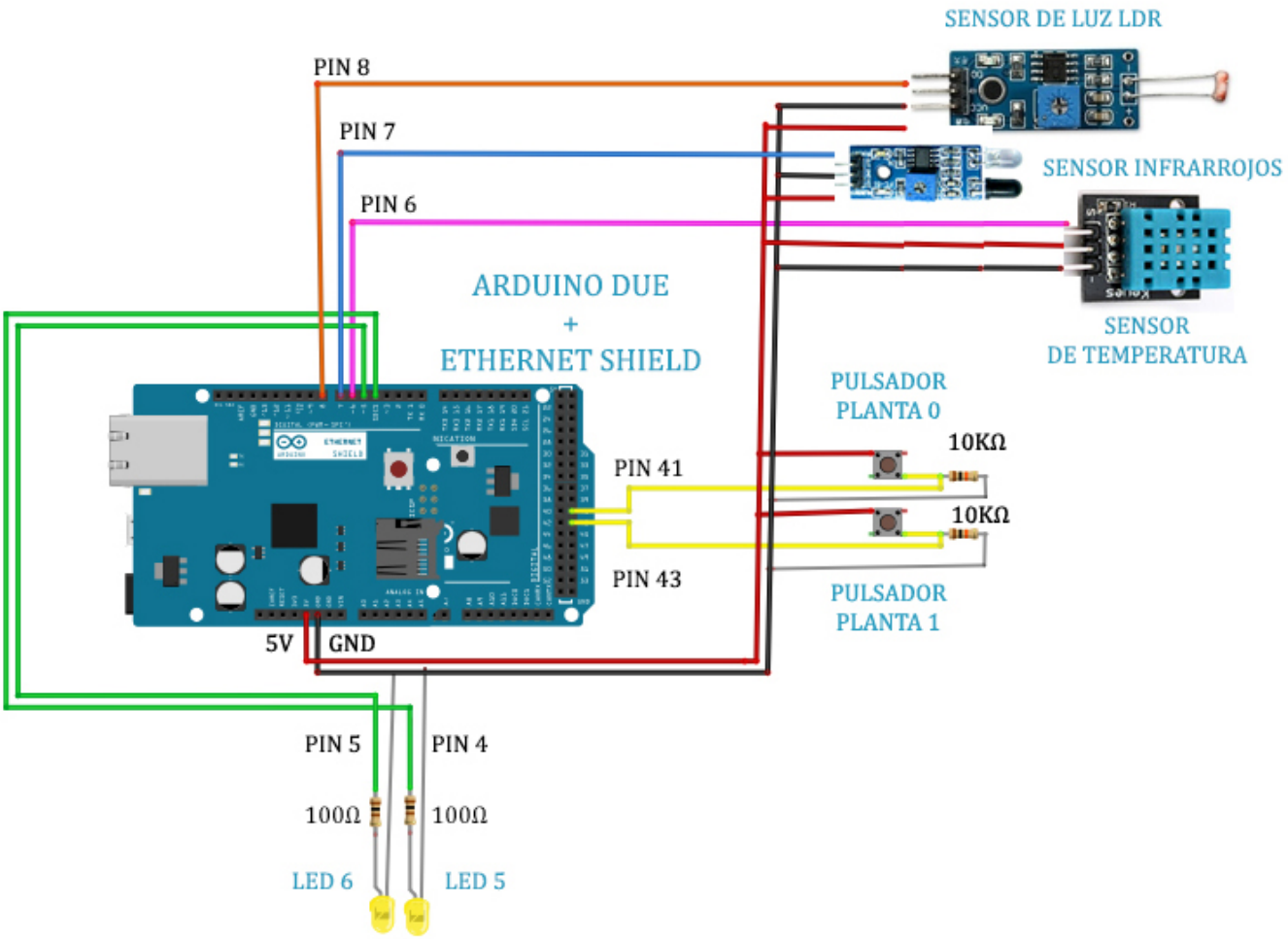


IMAGEN 65. CIRCUITO ARDUINO DUE - SENSORES Y ACTUADORES

8.2.2. Arduino UNO - 1

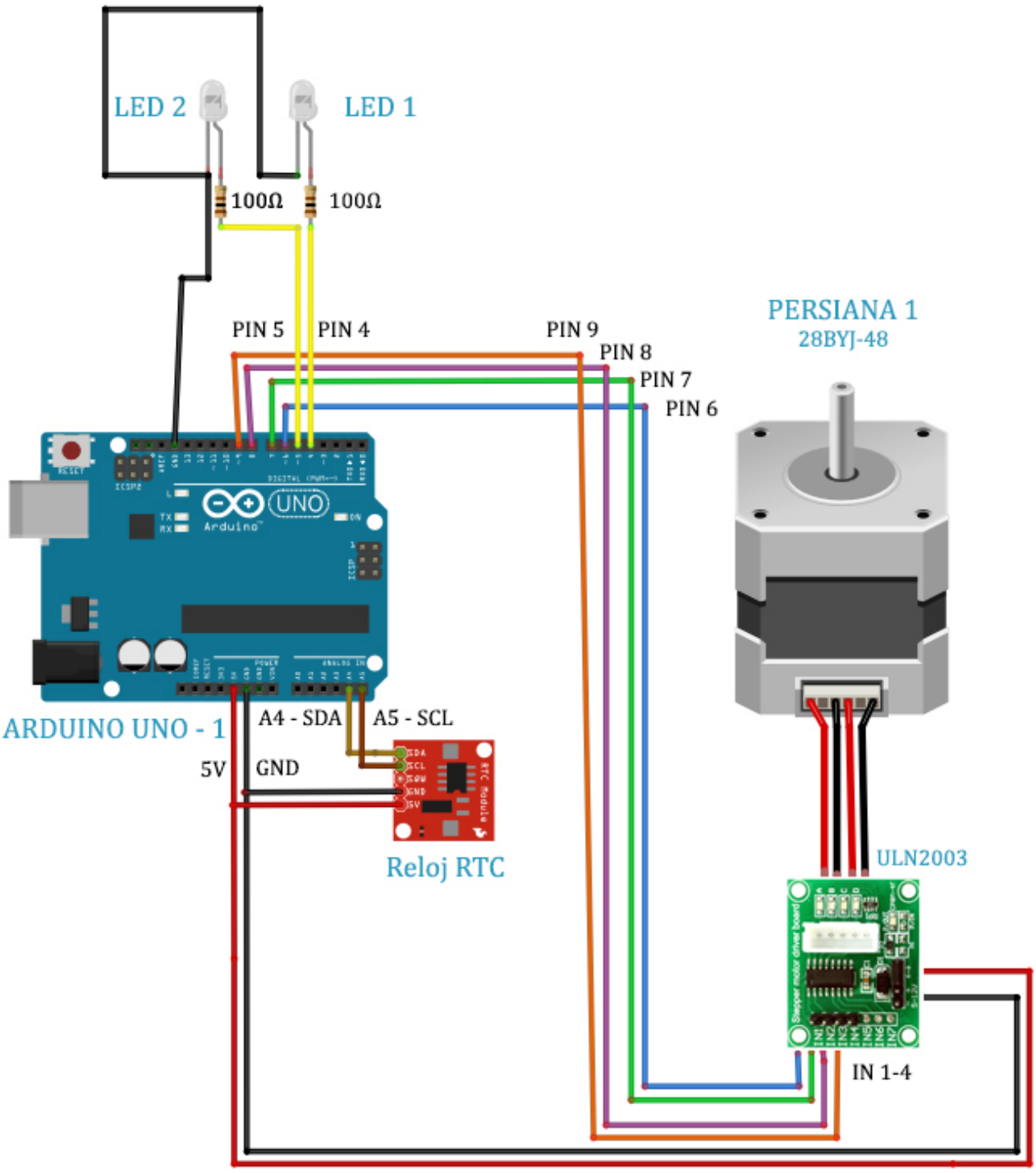


IMAGEN 66. CIRCUITO ARDUINO UNO-1 - SENSORES Y ACTUADORES

8.2.3. Arduino UNO - 2

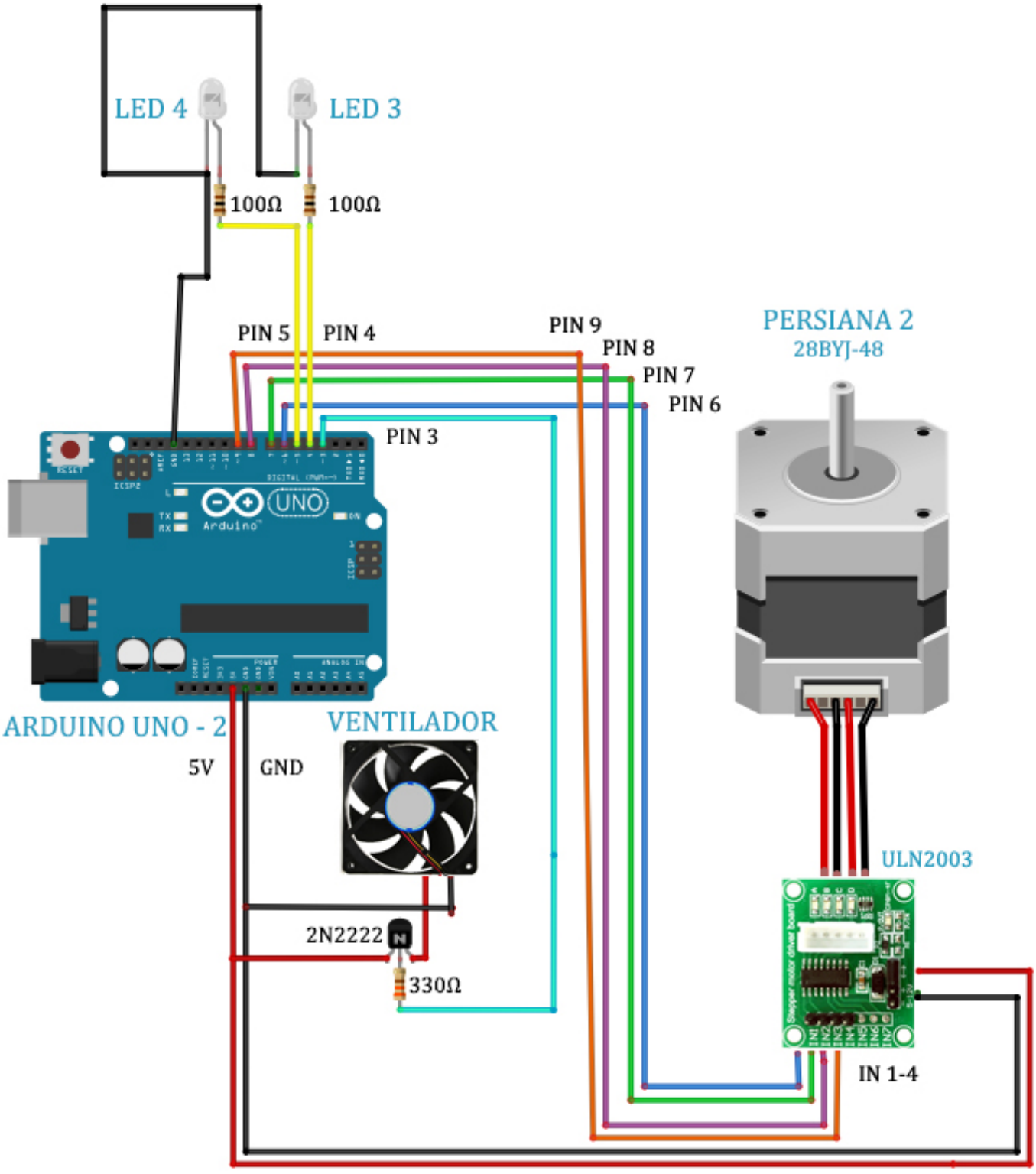


IMAGEN 67. CIRCUITO ARDUINO UNO-2 - SENSORES Y ACTUADORES

8.3. Cálculos del sistema

A lo largo del proyecto se han utilizado diferentes componentes electrónicos con valores no justificados. Sobretodo, se trata del uso de resistencias para disminuir la potencia utilizada para actuar las luces LEDs, los pulsadores y el ventilador.

Para empezar, se ha tomado una referencia aproximada de la caída de tensión de los diodos LEDs según su color. Cabe recordar que existen dos valores en la alimentación del sistema, 3,3V (Arduino Due) y 5V (Arduino UNO). Es aconsejable que la corriente de cada pin no supere los 20mA, así que se ha tomado ese valor como referencia.

COLOR DEL LED	TENSIÓN UMBRAL
Rojo	1,6V
Rojo alta luminosidad	1,9V
Amarillo	1,7V a 2V
Verde	2,4V
Naranja	2,4V
Blanco brillante	3,4V
Azul	3,4V
Azul 430nm	4,6V

TABLA 23. VALORES DE CAIDA DE TENSIÓN DE LED

Los LEDs utilizados son de color amarillo y blanco. Por una simple razón, los LEDs que dependen de la tensión de salida de Arduino Due no pueden ser de color blanco, ya que su caída de tensión es superior a la tensión que puede proporcionar el controlador.

De todos modos, en un hogar puede haber variedad de colores de luces, y los colores escogidos no desvarían mucho. A continuación se procede a realizar el cálculo de las resistencias utilizadas para los LEDs según su color, teniendo en cuenta la ley de Ohm:

LEY DE OHM:	$\Delta V = I \cdot R$	$P = I \cdot V$
LED BLANCO:	$5\ V - 3{,}4\ V = 20mA \cdot R$	$R = 80\Omega \simeq 100\ \Omega$
	$P = 1{,}6V \cdot 20mA$	$P = 0{,}032W$
LED AMARILLO:	$3{,}3\ V - 2\ V = 20mA \cdot R$	$R = 65\Omega \simeq 100\ \Omega$
	$P = 1{,}3V \cdot 20mA$	$P = 0{,}026W$

Existen diferentes tipos de resistencias según el valor de potencia, en éste caso los valores utilizados no superan el cuarto de Wattio.

Además, se han utilizado pulsadores para la interfaz física. Éstos pulsadores emiten pulsos difícil de detectar por el controlador mediante las entradas digitales.

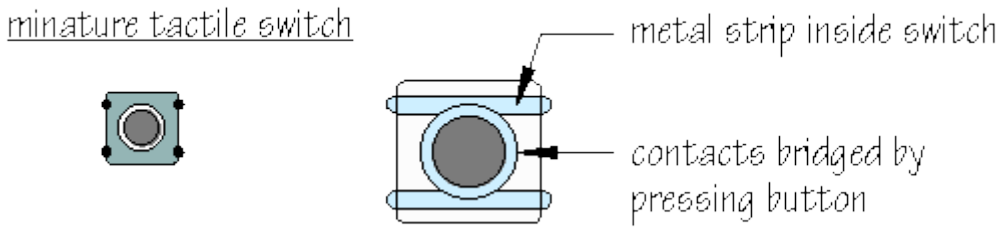


IMAGEN 68. PULSADOR

Ésto provoca que a veces no llegue la señal del pulso al controlador y pueda provocar algún fallo en el sistema. Para tratar de evitar ésta problemática se ha incluido una resistencia de pull-down.

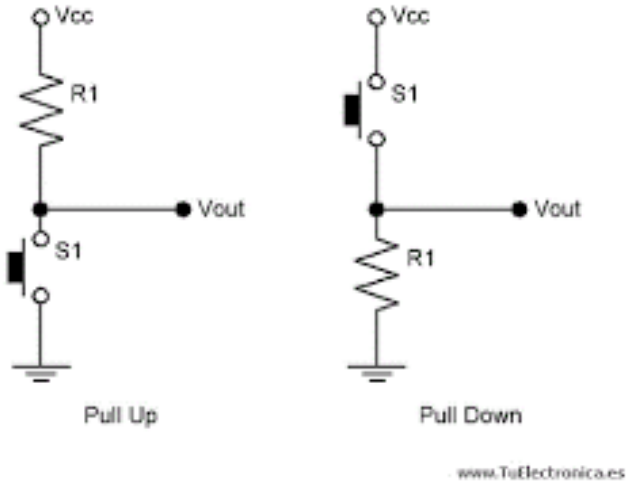


IMAGEN 69. RESISTENCIAS PULL-UP Y PULL-DOWN

Como se ve en el esquema la resistencia pull-down se conecta a tierra (GND), de esta manera cuando el interruptor este abierto la corriente se dirigida hacia la resistencia dejando un valor 0 en Vout y si el interruptor esta cerrado la corriente se moverá hacia Vout dejando un valor lógico HIGH.

$$R = \frac{5V}{20mA} = 250\Omega$$

El valor las resistencias utilizadas es de 10KΩ, que sería lo equivalente a que pasaran 0,5 mA, como medida de protección, para que circule la menor corriente posible, ya que solamente se quiere leer un estado alto o bajo de tensión.

Además, para asegurar que la señal del pulsador llega a la entrada digital, se podría añadir un condensador en paralelo a la resistencia, formando un filtro RC. Para un valor de resistencia de 10KΩ se recomienda utilizar un condensador de 1µF.

Finalmente, el valor de la resistencia utilizada en el control del ventilador, además del transistor es el siguiente:

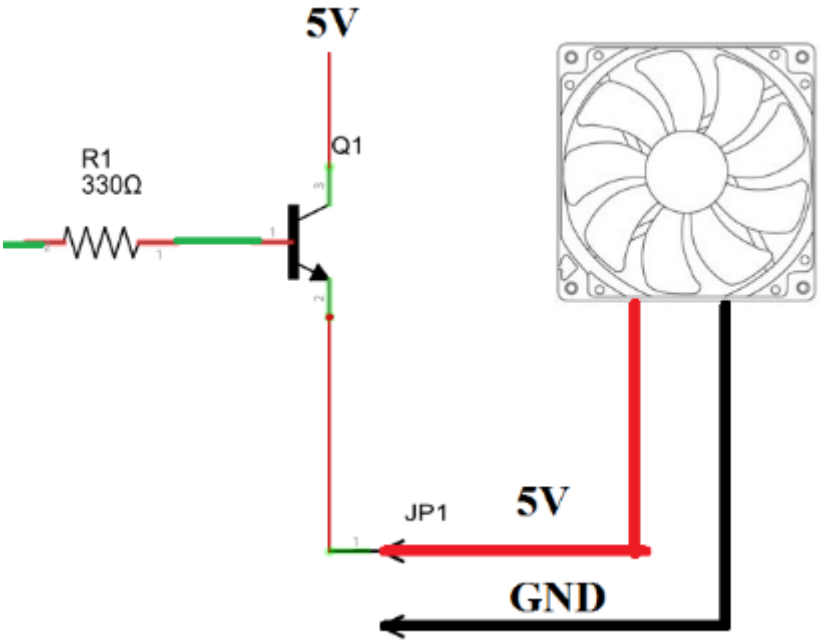


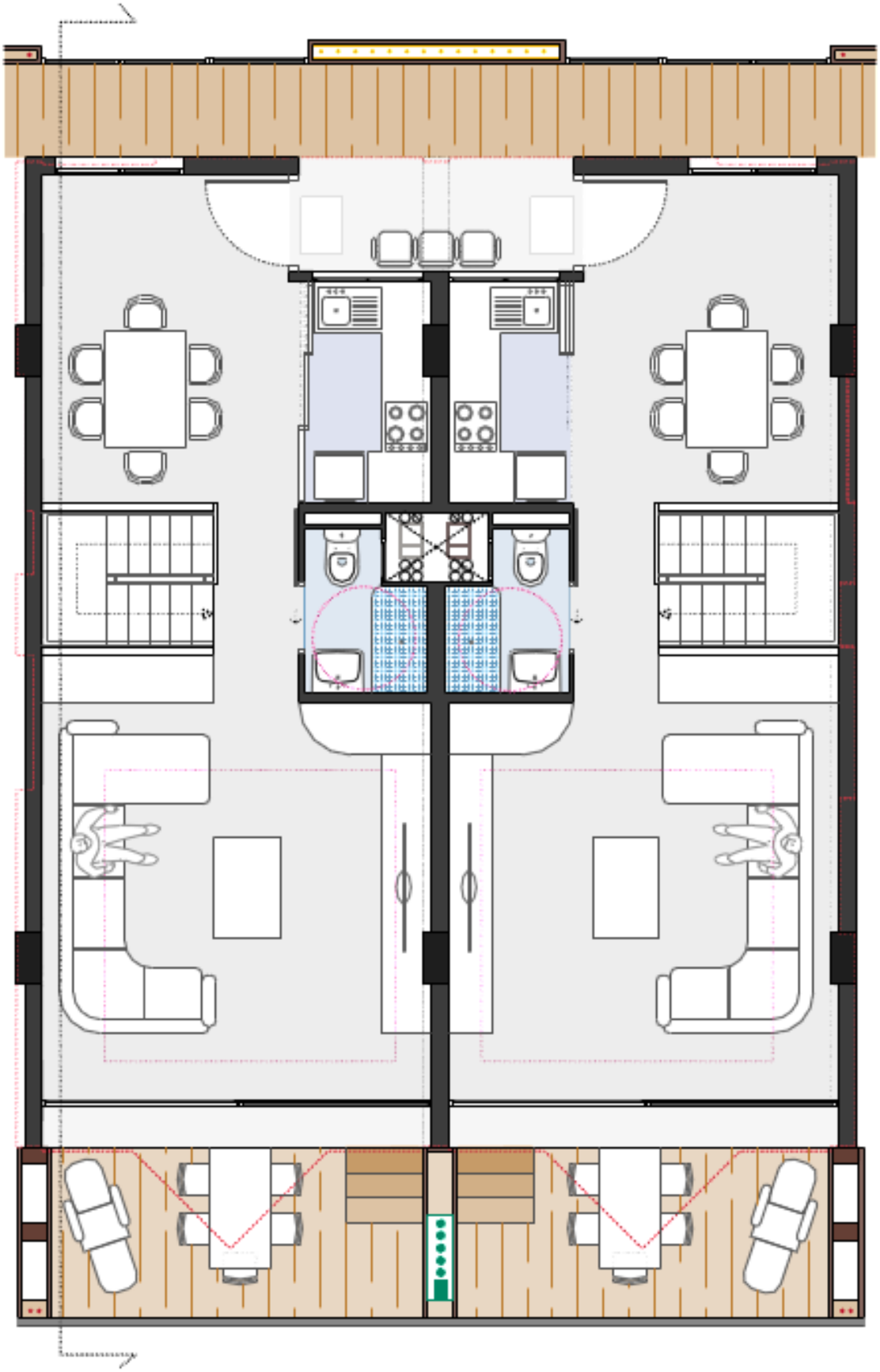
IMAGEN 70. CONEXIÓN DEL VENTILADOR

Siguiendo la misma ley para calcular las resistencias anteriores:

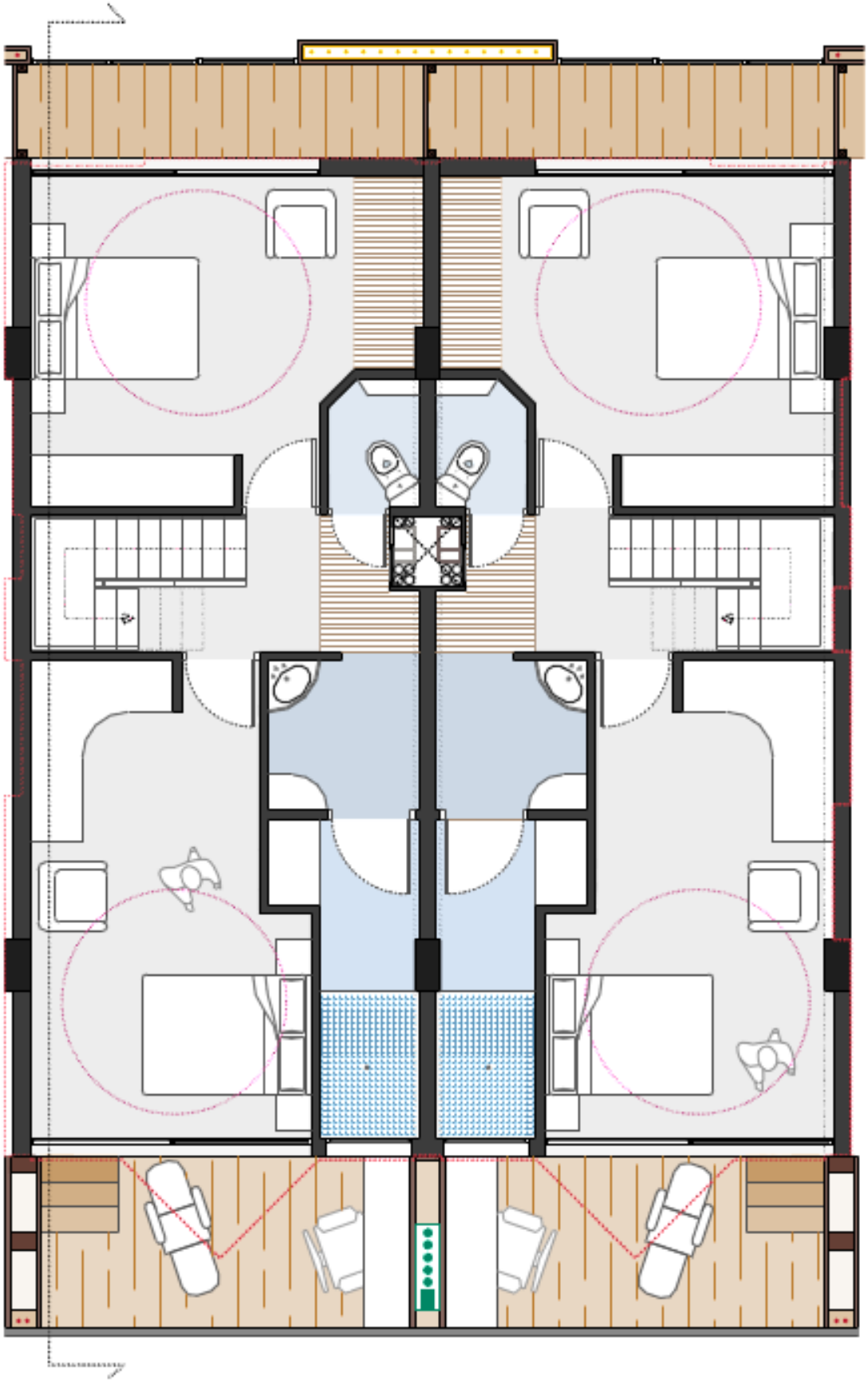
$$R = \frac{5V}{20mA} = 250\Omega$$

El valor de la resistencia utilizada es de 330 Ω, ya que supondría que pasara una corriente de 15mA en un Arduino UNO (5V) y una corriente de 10mA en un Arduino UNO (3,3V), asegurando la protección del sistema.

8.4. Planos de la maqueta



PLANO PLANTA 0



PLANO PLANTA 1

8.5. Obtención de librerías

Las librerías utilizadas en el proyecto se pueden obtener a partir de los siguientes enlaces:

- *due_can*: https://github.com/collin80/due_can
- *CAN_BUS_Shield*: https://github.com/Seeed-Studio/CAN_BUS_Shield
- *DHT*: https://github.com/adafruit/DHT-sensor-library
- *RTClib*: https://github.com/adafruit/RTClib

Las librerías *Ethernet*, *SPI* y *Wire*, ya van incluidas en el propio software de Arduino.

8.6. Datasheets

Los datasheet añadidos al anexo del proyecto, se corresponden a información valiosa para poder entender con más profundidad el bus de transmisión CAN. Por lo que se ha creído conveniente añadir los siguientes datasheets:

CAN Specification 2.0 - Part A : Correspondiente a la información, proporcionada por BOSCH, del protocolo bus CAN, especificamente al formato de mensajes estándar, declarada en la parte A del documento.

SN65HVD230 CAN Transceiver: Datasheet correspondiente al transceptor utilizado para permitir la transmisión de datos mediante bus CAN para Arduino Due.

TJA1050 CAN Transceiver: Datasheet correspondiente al transceptor utilizado para permitir la transmisión de datos mediante bus CAN para Arduino UNO.

BOSCH

CAN Specification

Version 2.0

Imported into Framemaker 4 by:

Chuck Powers, Motorola MCTG Multiplex Applications, April 5,1995.

Recital

The acceptance and introduction of serial communication to more and more applications has led to requirements that the assignment of message identifiers to communication functions be standardized for certain applications. These applications can be realized with CAN more comfortably, if the address range that originally has been defined by 11 identifier bits is enlarged
Therefore a second message format ('extended format') is introduced that provides a larger address range defined by 29 bits. This will relieve the system designer from compromises with respect to defining well-structured naming schemes. Users of CAN who do not need the identifier range offered by the extended format, can rely on the conventional 11 bit identifier range ('standard format') further on. In this case they can make use of the CAN implementations that are already available on the market, or of new controllers that implement both formats.
In order to distinguish standard and extended format the first reserved bit of the CAN message format, as it is defined in CAN Specification 1.2, is used. This is done in such a way that the message format in CAN Specification 1.2 is equivalent to the standard format and therefore is still valid. Furthermore, the extended format has been defined so that messages in standard format and extended format can coexist within the same network.

This CAN Specification consists of two parts, with

- Part A describing the CAN message format as it is defined in CAN Specification 1.2;
- Part B describing both standard and extended message formats.

In order to be compatible with this CAN Specification 2.0 it is required that a CAN implementation be compatible with either Part A or Part B.

Note

CAN implementations that are designed according to part A of this or according to previous CAN Specifications, and CAN implementations that are designed according to part B of this specification can communicate with each other as long as it is not made use of the extended format.

PART A

1 INTRODUCTION

The Controller Area Network (CAN) is a serial communications protocol which efficiently supports distributed realtime control with a very high level of security.
Its domain of application ranges from high speed networks to low cost multiplex wiring. In automotive electronics, engine control units, sensors, anti-skid-systems, etc. are connected using CAN with bitrates up to 1 Mbit/s. At the same time it is cost effective to build into vehicle body electronics, e.g. lamp clusters, electric windows etc. to replace the wiring harness otherwise required.
The intention of this specification is to achieve compatibility between any two CAN implementations. Compatibility, however, has different aspects regarding e.g. electrical features and the interpretation of data to be transferred. To achieve design transparency and implementation flexibility CAN has been subdivided into different layers.

- the (CAN-) object layer
- the (CAN-) transfer layer
- the physical layer

The object layer and the transfer layer comprise all services and functions of the data link layer defined by the ISO/OSI model. The scope of the object layer includes

- finding which messages are to be transmitted
- deciding which messages received by the transfer layer are actually to be used,
- providing an interface to the application layer related hardware.

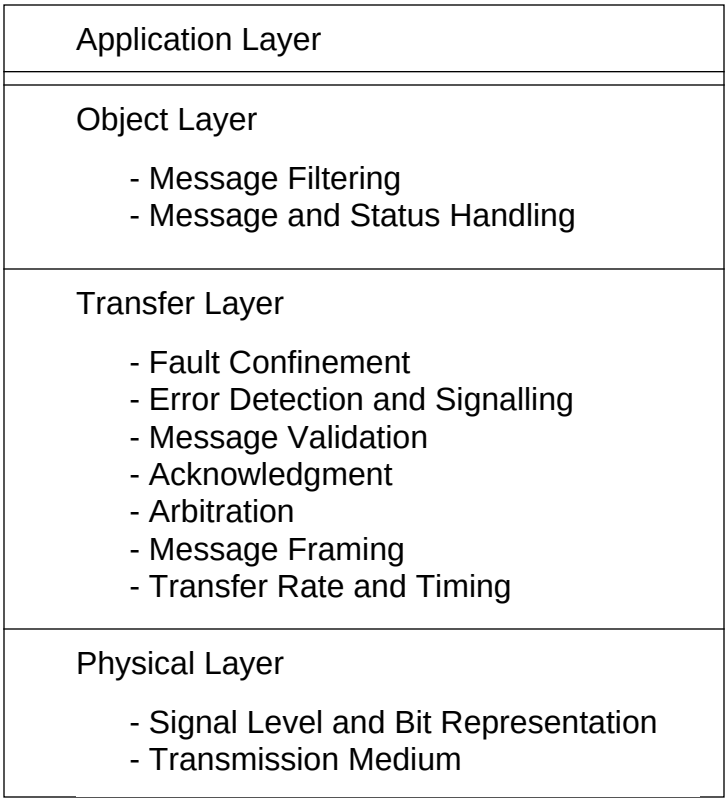
There is much freedom in defining object handling. The scope of the transfer layer mainly is the transfer protocol, i.e. controlling the framing, performing arbitration, error checking, error signalling and fault confinement. Within the transfer layer it is decided whether the bus is free for starting a new transmission or whether a reception is just starting. Also some general features of the bit timing are regarded as part of the transfer layer. It is in the nature of the transfer layer that there is no freedom for modifications.
The scope of the physical layer is the actual transfer of the bits between the different nodes with respect to all electrical properties. Within one network the physical layer, of course, has to be the same for all nodes. There may be, however, much freedom in selecting a physical layer.
The scope of this specification is to define the transfer layer and the consequences of the CAN protocol on the surrounding layers.

2 BASIC CONCEPTS

CAN has the following properties

- prioritization of messages
- guarantee of latency times
- configuration flexibility
- multicast reception with time synchronization
- system wide data consistency
- multimaster
- error detection and signalling
- automatic retransmission of corrupted messages as soon as the bus is idle again
- distinction between temporary errors and permanent failures of nodes and autonomous switching off of defect nodes

Layered Structure of a CAN Node



- The Physical Layer defines how signals are actually transmitted. Within this specification the physical layer is not defined so as to allow transmission medium and signal level implementations to be optimized for their application.
- The Transfer Layer represents the kernel of the CAN protocol. It presents messages received to the object layer and accepts messages to be transmitted from the object layer. The transfer layer is responsible for bit timing and synchronization, message framing, arbitration, acknowledgment, error detection and signalling, and fault confinement.
- The Object Layer is concerned with message filtering as well as status and message handling.

The scope of this specification is to define the transfer layer and the consequences of the CAN protocol on the surrounding layers.

Messages

Information on the bus is sent in fixed format messages of different but limited length (see section 3: Message Transfer). When the bus is free any connected unit may start to transmit a new message.

Information Routing

In CAN systems a CAN node does not make use of any information about the system configuration (e.g. station addresses). This has several important consequences.

System Flexibility: Nodes can be added to the CAN network without requiring any change in the software or hardware of any node and application layer.

Message Routing: The content of a message is named by an IDENTIFIER. The IDENTIFIER does not indicate the destination of the message, but describes the meaning of the data, so that all nodes in the network are able to decide by MESSAGE FILTERING whether the data is to be acted upon by them or not.

Multicast: As a consequence of the concept of MESSAGE FILTERING any number of nodes can receive and simultaneously act upon the same message.

Data Consistency: Within a CAN network it is guaranteed that a message is simultaneously accepted either by all nodes or by no node. Thus data consistency of a system is achieved by the concepts of multicast and by error handling.

Bit rate
The speed of CAN may be different in different systems. However, in a given system the bitrate is uniform and fixed.

Priorities
The IDENTIFIER defines a static message priority during bus access.

Remote Data Request
By sending a REMOTE FRAME a node requiring data may request another node to send the corresponding DATA FRAME. The DATA FRAME and the corresponding REMOTE FRAME are named by the same IDENTIFIER.

Multimaster
When the bus is free any unit may start to transmit a message. The unit with the message of higher priority to be transmitted gains bus access.

Arbitration
Whenever the bus is free, any unit may start to transmit a message. If 2 or more units start transmitting messages at the same time, the bus access conflict is resolved by bitwise arbitration using the IDENTIFIER. The mechanism of arbitration guarantees that neither information nor time is lost. If a DATA FRAME and a REMOTE FRAME with the same IDENTIFIER are initiated at the same time, the DATA FRAME prevails over the REMOTE FRAME. During arbitration every transmitter compares the level of the bit transmitted with the level that is monitored on the bus. If these levels are equal the unit may continue to send. When a ’recessive’ level is sent and a ’dominant’ level is monitored (see Bus Values), the unit has lost arbitration and must withdraw without sending one more bit.

Safety
In order to achieve the utmost safety of data transfer, powerful measures for error detection, signalling and self-checking are implemented in every CAN node.

Error Detection
For detecting errors the following measures have been taken:
- Monitoring (transmitters compare the bit levels to be transmitted with the bit levels detected on the bus)
- Cyclic Redundancy Check
- Bit Stuffing
- Message Frame Check

Performance of Error Detection
The error detection mechanisms have the following properties:

- all global errors are detected.
- all local errors at transmitters are detected.
- up to 5 randomly distributed errors in a message are detected.
- burst errors of length less than 15 in a message are detected.
- errors of any odd number in a message are detected.

Total residual error probability for undetected corrupted messages: less than

$$\text{message error rate} * 4.7 * 10^{-11}.$$

Error Signalling and Recovery Time
Corrupted messages are flagged by any node detecting an error. Such messages are aborted and will be retransmitted automatically. The recovery time from detecting an error until the start of the next message is at most 29 bit times, if there is no further error.

Fault Confinement
CAN nodes are able to distinguish short disturbances from permanent failures. Defective nodes are switched off.

Connections
The CAN serial communication link is a bus to which a number of units may be connected. This number has no theoretical limit. Practically the total number of units will be limited by delay times and/or electrical loads on the bus line.

Single Channel
The bus consists of a single channel that carries bits. From this data resynchronization information can be derived. The way in which this channel is implemented is not fixed in this specification. E.g. single wire (plus ground), two differential wires, optical fibres, etc.

Bus values
The bus can have one of two complementary logical values: ’dominant’ or ’recessive’. During simultaneous transmission of ’dominant’ and ’recessive’ bits, the resulting bus value will be ’dominant’. For example, in case of a wired-AND implementation of the bus, the ’dominant’ level would be represented by a logical ’0’ and the ’recessive’ level by a logical ’1’. Physical states (e.g. electrical voltage, light) that represent the logical levels are not given in this specification.

Acknowledgment

All receivers check the consistency of the message being received and will acknowledge a consistent message and flag an inconsistent message.

Sleep Mode / Wake-up

To reduce the system's power consumption, a CAN-device may be set into sleep mode without any internal activity and with disconnected bus drivers. The sleep mode is finished with a wake-up by any bus activity or by internal conditions of the system. On wake-up, the internal activity is restarted, although the transfer layer will be waiting for the system's oscillator to stabilize and it will then wait until it has synchronized itself to the bus activity (by checking for eleven consecutive 'recessive' bits), before the bus drivers are set to "on-bus" again.

In order to wake up other nodes of the system, which are in sleep-mode, a special wake-up message with the dedicated, lowest possible IDENTIFIER (rrr rrrd rrrr; r = 'recessive' d = 'dominant') may be used.

3 MESSAGE TRANSFER

3.1 Frame Types

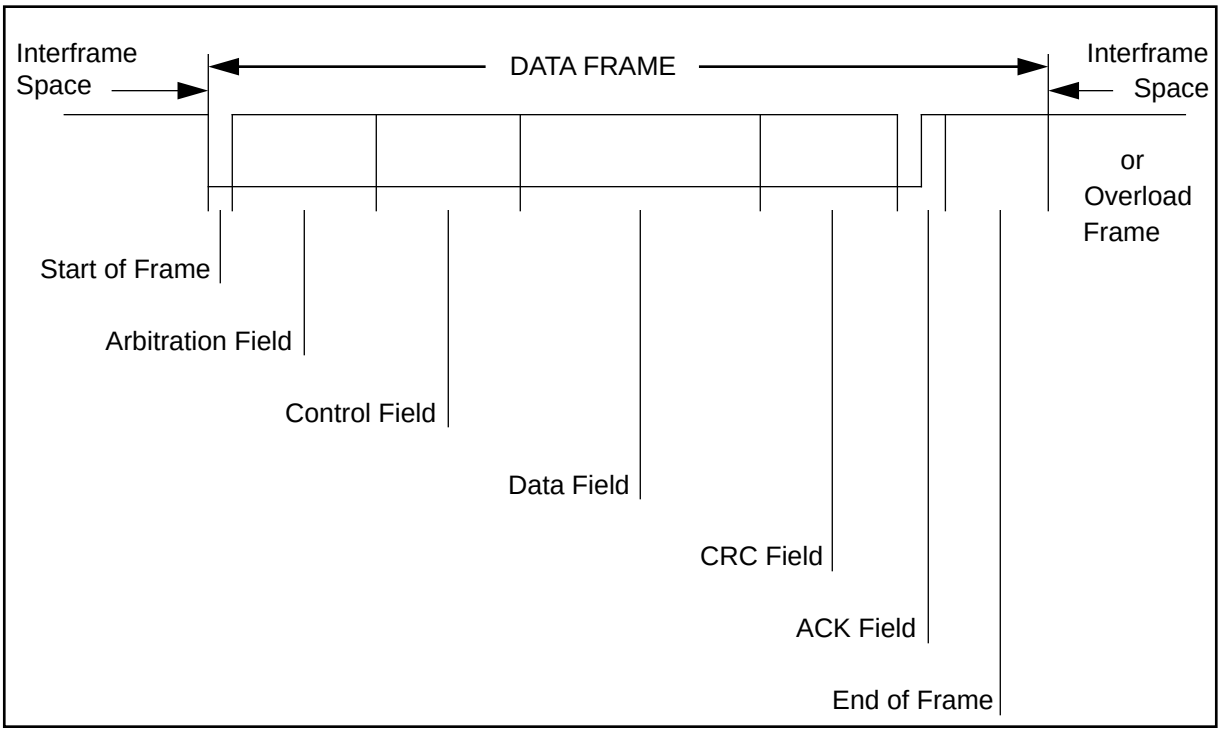
Message transfer is manifested and controlled by four different frame types:

A DATA FRAME carries data from a transmitter to the receivers.
A REMOTE FRAME is transmitted by a bus unit to request the transmission of the DATA FRAME with the same IDENTIFIER.
An ERROR FRAME is transmitted by any unit on detecting a bus error.
An OVERLOAD FRAME is used to provide for an extra delay between the preceding and the succeeding DATA or REMOTE FRAMEs.

DATA FRAMEs and REMOTE FRAMEs are separated from preceding frames by an INTERFRAME SPACE.

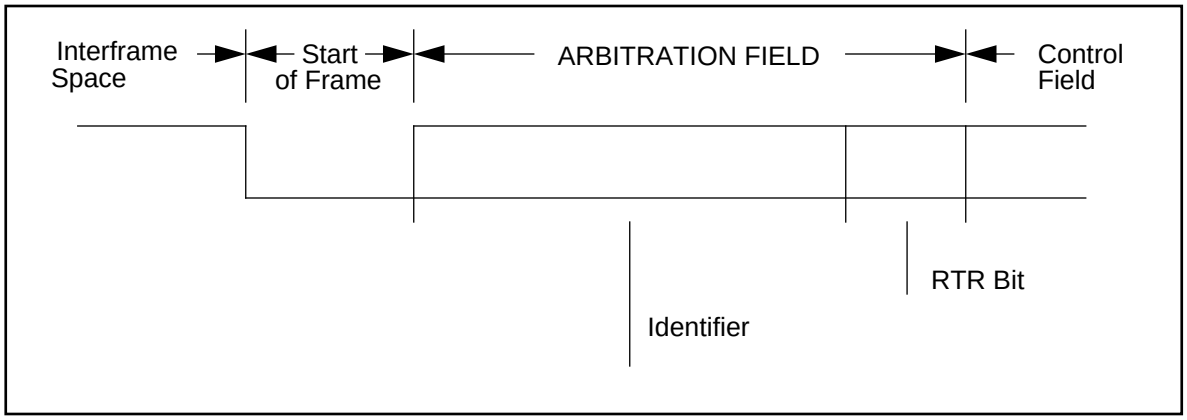
3.1.1 DATA FRAME

A DATA FRAME is composed of seven different bit fields:
START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, DATA FIELD, CRC FIELD, ACK FIELD, END OF FRAME. The DATA FIELD can be of length zero.



START OF FRAME
marks the beginning of DATA FRAMES and REMOTE FRAMEs. It consists of a single 'dominant' bit.
A station is only allowed to start transmission when the bus is idle (see BUS IDLE). All stations have to synchronize to the leading edge caused by START OF FRAME (see 'HARD SYNCHRONIZATION') of the station starting transmission first.

ARBITRATION FIELD
The ARBITRATION FIELD consists of the IDENTIFIER and the RTR-BIT.

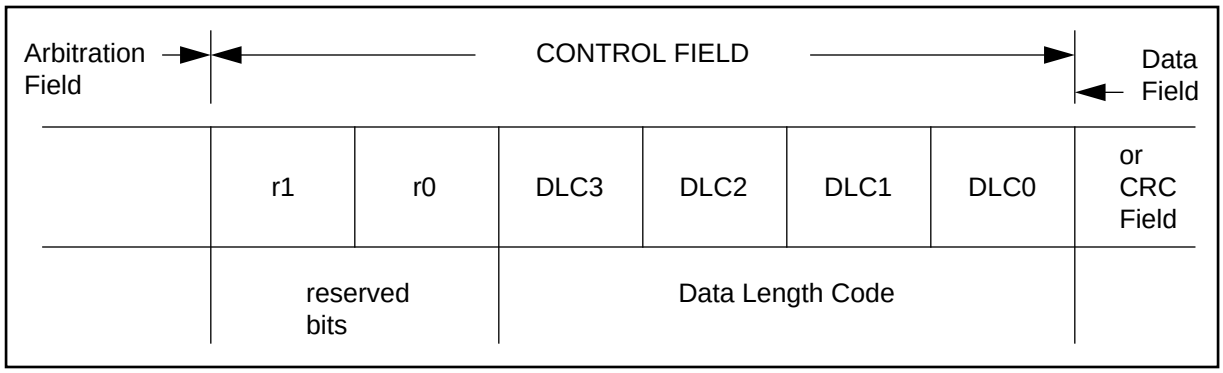


IDENTIFIER
The IDENTIFIER's length is 11 bits. These bits are transmitted in the order from ID-10 to ID-0. The least significant bit is ID-0. The 7 most significant bits (ID-10 - ID-4) must not be all 'recessive'.

RTR BIT
Remote Transmission Request BIT
In DATA FRAMEs the RTR BIT has to be 'dominant'. Within a REMOTE FRAME the RTR BIT has to be 'recessive'.

CONTROL FIELD
The CONTROL FIELD consists of six bits. It includes the DATA LENGTH CODE and two bits reserved for future expansion. The reserved bits have to be sent 'dominant'. Receivers accept 'dominant' and 'recessive' bits in all combinations.

DATA LENGTH CODE
The number of bytes in the DATA FIELD is indicated by the DATA LENGTH CODE. This DATA LENGTH CODE is 4 bits wide and is transmitted within the CONTROL FIELD.



Coding of the number of data bytes by the DATA LENGTH CODE

abbreviations: d 'dominant'
r 'recessive'

Number of Data Bytes	Data Length Code			
	DLC3	DLC2	DLC1	DLC0
0	d	d	d	d
1	d	d	d	r
2	d	d	r	d
3	d	d	r	r
4	d	r	d	d
5	d	r	d	r
6	d	r	r	d
7	d	r	r	r
8	r	d	d	d

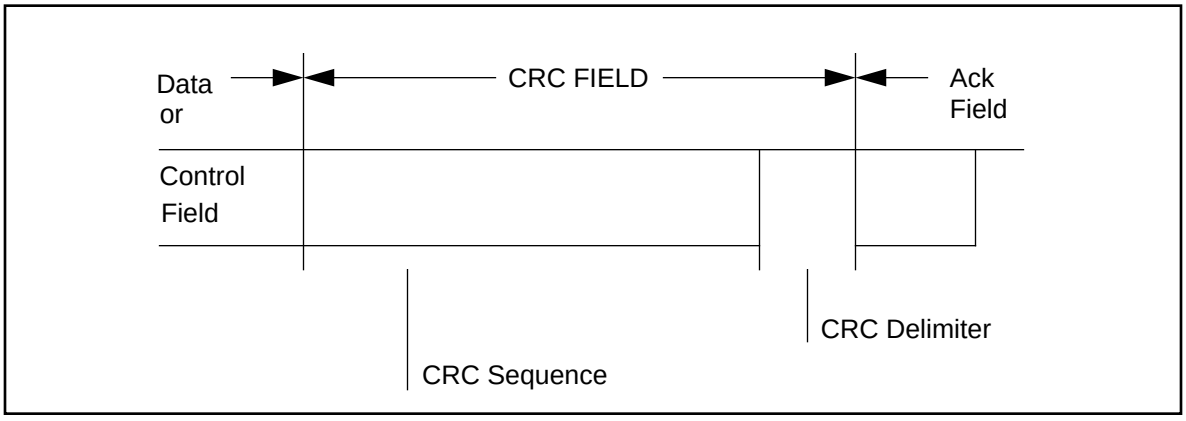
DATA FRAME: admissible numbers of data bytes: {0,1,....,7,8}.
Other values may not be used.

DATA FIELD

The DATA FIELD consists of the data to be transferred within a DATA FRAME. It can contain from 0 to 8 bytes, which each contain 8 bits which are transferred MSB first.

CRC FIELD

contains the CRC SEQUENCE followed by a CRC DELIMITER.



CRC SEQUENCE

The frame check sequence is derived from a cyclic redundancy code best suited for frames with bit counts less than 127 bits (BCH Code).

In order to carry out the CRC calculation the polynomial to be divided is defined as the polynomial, the coefficients of which are given by the destuffed bit stream consisting of START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, DATA FIELD (if present) and, for the 15 lowest coefficients, by 0. This polynomial is divided (the coefficients are calculated modulo-2) by the generator-polynomial:

$$X^{15} + X^{14} + X^{10} + X^8 + X^7 + X^4 + X^3 + 1.$$

The remainder of this polynomial division is the CRC SEQUENCE transmitted over the bus. In order to implement this function, a 15 bit shift register CRC_RG(14:0) can be used. If NXTBIT denotes the next bit of the bit stream, given by the destuffed bit sequence from START OF FRAME until the end of the DATA FIELD, the CRC SEQUENCE is calculated as follows:

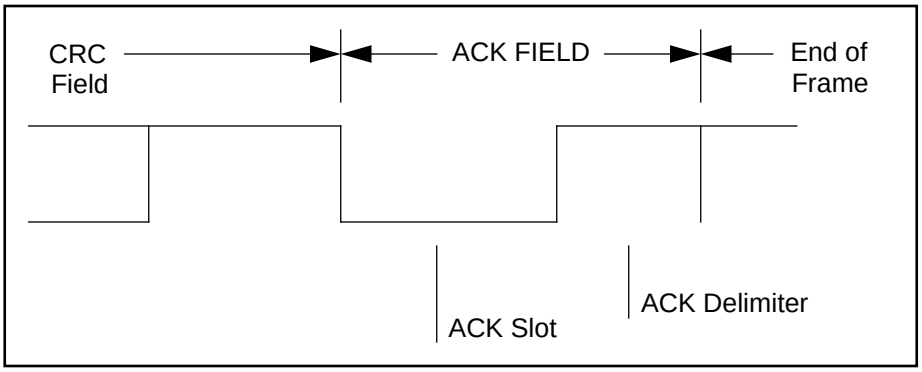
```
CRC_RG = 0;                                   // initialize shift register
REPEAT
        CRCNXT = NXTBIT EXOR CRC_RG(14);
        CRC_RG(14:1) = CRC_RG(13:0);          // shift left by
        CRC_RG(0) = 0;                        // 1 position
```

```
        IF CRCNXT THEN
                CRC_RG(14:0) = CRC_RG(14:0) EXOR (4599hex);
        ENDIF
UNTIL (CRC SEQUENCE starts or there is an ERROR condition)
```

After the transmission / reception of the last bit of the DATA FIELD, CRC_RG contains the CRC sequence.

CRC DELIMITER
The CRC SEQUENCE is followed by the CRC DELIMITER which consists of a single 'recessive' bit.

ACK FIELD
The ACK FIELD is two bits long and contains the ACK SLOT and the ACK DELIMITER. In the ACK FIELD the transmitting station sends two 'recessive' bits.
A RECEIVER which has received a valid message correctly, reports this to the TRANSMITTER by sending a 'dominant' bit during the ACK SLOT (it sends 'ACK').



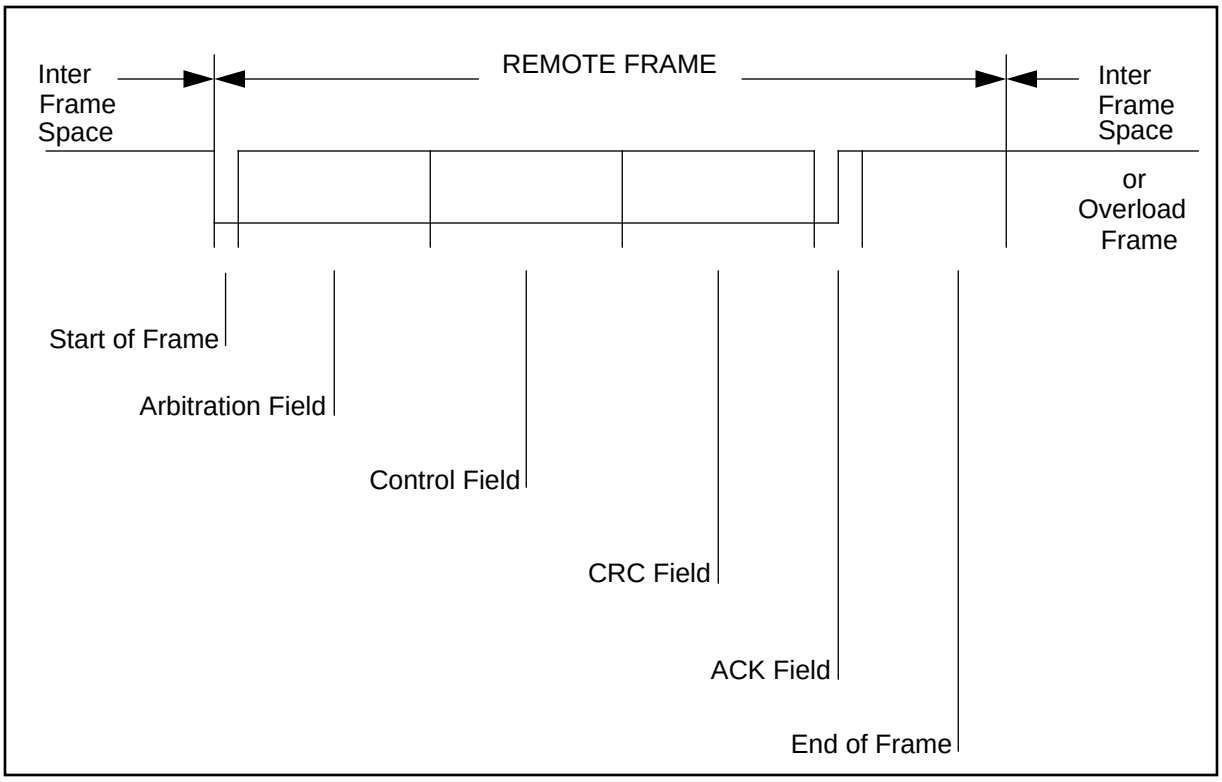
ACK SLOT
All stations having received the matching CRC SEQUENCE report this within the ACK SLOT by superscribing the 'recessive' bit of the TRANSMITTER by a 'dominant' bit.

ACK DELIMITER
The ACK DELIMITER is the second bit of the ACK FIELD and has to be a 'recessive' bit. As a consequence, the ACK SLOT is surrounded by two 'recessive' bits (CRC DELIMITER, ACK DELIMITER).

END OF FRAME
Each DATA FRAME and REMOTE FRAME is delimited by a flag sequence consisting of seven 'recessive' bits.

3.1.2 REMOTE FRAME

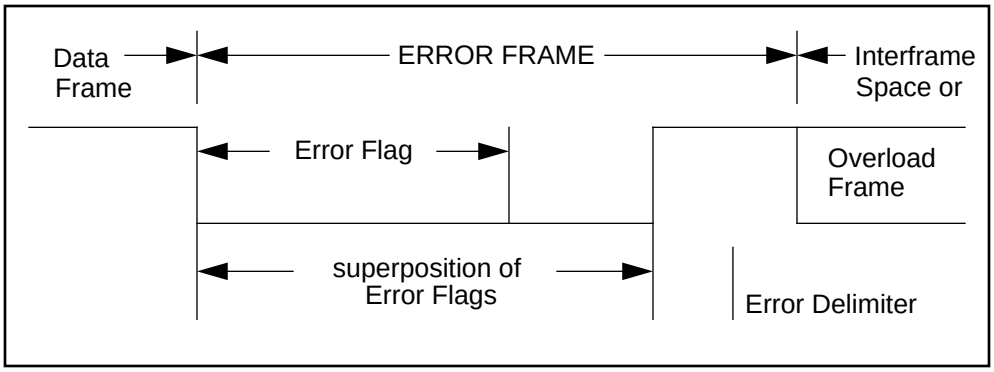
A station acting as a RECEIVER for certain data can initiate the transmission of the respective data by its source node by sending a REMOTE FRAME.
A REMOTE FRAME is composed of six different bit fields:
START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, CRC FIELD, ACK FIELD, END OF FRAME.
Contrary to DATA FRAMEs, the RTR bit of REMOTE FRAMEs is 'recessive'. There is no DATA FIELD, independent of the values of the DATA LENGTH CODE which may be signed any value within the admissible range 0...8. The value is the DATA LENGTH CODE of the corresponding DATA FRAME.



The polarity of the RTR bit indicates whether a transmitted frame is a DATA FRAME (RTR bit 'dominant') or a REMOTE FRAME (RTR bit 'recessive').

3.1.3 ERROR FRAME

The ERROR FRAME consists of two different fields. The first field is given by the superposition of ERROR FLAGs contributed from different stations. The following second field is the ERROR DELIMITER.



In order to terminate an ERROR FRAME correctly, an 'error passive' node may need the bus to be 'bus idle' for at least 3 bit times (if there is a local error at an 'error passive' receiver). Therefore the bus should not be loaded to 100%.

ERROR FLAG
There are 2 forms of an ERROR FLAG: an ACTIVE ERROR FLAG and a PASSIVE ERROR FLAG.

1. The ACTIVE ERROR FLAG consists of six consecutive 'dominant' bits.

2. The PASSIVE ERROR FLAG consists of six consecutive 'recessive' bits unless it is overwritten by 'dominant' bits from other nodes.

An 'error active' station detecting an error condition signals this by transmission of an ACTIVE ERROR FLAG. The ERROR FLAG's form violates the law of bit stuffing (see CODING) applied to all fields from START OF FRAME to CRC DELIMITER or destroys the fixed form ACK FIELD or END OF FRAME field. As a consequence, all other stations detect an error condition and on their part start transmission of an ERROR FLAG. So the sequence of 'dominant' bits which actually can be monitored on the bus results from a superposition of different ERROR FLAGs transmitted by individual stations. The total length of this sequence varies between a minimum of six and a maximum of twelve bits.
An 'error passive' station detecting an error condition tries to signal this by transmission of a PASSIVE ERROR FLAG. The 'error passive' station waits for six consecutive bits

of equal polarity, beginning at the start of the PASSIVE ERROR FLAG. The PASSIVE ERROR FLAG is complete when these 6 equal bits have been detected.

ERROR DELIMITER
The ERROR DELIMITER consists of eight 'recessive' bits.
After transmission of an ERROR FLAG each station sends 'recessive' bits and monitors the bus until it detects a 'recessive' bit. Afterwards it starts transmitting seven more 'recessive' bits.

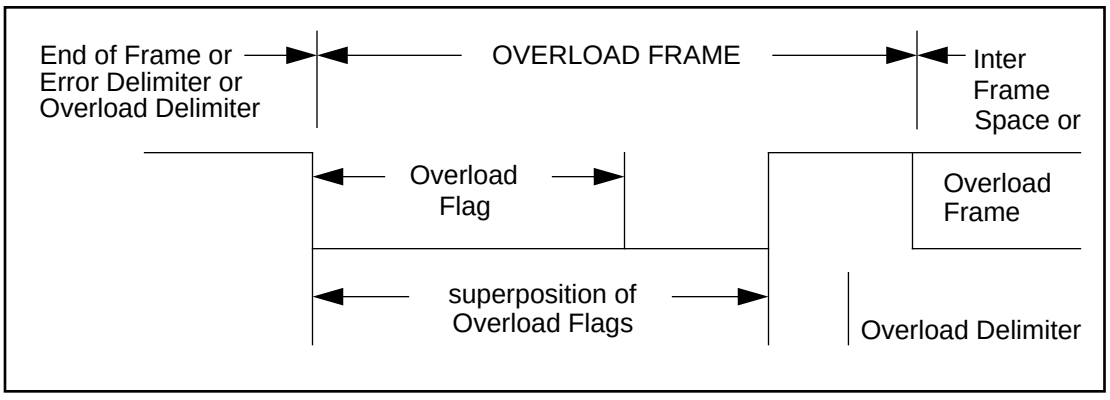
3.1.4 OVERLOAD FRAME

The OVERLOAD FRAME contains the two bit fields OVERLOAD FLAG and OVERLOAD DELIMITER.
There are two kinds of OVERLOAD conditions, which both lead to the transmission of an OVERLOAD FLAG:

1. The internal conditions of a receiver, which requires a delay of the next DATA FRAME or REMOTE FRAME.

2. Detection of a 'dominant' bit during INTERMISSION.

The start of an OVERLOAD FRAME due to OVERLOAD condition 1 is only allowed to be started at the first bit time of an expected INTERMISSION, whereas OVERLOAD FRAMEs due to OVERLOAD condition 2 start one bit after detecting the 'dominant' bit.



At most two OVERLOAD FRAMEs may be generated to delay the next DATA or REMOTE FRAME.

OVERLOAD FLAG
consists of six 'dominant' bits. The overall form corresponds to that of the ACTIVE ERROR FLAG.

The OVERLOAD FLAG's form destroys the fixed form of the INTERMISSION field. As a consequence, all other stations also detect an OVERLOAD condition and on their part start transmission of an OVERLOAD FLAG. (In case that there is a 'dominant' bit detected during the 3rd bit of INTERMISSION locally at some node, the other nodes will not interpret the OVERLOAD FLAG correctly, but interpret the first of these six 'dominant' bits as START OF FRAME. The sixth 'dominant' bit violates the rule of bit stuffing causing an error condition).

OVERLOAD DELIMITER
consists of eight 'recessive' bits.

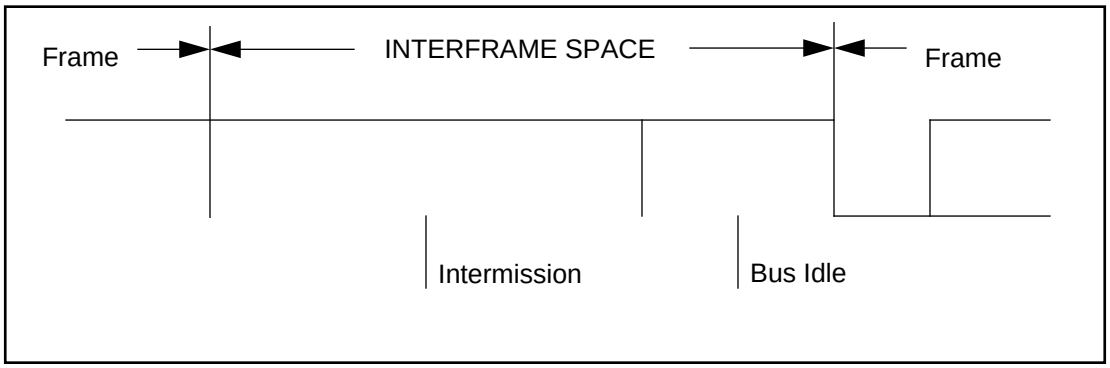
The OVERLOAD DELIMITER is of the same form as the ERROR DELIMITER. After transmission of an OVERLOAD FLAG the station monitors the bus until it detects a transition from a 'dominant' to a 'recessive' bit. At this point of time every bus station has finished sending its OVERLOAD FLAG and all stations start transmission of seven more 'recessive' bits in coincidence.

3.1.5 INTERFRAME SPACING

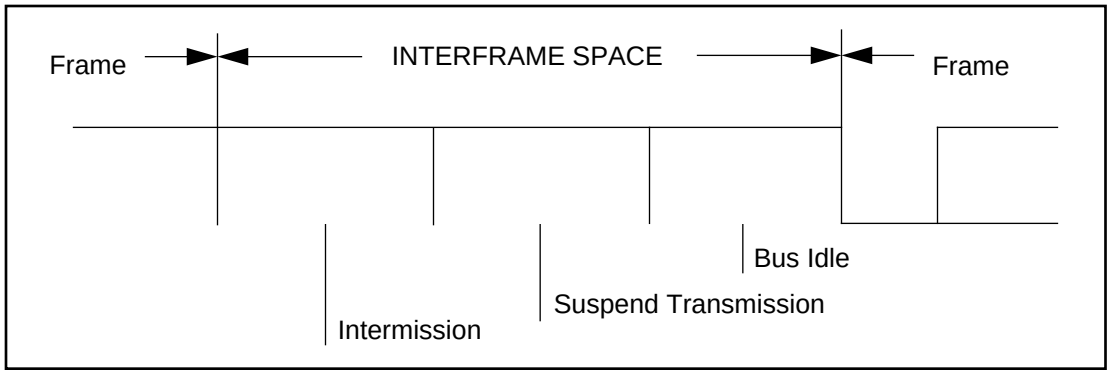
DATA FRAMEs and REMOTE FRAMEs are separated from preceding frames whatever type they are (DATA FRAME, REMOTE FRAME, ERROR FRAME, OVERLOAD FRAME) by a bit field called INTERFRAME SPACE. In contrast, OVERLOAD FRAMEs and ERROR FRAMEs are not preceded by an INTERFRAME SPACE and multiple OVERLOAD FRAMEs are not separated by an INTERFRAME SPACE.

INTERFRAME SPACE
contains the bit fields INTERMISSION and BUS IDLE and, for 'error passive' stations, which have been TRANSMITTER of the previous message, SUSPEND TRANSMISSION.

For stations which are not 'error passive' or have been RECEIVER of the previous message:



For 'error passive' stations which have been TRANSMITTER of the previous message:



INTERMISSION
consists of three 'recessive' bits.

During INTERMISSION no station is allowed to start transmission of a DATA FRAME or REMOTE FRAME. The only action to be taken is signalling an OVERLOAD condition.

BUS IDLE
The period of BUS IDLE may be of arbitrary length. The bus is recognized to be free and any station having something to transmit can access the bus. A message, which is pending for transmission during the transmission of another message, is started in the first bit following INTERMISSION.
The detection of a 'dominant' bit on the bus is interpreted as a START OF FRAME.

SUSPEND TRANSMISSION
After an 'error passive' station has transmitted a message, it sends eight 'recessive' bits following INTERMISSION, before starting to transmit a further message or recognizing the bus to be idle. If meanwhile a transmission (caused by another station) starts, the station will become receiver of this message.

3.2 Definition of TRANSMITTER / RECEIVER

TRANSMITTER
A unit originating a message is called “TRANSMITTER” of that message. The unit stays TRANSMITTER until the bus is idle or the unit loses ARBITRATION.

RECEIVER
A unit is called “RECEIVER” of a message, if it is not TRANSMITTER of that message and the bus is not idle.

4 MESSAGE VALIDATION

The point of time at which a message is taken to be valid, is different for the transmitter and the receivers of the message.

Transmitter:
The message is valid for the transmitter, if there is no error until the end of END OF FRAME. If a message is corrupted, retransmission will follow automatically and according to prioritization. In order to be able to compete for bus access with other messages, retransmission has to start as soon as the bus is idle.

Receivers:
The message is valid for the receivers, if there is no error until the last but one bit of END OF FRAME.

5 CODING

BIT STREAM CODING

The frame segments START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, DATA FIELD and CRC SEQUENCE are coded by the method of bit stuffing. Whenever a transmitter detects five consecutive bits of identical value in the bit stream to be transmitted it automatically inserts a complementary bit in the actual transmitted bit stream.
The remaining bit fields of the DATA FRAME or REMOTE FRAME (CRC DELIMITER, ACK FIELD, and END OF FRAME) are of fixed form and not stuffed. The ERROR FRAME and the OVERLOAD FRAME are of fixed form as well and not coded by the method of bit stuffing.
The bit stream in a message is coded according to the Non-Return-to-Zero (NRZ) method. This means that during the total bit time the generated bit level is either ’dominant’ or ’recessive’.

6 ERROR HANDLING

6.1 Error Detection

There are 5 different error types (which are not mutually exclusive):

- BIT ERROR
  A unit that is sending a bit on the bus also monitors the bus. A BIT ERROR has to be detected at that bit time, when the bit value that is monitored is different from the bit value that is sent. An exception is the sending of a 'recessive' bit during the stuffed bit stream of the ARBITRATION FIELD or during the ACK SLOT. Then no BIT ERROR occurs when a 'dominant' bit is monitored. A TRANSMITTER sending a PASSIVE ERROR FLAG and detecting a 'dominant' bit does not interpret this as a BIT ERROR.
- STUFF ERROR
  A STUFF ERROR has to be detected at the bit time of the 6th consecutive equal bit level in a message field that should be coded by the method of bit stuffing.
- CRC ERROR
  The CRC sequence consists of the result of the CRC calculation by the transmitter. The receivers calculate the CRC in the same way as the transmitter. A CRC ERROR has to be detected, if the calculated result is not the same as that received in the CRC sequence.
- FORM ERROR
  A FORM ERROR has to be detected when a fixed-form bit field contains one or more illegal bits.
- ACKNOWLEDGMENT ERROR
  An ACKNOWLEDGMENT ERROR has to be detected by a transmitter whenever it does not monitor a 'dominant' bit during the ACK SLOT.

6.2 Error Signalling

A station detecting an error condition signals this by transmitting an ERROR FLAG. For an 'error active' node it is an ACTIVE ERROR FLAG, for an 'error passive' node it is a PASSIVE ERROR FLAG. Whenever a BIT ERROR, a STUFF ERROR, a FORM ERROR or an ACKNOWLEDGMENT ERROR is detected by any station, transmission of an ERROR FLAG is started at the respective station at the next bit.
Whenever a CRC ERROR is detected, transmission of an ERROR FLAG starts at the bit following the ACK DELIMITER, unless an ERROR FLAG for another condition has already been started.

7 FAULT CONFINEMENT

With respect to fault confinement a unit may be in one of three states:

- 'error active'
- 'error passive'
- 'bus off'

An 'error active' unit can normally take part in bus communication and sends an ACTIVE ERROR FLAG when an error has been detected.
An 'error passive' unit must not send an ACTIVE ERROR FLAG. It takes part in bus communication but when an error has been detected only a PASSIVE ERROR FLAG is sent. Also after a transmission, an 'error passive' unit will wait before initiating a further transmission. (See SUSPEND TRANSMISSION)
A 'bus off' unit is not allowed to have any influence on the bus. (E.g. output drivers switched off.)
For fault confinement two counts are implemented in every bus unit:

1) TRANSMIT ERROR COUNT
2) RECEIVE ERROR COUNT

These counts are modified according to the following rules:
(note that more than one rule may apply during a given message transfer)

1. When a RECEIVER detects an error, the RECEIVE ERROR COUNT will be increased by 1, except when the detected error was a BIT ERROR during the sending of an ACTIVE ERROR FLAG or an OVERLOAD FLAG.

2. When a RECEIVER detects a 'dominant' bit as the first bit after sending an ERROR FLAG the RECEIVE ERROR COUNT will be increased by 8.

3. When a TRANSMITTER sends an ERROR FLAG the TRANSMIT ERROR COUNT is increased by 8.

   Exception 1:
   If the TRANSMITTER is 'error passive' and detects an ACKNOWLEDGMENT

ERROR because of not detecting a 'dominant' ACK and does not detect a 'dominant' bit while sending its PASSIVE ERROR FLAG.

Exception 2:
If the TRANSMITTER sends an ERROR FLAG because a STUFF ERROR occurred during ARBITRATION whereby the STUFFBIT is located before the RTR bit, and should have been 'recessive', and has been sent as 'recessive' but monitored as 'dominant'.

In exceptions 1 and 2 the TRANSMIT ERROR COUNT is not changed.

4. If an TRANSMITTER detects a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG the TRANSMIT ERROR COUNT is increased by 8.

5. If an RECEIVER detects a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG the RECEIVE ERROR COUNT is increased by 8.

6. Any node tolerates up to 7 consecutive 'dominant' bits after sending an ACTIVE ERROR FLAG, PASSIVE ERROR FLAG or OVERLOAD FLAG. After detecting the 14th consecutive 'dominant' bit (in case of an ACTIVE ERROR FLAG or an OVERLOAD FLAG) or after detecting the 8th consecutive 'dominant' bit following a PASSIVE ERROR FLAG, and after each sequence of additional eight consecutive 'dominant' bits every TRANSMITTER increases its TRANSMIT ERROR COUNT by 8 and every RECEIVER increases its RECEIVE ERROR COUNT by 8.

7. After the successful transmission of a message (getting ACK and no error until END OF FRAME is finished) the TRANSMIT ERROR COUNT is decreased by 1 unless it was already 0.

8. After the successful reception of a message (reception without error up to the ACK SLOT and the successful sending of the ACK bit), the RECEIVE ERROR COUNT is decreased by 1, if it was between 1 and 127. If the RECEIVE ERROR COUNT was 0, it stays 0, and if it was greater than 127, then it will be set to a value between 119 and 127.

9. A node is 'error passive' when the TRANSMIT ERROR COUNT equals or exceeds 128, or when the RECEIVE ERROR COUNT equals or exceeds 128. An error condition letting a node become 'error passive' causes the node to send an ACTIVE ERROR FLAG.

10. A node is ’bus off’ when the TRANSMIT ERROR COUNT is greater than or equal to 256.

11. An ’error passive’ node becomes ’error active’ again when both the TRANSMIT ERROR COUNT and the RECEIVE ERROR COUNT are less than or equal to 127.

12. An node which is ’bus off’ is permitted to become ’error active’ (no longer ’bus off’) with its error counters both set to 0 after 128 occurrence of 11 consecutive ’recessive’ bits have been monitored on the bus.

Note:
An error count value greater than about 96 indicates a heavily disturbed bus. It may be of advantage to provide means to test for this condition.

Note:
Start-up / Wake-up:
If during start-up only 1 node is online, and if this node transmits some message, it will get no acknowledgment, detect an error and repeat the message. It can become ’error passive’ but not ’bus off’ due to this reason.

8 BIT TIMING REQUIREMENTS

NOMINAL BIT RATE

The Nominal Bit Rate is the number of bits per second transmitted in the absence of resynchronization by an ideal transmitter.

NOMINAL BIT TIME

NOMINAL BIT TIME = 1 / NOMINAL BIT RATE

The Nominal Bit Time can be thought of as being divided into separate non-overlapping time segments. These segments

- SYNCHRONIZATION SEGMENT (SYNC_SEG)
- PROPAGATION TIME SEGMENT (PROP_SEG)
- PHASE BUFFER SEGMENT1 (PHASE_SEG1)
- PHASE BUFFER SEGMENT2 (PHASE_SEG2)

form the bit time as shown in figure 1.

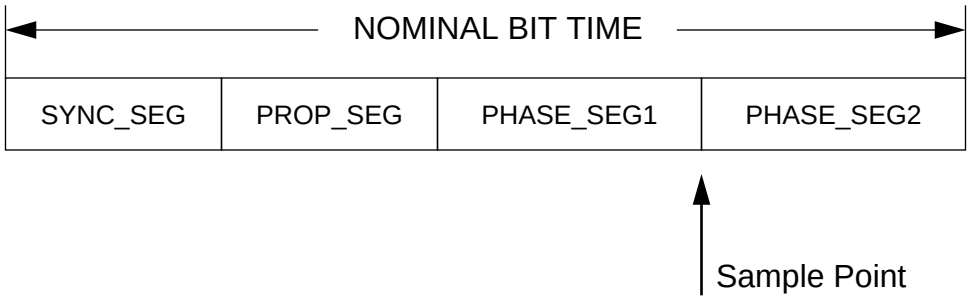


Fig. 1 Partition of the Bit Time

SYNC SEG

This part of the bit time is used to synchronize the various nodes on the bus. An edge is expected to lie within this segment.

PROP SEG

This part of the bit time is used to compensate for the physical delay times within the network.

It is twice the sum of the signal's propagation time on the bus line, the input comparator delay, and the output driver delay.

PHASE SEG1, PHASE SEG2
These Phase-Buffer-Segments are used to compensate for edge phase errors. These segments can be lengthened or shortened by resynchronization.

SAMPLE POINT
The SAMPLE POINT is the point of time at which the bus level is read and interpreted as the value of that respective bit. It's location is at the end of PHASE_SEG1.

INFORMATION PROCESSING TIME
The INFORMATION PROCESSING TIME is the time segment starting with the SAMPLE POINT reserved for calculation the subsequent bit level.

TIME QUANTUM
The TIME QUANTUM is a fixed unit of time derived from the oscillator period. There exists a programmable prescaler, with integral values, ranging at least from 1 to 32. Starting with the MINIMUM TIME QUANTUM, the TIME QUANTUM can have a length of

TIME QUANTUM = m * MINIMUM TIME QUANTUM

with m the value of the prescaler.

Length of Time Segments

- SYNC_SEG is 1 TIME QUANTUM long.
- PROP_SEG is programmable to be 1,2,...,8 TIME QUANTA long.
- PHASE_SEG1 is programmable to be 1,2,...,8 TIME QUANTA long.
- PHASE_SEG2 is the maximum of PHASE_SEG1 and the INFORMATION PROCESSING TIME
- The INFORMATION PROCESSING TIME is less than or equal to 2 TIME QUANTA long.

The total number of TIME QUANTA in a bit time has to be programmable at least from 8 to 25.

SYNCHRONIZATION

HARD SYNCHRONIZATION
After a HARD SYNCHRONIZATION the internal bit time is restarted with SYNC_SEG. Thus HARD SYNCHRONIZATION forces the edge which has caused the HARD SYNCHRONIZATION to lie within the SYNCHRONIZATION SEGMENT of the restarted bit time.

RESYNCHRONIZATION JUMP WIDTH
As a result of RESYNCHRONIZATION PHASE_SEG1 may be lengthened or PHASE_SEG2 may be shortened. The amount of lengthening or shortening of the PHASE BUFFER SEGMENTs has an upper bound given by the RESYNCHRONIZATION JUMP WIDTH. The RESYNCHRONIZATION JUMP WIDTH shall be programmable between 1 and min(4, PHASE_SEG1).
Clocking information may be derived from transitions from one bit value to the other. The property that only a fixed maximum number of successive bits have the same value provides the possibility of resynchronizing a bus unit to the bit stream during a frame. The maximum length between two transitions which can be used for resynchronization is 29 bit times.

PHASE ERROR of an edge
The PHASE ERROR of an edge is given by the position of the edge relative to SYNC_SEG, measured in TIME QUANTA. The sign of PHASE ERROR is defined as follows:

- e = 0 if the edge lies within SYNC_SEG.
- e > 0 if the edge lies before the SAMPLE POINT.
- e < 0 if the edge lies after the SAMPLE POINT of the previous bit.

RESYNCHRONIZATION
The effect of a RESYNCHRONIZATION is the same as that of a HARD SYNCHRONIZATION, when the magnitude of the PHASE ERROR of the edge which causes the RESYNCHRONIZATION is less than or equal to the programmed value of the RESYNCHRONIZATION JUMP WIDTH. When the magnitude of the PHASE ERROR is larger than the RESYNCHRONIZATION JUMP WIDTH,

- and if the PHASE ERROR is positive, then PHASE_SEG1 is lengthened by an amount equal to the RESYNCHRONIZATION JUMP WIDTH.
- and if the PHASE ERROR is negative, then PHASE_SEG2 is shortened by an amount equal to the RESYNCHRONIZATION JUMP WIDTH.

SYNCHRONIZATION RULES

HARD SYNCHRONIZATION and RESYNCHRONIZATION are the two forms of SYNCHRONIZATION. They obey the following rules:

1. Only one SYNCHRONIZATION within one bit time is allowed.

2. An edge will be used for SYNCHRONIZATION only if the value detected at the previous SAMPLE POINT (previous read bus value) differs from the bus value immediately after the edge.

3. HARD SYNCHRONIZATION is performed whenever there is a 'recessive' to 'dominant' edge during BUS IDLE.

4. All other 'recessive' to 'dominant' edges (and optionally 'dominant' to 'recessive' edges in case of low bit rates) fulfilling the rules 1 and 2 will be used for RESYNCHRONIZATION with the exception that a node transmitting a dominant bit will not perform a RESYNCHRONIZATION as a result of a 'recessive' to 'dominant' edge with a positive PHASE ERROR, if only 'recessive' to 'dominant' edges are used for resynchronization.

9 INCREASING CAN OSCILLATOR TOLERANCE

This section describes an upwards compatible modification of the CAN protocol, as specified in sections 1 to 8.

9.1 Protocol Modifications

In order to increase the maximum oscillator tolerance from the 0.5% currently possible to 1.5%, the following modifications, which are upwards compatible to the existing CAN specification, are necessary:

[1] If a CAN node samples a dominant bit at the third bit of INTERMISSION, then it will interpret this bit as a START OF FRAME bit.

[2] If a CAN node has a message waiting for transmission and it samples a dominant bit at the third bit of INTERMISSION, it will interpret this as a START OF FRAME bit, and, with the next bit, start transmitting its message with the first bit of the IDENTIFIER without first transmitting a START OF FRAME bit and without becoming a receiver.

[3] If a CAN node samples a dominant bit at the eighth bit (the last bit) of an ERROR DELIMITER or OVERLOAD DELIMITER, it will, at the next bit, start transmitting an OVERLOAD FRAME (not an ERROR FRAME). The Error Counters will not be incremented.

[4] Only recessive to dominant edges will be used for synchronization.

In agreement with the existing specification, the following rules are still valid.

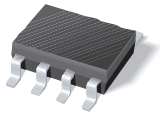
[5] All CAN controllers synchronize on the START OF FRAME bit with a hard synchronization.

[6] No CAN controller will send a START OF FRAME bit until it has counted three recessive bits of INTERMISSION.

This modifications allow a maximum oscillator tolerance of 1.58% and the use of a ceramic resonator at a bus speed of up to 125 Kbits/second. For the full bus speed range of the CAN protocol, still a quartz oscillator is required. The compatibility of the enhanced and the existing protocol is maintained, as long as:

[7] CAN controllers with the enhanced and existing protocols, used in one and the same network, have all to be provided with a quartz oscillator.

The chip with the highest requirement for its oscillator accuracy determines the oscillator accuracy which is required from all the other nodes. Ceramic resonators can only be used when all the nodes in the network use the enhanced protocol.



3.3-V CAN TRANSCEIVERS

FEATURES

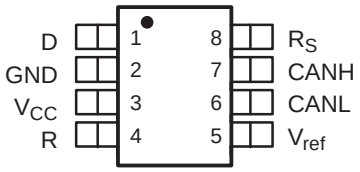
- **Operates With a 3.3-V Supply**
- **Low Power Replacement for the PCA82C250 Footprint**
- **Bus/Pin ESD Protection Exceeds 16 kV HBM**
- **High Input Impedance Allows for 120 Nodes on a Bus**
- **Controlled Driver Output Transition Times for Improved Signal Quality on the SN65HVD230 and SN65HVD231**
- **Unpowered Node Does Not Disturb the Bus**
- **Compatible With the Requirements of the ISO 11898 Standard**
- **Low-Current SN65HVD230 Standby Mode 370 μA Typical**
- **Low-Current SN65HVD231 Sleep Mode 40 nA Typical**
- **Designed for Signaling Rates(1) up to 1 Megabit/Second (Mbps)**
- **Thermal Shutdown Protection**
- **Open-Circuit Fail-Safe Design**
- **Glitch-Free Power-Up and Power-Down Protection for Hot-Plugging Applications**

(1) The signaling rate of a line is the number of voltage transitions that are made per second expressed in the units bps (bits per second).

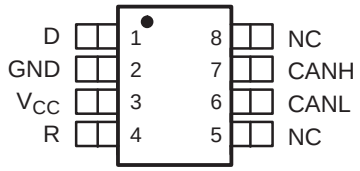
APPLICATIONS

- **Motor Control**
- **Industrial Automation**
- **Basestation Control and Status**
- **Robotics**
- **Automotive**
- **UPS Control**

SN65HVD230D (Marked as VP230)
SN65HVD231D (Marked as VP231)
(TOP VIEW)



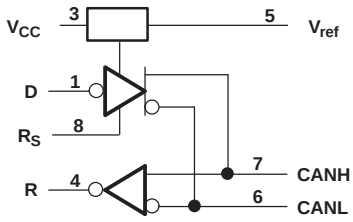
SN65HVD232D (Marked as VP232)
(TOP VIEW)



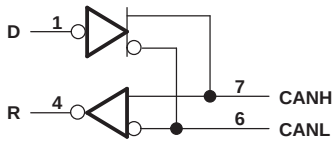
NC – No internal connection

LOGIC DIAGRAM (POSITIVE LOGIC)

SN65HVD230, SN65HVD231
Logic Diagram (Positive Logic)



SN65HVD232
Logic Diagram (Positive Logic)



These devices have limited built-in ESD protection. The leads should be shorted together or the device placed in conductive foam during storage or handling to prevent electrostatic damage to the MOS gates.

DESCRIPTION

The SN65HVD230, SN65HVD231, and SN65HVD232 controller area network (CAN) transceivers are designed for use with the Texas Instruments TMS320Lx240x™; 3.3-V DSPs with CAN controllers, or with equivalent devices. They are intended for use in applications employing the CAN serial communication physical layer in accordance with the ISO 11898 standard. Each CAN transceiver is designed to provide differential transmit capability to the bus and differential receive capability to a CAN controller at speeds up to 1 Mbps.

Designed for operation in especially-harsh environments, these devices feature cross-wire protection, loss-of-ground and overvoltage protection, overtemperature protection, as well as wide common-mode range.

The transceiver interfaces the single-ended CAN controller with the differential CAN bus found in industrial, building automation, and automotive applications. It operates over a -2-V to 7-V common-mode range on the bus, and it can withstand common-mode transients of ±25 V.

On the SN65HVD230 and SN65HVD231, pin 8 provides three different modes of operation: high-speed, slope control, and low-power modes. The high-speed mode of operation is selected by connecting pin 8 to ground, allowing the transmitter output transistors to switch on and off as fast as possible with no limitation on the rise and fall slopes. The rise and fall slopes can be adjusted by connecting a resistor to ground at pin 8, since the slope is proportional to the pin's output current. This slope control is implemented with external resistor values of 10 kΩ, to achieve a 15-V/µs slew rate, to 100 kΩ, to achieve a 2-V/µs slew rate. See the *Application Information* section of this data sheet.

The circuit of the SN65HVD230 enters a low-current standby mode during which the driver is switched off and the receiver remains active if a high logic level is applied to pin 8. The DSP controller reverses this low-current standby mode when a dominant state (bus differential voltage > 900 mV typical) occurs on the bus.

The unique difference between the SN65HVD230 and the SN65HVD231 is that both the driver and the receiver are switched off in the SN65HVD231 when a high logic level is applied to pin 8 and remain in this sleep mode until the circuit is reactivated by a low logic level on pin 8.

The V_{ref} pin 5 on the SN65HVD230 and SN65HVD231 is available as a $V_{CC}/2$ voltage reference.

The SN65HVD232 is a basic CAN transceiver with no added options; pins 5 and 8 are NC, no connection.

AVAILABLE OPTIONS(1)

PART NUMBER	LOW POWER MODE	INTEGRATED SLOPE CONTROL	V_{ref} PIN	T_A	MARKED AS:
SN65HVD230	Standby mode	Yes	Yes	40°C to 85°C	VP230
SN65HVD231	Sleep mode	Yes	Yes		VP231
SN65HVD232	No standby or sleep mode	No	No		VP232

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

FUNCTION TABLES

DRIVER (SN65HVD230, SN65HVD231)(1)				
INPUT D	R_S	OUTPUTS		BUS STATE
		CANH	CANL	
L	$V_{(Rs)}$ < 1.2 V	H	L	Dominant
H		Z	Z	Recessive
Open	X	Z	Z	Recessive
X	$V_{(Rs)}$ > 0.75 V_{CC}	Z	Z	Recessive

(1) H = high level; L = low level; X = irrelevant; ? = indeterminate; Z = high impedance

DRIVER (SN65HVD232)(1)			
INPUT D	OUTPUTS		BUS STATE
	CANH	CANL	
L	H	L	Dominant
H	Z	Z	Recessive
Open	Z	Z	Recessive

(1) H = high level; L = low level; Z = high impedance

RECEIVER (SN65HVD230)(1)		
DIFFERENTIAL INPUTS	R_S	OUTPUT R
$V_{ID} \geq 0.9$ V	X	L
0.5 V < V_{ID} < 0.9 V	X	?
$V_{ID} \leq 0.5$ V	X	H
Open	X	H

(1) H = high level; L = low level; X = irrelevant; ? = indeterminate

RECEIVER (SN65HVD231)(1)		
DIFFERENTIAL INPUTS	R_S	OUTPUT R
$V_{ID} \geq 0.9$ V	$V_{(Rs)} < 1.2$ V	L
0.5 V < V_{ID} < 0.9 V		?
$V_{ID} \leq 0.5$ V		H
X	$V_{(Rs)} > 0.75\ V_{CC}$	H
X	1.2 V < $V_{(Rs)}$ < 0.75 V_{CC}	?
Open	X	H

(1) H = high level; L = low level; X = irrelevant; ? = indeterminate

RECEIVER (SN65HVD232)(1)	
DIFFERENTIAL INPUTS	OUTPUT R
$V_{ID} \geq 0.9$ V	L
0.5 V < V_{ID} < 0.9 V	?
$V_{ID} \leq 0.5$ V	H
Open	H

(1) H = high level; L = low level; X = irrelevant; ? = indeterminate

TRANSCEIVER MODES (SN65HVD230, SN65HVD231)	
$V_{(Rs)}$	OPERATING MODE
$V_{(Rs)} > 0.75\ V_{CC}$	Standby
10 kΩ to 100 kΩ to ground	Slope control
$V_{(Rs)} < 1$ V	High speed (no slope control)

TERMINAL FUNCTIONS

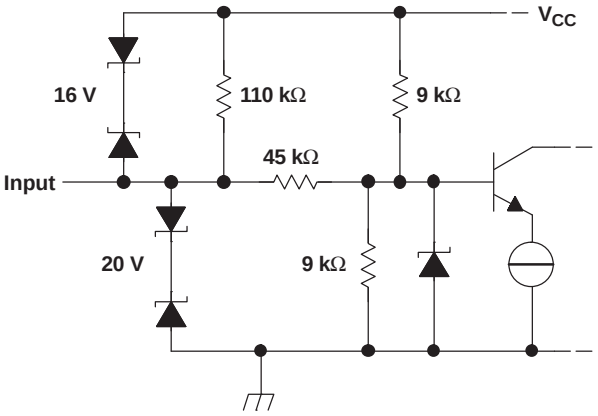
TERMINAL		DESCRIPTION
NAME	NO.	
SN65HVD230, SN65HVD231		
CANL	6	Low bus output
CANH	7	High bus output
D	1	Driver input
GND	2	Ground
R	4	Receiver output

TERMINAL FUNCTIONS (continued)

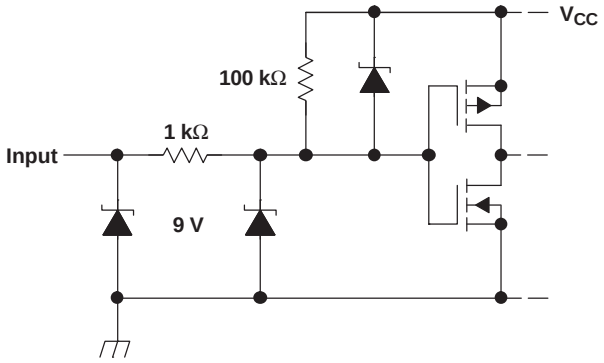
TERMINAL		DESCRIPTION
NAME	NO.	
R_S	8	Standby/slope control
V_{CC}	3	Supply voltage
V_{ref}	5	Reference output
SN65HVD232		
CANL	6	Low bus output
CANH	7	High bus output
D	1	Driver input
GND	2	Ground
NC	5, 8	No connection
R	4	Receiver output
V_{CC}	3	Supply voltage

EQUIVALENT INPUT AND OUTPUT SCHEMATIC DIAGRAMS

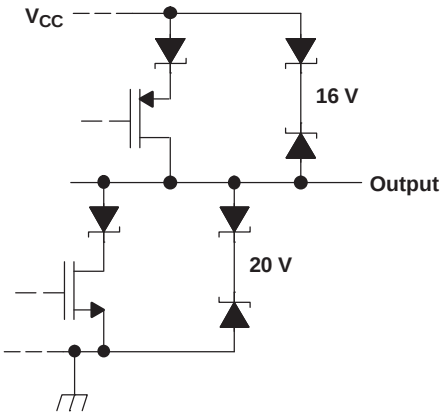
CANH and CANL Inputs



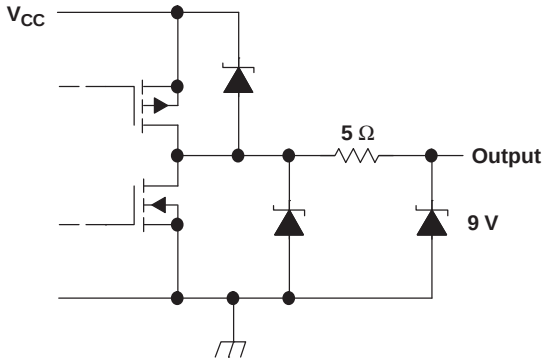
D Input



CANH and CANL Outputs



R Output



ABSOLUTE MAXIMUM RATINGS

over operating free-air temperature range (unless otherwise noted)(1)(2)

			UNIT
Supply voltage range, V_{CC}			-0.3 V to 6 V
Voltage range at any bus terminal (CANH or CANL)			-4 V to 16 V
Voltage input range, transient pulse, CANH and CANL, through 100 Ω (see Figure 7)			-25 V to 25 V
Input voltage range, V_I (D or R)			-0.5 V to V_{CC} + 0.5 V
Receiver output current, I_O			±11 mA
Electrostatic discharge	Human body model(3)	CANH, CANL and GND	16 kV
		All Pins	4 kV
	Charged-device model(4)	All pins	1 kV
Continuous total power dissipation			See Dissipation Rating Table

(1) Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods amy affect device reliability.
(2) All voltage values, except differential I/O bus voltages, are with respect to network ground terminal.
(3) Tested in accordance with JEDEC Standard 22, Test Method A114-A.
(4) Tested in accordance with JEDEC Standard 22, Test Method C101.

DISSIPATION RATING TABLE

PACKAGE	$T_A \leq 25°C$ POWER RATING	DERATING FACTOR(1) ABOVE $T_A = 25°C$	$T_A = 70°C$ POWER RATING	$T_A = 85°C$ POWER RATING
D	725 mW	5.8 mW/°C	464 mW	377 mW

(1) This is the inverse of the junction-to-ambient thermal resistance when board-mounted and with no air flow.

RECOMMENDED OPERATING CONDITIONS

		MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}		3		3.6	V
Voltage at any bus terminal (common mode) V_{IC}		-2(1)		7	V
Voltage at any bus terminal (separately) V_I		-2.5		7.5	V
High-level input voltage, V_{IH}	D, R	2			V
Low-level input voltage, V_{IL}	D, R			0.8	V
Differential input voltage, V_{ID} (see Figure 5)		-6		6	V
Input voltage, $V_{(Rs)}$		0		V_{CC}	V
Input voltage for standby or sleep, $V_{(Rs)}$		0.75 V_{CC}		V_{CC}	V
Wave-shaping resistance, Rs		0		100	kΩ
High-level output current, I_{OH}	Driver	-40			mA
	Receiver	-8			
Low-level output current, I_{OL}	Driver			48	mA
	Receiver			8	
Operating free-air temperature, T_A		-40		85	°C

(1) The algebraic convention, in which the least positive (most negative) limit is designated as minimum is used in this data sheet.

DRIVER ELECTRICAL CHARACTERISTICS

over recommended operating conditions (unless otherwise noted)

PARAMETER				TEST CONDITIONS		MIN	TYP(1)	MAX	UNIT
V_{OH}	Bus output voltage	Dominant		V_I = 0 V, See Figure 1 and Figure 3	CANH	2.45		V_{CC}	V
					CANL	0.5		1.25	
V_{OL}		Recessive		V_I = 3 V, See Figure 1 and Figure 3	CANH		2.3		
					CANL		2.3		
$V_{OD(D)}$	Differential output voltage	Dominant		V_I = 0 V, See Figure 1		1.5	2	3	V
				V_I = 0 V, See Figure 2		1.2	2	3	
$V_{OD(R)}$		Recessive		V_I = 3 V, See Figure 1		-120	0	12	mV
				V_I = 3 V, No load		-0.5	-0.2	0.05	V
I_{IH}	High-level input current			V_I = 2 V		-30			µA
I_{IL}	Low-level input current			V_I = 0.8 V		-30			µA
I_{OS}	Short-circuit output current			V_{CANH} = -2 V		-250		250	mA
				V_{CANL} = 7 V		-250		250	
C_o	Output capacitance			See receiver					
I_{CC}	Supply current	Standby	SN65HVD230	$V_{(Rs)}$ = V_{CC}			370	600	µA
		Sleep	SN65HVD231	$V_{(Rs)}$ = V_{CC}, D at V_{CC}			0.04	1	
		All devices	Dominant	V_I = 0 V, No load	Dominant		10	17	mA
			Recessive	V_I = V_{CC}, No load	Recessive		10	17	

(1) All typical values are at 25°C and with a 3.3-V supply.

DRIVER SWITCHING CHARACTERISTICS

over recommended operating conditions (unless otherwise noted)

PARAMETER			TEST CONDITIONS	MIN	TYP	MAX	UNIT
SN65HVD230 AND SN65HVD231							
t_{PLH}	Propagation delay time, low-to-high-level output	$V_{(Rs)}$ = 0 V	C_L = 50 pF, See Figure 4		35	85	ns
		R_S with 10 kΩ to ground			70	125	
		R_S with 100 kΩ to ground			500	870	
t_{PHL}	Propagation delay time, high-to-low-level output	$V_{(Rs)}$ = 0 V			70	120	ns
		R_S with 10 kΩ to ground			130	180	
		R_S with 100 kΩ to ground			870	1200	
$t_{sk(p)}$	Pulse skew ($\|t_{PHL} - t_{PLH}\|$)	$V_{(Rs)}$ = 0 V			35		ns
		R_S with 10 kΩ to ground			60		
		R_S with 100 kΩ to ground			370		
t_r	Differential output signal rise time	$V_{(Rs)}$ = 0 V		25	50	100	ns
t_f	Differential output signal fall time			40	55	80	ns
t_r	Differential output signal rise time	R_S with 10 kΩ to ground		80	120	160	ns
t_f	Differential output signal fall time			80	125	150	ns
t_r	Differential output signal rise time	R_S with 100 kΩ to ground		600	800	1200	ns
t_f	Differential output signal fall time			600	825	1000	ns
SN65HVD232							
t_{PLH}	Propagation delay time, low-to-high-level output		C_L = 50 pF, See Figure 4		35	85	ns
t_{PHL}	Propagation delay time, high-to-low-level output				70	120	
$t_{sk(p)}$	Pulse skew ($\|t_{PHL} - t_{PLH}\|$)				35		
t_r	Differential output signal rise time			25	50	100	
t_f	Differential output signal fall time			40	55	80	

RECEIVER ELECTRICAL CHARACTERISTICS

over recommended operating conditions (unless otherwise noted)

PARAMETER		TEST CONDITIONS		MIN	TYP(1)	MAX	UNIT
V_{IT+}	Positive-going input threshold voltage	See Table 1			750	900	mV
V_{IT-}	Negative-going input threshold voltage			500	650		mV
V_{hys}	Hysteresis voltage (V_{IT+} - V_{IT-})				100		
V_{OH}	High-level output voltage	-6 V ≤ V_{ID} ≤ 500 mV, I_O = -8 mA, See Figure 5		2.4			V
V_{OL}	Low-level output voltage	900 mV ≤ V_{ID} ≤ 6 V, I_O = 8 mA, See Figure 5				0.4	V
I_I	Bus input current	V_{IH} = 7 V	Other input at 0 V, D = 3 V	100		250	µA
		V_{IH} = 7 V, V_{CC} = 0 V		100		350	
		V_{IH} = -2 V		-200		-30	µA
		V_{IH} = -2 V, V_{CC} = 0 V		-100		-20	
C_i	CANH, CANL input capacitance	Pin-to-ground, V_I = 0.4 sin(4E6πt) + 0.5 V	$V_{(D)}$ = 3 V,		32		pF
C_{diff}	Differential input capacitance	Pin-to-pin, V_I = 0.4 sin(4E6πt) + 0.5 V	$V_{(D)}$ = 3 V,		16		pF
R_{diff}	Differential input resistance	Pin-to-pin, $V_{(D)}$ = 3 V		40	70	100	kΩ
R_I	CANH, CANL input resistance			20	35	50	kΩ
I_{CC}	Supply current	See driver					

(1) All typical values are at 25°C and with a 3.3-V supply.

RECEIVER SWITCHING CHARACTERISTICS

over recommended operating conditions (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output	See Figure 6		35	50	ns
t_{PHL}	Propagation delay time, high-to-low-level output			35	50	ns
$t_{sk(p)}$	Pulse skew ($\|t_{PHL} - t_{PLH}\|$)				10	ns
t_r	Output signal rise time	See Figure 6		1.5		ns
t_f	Output signal fall time			1.5		ns

DEVICE SWITCHING CHARACTERISTICS

over recommended operating conditions (unless otherwise noted)

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
$t_{(LOOP1)}$	Total loop delay, driver input to receiver output, recessive to dominant	$V_{(Rs)}$ = 0 V,	See Figure 9		70	115	ns
		R_S with 10 kΩ to ground,	See Figure 9		105	175	
		R_S with 100 kΩ to ground,	See Figure 9		535	920	
$t_{(LOOP2)}$	Total loop delay, driver input to receiver output, dominant to recessive	$V_{(Rs)}$ = 0 V,	See Figure 9		100	135	ns
		R_S with 10 kΩ to ground,	See Figure 9		155	185	
		R_S with 100 kΩ to ground,	See Figure 9		830	990	

DEVICE CONTROL-PIN CHARACTERISTICS

over recommended operating conditions (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP(1)	MAX	UNIT
$t_{(WAKE)}$	SN65HVD230 wake-up time from standby mode with R_S	See Figure 8		0.55	1.5	µs
	SN65HVD231 wake-up time from sleep mode with R_S			3	5	µs

(1) All typical values are at 25°C and with a 3.3-V supply.

DEVICE CONTROL-PIN CHARACTERISTICS (continued)

over recommended operating conditions (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP(1)	MAX	UNIT
V_{ref}	Reference output voltage	-5 µA < $I_{(Vref)}$ < 5 µA	0.45 V_{CC}		0.55 V_{CC}	V
		-50 µA < $I_{(Vref)}$ < 50 µA	0.4 V_{CC}		0.6 V_{CC}	
$I_{(Rs)}$	Input current for high-speed	$V_{(Rs)}$ < 1 V	-450		0	µA

PARAMETER MEASUREMENT INFORMATION

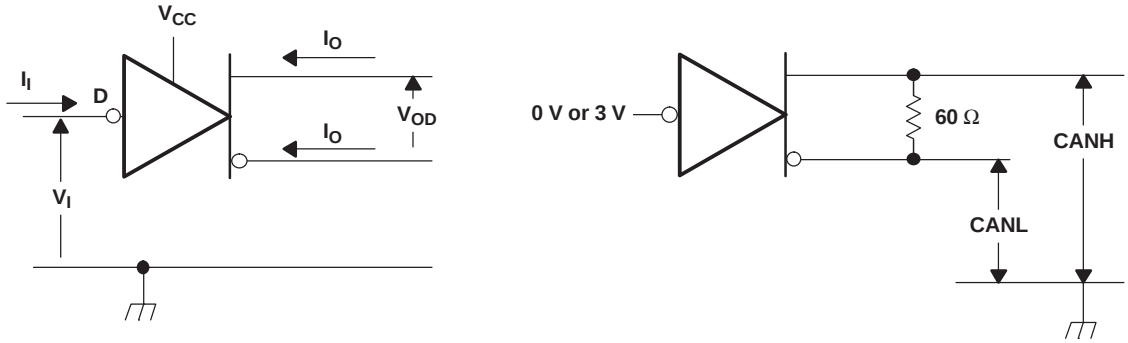


Figure 1. Driver Voltage and Current Definitions

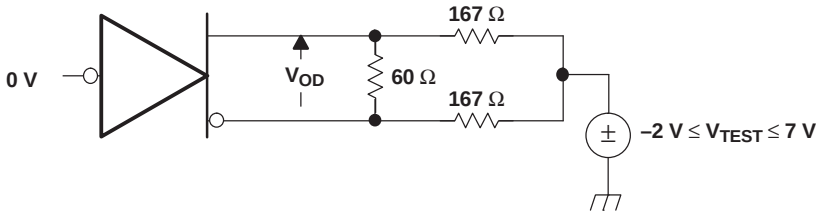


Figure 2. Driver V_{OD}

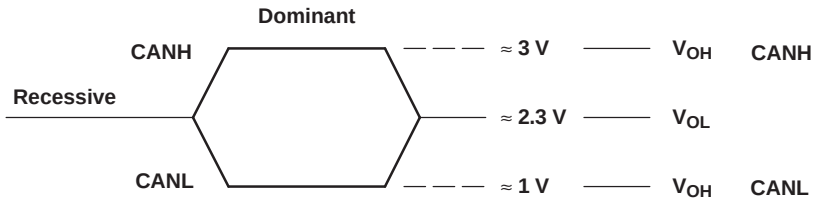
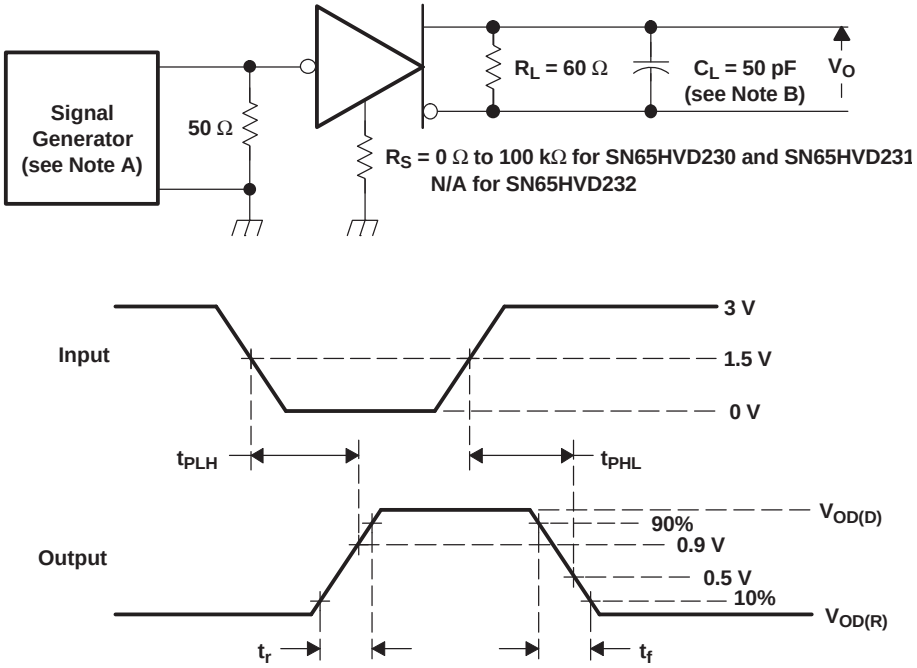


Figure 3. Driver Output Voltage Definitions

PARAMETER MEASUREMENT INFORMATION (continued)



A. The input pulse is supplied by a generator having the following characteristics: PRR ≤ 500 kHz, 50% duty cycle, t_r ≤ 6 ns, t_f ≤ 6 ns, Z_o = 50 Ω.

B. C_L includes probe and jig capacitance.

Figure 4. Driver Test Circuit and Voltage Waveforms

$$V_{IC} = \frac{V_{CANH} + V_{CANL}}{2}$$

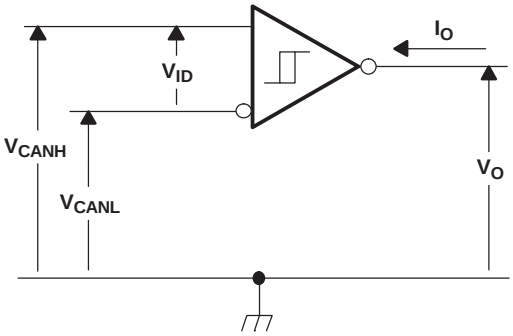
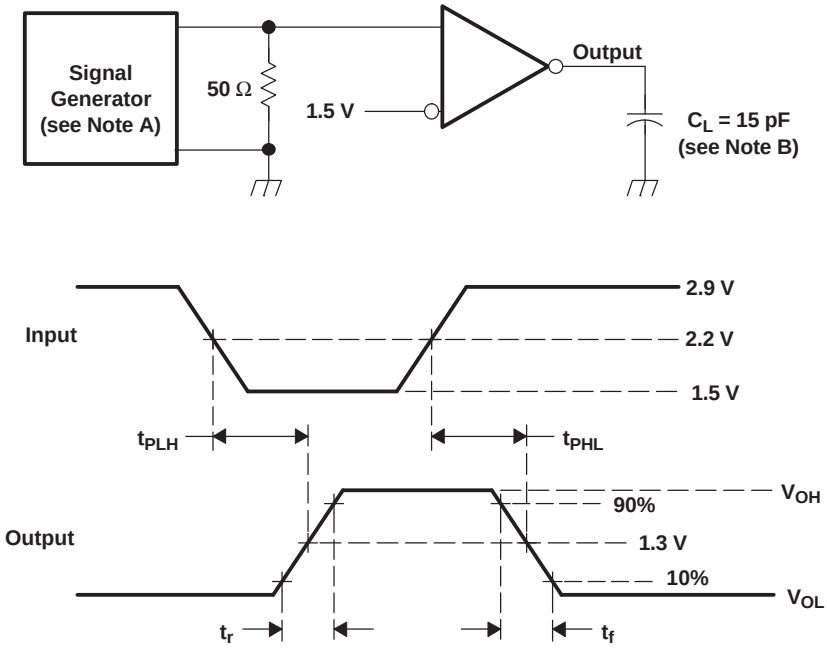


Figure 5. Receiver Voltage and Current Definitions

PARAMETER MEASUREMENT INFORMATION (continued)



A. The input pulse is supplied by a generator having the following characteristics: PRR ≤ 500 kHz, 50% duty cycle, t_r ≤ 6 ns, t_f ≤ 6 ns, Z_o = 50 Ω.

B. C_L includes probe and jig capacitance.

Figure 6. Receiver Test Circuit and Voltage Waveforms

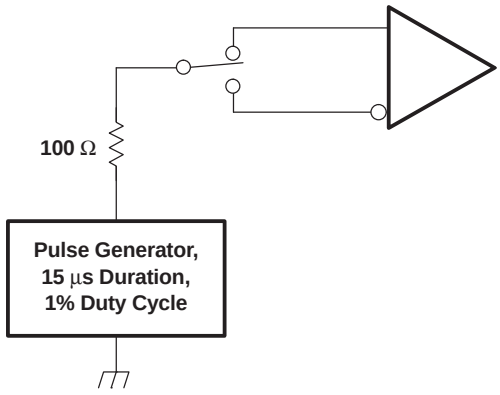


Figure 7. Overvoltage Protection

PARAMETER MEASUREMENT INFORMATION (continued)

Table 1. Receiver Characteristics Over Common Mode With $V_{(Rs)}$ = 1.2 V

V_{IC}	V_{ID}	V_{CANH}	V_{CANL}	R OUTPUT	
-2 V	900 mV	-1.55 V	-2.45 V	L	V_{OL}
7 V	900 mV	8.45 V	6.55 V	L	
1 V	6 V	4 V	-2 V	L	
4 V	6 V	7 V	1 V	L	
-2 V	500 mV	-1.75 V	-2.25 V	H	V_{OH}
7 V	500 mV	7.25 V	6.75 V	H	
1 V	-6 V	-2 V	4 V	H	
4 V	-6 V	1 V	7 V	H	
X	X	Open	Open	H	

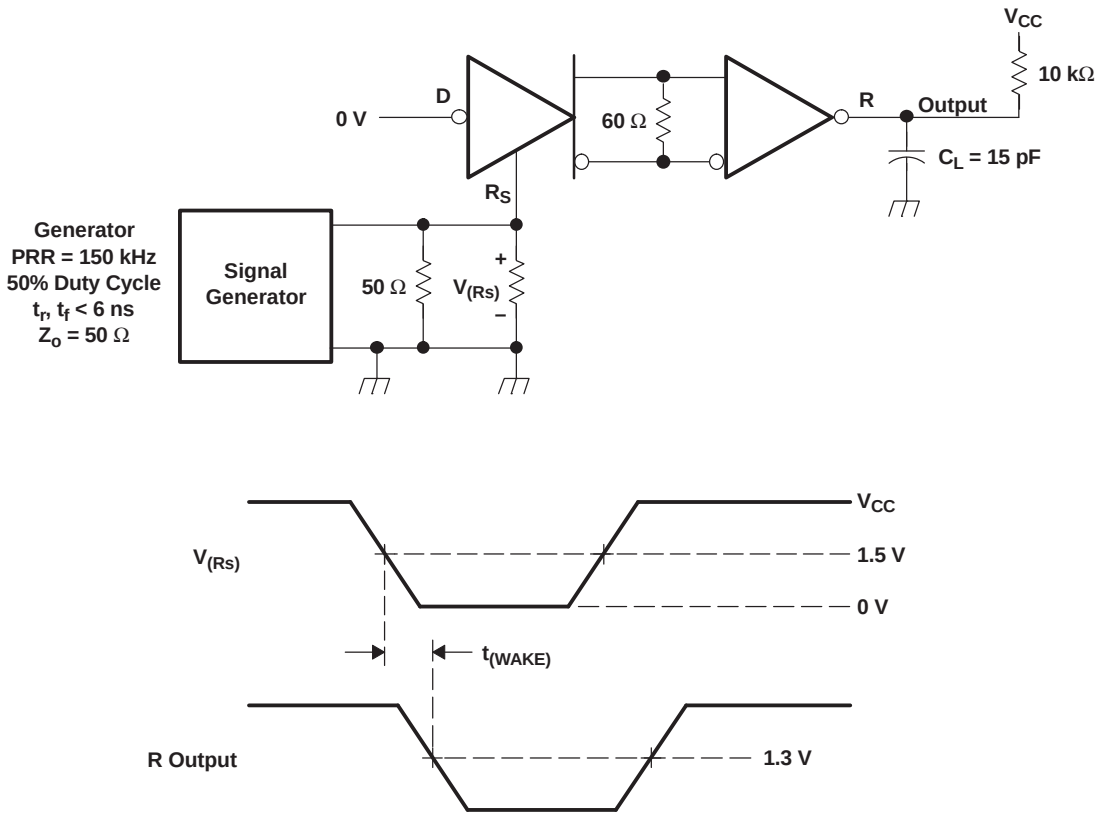
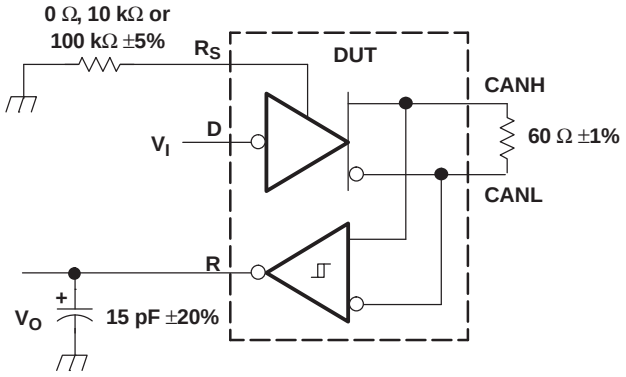
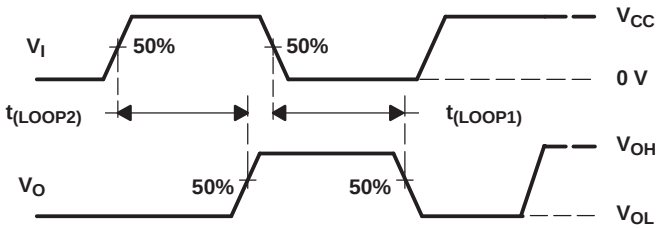


Figure 8. $t_{(WAKE)}$ Test Circuit and Voltage Waveforms





A. All V_I input pulses are supplied by a generator having the following characteristics: t_r or $t_f \leq 6$ ns, Pulse Repetition Rate (PRR) = 125 kHz, 50% duty cycle.

Figure 9. $t_{(LOOP)}$ Test Circuit and Voltage Waveforms

TYPICAL CHARACTERISTICS

SUPPLY CURRENT (RMS) vs FREQUENCY

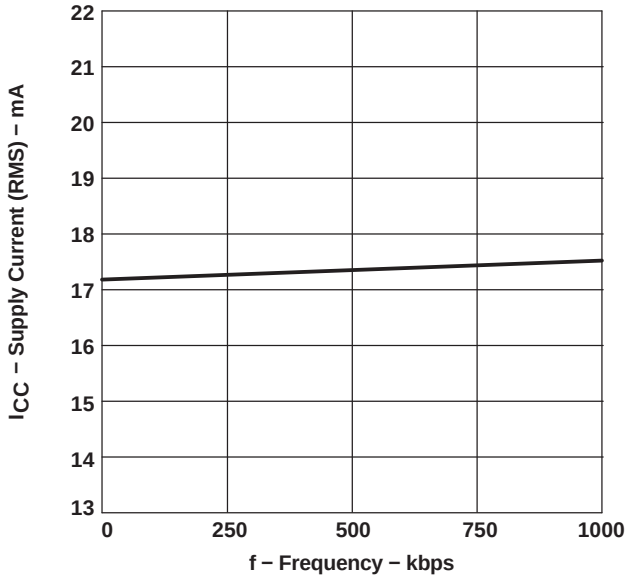


Figure 10.

LOGIC INPUT CURRENT (PIN D) vs INPUT VOLTAGE

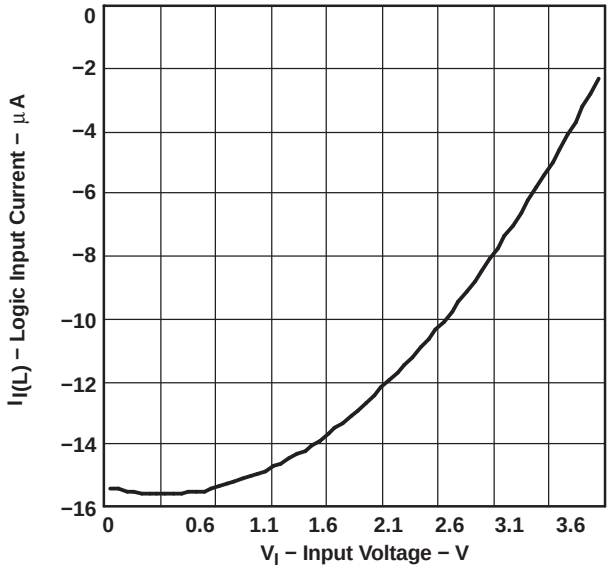


Figure 11.

TYPICAL CHARACTERISTICS (continued)

BUS INPUT CURRENT vs BUS INPUT VOLTAGE

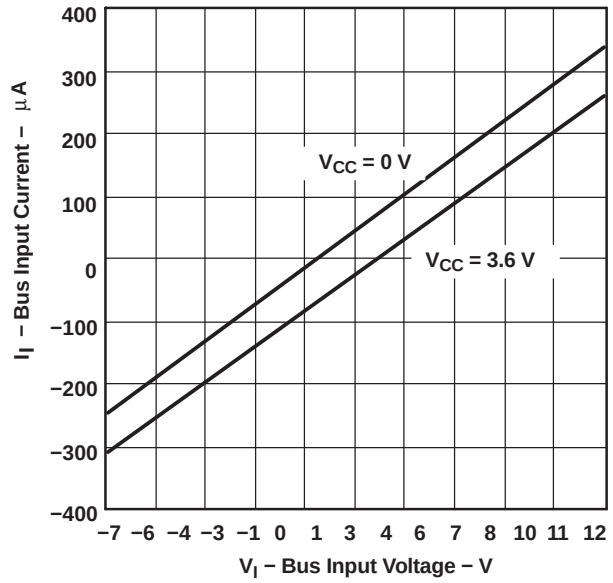


Figure 12.

DRIVER LOW-LEVEL OUTPUT CURRENT vs LOW-LEVEL OUTPUT VOLTAGE

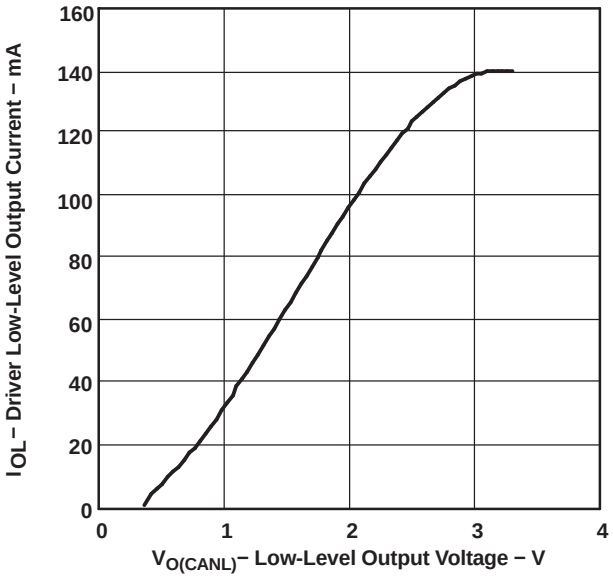


Figure 13.

DRIVER HIGH-LEVEL OUTPUT CURRENT vs HIGH-LEVEL OUTPUT VOLTAGE

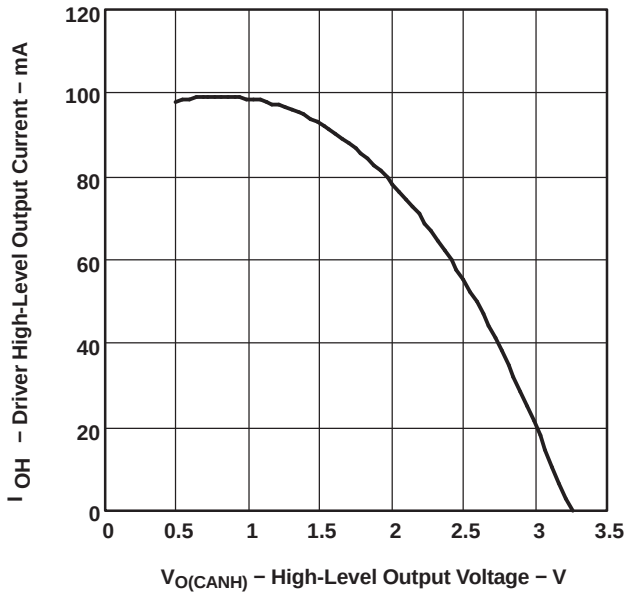


Figure 14.

DOMINANT VOLTAGE (V_{OD}) vs FREE-AIR TEMPERATURE

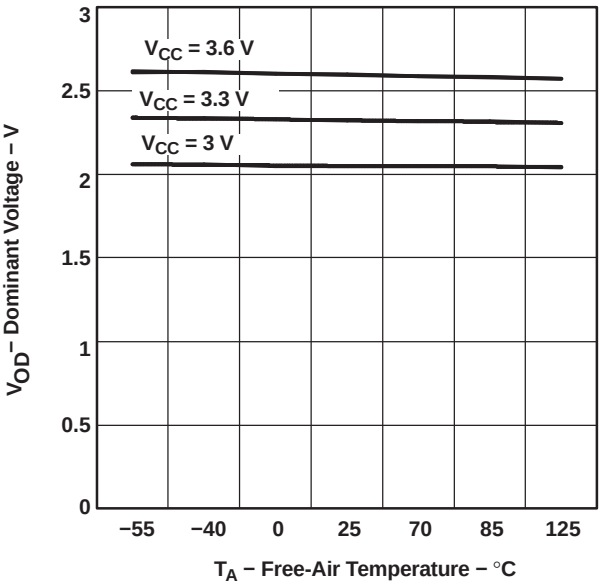


Figure 15.

TYPICAL CHARACTERISTICS (continued)

RECEIVER LOW-TO-HIGH PROPAGATION DELAY TIME vs FREE-AIR TEMPERATURE

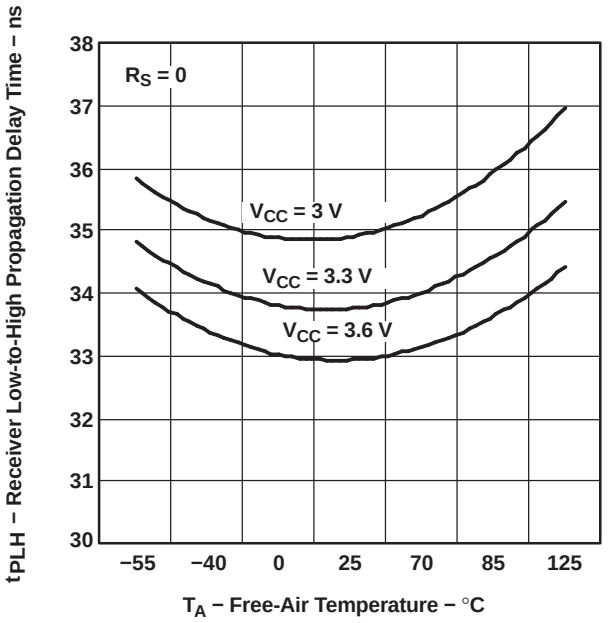


Figure 16.

RECEIVER HIGH-TO-LOW PROPAGATION DELAY TIME vs FREE-AIR TEMPERATURE

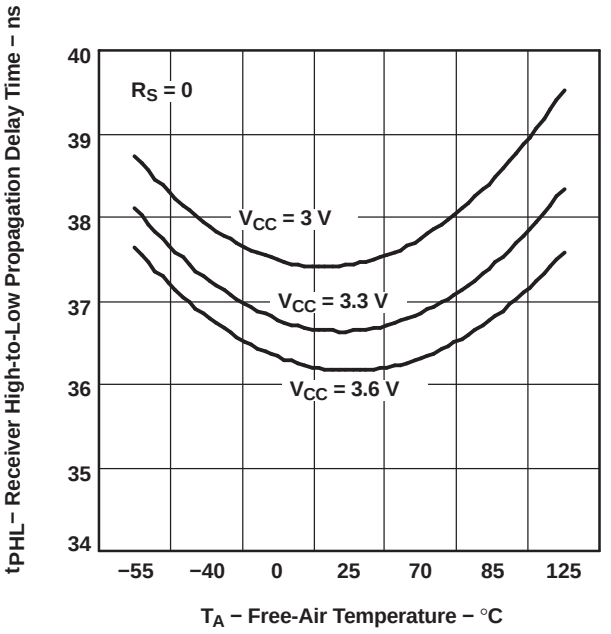


Figure 17.

DRIVER LOW-TO-HIGH PROPAGATION DELAY TIME vs FREE-AIR TEMPERATURE

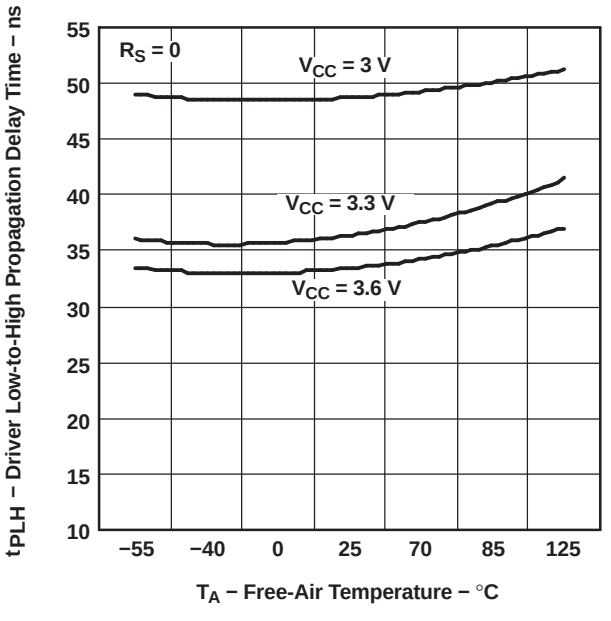


Figure 18.

DRIVER HIGH-TO-LOW PROPAGATION DELAY TIME vs FREE-AIR TEMPERATURE

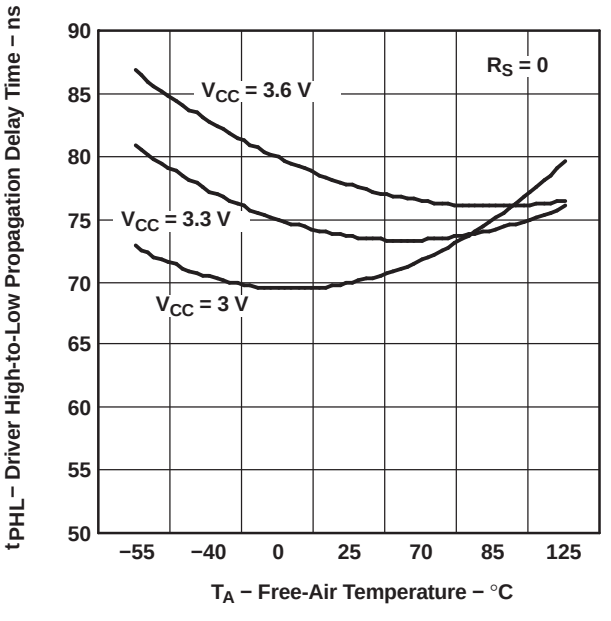


Figure 19.

TYPICAL CHARACTERISTICS (continued)

DRIVER LOW-TO-HIGH PROPAGATION DELAY TIME vs FREE-AIR TEMPERATURE

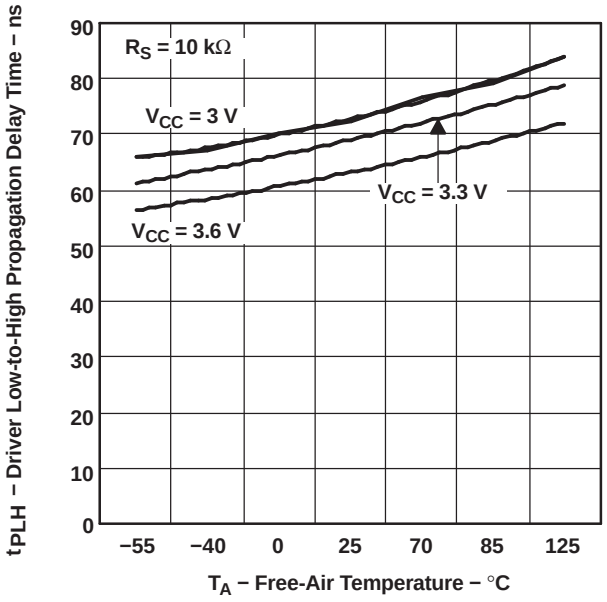


Figure 20.

DRIVER HIGH-TO-LOW PROPAGATION DELAY TIME vs FREE-AIR TEMPERATURE

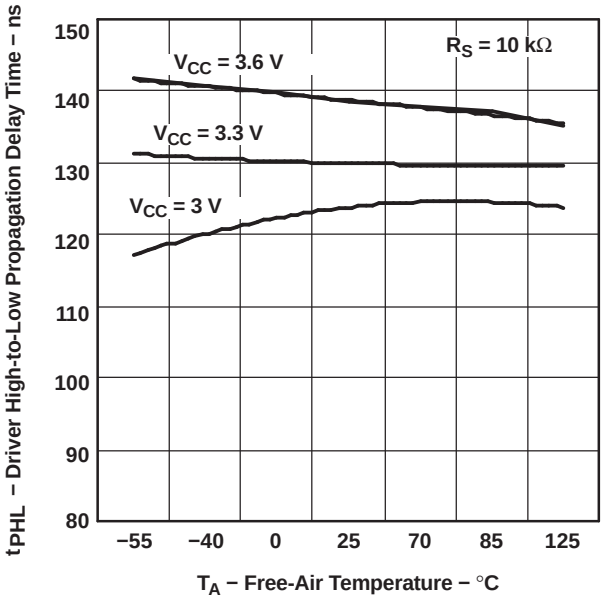


Figure 21.

DRIVER LOW-TO-HIGH PROPAGATION DELAY TIME vs FREE-AIR TEMPERATURE

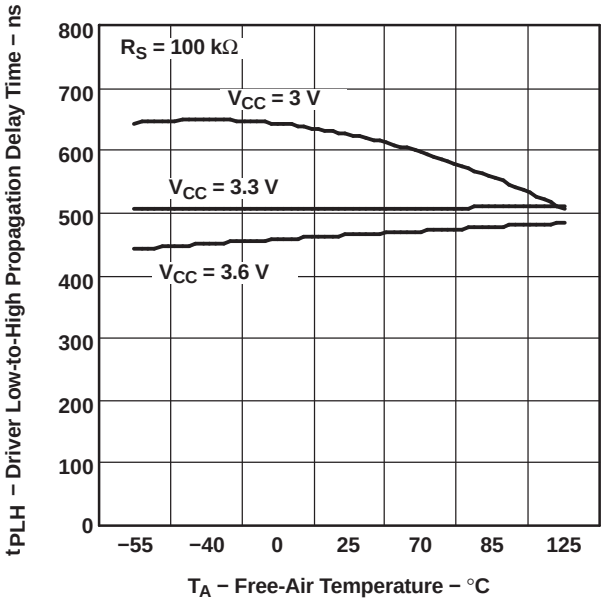


Figure 22.

DRIVER HIGH-TO-LOW PROPAGATION DELAY TIME vs FREE-AIR TEMPERATURE

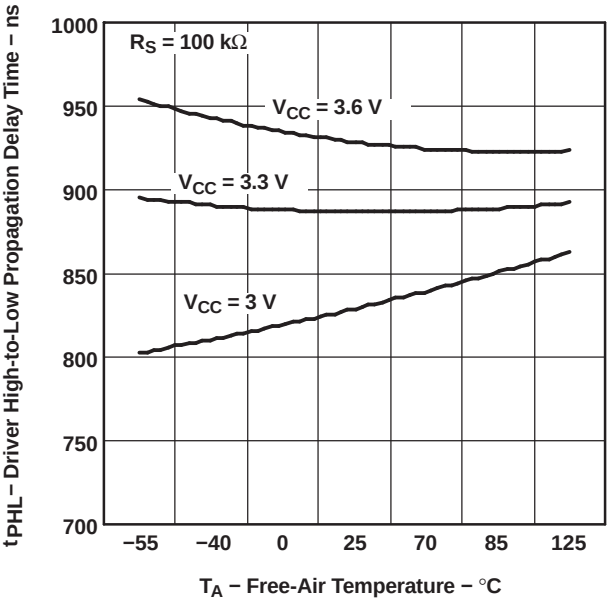


Figure 23.

TYPICAL CHARACTERISTICS (continued)

DRIVER OUTPUT CURRENT
vs
SUPPLY VOLTAGE

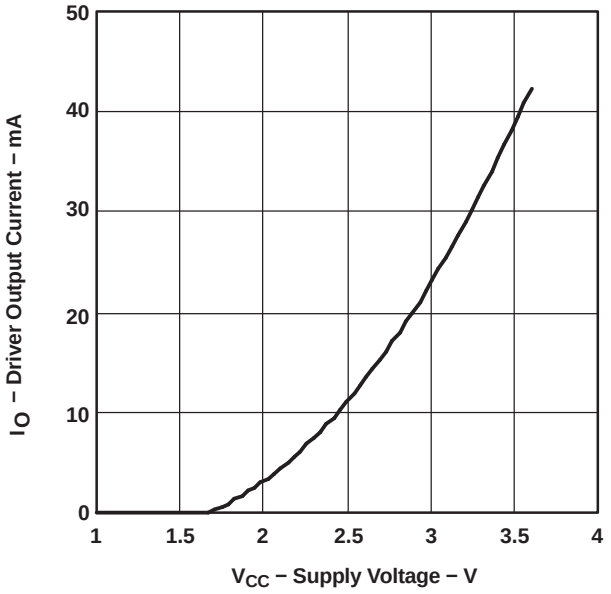


Figure 24.

DIFFERENTIAL DRIVER OUTPUT FALL TIME
vs
SOURCE RESISTANCE (R_s)

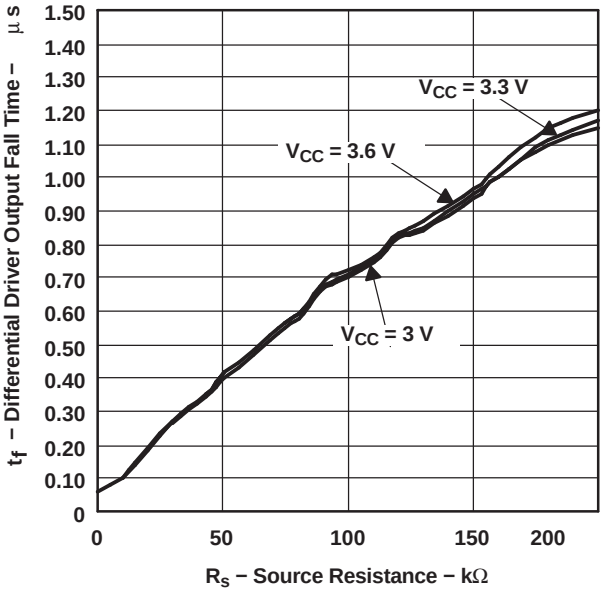


Figure 25.

REFERENCE VOLTAGE
vs
REFERENCE CURRENT

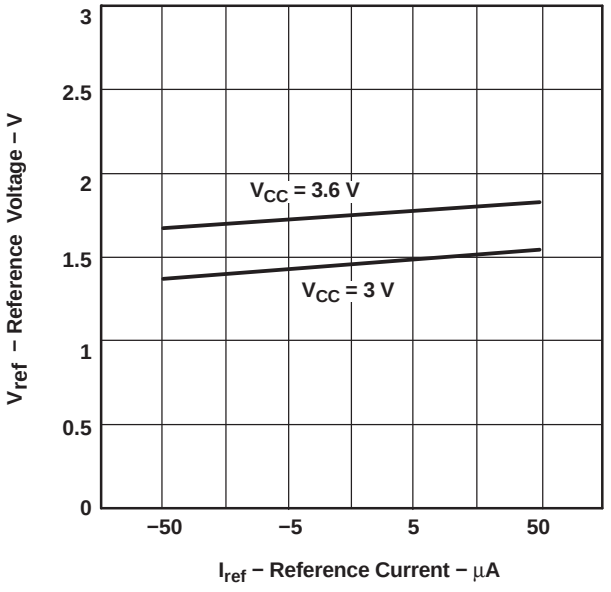


Figure 26.

HVD230, HVD231 LOW-TO-HIGH PROPAGATION DELAY TIME
vs
COMMON-MODE INPUT VOLTAGE

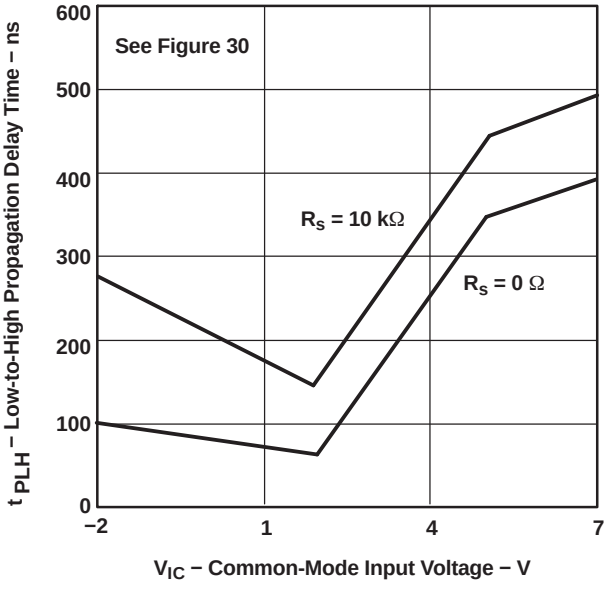


Figure 27.

TYPICAL CHARACTERISTICS (continued)

HVD230, HVD231 LOW-TO-HIGH PROPAGATION DELAY TIME vs COMMON-MODE INPUT VOLTAGE

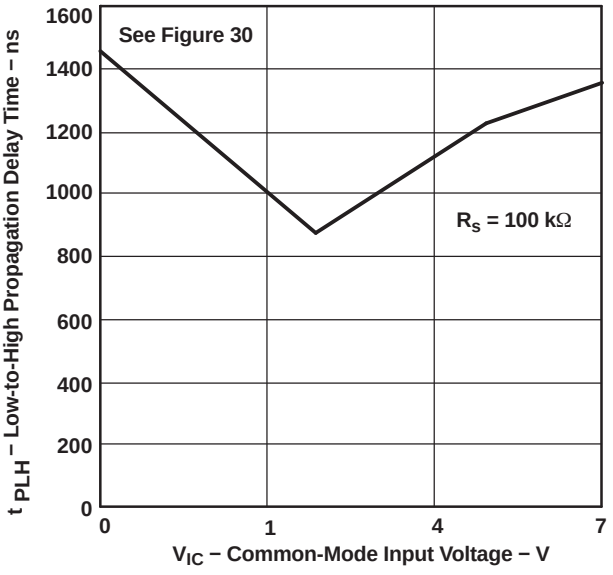


Figure 28.

HVD232 LOW-TO-HIGH PROPAGATION DELAY TIME vs COMMON-MODE INPUT VOLTAGE

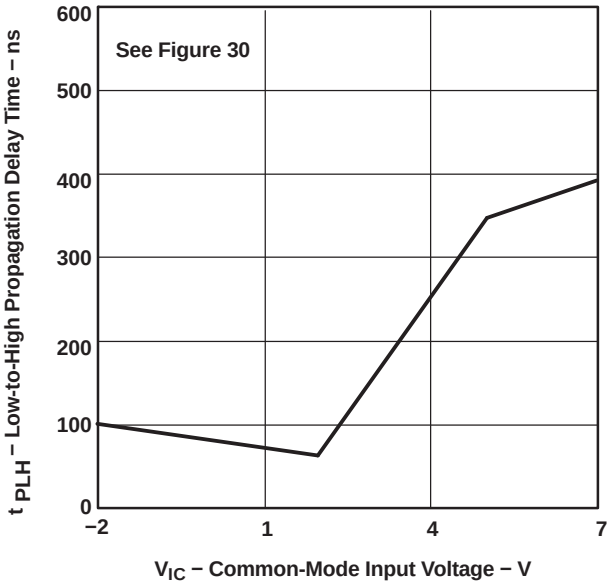


Figure 29.

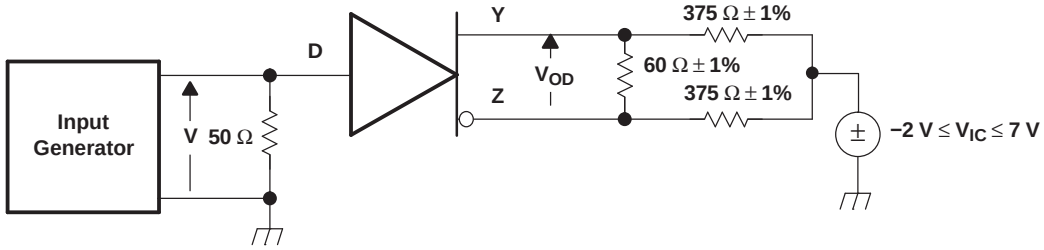


Figure 30. Driver Schematic

APPLICATION INFORMATION

This application provides information concerning the implementation of the physical medium attachment layer in a CAN network according to the ISO 11898 standard. It presents a typical application circuit and test results, as well as discussions on slope control, total loop delay, and interoperability in 5-V systems.

INTRODUCTION

ISO 11898 is the international standard for high-speed serial communication using the controller area network (CAN) bus protocol. It supports multimaster operation, real-time control, programmable data rates up to 1 Mbps, and powerful redundant error checking procedures that provide reliable data transmission. It is suited for networking *intelligent* devices as well as sensors and actuators within the rugged electrical environment of a machine chassis or factory floor. The SN65HVD230 family of 3.3-V CAN transceivers implement the lowest layers of the ISO/OSI reference model. This is the interface with the physical signaling output of the CAN controller of the Texas Instruments TMS320Lx240x 3.3-V DSPs, as illustrated in Figure 31.

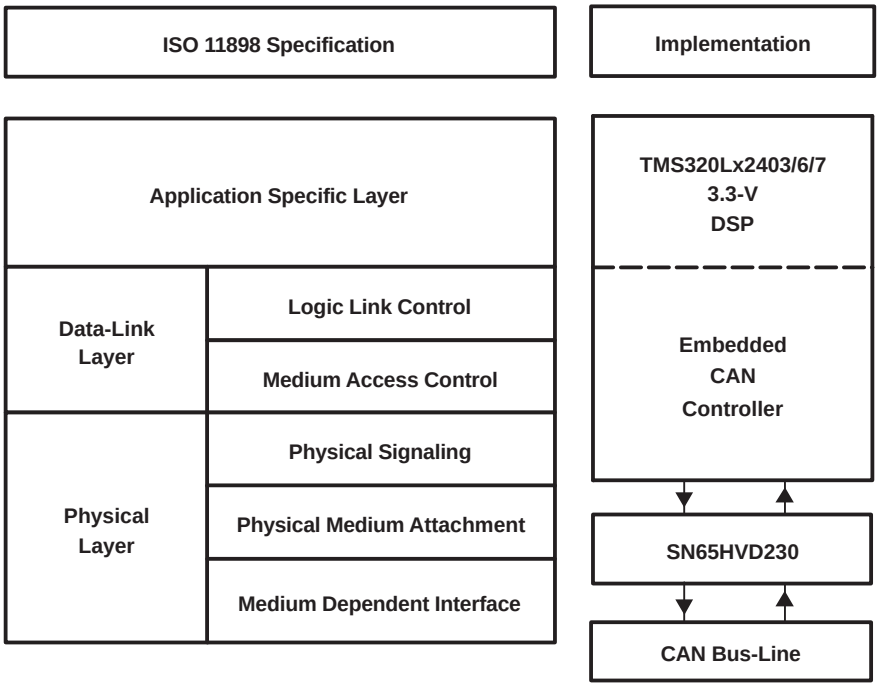


Figure 31. The Layered ISO 11898 Standard Architecture

The SN65HVD230 family of CAN transceivers are compatible with the ISO 11898 standard; this ensures interoperability with other standard-compliant products.

APPLICATION OF THE SN65HVD230

Figure 32 illustrates a typical application of the SN65HVD230 family. The output of a DSP's CAN controller is connected to the serial driver input, pin D, and receiver serial output, pin R, of the transceiver. The transceiver is then attached to the differential bus lines at pins CANH and CANL. Typically, the bus is a twisted pair of wires with a characteristic impedance of 120 Ω, in the standard half-duplex multipoint topology of Figure 33. Each end of the bus is terminated with 120-Ω resistors in compliance with the standard to minimize signal reflections on the bus.

APPLICATION INFORMATION (continued)

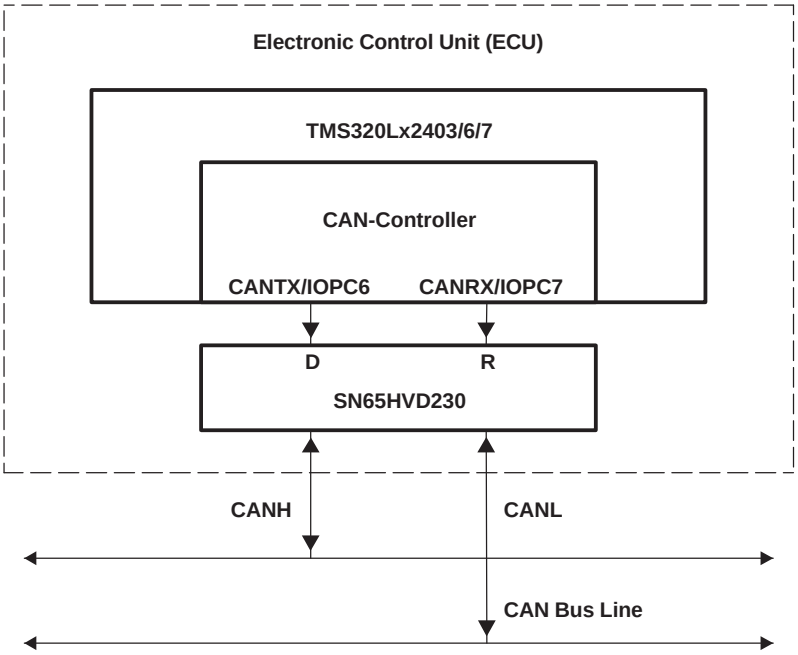


Figure 32. Details of a Typical CAN Node

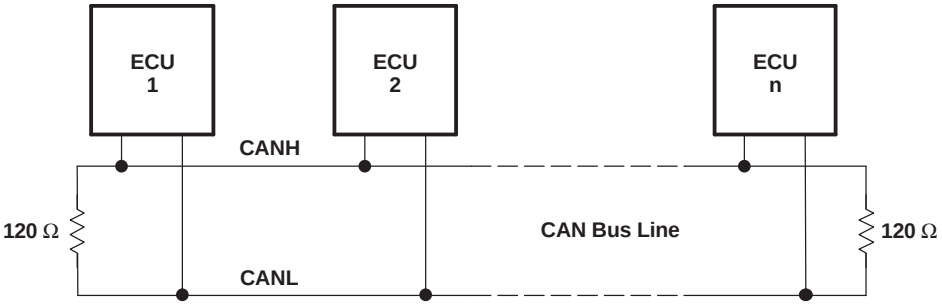


Figure 33. Typical CAN Network

The SN65HVD230/231/232 3.3-V CAN transceivers provide the interface between the 3.3-V TMS320Lx2403/6/7 CAN DSPs and the differential bus line, and are designed to transmit data at signaling rates up to 1 Mbps as defined by the ISO 11898 standard.

FEATURES of the SN65HVD230, SN65HVD231, and SN65HVD232

The SN65HVD230/231/232 are pin-compatible (but not functionally identical) with one another and, depending upon the application, may be used with identical circuit boards.

These transceivers feature 3.3-V operation and standard compatibility with signaling rates up to 1 Mbps, and also offer 16-kV HBM ESD protection on the bus pins, thermal shutdown protection, bus fault protection, and open-circuit receiver failsafe. The fail-safe design of the receiver assures a logic high at the receiver output if the bus wires become open circuited. If a high ambient operating environment temperature or excessive output current result in thermal shutdown, the bus pins become high impedance, while the D and R pins default to a logic high.

APPLICATION INFORMATION (continued)

The bus pins are also maintained in a high-impedance state during low V_{CC} conditions to ensure glitch-free power-up and power-down bus protection for hot-plugging applications. This high-impedance condition also means that an unpowered node does not disturb the bus. Transceivers without this feature usually have a very low output impedance. This results in a high current demand when the transceiver is unpowered, a condition that could affect the entire bus.

OPERATING MODES

R_S (pin 8) of the SN65HVD230 and SN65HVD231 provides for three different modes of operation: high-speed mode, slope-control mode, and low-power mode.

High-Speed

The high-speed mode can be selected by applying a logic low to R_S (pin 8). The high-speed mode of operation is commonly employed in industrial applications. High-speed allows the output to switch as fast as possible with no internal limitation on the output rise and fall slopes. The only limitations of the high-speed operation are cable length and radiated emission concerns, each of which is addressed by the slope control mode of operation.

If the low-power standby mode is to be employed in the circuit, direct connection to a DSP output pin can be used to switch between a logic-low level (< 1 V) for high speed operation, and the logic-high level (> 0.75 V_{CC}) for standby. Figure 34 shows a typical DSP connection, and Figure 35 shows the HVD230 driver output signal in high-speed mode on the CAN bus.

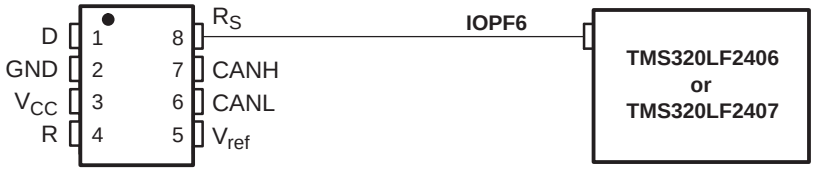


Figure 34. R_S (Pin 8) Connection to a TMS320LF2406/07 for High Speed/Standby Operation

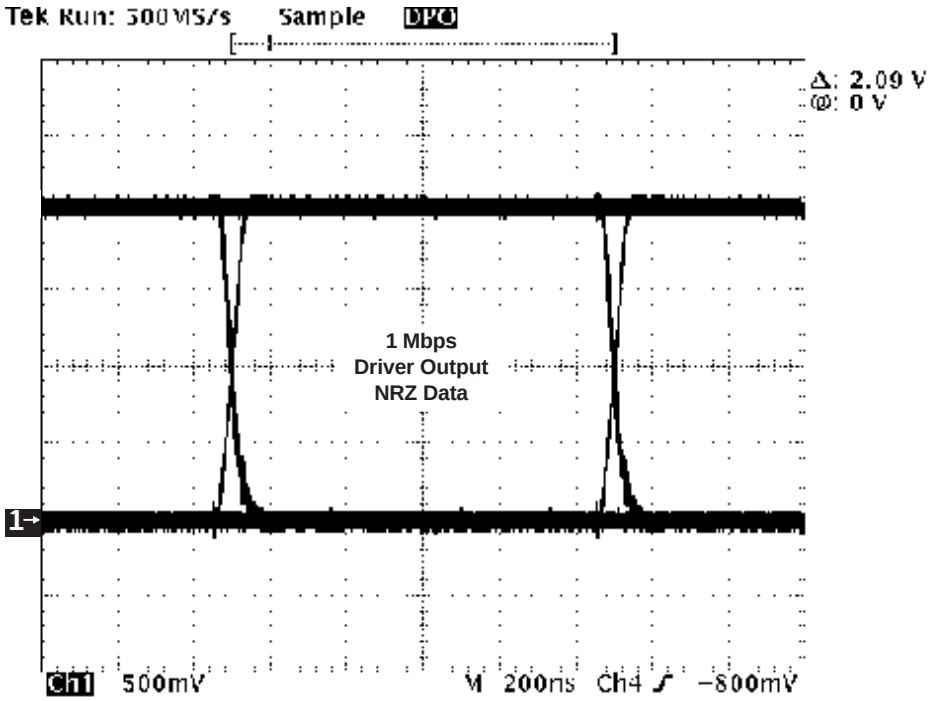


Figure 35. Typical High Speed SN65HVD230 Output Waveform Into a 60-Ω Load

APPLICATION INFORMATION (continued)

Slope Control

Electromagnetic compatibility is essential in many applications using unshielded bus cable to reduce system cost. To reduce the electromagnetic interference generated by fast rise times and resulting harmonics, the rise and fall slopes of the SN65HVD230 and SN65HVD231 driver outputs can be adjusted by connecting a resistor from R_S (pin 8) to ground or to a logic low voltage, as shown in Figure 36. The slope of the driver output signal is proportional to the pin's output current. This slope control is implemented with an external resistor value of 10 kΩ to achieve a ≈ 15 V/µs slew rate, and up to 100 kΩ to achieve a ≈ 2.0 V/µs slew rate as displayed in Figure 37. Typical driver output waveforms from a pulse input signal with and without slope control are displayed in Figure 38. A pulse input is used rather than NRZ data to clearly display the actual slew rate.

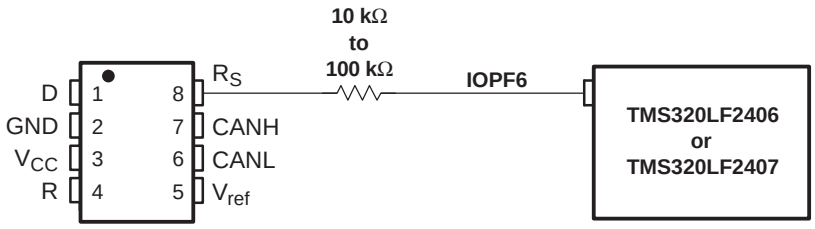


Figure 36. Slope Control/Standby Connection to a DSP

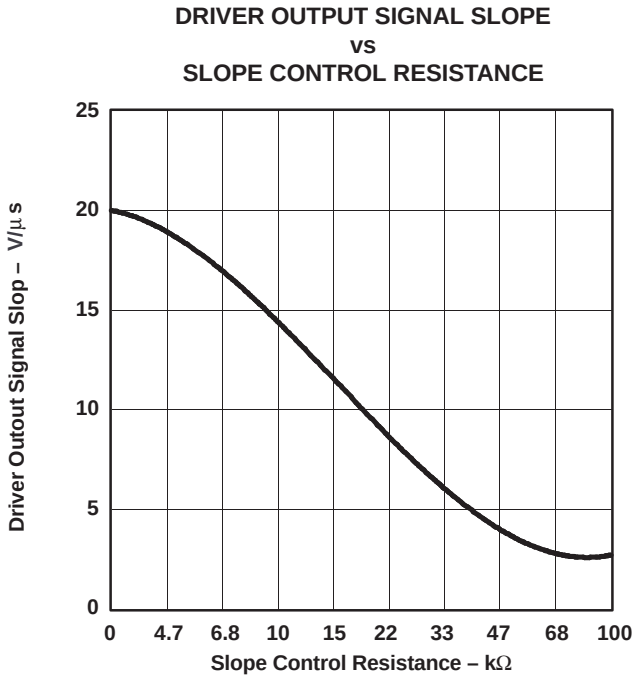


Figure 37. HVD230 Driver Output Signal Slope vs Slope Control Resistance Value

APPLICATION INFORMATION (continued)

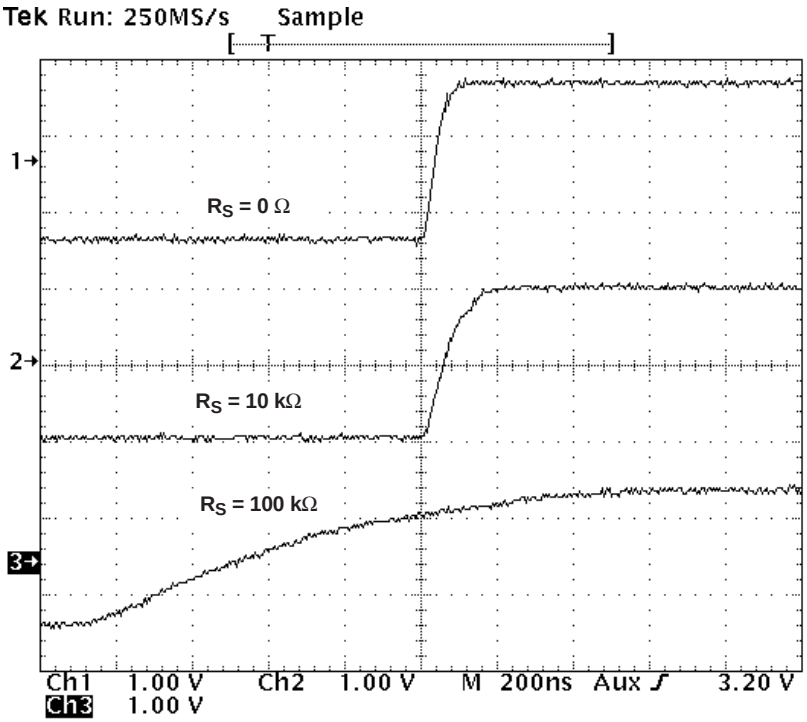


Figure 38. Typical SN65HVD230 250-kbps Output Pulse Waveforms With Slope Control

Standby Mode (Listen Only Mode) of the HVD230

If a logic high (> 0.75 V_{CC}) is applied to R_S (pin 8) in Figure 34 and Figure 36, the circuit of the SN65HVD230 enters a low-current, *listen only* standby mode, during which the driver is switched off and the receiver remains active. In this *listen only* state, the transceiver is completely passive to the bus. It makes no difference if a slope control resistor is in place as shown in Figure 36. The DSP can reverse this low-power standby mode when the rising edge of a dominant state (bus differential voltage > 900 mV typical) occurs on the bus. The DSP, sensing bus activity, reactivates the driver circuit by placing a logic low (< 1.2 V) on R_S (pin 8).

The Babbling Idiot Protection of the HVD230

Occasionally, a runaway CAN controller unintentionally sends messages that completely tie up the bus (what is referred to in CAN jargon as a babbling idiot). When this occurs, the DSP can engage the *listen-only* standby mode to disengage the driver and release the bus, even when access to the CAN controller has been lost. When the driver circuit is deactivated, its outputs default to a high-impedance state.

Sleep Mode of the HVD231

The unique difference between the SN65HVD230 and the SN65HVD231 is that both driver and receiver are switched off in the SN65HVD231 when a logic high is applied to R_S (pin 8). The device remains in a very low power-sleep mode until the circuit is reactivated with a logic low applied to R_S (pin 8). While in this sleep mode, the bus-pins are in a high-impedance state, while the D and R pins default to a logic high.

LOOP PROPAGATION DELAY

Transceiver loop delay is a measure of the overall device propagation delay, consisting of the delay from the driver input to the differential outputs, plus the delay from the receiver inputs to its output.

The loop delay of the transceiver displayed in Figure 39 increases accordingly when slope control is being used. This increased loop delay means that the total bus length must be reduced to meet the CAN bit-timing requirements of the overall system. The loop delay becomes ≈ 100 ns when employing slope control with a

APPLICATION INFORMATION (continued)

10-kΩ resistor, and ≈ 500 ns with a 100-kΩ resistor. Therefore, considering that the rule-of-thumb propagation delay of typical bus cable is 5 ns/m, slope control with the 100-kΩ resistor decreases the allowable bus length by the difference between the 500-ns max loop delay and the loop delay with no slope control, 70.7 ns. This equates to (500-70.7 ns)/5 ns, or approximately 86 m less bus length. This slew-rate/bus length trade-off to reduce electromagnetic interference to adjoining circuits from the bus can also be solved with a quality shielded bus cable.

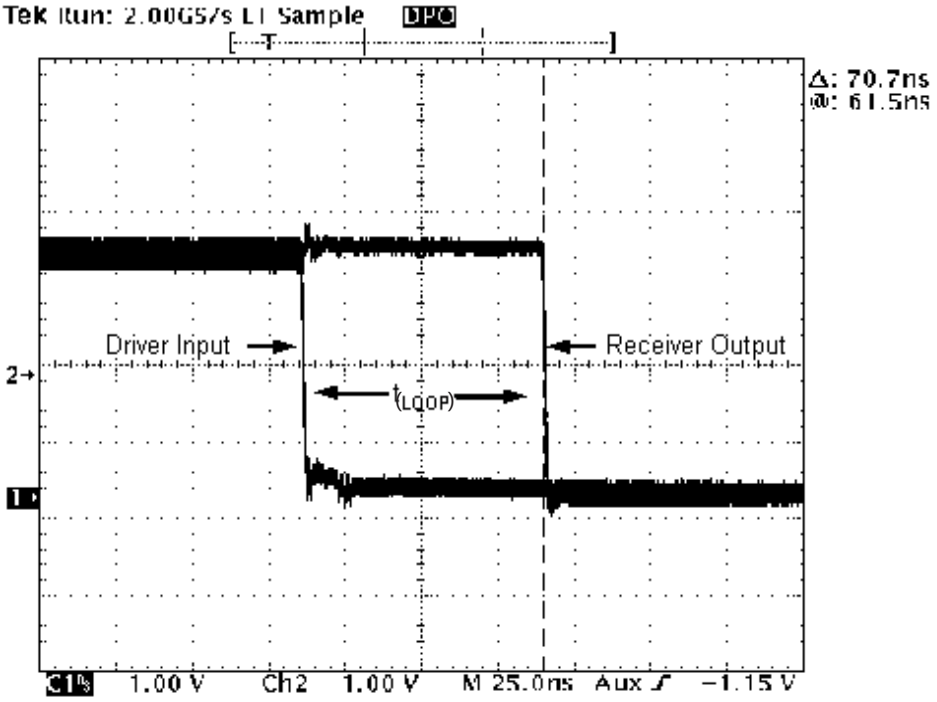


Figure 39. 70.7-ns Loop Delay Through the HVD230 With R_S = 0

INTEROPERABILITY WITH 5-V CAN SYSTEMS

It is essential that the 3.3-V HVD230 family performs seamlessly with 5-V transceivers because of the large number of 5-V devices installed. Figure 40 displays a test bus of a 3.3-V node with the HVD230, and three 5-V nodes: one for each of TI's SN65LBC031 and UC5350 transceivers, and one using a competitor X250 transceiver.

APPLICATION INFORMATION (continued)

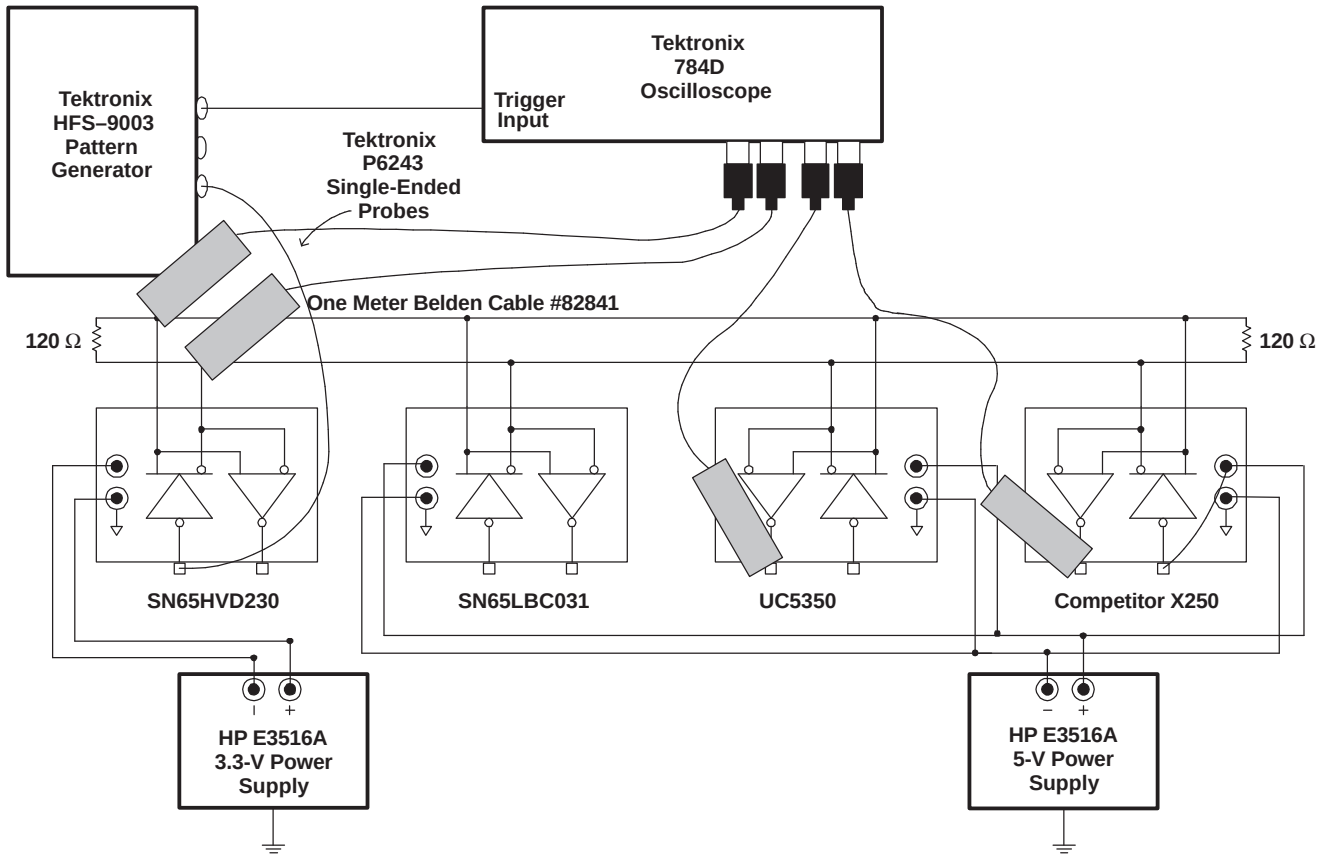


Figure 40. 3.3-V/5-V CAN Transceiver Test Bed

APPLICATION INFORMATION (continued)

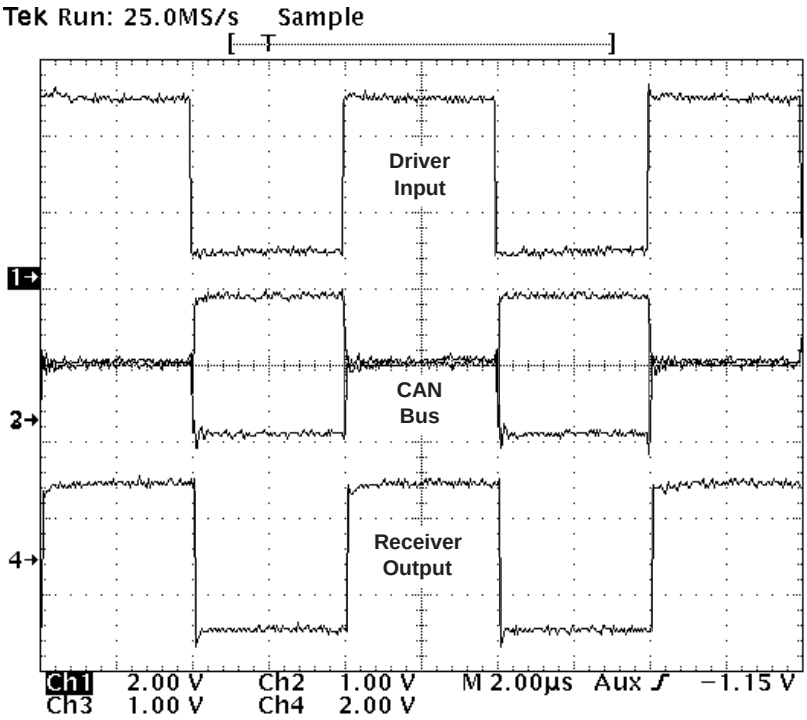


Figure 41. The HVD230's Input, CAN Bus, and X250's RXD Output Waveforms

Figure 41 displays the HVD230's input signal, the CAN bus, and the competitor X250's receiver output waveforms. The input waveform from the Tektronix HFS-9003 Pattern Generator in Figure 40 to the HVD230 is a 250-kbps pulse for this test. The circuit is monitored with Tektronix P6243, 1-GHz single-ended probes in order to display the CAN dominant and recessive bus states.

Figure 41 displays the 250-kbps pulse input waveform to the HVD230 on channel 1. Channels 2 and 3 display CANH and CANL respectively, with their recessive bus states overlaying each other to clearly display the dominant and recessive CAN bus states. Channel 4 is the receiver output waveform of the competitor X250.

PACKAGING INFORMATION

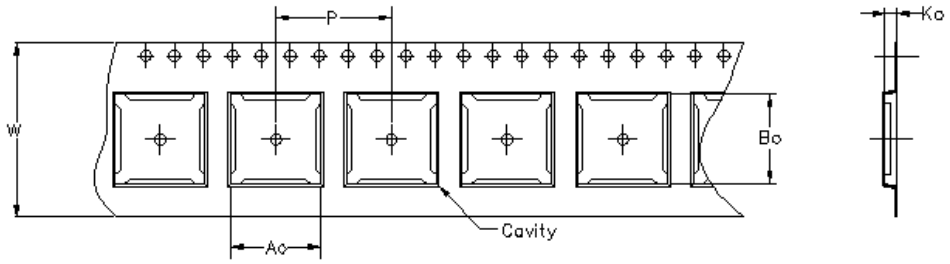
Orderable Device	Status [1]	Package Type	Package Drawing	Pins	Package Qty	Eco Plan [2]	Lead/Ball Finish	MSL Peak Temp [3]
SN65HVD230D	ACTIVE	SOIC	D	8	75	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN65HVD230DG4	ACTIVE	SOIC	D	8	75	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN65HVD230DR	ACTIVE	SOIC	D	8	2500	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN65HVD230DRG4	ACTIVE	SOIC	D	8	2500	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN65HVD231D	ACTIVE	SOIC	D	8	75	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN65HVD231DG4	ACTIVE	SOIC	D	8	75	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN65HVD231DR	ACTIVE	SOIC	D	8	2500	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN65HVD231DRG4	ACTIVE	SOIC	D	8	2500	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN65HVD232D	ACTIVE	SOIC	D	8	75	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN65HVD232DG4	ACTIVE	SOIC	D	8	75	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN65HVD232DR	ACTIVE	SOIC	D	8	2500	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM
SN65HVD232DRG4	ACTIVE	SOIC	D	8	2500	Green (RoHS & no Sb/Br)	CU NIPDAU	Level-1-260C-UNLIM

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

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

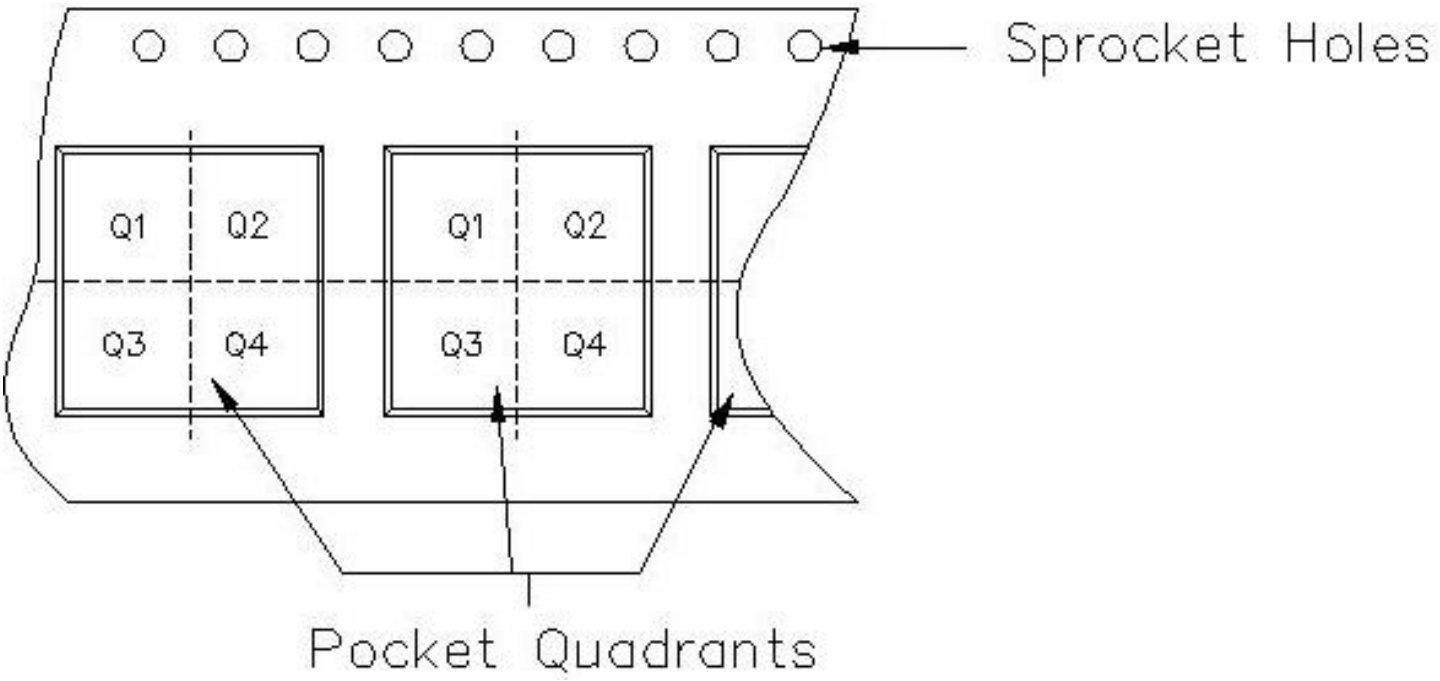
[3] MSL, Peak Temp. -- The Moisture Sensitivity Level rating according to the JEDEC industry standard classifications, and peak solder temperature.

In no event shall TI's liability arising out of such information exceed the total purchase price of the TI part(s) at issue in this document sold by TI to Customer on an annual basis.



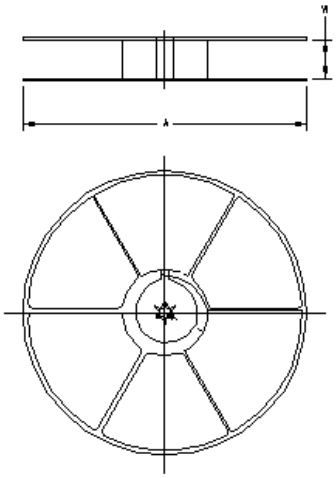
Carrier tape design is defined largely by the component lentgh, width, and thickness.

Ao = Dimension designed to accommodate the component width.
Bo = Dimension designed to accommodate the component length.
Ko = Dimension designed to accommodate the component thickness.
W = Overall width of the carrier tape.
P = Pitch between successive cavity centers.



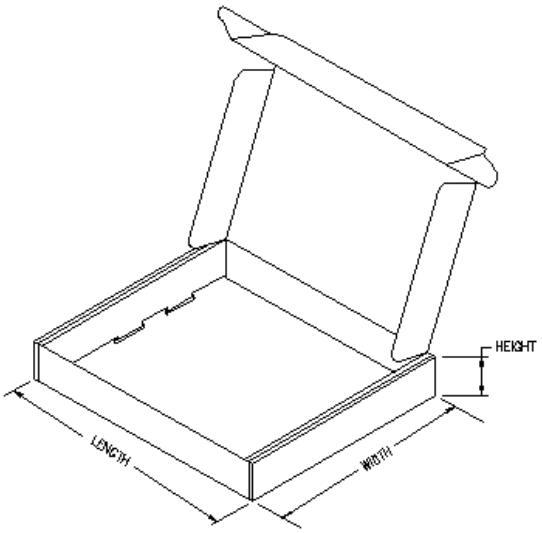
TAPE AND REEL INFORMATION

Device	Package	Pins	Site	Reel Diameter (mm)	Reel Width (mm)	A0 (mm)	B0 (mm)	K0 (mm)	P1 (mm)	W (mm)	Pin1 Quadrant
SN65HVD230DR	D	8	FMX	330	0	6.4	5.2	2.1	8	12	Q1
SN65HVD231DR	D	8	FMX	330	0	6.4	5.2	2.1	8	12	Q1
SN65HVD232DR	D	8	FMX	330	0	6.4	5.2	2.1	8	12	Q1

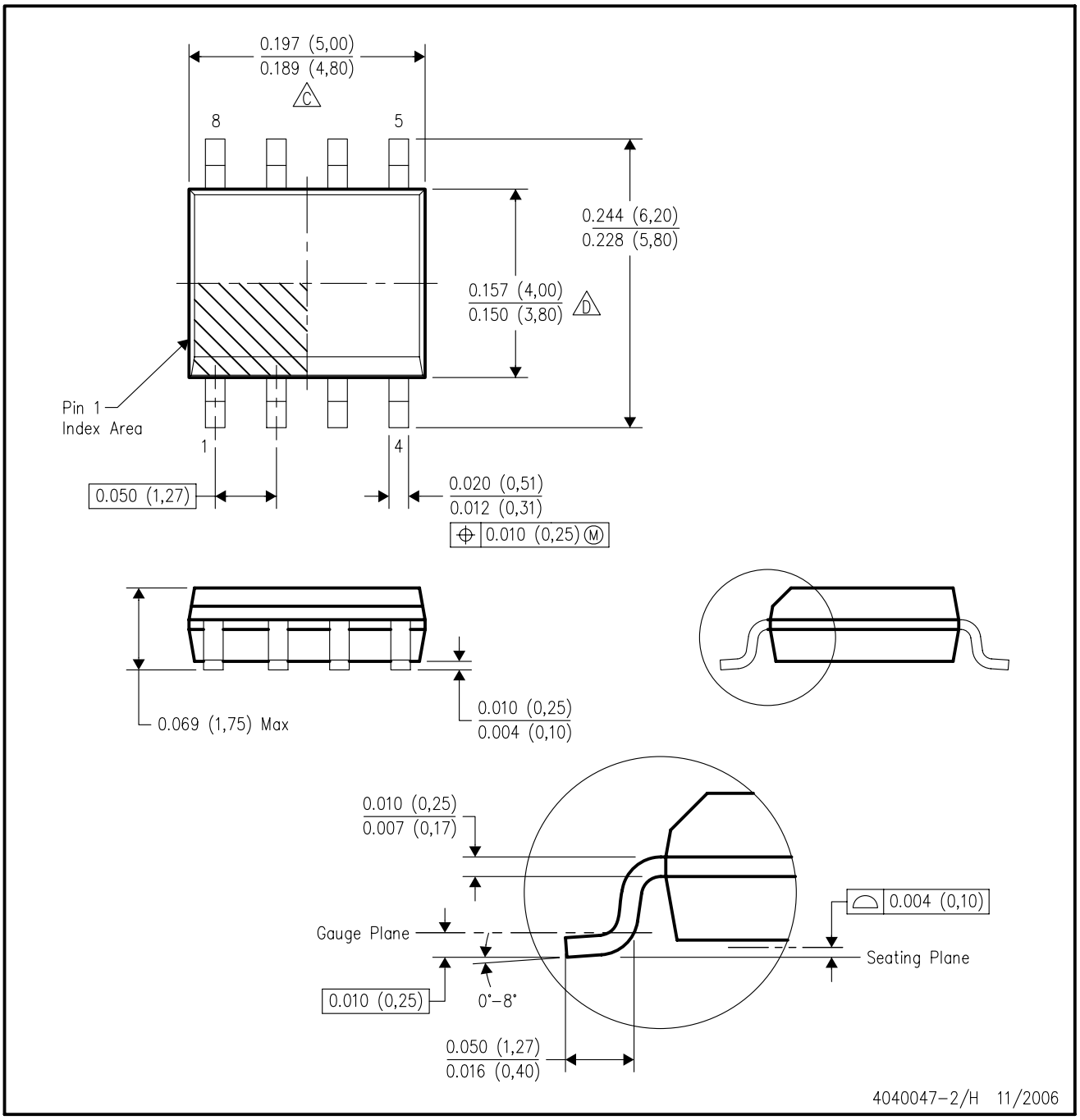


TAPE AND REEL BOX INFORMATION

Device	Package	Pins	Site	Length (mm)	Width (mm)	Height (mm)
SN65HVD230DR	D	8	FMX	342.9	336.6	20.64
SN65HVD231DR	D	8	FMX	342.9	336.6	20.64
SN65HVD232DR	D	8	FMX	342.9	336.6	20.64


HEIGHT
LENGTH
WIDTH

D (R–PDSO–G8) PLASTIC SMALL–OUTLINE PACKAGE



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

IMPORTANT NOTICE

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

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

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

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

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

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

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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

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

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

DATA SHEET

TJA1050
High speed CAN transceiver

Product specification
Supersedes data of 2002 May 16

2003 Oct 22

Philips
Semiconductors

PHILIPS

High speed CAN transceiver TJA1050

FEATURES

- Fully compatible with the *"ISO 11898"* standard
- High speed (up to 1 Mbaud)
- Very low ElectroMagnetic Emission (EME)
- Differential receiver with wide common-mode range for high ElectroMagnetic Immunity (EMI)
- An unpowered node does not disturb the bus lines
- Transmit Data (TXD) dominant time-out function
- Silent mode in which the transmitter is disabled
- Bus pins protected against transients in an automotive environment
- Input levels compatible with 3.3 V and 5 V devices
- Thermally protected
- Short-circuit proof to battery and to ground
- At least 110 nodes can be connected.

GENERAL DESCRIPTION

The TJA1050 is the interface between the Controller Area Network (CAN) protocol controller and the physical bus. The device provides differential transmit capability to the bus and differential receive capability to the CAN controller.

The TJA1050 is the third Philips high-speed CAN transceiver after the PCA82C250 and the PCA82C251. The most important differences are:

- Much lower electromagnetic emission due to optimal matching of the output signals CANH and CANL
- Improved behaviour in case of an unpowered node
- No standby mode.

This makes the TJA1050 eminently suitable for use in nodes that are in a power-down situation in partially powered networks.

QUICK REFERENCE DATA

SYMBOL	PARAMETER	CONDITIONS	MIN.	MAX.	UNIT
V_{CC}	supply voltage		4.75	5.25	V
V_{CANH}	DC voltage at pin CANH	$0 < V_{CC} < 5.25$ V; no time limit	–27	+40	V
V_{CANL}	DC voltage at pin CANL	$0 < V_{CC} < 5.25$ V; no time limit	–27	+40	V
$V_{i(dif)(bus)}$	differential bus input voltage	dominant	1.5	3	V
$t_{PD(TXD-RXD)}$	propagation delay TXD to RXD	V_S = 0 V; see Fig.7	–	250	ns
T_{vj}	virtual junction temperature		–40	+150	°C

ORDERING INFORMATION

TYPE NUMBER	PACKAGE		
	NAME	DESCRIPTION	VERSION
TJA1050T	SO8	plastic small outline package; 8 leads; body width 3.9 mm	SOT96-1
TJA1050U	–	bare die; die dimensions 1700 × 1280 × 380 μm	–

BLOCK DIAGRAM

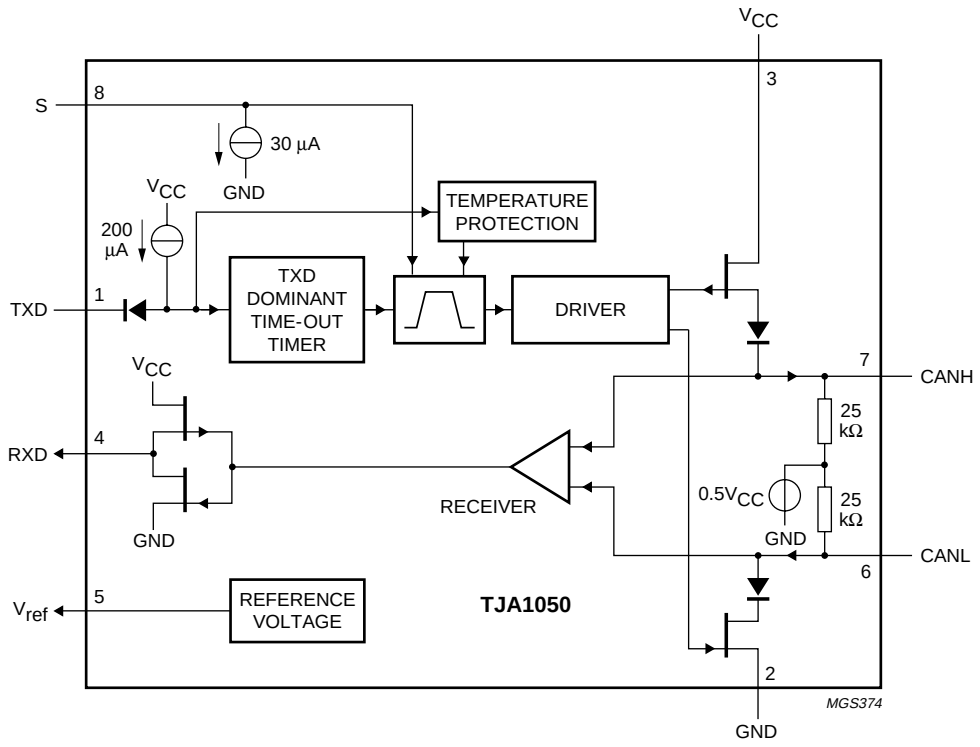


Fig.1 Block diagram.

PINNING

SYMBOL	PIN	DESCRIPTION
TXD	1	transmit data input; reads in data from the CAN controller to the bus line drivers
GND	2	ground
V_{CC}	3	supply voltage
RXD	4	receive data output; reads out data from the bus lines to the CAN controller
V_{ref}	5	reference voltage output
CANL	6	LOW-level CAN bus line
CANH	7	HIGH-level CAN bus line
S	8	select input for high-speed mode or silent mode

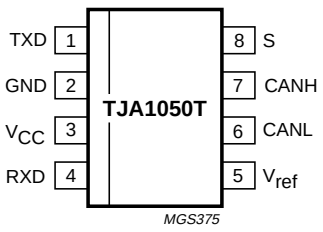


Fig.2 Pin configuration.

FUNCTIONAL DESCRIPTION

The TJA1050 is the interface between the CAN protocol controller and the physical bus. It is primarily intended for high-speed automotive applications using baud rates from 60 kbaud up to 1 Mbaud. It provides differential transmit capability to the bus and differential receiver capability to the CAN protocol controller. It is fully compatible to the *"ISO 11898"* standard.

A current-limiting circuit protects the transmitter output stage from damage caused by accidental short-circuit to either positive or negative supply voltage, although power dissipation increases during this fault condition.

A thermal protection circuit protects the IC from damage by switching off the transmitter if the junction temperature exceeds a value of approximately 165 °C. Because the transmitter dissipates most of the power, the power dissipation and temperature of the IC is reduced. All other IC functions continue to operate. The transmitter off-state resets when pin TXD goes HIGH. The thermal protection circuit is particularly needed when a bus line short-circuits.

The pins CANH and CANL are protected from automotive electrical transients (according to *"ISO 7637"*; see Fig.4).

Control pin S allows two operating modes to be selected: high-speed mode or silent mode.

The high-speed mode is the normal operating mode and is selected by connecting pin S to ground. It is the default mode if pin S is not connected. However, to ensure EMI performance in applications using only the high-speed mode, it is recommended that pin S is connected to ground.

In the silent mode, the transmitter is disabled. All other IC functions continue to operate. The silent mode is selected by connecting pin S to V_{CC} and can be used to prevent network communication from being blocked, due to a CAN controller which is out of control.

A 'TXD dominant time-out' timer circuit prevents the bus lines being driven to a permanent dominant state (blocking all network communication) if pin TXD is forced permanently LOW by a hardware and/or software application failure. The timer is triggered by a negative edge on pin TXD. If the duration of the LOW-level on pin TXD exceeds the internal timer value, the transmitter is disabled, driving the bus into a recessive state. The timer is reset by a positive edge on pin TXD.

Table 1 Function table of the CAN transceiver; X = don't care

V_{CC}	TXD	S	CANH	CANL	BUS STATE	RXD
4.75 V to 5.25 V	LOW	LOW (or floating)	HIGH	LOW	dominant	LOW
4.75 V to 5.25 V	X	HIGH	$0.5V_{CC}$	$0.5V_{CC}$	recessive	HIGH
4.75 V to 5.25 V	HIGH (or floating)	X	$0.5V_{CC}$	$0.5V_{CC}$	recessive	HIGH
<2 V (not powered)	X	X	$0 V < V_{CANH} < V_{CC}$	$0 V < V_{CANL} < V_{CC}$	recessive	X
$2 V < V_{CC} < 4.75 V$	>2 V	X	$0 V < V_{CANH} < V_{CC}$	$0 V < V_{CANL} < V_{CC}$	recessive	X

LIMITING VALUES

In accordance with the Absolute Maximum Rating System (IEC 60134). All voltages are referenced to GND (pin 2). Positive currents flow into the IC.

SYMBOL	PARAMETER	CONDITIONS	MIN.	MAX.	UNIT
V_{CC}	supply voltage		–0.3	+6	V
V_{CANH}	DC voltage at pin CANH	$0 < V_{CC} < 5.25$ V; no time limit	–27	+40	V
V_{CANL}	DC voltage at pin CANL	$0 < V_{CC} < 5.25$ V; no time limit	–27	+40	V
V_{TXD}	DC voltage at pin TXD		–0.3	V_{CC} + 0.3	V
V_{RXD}	DC voltage at pin RXD		–0.3	V_{CC} + 0.3	V
V_{ref}	DC voltage at pin V_{ref}		–0.3	V_{CC} + 0.3	V
V_S	DC voltage at pin S		–0.3	V_{CC} + 0.3	V
$V_{trt(CANH)}$	transient voltage at pin CANH	note 1	–200	+200	V
$V_{trt(CANL)}$	transient voltage at pin CANL	note 1	–200	+200	V
V_{esd}	electrostatic discharge voltage at all pins	note 2	–4000	+4000	V
		note 3	–200	+200	V
T_{stg}	storage temperature		–55	+150	°C
T_{vj}	virtual junction temperature	note 4	–40	+150	°C

Notes

1. The waveforms of the applied transients shall be in accordance with *"ISO 7637 part 1"*, test pulses 1, 2, 3a and 3b (see Fig.4).
2. Human body model: C = 100 pF and R = 1.5 kΩ.
3. Machine model: C = 200 pF, R = 10 Ω and L = 0.75 μH.
4. In accordance with *"IEC 60747-1"*. An alternative definition of T_{vj} is: $T_{vj} = T_{amb} + P \times R_{th(vj-a)}$, where $R_{th(vj-a)}$ is a fixed value to be used for the calculation of T_{vj}. The rating for T_{vj} limits the allowable combinations of power dissipation (P) and ambient temperature (T_{amb}).

THERMAL CHARACTERISTICS

According to IEC 60747-1.

SYMBOL	PARAMETER	CONDITIONS	VALUE	UNIT
$R_{th(vj-a)}$	thermal resistance from junction to ambient in SO8 package	in free air	145	K/W
$R_{th(vj-s)}$	thermal resistance from junction to substrate of bare die	in free air	50	K/W

QUALITY SPECIFICATION

Quality specification *"SNW-FQ-611 part D"* is applicable.

CHARACTERISTICS

V_{CC} = 4.75 V to 5.25 V; T_{vj} = –40 °C to +150 °C; R_L = 60 Ω unless specified otherwise; all voltages are referenced to GND (pin 2); positive currents flow into the IC; see notes 1 and 2.

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
Supply (pin V_{CC})						
I_{CC}	supply current	dominant; V_{TXD} = 0 V	25	50	75	mA
		recessive; $V_{TXD} = V_{CC}$	2.5	5	10	mA
Transmitter data input (pin TXD)						
V_{IH}	HIGH-level input voltage	output recessive	2.0	–	V_{CC} + 0.3	V
V_{IL}	LOW-level input voltage	output dominant	–0.3	–	+0.8	V
I_{IH}	HIGH-level input current	$V_{TXD} = V_{CC}$	–5	0	+5	µA
I_{IL}	LOW-level input current	V_{TXD} = 0 V	–100	–200	–300	µA
C_i	input capacitance	not tested	–	5	10	pF
Mode select input (pin S)						
V_{IH}	HIGH-level input voltage	silent mode	2.0	–	V_{CC} + 0.3	V
V_{IL}	LOW-level input voltage	high-speed mode	–0.3	–	+0.8	V
I_{IH}	HIGH-level input current	V_S = 2 V	20	30	50	µA
I_{IL}	LOW-level input current	V_S = 0.8 V	15	30	45	µA
Receiver data output (pin RXD)						
I_{OH}	HIGH-level output current	$V_{RXD} = 0.7V_{CC}$	–2	–6	–15	mA
I_{OL}	LOW-level output current	V_{RXD} = 0.45 V	2	8.5	20	mA
Reference voltage output (pin V_{ref})						
V_{ref}	reference output voltage	–50 µA < I_{Vref} < +50 µA	$0.45V_{CC}$	$0.5V_{CC}$	$0.55V_{CC}$	V
Bus lines (pins CANH and CANL)						
$V_{o(reces)(CANH)}$	recessive bus voltage at pin CANH	$V_{TXD} = V_{CC}$; no load	2.0	2.5	3.0	V
$V_{o(reces)(CANL)}$	recessive bus voltage at pin CANL	$V_{TXD} = V_{CC}$; no load	2.0	2.5	3.0	V
$I_{o(reces)(CANH)}$	recessive output current at pin CANH	–27 V < V_{CANH} < +32 V; 0 V < V_{CC} < 5.25 V	–2.0	–	+2.5	mA
$I_{o(reces)(CANL)}$	recessive output current at pin CANL	–27 V < V_{CANL} < +32 V; 0 V < V_{CC} < 5.25 V	–2.0	–	+2.5	mA
$V_{o(dom)(CANH)}$	dominant output voltage at pin CANH	V_{TXD} = 0 V	3.0	3.6	4.25	V
$V_{o(dom)(CANL)}$	dominant output voltage at pin CANL	V_{TXD} = 0 V	0.5	1.4	1.75	V
$V_{i(dif)(bus)}$	differential bus input voltage ($V_{CANH} - V_{CANL}$)	V_{TXD} = 0 V; dominant; 42.5 Ω < R_L < 60 Ω	1.5	2.25	3.0	V
		$V_{TXD} = V_{CC}$; recessive; no load	–50	0	+50	mV

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
$I_{o(sc)(CANH)}$	short-circuit output current at pin CANH	V_{CANH} = 0 V; V_{TXD} = 0 V	–45	–70	–95	mA
$I_{o(sc)(CANL)}$	short-circuit output current at pin CANL	V_{CANL} = 36 V; V_{TXD} = 0 V	45	70	100	mA
$V_{i(dif)(th)}$	differential receiver threshold voltage	–12 V < V_{CANL} < +12 V; –12 V < V_{CANH} < +12 V; see Fig.5	0.5	0.7	0.9	V
$V_{i(dif)(hys)}$	differential receiver input voltage hysteresis	–12 V < V_{CANL} < +12 V; –12 V < V_{CANH} < +12 V; see Fig.5	50	70	100	mV
$R_{i(cm)(CANH)}$	common mode input resistance at pin CANH		15	25	35	kΩ
$R_{i(cm)(CANL)}$	common mode input resistance at pin CANL		15	25	35	kΩ
$R_{i(cm)(m)}$	matching between pin CANH and pin CANL common mode input resistance	V_{CANH} = V_{CANL}	–3	0	+3	%
$R_{i(dif)}$	differential input resistance		25	50	75	kΩ
$C_{i(CANH)}$	input capacitance at pin CANH	V_{TXD} = V_{CC}; not tested	–	7.5	20	pF
$C_{i(CANL)}$	input capacitance at pin CANL	V_{TXD} = V_{CC}; not tested	–	7.5	20	pF
$C_{i(dif)}$	differential input capacitance	V_{TXD} = V_{CC}; not tested	–	3.75	10	pF
$I_{LI(CANH)}$	input leakage current at pin CANH	V_{CC} = 0 V; V_{CANH} = 5 V	100	170	250	µA
$I_{LI(CANL)}$	input leakage current at pin CANL	V_{CC} = 0 V; V_{CANL} = 5 V	100	170	250	µA
Thermal shutdown						
$T_{j(sd)}$	shutdown junction temperature		155	165	180	°C
Timing characteristics (see Figs.6 and 7)						
$t_{d(TXD-BUSon)}$	delay TXD to bus active	V_S = 0 V	25	55	110	ns
$t_{d(TXD-BUSoff)}$	delay TXD to bus inactive	V_S = 0 V	25	60	95	ns
$t_{d(BUSon-RXD)}$	delay bus active to RXD	V_S = 0 V	20	50	110	ns
$t_{d(BUSoff-RXD)}$	delay bus inactive to RXD	V_S = 0 V	45	95	155	ns
$t_{dom(TXD)}$	TXD dominant time for time-out	V_{TXD} = 0 V	250	450	750	µs

Notes

1. All parameters are guaranteed over the virtual junction temperature range by design, but only 100 % tested at 125 °C ambient temperature for dies on wafer level and in addition to this 100 % tested at 25 °C ambient temperature for cased products, unless specified otherwise.
2. For bare die, all parameters are only guaranteed if the backside of the bare die is connected to ground.

APPLICATION AND TEST INFORMATION

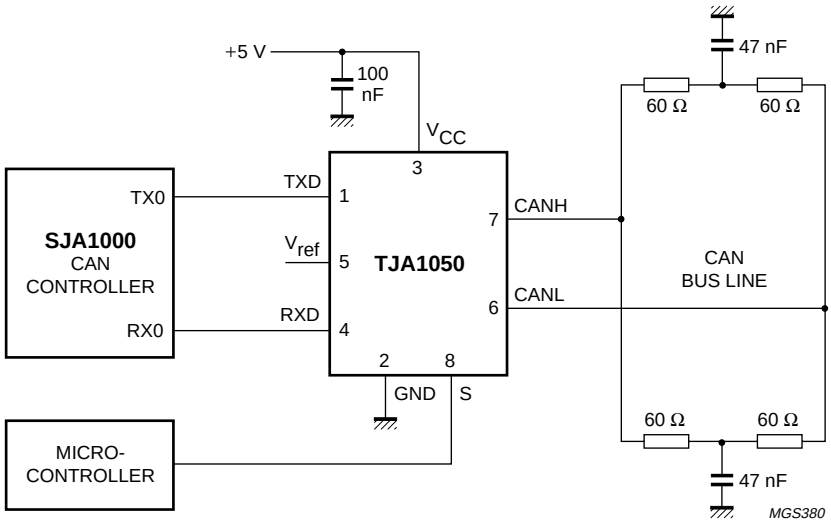
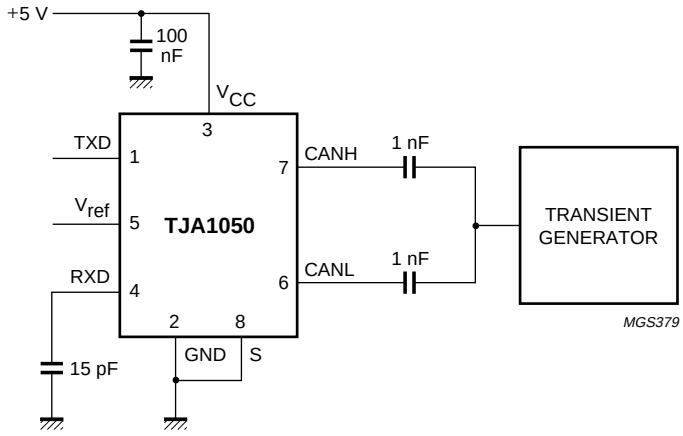


Fig.3 Application information.



The waveforms of the applied transients shall be in accordance with *"ISO 7637 part 1"*, test pulses 1, 2, 3a and 3b.

Fig.4 Test circuit for automotive transients.

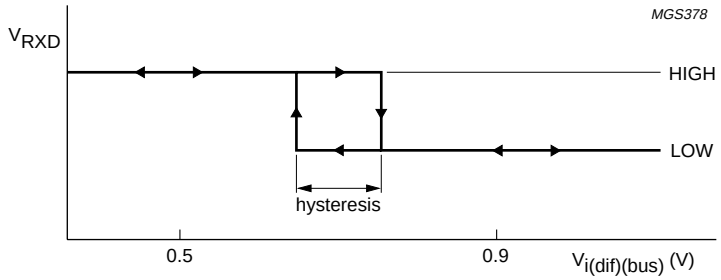


Fig.5 Hysteresis of the receiver.

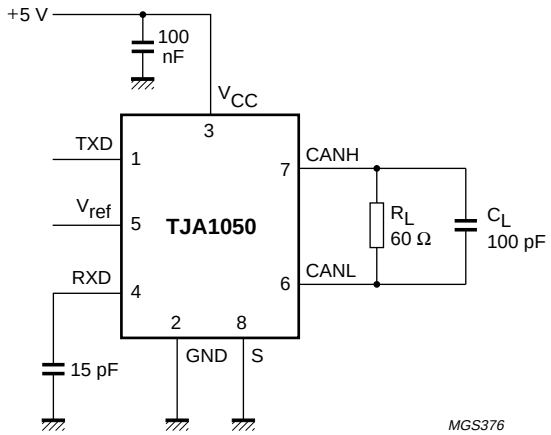
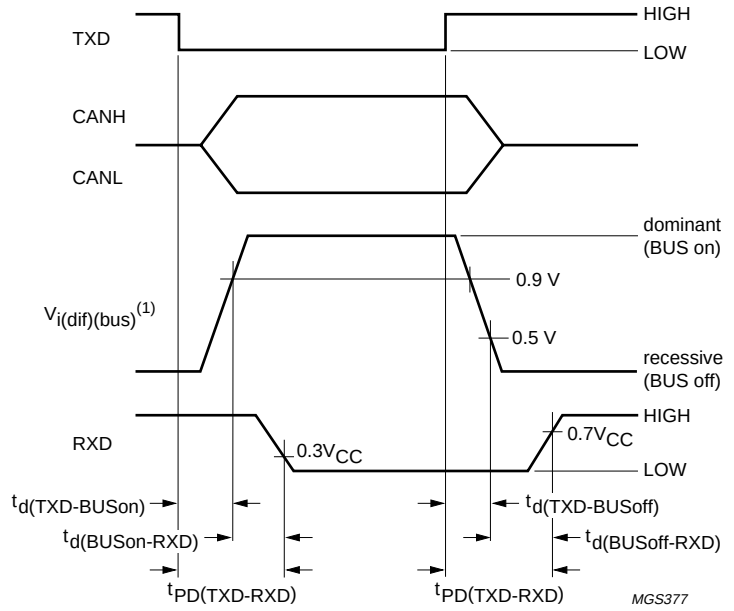


Fig.6 Test circuit for timing characteristics.



(1) $V_{i(dif)(bus)} = V_{CANH} - V_{CANL}$.

Fig.7 Timing diagram for AC characteristics.

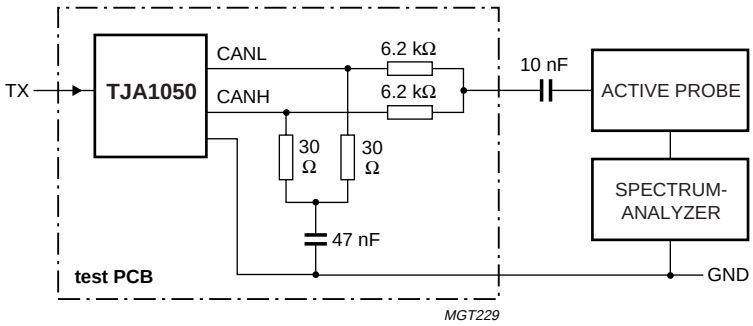
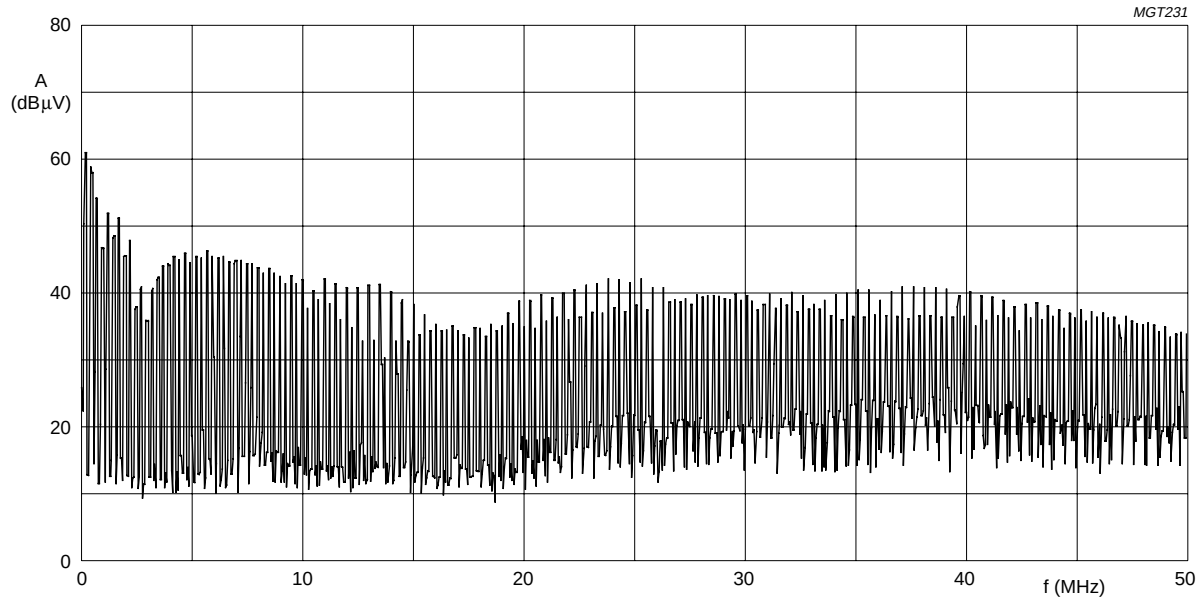
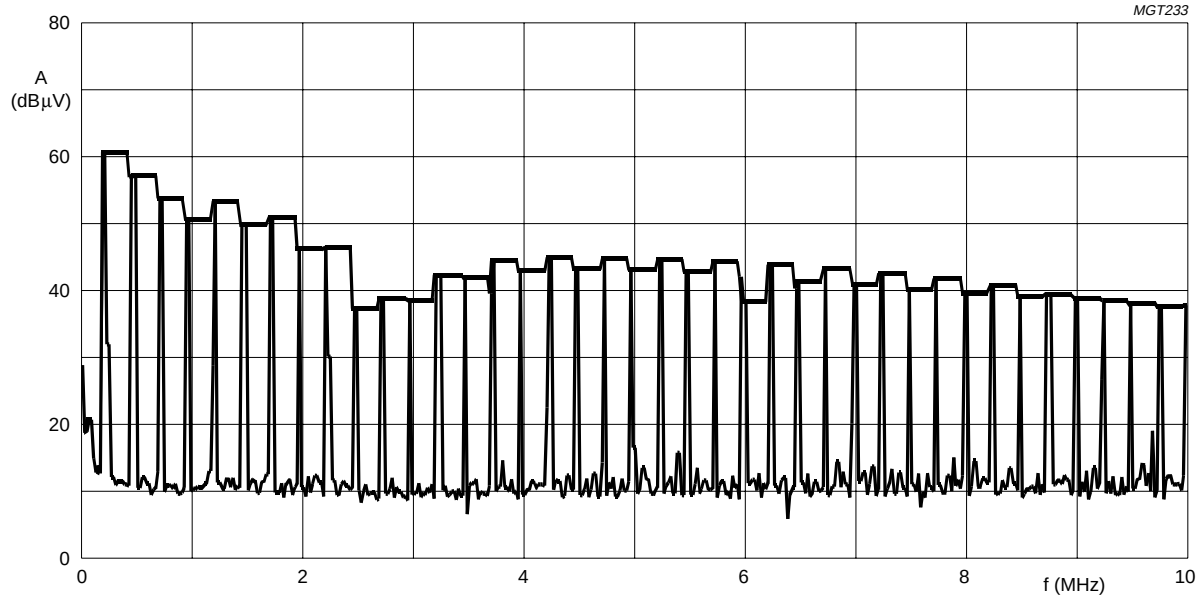


Fig.8 Basic test set-up (with split termination) for electromagnetic emission measurement (see Figs 9 and 10).



Data rate of 500 kbits/s.

Fig.9 Typical electromagnetic emission up to 50 MHz (peak amplitude measurement).



Data rate of 500 kbits/s.

Fig.10 Typical electromagnetic emission up to 10 MHz (peak amplitude measurement and envelope on peak amplitudes).

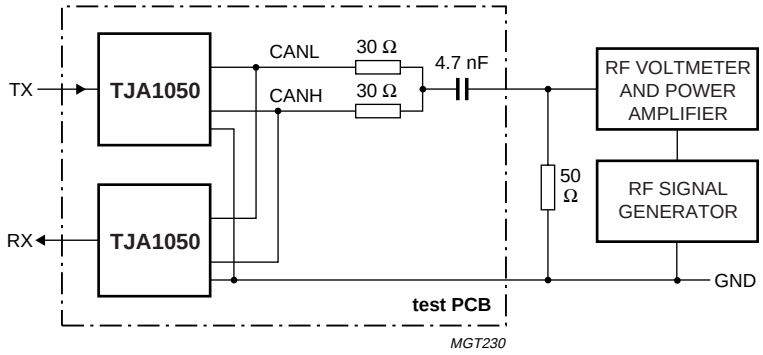
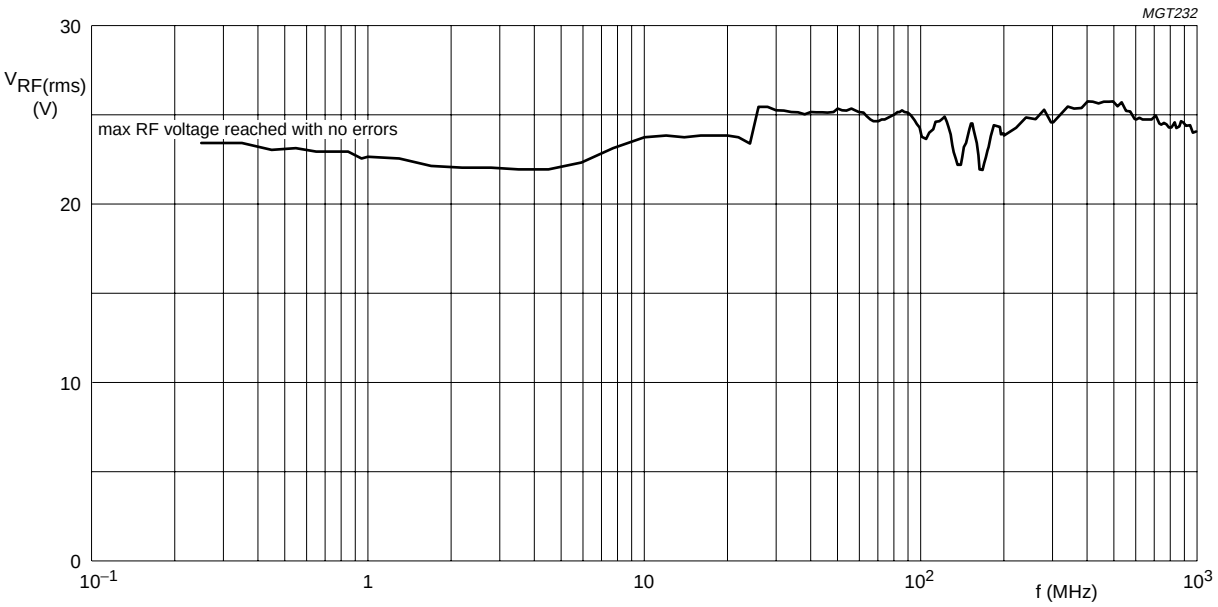


Fig.11 Basic test set-up for electromagnetic immunity measurement (see Fig.12).



Data rate of 500 kbits/s.

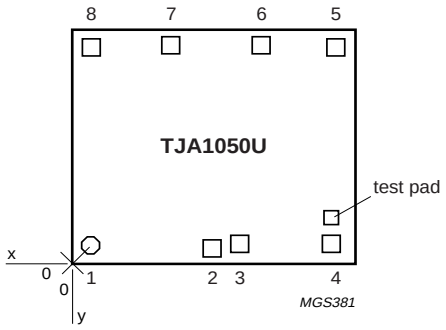
Fig.12 Typical electromagnetic immunity.

BONDING PAD LOCATIONS

SYMBOL	PAD	COORDINATES(1)	
		x	y
TXD	1	103	103
GND	2	740	85
V_{CC}	3	886.5	111
RXD	4	1371.5	111
V_{ref}	5	1394	1094
CANL	6	998	1115
CANH	7	538.5	1115
S	8	103	1097

Note

1. All x/y coordinates represent the position of the centre of each pad (in μm) with respect to the lefthand bottom corner of the top aluminium layer (see Fig.13).

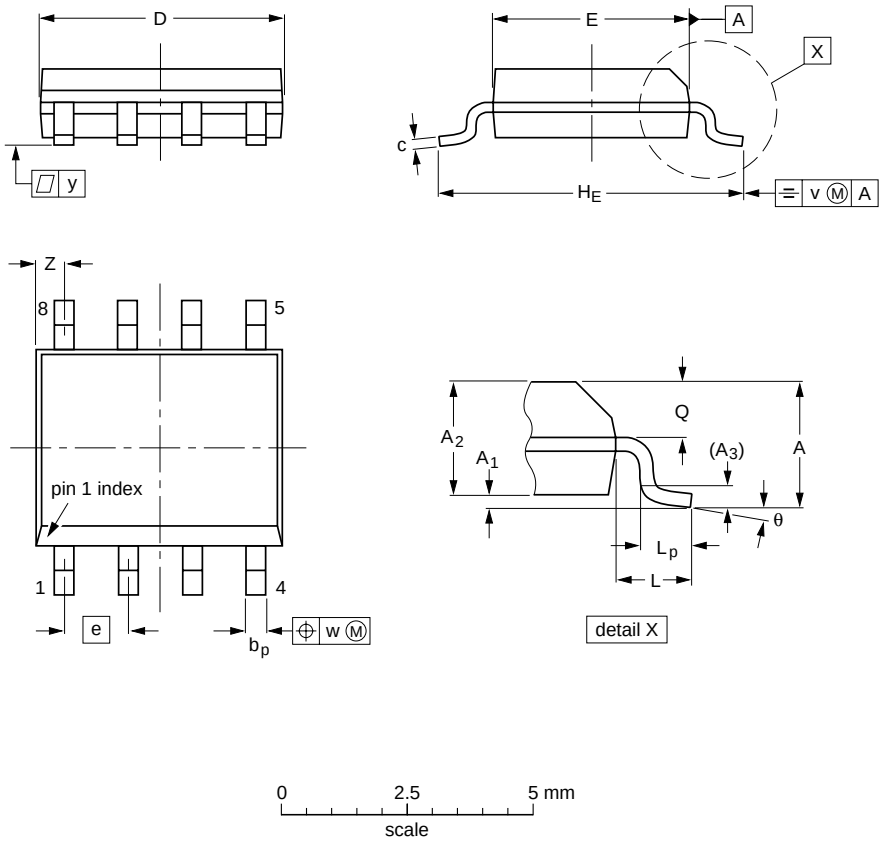


The backside of the bare die must be connected to ground.

Fig.13 Bonding pad locations.

PACKAGE OUTLINE

SO8: plastic small outline package; 8 leads; body width 3.9 mm **SOT96-1**



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

UNIT	A max.	A_1	A_2	A_3	b_p	c	$D^{(1)}$	$E^{(2)}$	e	H_E	L	L_p	Q	v	w	y	$Z^{(1)}$	θ
mm	1.75	0.25 0.10	1.45 1.25	0.25	0.49 0.36	0.25 0.19	5.0 4.8	4.0 3.8	1.27	6.2 5.8	1.05	1.0 0.4	0.7 0.6	0.25	0.25	0.1	0.7 0.3	8^o 0^o
inches	0.069	0.010 0.004	0.057 0.049	0.01	0.019 0.014	0.0100 0.0075	0.20 0.19	0.16 0.15	0.05	0.244 0.228	0.041	0.039 0.016	0.028 0.024	0.01	0.01	0.004	0.028 0.012	

Notes

1. Plastic or metal protrusions of 0.15 mm (0.006 inch) maximum per side are not included.
2. Plastic or metal protrusions of 0.25 mm (0.01 inch) maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	JEITA			
SOT96-1	076E03	MS-012				~~99-12-27~~ 03-02-18

SOLDERING

Introduction to soldering surface mount packages

This text gives a very brief insight to a complex technology. A more in-depth account of soldering ICs can be found in our *"Data Handbook IC26; Integrated Circuit Packages"* (document order number 9398 652 90011).

There is no soldering method that is ideal for all surface mount IC packages. Wave soldering can still be used for certain surface mount ICs, but it is not suitable for fine pitch SMDs. In these situations reflow soldering is recommended.

Reflow soldering

Reflow soldering requires solder paste (a suspension of fine solder particles, flux and binding agent) to be applied to the printed-circuit board by screen printing, stencilling or pressure-syringe dispensing before package placement. Driven by legislation and environmental forces the worldwide use of lead-free solder pastes is increasing.

Several methods exist for reflowing; for example, convection or convection/infrared heating in a conveyor type oven. Throughput times (preheating, soldering and cooling) vary between 100 and 200 seconds depending on heating method.

Typical reflow peak temperatures range from 215 to 270 °C depending on solder paste material. The top-surface temperature of the packages should preferably be kept:

- below 220 °C (SnPb process) or below 245 °C (Pb-free process)
  - for all BGA and SSOP-T packages
  - for packages with a thickness ≥ 2.5 mm
  - for packages with a thickness < 2.5 mm and a volume ≥ 350 mm^3 so called thick/large packages.
- below 235 °C (SnPb process) or below 260 °C (Pb-free process) for packages with a thickness < 2.5 mm and a volume < 350 mm^3 so called small/thin packages.

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

Wave soldering

Conventional single wave soldering is not recommended for surface mount devices (SMDs) or printed-circuit boards with a high component density, as solder bridging and non-wetting can present major problems.

To overcome these problems the double-wave soldering method was specifically developed.

If wave soldering is used the following conditions must be observed for optimal results:

- Use a double-wave soldering method comprising a turbulent wave with high upward pressure followed by a smooth laminar wave.
- For packages with leads on two sides and a pitch (e):
  - larger than or equal to 1.27 mm, the footprint longitudinal axis is **preferred** to be parallel to the transport direction of the printed-circuit board;
  - smaller than 1.27 mm, the footprint longitudinal axis **must** be parallel to the transport direction of the printed-circuit board.

  The footprint must incorporate solder thieves at the downstream end.
- For packages with leads on four sides, the footprint must be placed at a 45° angle to the transport direction of the printed-circuit board. The footprint must incorporate solder thieves downstream and at the side corners.

During placement and before soldering, the package must be fixed with a droplet of adhesive. The adhesive can be applied by screen printing, pin transfer or syringe dispensing. The package can be soldered after the adhesive is cured.

Typical dwell time of the leads in the wave ranges from 3 to 4 seconds at 250 °C or 265 °C, depending on solder material applied, SnPb or Pb-free respectively.

A mildly-activated flux will eliminate the need for removal of corrosive residues in most applications.

Manual soldering

Fix the component by first soldering two diagonally-opposite end leads. Use a low voltage (24 V or less) soldering iron applied to the flat part of the lead. Contact time must be limited to 10 seconds at up to 300 °C.

When using a dedicated tool, all other leads can be soldered in one operation within 2 to 5 seconds between 270 and 320 °C.

Suitability of surface mount IC packages for wave and reflow soldering methods

PACKAGE[1]	SOLDERING METHOD	
	WAVE	REFLOW[2]
BGA, LBGA, LFBGA, SQFP, SSOP-T[3], TFBGA, VFBGA	not suitable	suitable
DHVQFN, HBCC, HBGA, HLQFP, HSQFP, HSOP, HTQFP, HTSSOP, HVQFN, HVSON, SMS	not suitable[4]	suitable
PLCC[5], SO, SOJ	suitable	suitable
LQFP, QFP, TQFP	not recommended[5][6]	suitable
SSOP, TSSOP, VSO, VSSOP	not recommended[7]	suitable
PMFP[8]	not suitable	not suitable

Notes

1. For more detailed information on the BGA packages refer to the *"(LF)BGA Application Note*" (AN01026); order a copy from your Philips Semiconductors sales office.
2. All surface mount (SMD) packages are moisture sensitive. Depending upon the moisture content, the maximum temperature (with respect to time) and body size of the package, there is a risk that internal or external package cracks may occur due to vaporization of the moisture in them (the so called popcorn effect). For details, refer to the Drypack information in the *"Data Handbook IC26; Integrated Circuit Packages; Section: Packing Methods"*.
3. These transparent plastic packages are extremely sensitive to reflow soldering conditions and must on no account be processed through more than one soldering cycle or subjected to infrared reflow soldering with peak temperature exceeding 217 °C ± 10 °C measured in the atmosphere of the reflow oven. The package body peak temperature must be kept as low as possible.
4. These packages are not suitable for wave soldering. On versions with the heatsink on the bottom side, the solder cannot penetrate between the printed-circuit board and the heatsink. On versions with the heatsink on the top side, the solder might be deposited on the heatsink surface.
5. If wave soldering is considered, then the package must be placed at a 45° angle to the solder wave direction. The package footprint must incorporate solder thieves downstream and at the side corners.
6. Wave soldering is suitable for LQFP, TQFP and QFP packages with a pitch (e) larger than 0.8 mm; it is definitely not suitable for packages with a pitch (e) equal to or smaller than 0.65 mm.
7. Wave soldering is suitable for SSOP, TSSOP, VSO and VSSOP packages with a pitch (e) equal to or larger than 0.65 mm; it is definitely not suitable for packages with a pitch (e) equal to or smaller than 0.5 mm.
8. Hot bar or manual soldering is suitable for PMFP packages.

REVISION HISTORY

REV	DATE	CPCN	DESCRIPTION
4	20031013	–	**Product specification (9397 750 12157)** Modification: • Added recommendation to connect unused pin S to ground • Added Chapter REVISION HISTORY
3	20020516	–	**Product specification (9397 750 09778)**

DATA SHEET STATUS

LEVEL	DATA SHEET STATUS(1)	PRODUCT STATUS(2)(3)	DEFINITION
I	Objective data	Development	This data sheet contains data from the objective specification for product development. Philips Semiconductors reserves the right to change the specification in any manner without notice.
II	Preliminary data	Qualification	This data sheet contains data from the preliminary specification. Supplementary data will be published at a later date. Philips Semiconductors reserves the right to change the specification without notice, in order to improve the design and supply the best possible product.
III	Product data	Production	This data sheet contains data from the product specification. Philips Semiconductors reserves the right to make changes at any time in order to improve the design, manufacturing and supply. Relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN).

Notes

1. Please consult the most recently issued data sheet before initiating or completing a design.
2. The product status of the device(s) described in this data sheet may have changed since this data sheet was published. The latest information is available on the Internet at URL http://www.semiconductors.philips.com.
3. For data sheets describing multiple type numbers, the highest-level product status determines the data sheet status.

DEFINITIONS

Short-form specification — The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information see the relevant data sheet or data handbook.

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

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

DISCLAIMERS

Life support applications — These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes in the products - including circuits, standard cells, and/or software - described or contained herein in order to improve design and/or performance. When the product is in full production (status ‘Production’), relevant changes will be communicated via a Customer Product/Process Change Notification (CPCN). Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no licence or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Philips Semiconductors – a worldwide company

Contact information

For additional information please visit **http://www.semiconductors.philips.com**. Fax: **+31 40 27 24825**
For sales offices addresses send e-mail to: **sales.addresses@www.semiconductors.philips.com**.

Printed in The Netherlands R16/04/pp18 Date of release: 2003 Oct 22 Document order number: 9397 750 12157

Let's make things better.

Philips
Semiconductors

PHILIPS