Creación de un Master DALI. Control de una/varias luminarias basado en un microcontrolador Cortex M4.

Pascual Martínez Pérez
Máster Universitario Ingeniería de Telecomunicación

Consultor: Aleix López Antón

Junio 2016.


Esta obra está sujeta a una licencia de
Reconocimiento-NoComercial-SinObraDerivada
3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Creación de un Master DALI. Control de una/varias luminarias basado en un microcontrolador Cortex M4*
Nombre del autor:	*Pascual Martínez Pérez*
Nombre del consultor:	*Aleix López Antón.*
Fecha de entrega:	*06/2015*
Área del Trabajo Final:	
Titulación:	*Máster Universitario Ingeniería de Telecomunicación*

Resumen del Trabajo:

La automatización está evolucionando cada vez más, el hombre ha decidido aplicar la tecnología a todos los lugares para sacar provecho de ella, beneficiándose de las ventajas que le brinda. Para llevar a cabo dichas automatizaciones, siempre nos hacen falta dos elementos básicos, el Control sobre el sistema a automatizar y protocolo de comunicaciones. Los dos elementos son complementarios de tal modo que observamos y controlamos el sistema mediante el protocolo de comunicaciones.

Cuando aplicamos la automatización a diversas índoles, como en la industria podemos hablar de procesos industriales, si la aplicamos a sistemas en general, iluminación, calefacción, posicionamiento, energía etc... Hablamos de control de iluminación, control de calefacción, control ambiental, gps. Etc. Si finalmente aplicamos a viviendas, usamos el termino domótica.

Para el control de cada sistema se desarrollan protocolos de comunicaciones específicos o se integran los protocolos estándares ampliamente conocidos. Existen casos donde los protocolos propietarios se convierten es estándares de facto ya que son usados ampliamente por una gran diversidad de fabricantes.

De todo este abanico de posibilidades que tenemos disponible, para el desarrollo de este proyecto he escogido el área de sistemas generalistas y dentro de ello la aplicación práctica de un protocolo de iluminación. El Protocolo DALI.

Como indica el significado de este acrónimo, Digital Addresable Lighting Interface, DALI, es un interfaz de comunicación digital y direccionable para sistemas de iluminación. DALI Es usado ampliamente por la mayoría de fabricantes de reactancias y sistema de control de luz. Es sistema es un estándar internacional, de acuerdo a la norma IEC 62386, que asegura la compatibilidad e intercambiabilidad entre equipos de diferentes fabricantes.

Este interfaz puede utilizarse en aplicaciones sencillas cómo puede ser el control de una luminaria o una pequeña sala de forma independiente. En aplicaciones de alto nivel, como por ejemplo: oficinas, edificios, barcos, salas de conferencias, etc… Espacios donde se requieren unos requisitos de iluminación flexible. Además se integra fácilmente mediante pasarelas en sistemas de control inteligente de edificios.

Así pues el proyecto a desarrollar consiste primeramente en un "estudio del arte" sobre el protocolo y sus aplicaciones prácticas de hoy en día. Seguidamente haremos un breve estudio técnico sobre la norma y el protocolo asociado a ella. Responderemos a preguntas de bajo nivel tales como. Tipo de señales, voltajes asociados, codificación, velocidades, etc… Pasaremos del protocolo y la norma al sistema. ¿Qué elementos lo integran?, ¿Cómo se produce la comunicación entre ellos?, ¿Qué señales permiten esa comunicación y quien las producen? Finalmente hare una implementación práctica de dicho protocolo. Crearé un Master DALI para el control de una/varias luminarias. Esto sobre una plataforma de desarrollo de Hardware basada en microcontroladro Cortex M4.

Abstract:

Automation is evolving increasingly, the man has decided to apply the technology to all places to take advantage of it, benefiting from the advantages that offers. To carry out these automations, always do we need two basic elements, control over the system to automate and communications protocol. The two elements are complementary so we observe and control the system using the communication protocol.

When we apply automation to various kinds, and in industry we can talk about industrial processes, if we apply it to systems in general, lighting, heating, positioning, energy etc ... We talk lighting control, heating control, environmental control, gps. Etc. If finally we apply to houses, we use the term smart home.

Each control system specific protocols are developed communications widely known or standard protocols are integrated. There are cases where proprietary protocols is become de facto standards as they are widely used by a wide variety of manufacturers.

This whole range of possibilities available to us, for the development of this project I have chosen the area of general systems and within it the practical application of a lighting protocol. The DALI protocol.

As the meaning of this acronym, Digital Addresable Lighting Interface, DALI is a digital interface and communication addressable lighting systems. DALI is widely used by most manufacturers of ballasts and lighting control system. The system is an international standard, according to the IEC 62386 standard, which ensures compatibility and interchangeability between equipment from different manufacturers.

This interface can be used in simple applications can be control how a luminaire or a small living independently. In high-level applications such as offices, buildings, ships, conference rooms, etc ... areas where a flexible lighting requirements are required. In addition it is easily integrated through gateways in intelligent control systems of buildings.

Thus the project to develop consists primarily of a "studio art" on the protocol and its practical applications today. Then we will make a brief technical study on the standard and protocol associated with it. We will answer questions such as low level. Type signals associated voltages, coding, speeds, etc ... We will spend the protocol and standard system. What elements compose ?, how communication occurs between them ?, What signs allow such communication and who produce them? Finally I'll make a practical implementation of the protocol. I will build a DALI Master control one / several luminaires. This on a development platform based on microcontroladro Hardware Cortex M4.

Palabras Clave:

Iluminación, Dali, Master-Dali, cortex-M.

Índice

Lista de figuras

1. Introducción

1.1 Motivación

Cuando hablamos de productos electrónicos hoy en día, además del diseño de hardware asociado no se puede prescindir del uso de microcontroladores para cualquier tarea de control. Existe cierta dificultad relativa que supone adaptar ambos elementos para la creación de un producto. La motivación para el proyecto viene dada por ese mismo reto de integración del software con un diseño hardware y la posterior construcción de un producto.

1.2 Objetivos del Trabajo

- Conocer el protocolo/norma de iluminación DALI.
- Estudio de una plataforma de desarrollo de hardware basada en microprocesador Cortex-M4.
- Diseño del hardware.
- Implementación de un protocolo en una plataforma de desarrollo. (firmware)
- Estudio de la integración de conectividad para un entorno “real”.
- Creación de un master DALI para el control de luminarias.
- Crear aplicaciones sencillas para entornos móviles/fijos.

1.3 Enfoque y método seguido

Para afrontar el proyecto, el método seguido ha consistido en la división en tareas del trabajo previsto. Estas tareas se corresponden con los siguientes pasos: Inicialmente realizaremos un Estado del arte para conocer algunos aspectos de la iluminación en general. Seguidamente un breve estudio sobre protocolo norma DALI. Posteriormente describiremos los elementos disponibles y necesarios para el desarrollo del proyecto. Una vez tomadas las decisiones sobre las plataformas de desarrollo y el lenguaje de programación, nos centraremos en hacer un estudio de sobre la plataforma de hardware y la realización de varios programas de test para conocer los elementos de los que está compuesta, aspecto no comentado en este proyecto. Lo siguiente consiste en el diseño de todo el hardware necesario para consecuentemente implementar sobre el software.

Una vez tenemos todos los elementos completos realizaremos la maqueta de sistema completo. Para con ello empezar a probar ajustar y modificar los elementos asociados para conseguir la comunicación del sistema. El objetivo finalmente es conseguir apagar encender la luminaria con el prototipo desarrollado.

Para finalizar, intentaremos dotar nuestro sistema de algún tipo de comunicación "exterior". Pensamos en un módulo wifi, y el desarrollo de una aplicación en pc y/o en un móvil. Una vez conseguidos esto, como colofón final consiste en el control de varias luminarias en el mismo bus.

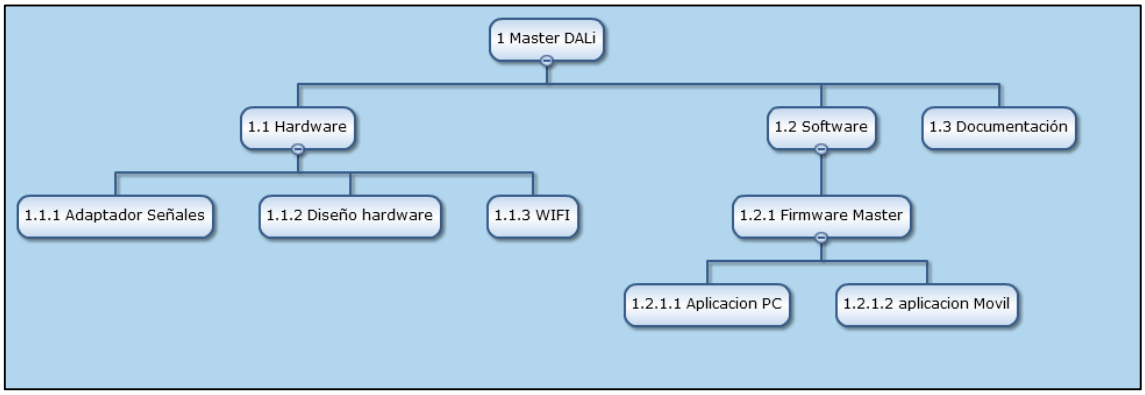


Figura 1: Enfoque y método seguido. Wbs. Resumen

1.4 Productos Obtenidos

Básicamente el resultado final de este proyecto, es la creación de un dispositivo que pueda controlar una lámpara mediante el protocolo/norma Dali. Este producto, es el colofón final de todos los productos intermedios obtenidos que intervienen en su desarrollo tanto materiales como en conocimiento. Por citar alguno de ellos la generación de la documentación, el conocimiento en arquitectura ARM, afianzación de conocimiento en un lenguaje de programación etc…

1.5 Descripción del resto de capítulos de la memoria

Capítulo 2. Estado del arte. Es un breve recorrido que abarca desde los tipos de lámparas hasta los sistemas inteligentes de iluminación.

Capítulo 3. Protocolo/Norma Dali. Consiste en conocer cuáles son las especificaciones que debemos cumplir para tener éxito entre la comunicación de dispositivos Dali.

Capítulo 4. Diseño e Implementación. Abarca todo el proceso dela creación del dispositivo, tanto el diseño de hardware necesario, como la implementación del protocolo en un lenguaje de programación

Capítulo 5. Conclusiones. Una descripción de lo que ha supuesto el proyecto, los problemas encontrados y trabajos futuros tomándolo como base para continuar con su desarrollo.

2. Estado del Arte

2.1 Iluminación.

Una lámpara es un convertidor de energía. Aunque pueda realizar funciones secundarias, su principal propósito es la transformación de energía eléctrica en radiación electromagnética visible. Hay muchas maneras de crear luz, pero el método normalmente utilizado en la iluminación general es la conversión de energía eléctrica en luz.

2.2 Tipos de luz

2.2.1 Incandescencia

Los materiales sólidos y líquidos, al calentarse, emiten radiación visible a temperaturas superiores a 1.000 K; este fenómeno recibe el nombre de incandescencia.

Las lámparas de filamentos se basan en este calentamiento para generar luz: una corriente eléctrica pasa a través de un fino hilo de tungsteno, cuya temperatura se eleva hasta alcanzar entre 2.500 y 3.200 K, en función del tipo de lámpara y su aplicación. Es una temperatura cercana al punto de fusión del tungsteno, que es el material utilizado como filamento, de modo que, en la práctica, el límite de temperatura es de unos 2.700 K, por encima del cual la evaporación del filamento resulta excesiva. Una consecuencia de estos desplazamientos espectrales es que una gran parte de la radiación desprendida no se emite en forma de luz, sino en forma de calor en la región de infrarrojos. Por consiguiente, las bombillas de filamentos pueden ser dispositivos de calefacción eficaces y se utilizan en lámparas diseñadas para secar materiales impresos, preparar alimentos y criar animales.

2.2.2 Descarga eléctrica

La descarga eléctrica es una técnica utilizada en las modernas fuentes de luz para el comercio y la industria, debido a que la producción de luz es más eficaz. Algunos tipos de lámparas combinan la descarga eléctrica con la fotoluminiscencia. Una corriente eléctrica que pasa a través de un gas excita los átomos y moléculas para emitir radiación con un espectro característico de los elementos presentes. Normalmente se utilizan dos metales, sodio y mercurio, porque sus características dan lugar a radiaciones útiles en el espectro visible. Ninguno de estos metales emite un espectro continuo y las lámparas de descarga tienen espectros selectivos. La reproducción del color nunca será idéntica a la obtenida con espectros continuos. Las lámparas de descarga suelen dividirse en las categorías de baja o alta presión, aunque estos términos sólo son relativos, y una lámpara de sodio de alta presión funciona a menos de una atmósfera.

2.3 Evolución de las lámparas eléctricas

Aunque el progreso tecnológico ha permitido producir diferentes lámparas, los principales factores que han influido en su desarrollo han sido fuerzas externas al mercado. Por ejemplo, la producción de las lámparas de filamentos que se utilizaban a principios de siglo sólo fue posible cuando se dispuso de buenas bombas de vacío y del proceso de trefilado del tungsteno. Con todo, fue la generación y distribución de electricidad a gran escala, para satisfacer la demanda de iluminación eléctrica, la que determinó el crecimiento del mercado.

La iluminación eléctrica ofrecía muchas ventajas en comparación con la luz generada por gas o aceite, como la estabilidad de la luz, el escaso mantenimiento, la mayor seguridad que supone no tener una llama desnuda y la ausencia de subproductos locales de combustión.

Durante el período de recuperación que siguió a la segunda Guerra Mundial, lo importante era la productividad. La lámpara fluorescente tubular se convirtió en la fuente de luz dominante porque con ella era posible iluminar fábricas y oficinas sin sombras y comparativamente sin calor, aprovechando al máximo el espacio disponible. En el decenio de 1970 aumentó el precio del petróleo y los costes energéticos se convirtieron en una parte importante de los costes de explotación. El mercado demandaba lámparas fluorescentes que produjesen la misma cantidad de luz con un menor consumo eléctrico, por lo que se perfeccionó el diseño de la lámpara de varias maneras. A medida que se aproxima el fin de siglo, aumenta la conciencia de los problemas ambientales globales. Factores como el mejor aprovechamiento de las materias primas escasas, el reciclaje o la seguridad en el vertido de los productos y la continua preocupación por el consumo de energía (sobre todo de la generada a partir de combustibles fósiles) influyen en el diseño de las lámparas actuales.

Figura 2: Evolución de las Lámparas

2.4 Principales tipos de lámparas

A lo largo de los años, se han ido desarrollando los diferentes tipos de lámparas en función de las exigencias de mercado, dando lugar a los diversos tipos de lámparas de los cuales haremos una breve descripción

2.3.1 Lámparas incandescentes

Utilizan un filamento de tungsteno dentro de un globo de vidrio al vacío o lleno de un gas inerte que evite la evaporación del tungsteno y reduzca el ennegrecimiento del globo. Existen lámparas de muy diversas formas, que pueden resultar muy decorativas. Se trata de unas lámparas que siguen teniendo aceptación en la iluminación doméstica debido a su bajo coste y pequeño tamaño. Con todo, su baja eficiencia genera costes de explotación muy altos en la iluminación comercial e industrial, por lo que normalmente se prefieren las lámparas de descarga. Una lámpara de 100 W tiene una eficiencia típica de 14 lúmenes/vatio en comparación con los 96 lúmenes/vatio de una lámpara fluorescente de 36 W.

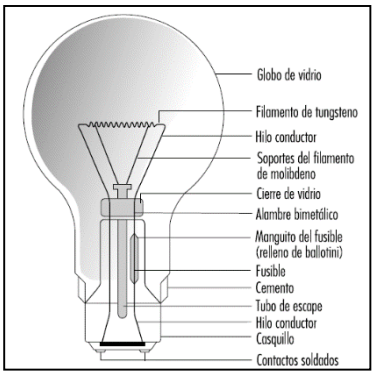


Figura 3: Partes de una lámpara incandescente

Las lámparas incandescentes todavía se utilizan cuando la atenuación de la luz es una característica de control conveniente, ya que resulta fácil atenuarlas reduciendo la tensión de alimentación. El filamento de tungsteno es una fuente de luz de tamaño reducido, que puede enfocarse fácilmente con reflectores o lentes. Las lámparas incandescentes son útiles en la iluminación de expositores, donde se requiere control direccional.

2.3.2 Lámparas halógenas de tungsteno

Son parecidas a las lámparas incandescentes y producen luz de la misma manera, a partir de un filamento de tungsteno. Ahora bien, el globo contiene gas halógeno (bromo o yodo) que actúa controlando la evaporación del tungsteno. Es fundamental para el ciclo del halógeno que la bombilla se mantenga a una temperatura mínima de 250 °C para que el haluro de tungsteno permanezca en estado gaseoso y no se condense sobre la superficie del globo. Tal temperatura da lugar a que las bombillas se fabriquen con cuarzo en lugar de vidrio.

El cuarzo permite reducir el tamaño de la bombilla. La mayoría de las lámparas halógenas de tungsteno duran más tiempo que sus equivalentes incandescentes y el filamento alcanza una temperatura más alta, creando más luz y un color más blanco.

Las lámparas halógenas de tungsteno han encontrado aceptación en situaciones cuyos principales requisitos son un tamaño reducido y un alto rendimiento. Como ejemplo típico cabe citar la iluminación de escenarios, incluyendo el cine y la televisión, donde el control direccional y la atenuación son requisitos habituales.

2.3.4 Lámparas halógenas de tunstegno de baja tensión

Fueron diseñadas originalmente para proyectores de diapositivas y películas. A 12 V, un filamento diseñado para los mismos vatios que en el caso de una corriente de 230 V se hace más pequeño y grueso. Puede enfocarse más eficazmente, y la mayor masa del filamento permite una temperatura de trabajo más alta, aumentando el rendimiento lumínico. El filamento grueso es más robusto. Son características que se han considerado ventajosas en el mercado de los expositores comerciales y, aunque es necesario incorporar un transformador reductor, estas lámparas dominan actualmente la iluminación de escaparates, así como en equipos de proyección.

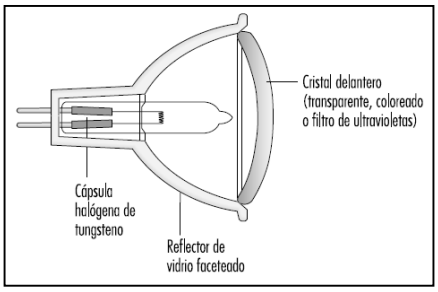


Figura 4: Partes de una lámpara de tunstegno

Sensibilidad a la tensión: Todas las lámparas de filamentos son sensibles a las variaciones de tensión, viéndose afectadas en términos de rendimiento lumínico y vida útil. Las lámparas de descarga también se verán afectadas por tan grandes variaciones de tensión, de modo que será importante la correcta especificación del equipo de control para un buen rendimiento.

2.3.5 Lámparas fluorescentes tubulares

Son lámparas de mercurio de baja presión que están disponibles en versiones de "cátodo caliente" y "cátodo frío".

La primera versión es el tubo fluorescente convencional para fábricas y oficinas; "cátodo caliente" se refiere al cebado de la lámpara por precalentamiento de los electrodos para que la ionización del gas y del vapor de mercurio sea suficiente para realizar la descarga.

Las lámparas de cátodo frío se utilizan principalmente en letreros y anuncios publicitarios .Las lámparas fluorescentes necesitan equipo de control externo para efectuar el cebado y para regular la corriente de la lámpara. Además de la pequeña cantidad de vapor de mercurio, hay un gas de cebado (argón o criptón).

Figura 5: Lámpara fluorescente

2.3.6 Lámparas fluorescentes de tamaño reducido

El tubo fluorescente no es un sustituto práctico para la lámpara incandescente debido a su forma alargada. Pueden hacerse tubos cortos y estrechos de aproximadamente el mismo tamaño que la lámpara incandescente, pero esto impone una carga eléctrica muy superior al material fosfórico. Para que la lámpara tenga una vida útil aceptable es esencial utilizar trifosfóricos.

Algunas lámparas de tamaño reducido incluyen el equipo de control necesario para crear dispositivos de conversión para lámparas incandescentes. La gama va en aumento y permite actualizar fácilmente las instalaciones de alumbrado ya existentes para utilizar más eficazmente la energía. En el caso de que los controles originales lo permitieran, estas unidades integradas no serían adecuadas para el efecto de atenuación.

Equipo electrónico de control de alta frecuencia: si la frecuencia normal de alimentación de 50 o 60 Hz aumenta a 30 kHz, se produce un 10 % de aumento en la eficiencia de los tubos fluorescentes. Los circuitos electrónicos pueden manejar las lámparas individualmente a tales frecuencias. El circuito electrónico está diseñado para proporcionar el mismo rendimiento lumínico que el equipo de control de hilo bobinado, con menor potencia en la lámpara. Con ello es posible compatibilizar el paquete lumínico, con la ventaja de que la menor carga en la lámpara aumentará notablemente la vida útil de ésta. El equipo de control electrónico puede trabajar en toda una gama de tensiones de alimentación. El uso de equipo electrónico de alta frecuencia elimina el problema normal de parpadeo de la luz, al que algunos ocupantes pueden ser sensibles.

2.3.7 Lámparas de inducción

Recientemente han aparecido en el mercado lámparas que utilizan el principio de inducción. Son lámparas de mercurio de baja presión con revestimientos trifosfóricos y cuya producción de luz es similar a la de las lámparas fluorescentes. La energía se transmite a la lámpara por radiación de alta frecuencia, aproximadamente a 2,5 MHz, desde una antena situada en el centro de la lámpara. No existe conexión física entre la bombilla y la bobina. Sin electrodos u otras conexiones alámbricas, la construcción del recipiente de descarga es más sencilla y duradera. La vida útil de la lámpara se determina principalmente por la fiabilidad de los componentes electrónicos y la constancia del flujo luminoso del revestimiento fosfórico.

2.3.8 Lámparas de mercurio de alta presión.

Las descargas de alta presión son más compactas y tienen mayores cargas eléctricas; por consiguiente, requieren tubos de descarga de arco hechos de cuarzo para soportar la presión y la temperatura. El tubo de descarga de arco va dentro de una envoltura exterior de vidrio con una atmósfera de nitrógeno o argón-nitrógeno para reducir la oxidación y el chisporroteo. La bombilla filtra eficazmente la radiación ultravioleta del tubo de descarga de arco.

A todas las lámparas de descarga de alta presión les cuesta alcanzar su pleno rendimiento. Las lámparas de descarga tienen una característica de resistencia negativa, por lo que es necesario el equipo de control externo para regular la corriente. Existen pérdidas debidas a los componentes del estos equipos de control, de modo que el usuario deberá tener en cuenta el voltaje total al estudiar los costes de explotación y la instalación eléctrica. Las lámparas de mercurio de alta presión constituyen una excepción, y uno de sus tipos contiene un filamento de tungsteno que actúa como dispositivo limitador de corriente y además agrega colores cálidos a la descarga verde/azul. Con lo cual, las lámparas incandescentes pueden reemplazarse directamente.

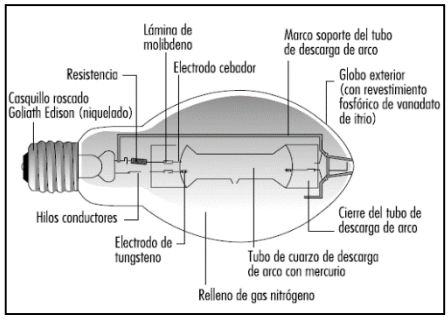


Figura 6:Partes de una lámpara de mercurio

Aunque las lámparas de mercurio tienen una larga vida útil, de alrededor de 20.000 horas, su rendimiento lumínico disminuye hasta aproximadamente el 55 % del inicial al final de este período y, por consiguiente, su vida económica puede ser menor.

2.3.9 Lámparas de haluro metálico

Es posible mejorar el color y el rendimiento lumínico de las lámparas de descarga de mercurio añadiendo diferentes metales al arco de mercurio. La dosis es pequeña en cada lámpara y, a efectos de precisión en la aplicación, es más conveniente manejar los metales en polvo, en forma de haluros, que se disgrega cuando la lámpara se calienta y libera el metal. Una lámpara de haluro metálico puede utilizar varios metales diferentes, cada uno de los cuales emite un color característico específico.

No existe una mezcla estándar de metales, por lo que puede ser que las lámparas de haluro metálico de diferentes fabricantes no sean compatibles en aspecto o funcionamiento. En las lámparas de menor voltaje, de 35 a 150 W, existe una compatibilidad física y eléctrica más próxima a una norma común. Las lámparas de haluro metálico necesitan equipo de control, pero la falta de compatibilidad significa que es necesario combinar bien cada lámpara con su equipo para que las condiciones de cebado y funcionamiento sean correctas.

2.3.10 Lámparas de sodio de baja presión

El tubo de descarga de arco tiene un tamaño similar al tubo fluorescente, pero está hecho de en vidrio contrachapado especial con una capa interior resistente al sodio. El tubo de descarga de arco tiene forma de "U" estrecha y va dentro de una envoltura exterior al vacío para asegurar la estabilidad térmica. Durante el cebado, el gas neón del interior de la lámpara produce un intenso resplandor rojo. La radiación característica del vapor de sodio a baja presión es de un amarillo monocromático. Es un color próximo a la sensibilidad máxima del ojo humano y las lámparas de sodio de baja presión son las más eficaces que existen, a casi 200 lúmenes/vatio. Ahora bien, su aplicación viene limitada por la condición de que la discriminación de los colores no tenga importancia visual, como en el caso de las carreteras principales, los pasos subterráneos y las calles residenciales. En muchas situaciones estas lámparas están siendo reemplazadas por lámparas de sodio de alta presión. Su menor tamaño ofrece mejor control óptico, particularmente en el alumbrado de carreteras, donde existe cada vez mayor preocupación por el excesivo resplandor del cielo.

2.3.11 Lámparas de sodio de alta presión

Son parecidas a las de mercurio de alta presión, pero ofrecen mejor eficiencia (más de 100 lúmenes/vatio) y una excelente constancia del flujo luminoso. La naturaleza reactiva del sodio requiere que el tubo de descarga de arco se fabrique con alúmina policristalina translúcida, ya que el vidrio o el cuarzo son inadecuados. El globo de vidrio exterior contiene un vacío para evitar el chisporroteo y la oxidación. La descarga de sodio no emite radiación ultravioleta, por lo que los revestimientos fosfóricos no tienen ninguna utilidad. Al aumentar la presión del sodio, la radiación se convierte en una banda ancha alrededor del pico amarillo y su coloración es de un blanco dorado. Ahora bien, al aumentar la presión, disminuye la eficiencia. Generalmente, se utilizan las lámparas normales para el alumbrado exterior, las lámparas de lujo para los interiores industriales y las blancas son para aplicaciones comerciales y de exposición.

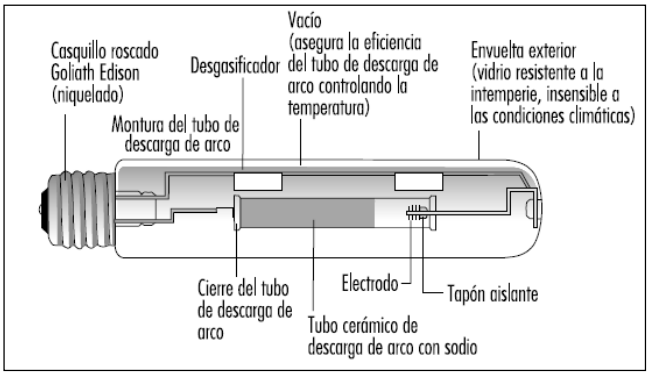


Figura 7: Partes de una lámpara de Sodio

2.3.12 Lámparas de LEDs

Las lámparas de LEDs de luz blanca son unos de los progresos más novedosos en el ámbito de la iluminación. Están muy bien posicionados para poder sustituir a las bombillas actuales. Se trata de un dispositivo semiconductor que emite luz cuando se polariza y es atravesado por la corriente eléctrica. El uso de lámparas basadas en la tecnología LED se está incrementando de una forma notable últimamente, ya que tiene una vida útil más prolongada que cualquier otro tipo de lámpara, una menor fragilidad y un mayor aprovechamiento de la energía.

Algunas características más concretas de este sistema de iluminación son: Su rendimiento es superior a otras lámparas: 100-150 lm/W. Su vida útil se encuentra entre las 50.000 y 100.000 horas. Su IRC es de aproximadamente el 90%. Consiguen una alta fiabilidad. Tienen una respuesta muy rápida. Conllevan menos riesgo para el medio ambiente. Es la tecnología más cara.

Aunque son bastante caros se prevé una rápida evolución. Buena prueba de ello es que los fabricantes cada vez más se decantan por la fabricación de productos basados en la tecnología LED para iluminación de interiores y exteriores, como calles o zonas de estacionamiento.

Figura 8: Lámparas de Led

2.4 Sistemas de iluminación.

Cuando hablamos de sistemas de iluminación, nos referimos a los sistemas que se utilizan para dar luz. Pero un sistema de iluminación no es solo para revelar nuestros alrededores en forma eficiente y segura, hoy por hoy, la iluminación es un arte, es entendida como una forma de crear atmósferas agradables y como un medio para proporcionar confort. La iluminación acentúa las características funcionales y decorativas de un espacio, así como sus proporciones. No existe sólo para mejorar nuestra percepción visual, sino también para estimular nuestros estados de ánimo: ambientes cálidos o fríos, dinámicos o relajantes, felices o solemnes.

2.4.1 Iluminación General

La misma proporciona un nivel uniforme sobre una superficie. En algunos sitios como por ejemplo armarios, cuartos de almacenamiento y garajes, una luminaria o un grupo de ellas, pueden proporcionar toda la iluminación necesaria. Este tipo de iluminación suele utilizarse en áreas donde el estilo y la apariencia son secundarios a los objetos que están siendo iluminados y el costo es un factor decisivo. El requerimiento para una buena distribución de la iluminación general es tener una iluminación horizontal sin sombras.

2.4.2 Iluminación áreas especificas

Proporciona iluminación en áreas específicas de trabajo como escritorios y mostradores. La iluminación de tareas es independiente de la iluminación general, proporcionando una iluminación de mejor calidad para tareas específicas, focalizada directamente en el área de trabajo. La mayoría de las luminarias para tareas son direccionales o locales.

2.4.3 Iluminación arquitectónica.

Caracterizada por acentuar las características y elementos específicos de un espacio en general, como sus paredes, techos, pisos, en vez de los objetos presentes.

2.4.4 Iluminación para acentuación

Es utilizada para realzar características específicas dentro de un espacio, tales como las obras de arte en museos, no debe crear altos niveles de resplandor y brillo.

2.4.5 Iluminación para ambientes

Es utilizada para darle carácter a los espacios a iluminar. Generalmente se caracteriza por la combinación de Los diversos tipos de iluminación: general, arquitectónica, de tareas y de acentuación para así crear atmósferas y espacios de trabajo confortables y estimulantes.

2.5 Regulación y control de sistemas iluminación

Hablar de sistemas de regulación y control del alumbrado es hablar del alumbrado de una sociedad moderna. Bajo la premisa de un uso inteligente de la luz, estos sistemas ofrecen un alumbrado que se adapta a las necesidades de cada instalación y situación, creando ambientes adecuados para cada momento y proporcionando tanto un alto grado de confort como un elevado ahorro de energía.

El ahorro de energía que proporcionan los sistemas de regulación y control del alumbrado, además del ahorro económico, tiene un efecto muy positivo desde el punto de vista ecológico, ya que el menor consumo de energía supone tanto la reducción de emisiones de CO2 como un uso sostenible de los recursos naturales y las fuentes de energía, preservando de esta forma el medioambiente.

2.5.1 Regulación por recorte de fase

Este tipo de regulación se realiza sin necesidad de una línea de control adicional, conectando un regulador en serie entre la línea de alimentación y el equipo.

El regulador recorta parte de la onda sinusoidal de la tensión de red en mayor o menor medida para obtener una regulación de flujo lumínico entre el 1-100%.

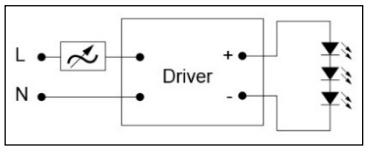


Figura 9: Regulación de alumbrado. Recortador de fase

Dependiendo como se realiza el recorte de la tensión de red se puede distinguir entre dos tipos de regulación:

2.5.2 Regulación al inicio de fase (Leading-edge dimming):

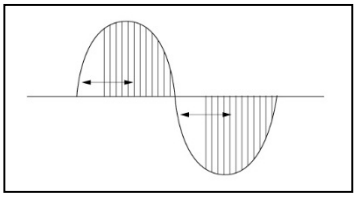


Figura 10: Regulación de alumbrado. Regulación por inicio de fase.

Regulación mediante recorte de la onda de red en su flanco de subida, desde el inicio (corte de fase en el encendido). Es el empleado habitualmente en lámparas halógenas alimentadas a través de transformadores electromagnéticos.

2.5.3 Regulación a final de fase (Trailing-edge dimming):

Regulación mediante recorte de la onda de red en su flanco de bajada, desde el final recortando hacia atrás (corte de fase en el apagado). Es más adecuado para lámparas halógenas alimentadas a través de transformadores electrónicos.

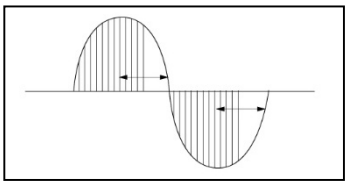


Figura 11: Regulación de alumbrado. Regulación por final de fase.

2.5.4 Reactancia electromagnética de doble nivel de potencia

Los balastros o reactancias para doble nivel, son reactancias de construcción similar a los modelos estándar pero a los cuales se les ha añadido un bobinado adicional sobre un mismo núcleo magnético. De esta forma pueden obtenerse dos niveles de potencia. El sistema con reactancia de doble nivel de potencia, se basan en balastros electromagnéticos como elementos de control inmediato de las lámparas. Esto implica pérdidas de potencia relativamente importantes

2.5.5 Regulación 1-10V:

El sistema 1-10V permite la regulación del flujo luminoso, entre alrededor del 1 y el 100%, mediante una señal analógica que llega a los equipos a través de una línea de control adicional de dos hilos. Estos hilos de control poseen una polaridad positiva y negativa respectivamente que hay que respetar a la hora de realizar el cableado. La señal analógica tiene un valor de tensión continua entre 1V y 10V, obteniéndose el nivel mínimo de luz con 1V o cortocircuitando la entrada de control del equipo, y el máximo nivel de luz con 10V o dejando la entrada de control en circuito abierto.

Mediante la línea de control solo se puede realizar la regulación del flujo luminoso, el encendido y el apagado de la luz, que puede tener lugar en cualquier punto de la regulación, se realiza mediante un interruptor colocado en la línea de alimentación del equipo. Ambas líneas, la de control y la de alimentación, se encuentran separadas eléctricamente entre sí.Para obtener una respuesta adaptada a la respuesta del ojo humano, se pueden usar potenciómetros de control logarítmicos.

En los equipos de iluminación con regulación 1-10V, la potencia de control es generada por éstos. A través de los bornes de control del equipo, se suministra una corriente al controlador que debe estar comprendida entre 10uA y 2mA. La máxima corriente por la línea de control se obtiene con la tensión de 1V y la mínima corriente con 10V.

Este sistema de regulación es unidireccional, es decir la información fluye en un único sentido, desde el controlador hacia el equipo de iluminación, no generando el equipo ningún tipo feedback hacia el control. No permite un direccionamiento vía software de los equipos, teniendo que realizarse la creación de grupos de forma cableada. Este sistema se puede integrar en sistemas de control de edificios.

La longitud del cableado de la línea de control está limitada por la caída de tensión que se produce a lo largo de la misma, por tanto la máxima distancia está limitada por el número de equipos a controlar conectados. Estos últimos fijan la corriente por la línea y la sección del cable usado.

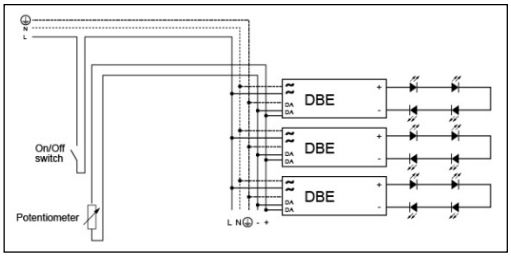


Figura 12: Sistema de regulación 1-10V

2.5.6 Regulación mediante pulsador Touch Control:

Touch Control es un sistema mediante el cual se consigue la regulación del flujo luminoso de una forma sencilla y económica, que utiliza la tensión de red como señal de control, aplicándola, a través de un pulsador estándar normalmente abierto, en una línea de control, sin necesidad de controladores específicos.

El sistema Touch Control permite realizar las funciones básicas de un sistema de regulación mediante el accionamiento de un pulsador libre de potencia. Dependiendo de la duración de la pulsación tiene lugar el encendido/apagado o la regulación de la luz. El encendido/apagado del alumbrado se consigue mediante una pulsación corta o "click" y mediante una pulsación continuada la regulación del flujo luminoso entre el nivel máximo y el mínimo alternativamente.

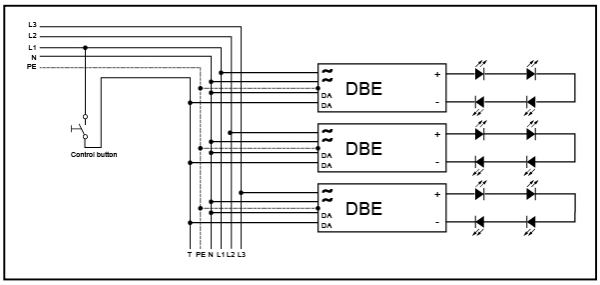


Figura 13: Regulación mediante pulsador Touch Control

Es un interfaz de regulación unidireccional, es decir la información fluye en un único sentido, no generando el equipo ningún tipo de feedback. No permite un direccionamiento vía software de los equipos, teniendo que realizarse la creación de grupos de forma cableada. Este sistema no se puede integrar en sistemas de control de edificios.

Debido a sus características, el uso de este método de regulación está indicado para oficinas individuales, pequeñas salas de conferencias o habitaciones, rellanos y áreas reducidas en general.

2.6 Métodos de regulación y control de alumbrado.

Las instalaciones de gestión del alumbrado tendrán una menor o mayor complejidad dependiendo de la solución escogida para cada una de ellas; del método de control elegido, el número y tipo de componentes, la interconexión entre ellos y la integración con sistemas de control de edificios.

Existen una gran variedad de posibilidades, desde las soluciones más sencillas compuestas por luminarias individuales, dotadas de equipos regulables y fotocélulas conectados directamente entre ellos, que regulan la luz independientemente del resto del alumbrado, hasta los sistemas de gestión del alumbrado más avanzados, integrados en el control inteligente de edificios, que controlan luminarias en diferentes salas y en diferentes plantas con múltiples usos, pudiendo crear diferentes ambientes adaptados a cada situación y reportar información de su estado en cada momento. El control y regulación tienen la implicación de poder seleccionar las lámparas para actuar sobre ellas (gestión), regulación, encender, apagar, color de luz...son características que podemos modificar. Además normalmente estos sistemas disponen de un bus digital.

2.6.1 DALI

Como indica el significado de este acrónimo, Digital Addresable Lighting Interface, DALI es un interfaz de comunicación digital y direccionarle para sistemas de iluminación. Este sistema es un estándar internacional, de acuerdo a la norma IEC 62386, que asegura la compatibilidad e intercambiabilidad entre equipos de diferentes fabricantes, los cuales están marcados con el siguiente logo:

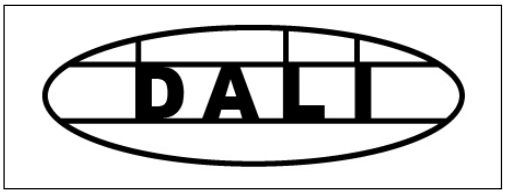


Figura 14: Logotipo DALI

2.6.1.1 Componentes del sistema de control.

Además de la fuente de luz que se pretende controlar, los sistemas de gestión del alumbrado están compuestos por otros componentes adicionales. Entre estos componentes se encuentran los equipos, accionamientos o elementos de mando, sensores, controladores, adaptadores, repetidores, convertidores, pasarelas y las herramientas de configuración y de monitorización.

2.6.1.2 Equipos:

Los equipos de iluminación, drivers para módulos LED, balastos para lámparas de fluorescencia y de descarga, transformadores para lámparas halógenas, son los componentes encargados de hacer funcionar las fuentes de luz de forma correcta. Éstos, para poder integrarse en un sistema de gestión de alumbrado, deben ser regulables por el método de control elegido.

2.6.1.3 Accionamientos o elementos de mando:

Son los componentes mediante los que el usuario interacciona con el sistema de gestión del alumbrado, permitiendo encender, apagar o regular la luz voluntariamente de forma manual y directa. En este grupo se encuentran los pulsadores, los mandos rotativos y paneles de control.

2.6.1.4 Los sensores o detectores:

Son dispositivos capaces de detectar magnitudes físicas o químicas y transformarlas en señales que pueden ser procesadas. En los sistemas de gestión de alumbrado destacan los detectores de presencia y las fotocélulas, mediante los cuales el encendido, apagado o regulación de la luz se realiza de forma automática dependiendo de la presencia de personas y el nivel de luz natural en la estancia.

2.6.1.5 Unidades de control o controladores:

Son los componentes encargados de recibir toda la información procedente del resto de componentes del sistema, procesarla y generar los comandos de control para distribuirlos de forma inteligente.

2.6.1.6 Repetidores:

Son componentes que amplifican el nivel o la potencia de las señales débiles, por lo que, en los sistemas de gestión de alumbrado, se deben utilizar cuando se necesitan mayores distancias de cableado o mayor número de equipos conectados de lo permitido.

2.6.1.7 Herramientas de configuración y de monitorización:

Para los sistemas de gestión del alumbrado más avanzados, son necesarias herramientas software que permitan el direccionamiento, la programación, la parametrización y la monitorización de los mismos.

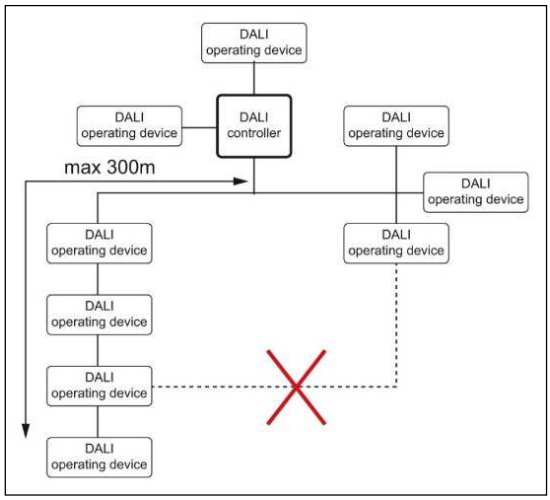


Figura 15: Diagrama de Bloques. Sistema Dali

2.6.2 KNX

KNX es un estándar (ISO/IEC 14543) de protocolo de comunicaciones de red, basado en OSI, para edificios inteligentes (domótica e inmótica). KNX es el sucesor y la convergencia de tres estándares previos: el European Home Systems Protocol (EHS), el European Installation Bus (EIB or 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). El estándar KNX está gestionado por la Asociación KNX.

Figura 16: Logotipo KNX

KNX es un protocolo de comunicaciones de red que está basado en la comunicación mediante cableado a través de bus de datos, que se ha desarrollado para realizar instalaciones domóticas en edificios y grandes infraestructuras, pero que también puede trasladarse a nuestras casas sin mayor complicación que la necesidad de cablear todas las zonas que queramos controlar.

KNX surge para realizar la comunicación de datos a todos los componentes en la gestión de un edificio, casa o grandes instalaciones, eliminando los problemas de dispositivos que se queden fuera de cobertura o aislados por un mal funcionamiento. A través del protocolo KNX todos los dispositivos están intercomunicados de una u otra forma pudiendo intercambiar información entre unos y otros y serán controlados desde un sistema uniforme sin necesidad de centros de control.

Pero si no se necesita de un centro de control como puede ser en la domótica Z-Wave, ¿cómo se controlan los dispositivos? Cada uno de ellos es independiente y está preparado para funcionar de forma individual o para interactuar con otros dispositivos KNX, así que no debemos de preocuparnos de nada sobre estos módulos ya que son inteligentes por sí mismos.

Las unidades de instalación en KNX son líneas y en cada una de estas líneas tenemos la posibilidad de instalar hasta un total de 64 dispositivos como máximo, permitiendo que todos los elementos estén comunicados entre sí y cualquier componente puede mandar sobre otro sin ningún tipo de problema al estar interconectados, sin preocuparnos de la distancia entre ellos, problemas de cobertura o tipos de ubicación.

KNX consta de básicamente 4 grupos de elementos:

2.6.2.1 actuadores

Los actuadores son los elementos del sistema que se conectan físicamente sobre los elementos a controlar en el edificio, por ejemplo las luces, electroválvulas, motores, contactos secos, etc. y hacen la traducción de las instrucciones que viajan del mundo KNX al mundo físico conmutado, regulando o accionando los dispositivos que son controlados.

2.6.2.2 sensores

Los sensores son los elementos del sistema que recogen datos o interpretan órdenes del usuario, por ejemplo pulsador, botonera, detector de movimiento, termostato, anemómetro, sensor crepuscular; muchos sensores incorporan visualizadores o pantallas donde se controla y monitoriza el sistema, como las botoneras o pantallas táctiles.

2.6.2.3 pasarelas

Las pasarelas (gateways o routers) enlazan otros sistemas con otros protocolos de comunicación con KNX, por ejemplo de DALI, BACnet, LONWORKS, RS485, IP, RS232, X10 etc. a KNX. Estos equipos permiten interaccionar con proyectores, otros sistemas inteligentes o incluso comunicarse en remoto con el sistema.

2.6.2.4 acopladores

Estos elementos realizan una separación física dentro del bus consiguiendo agrupar los dispositivos en un segmento de características determinadas para la cantidad de equipos, ubicaciones físicas o funciones determinadas y conectarlo con otro segmento para una mayor eficacia en el envío de datagramas a través del bus, alcanzar mayores distancias (repetidores), además de darle un direccionamiento físico muy entendible utilizando la división de Áreas, grupos y líneas.

2.6.2.5 software

Software de gestión: El software de gestión, es decir el que usaremos para configurar los dispositivos y ponerlos en marcha es el ETS .Nos permite relacionar actuadores con sensores y traducir las comunicaciones a través de las pasarelas. Esta herramienta es la única forma de configurar los dispositivos KNX y es creada, suministrada y regulada únicamente por la KNX Association.

Software de control: Es el programa de cómputo que sirve para tener acceso a la instalación para dotarnos de control y visualización desde un equipo de cómputo que puede tener varias funciones: Visualizar el estado de los elementos, Controlar la instalación, Registrar los eventos, Generar reportes y eventos, Crear funciones lógicas, Servir y dotar información a otros sistemas (interfaz o pasarela), Ejecutar funciones de diagnóstico, escenas y rutinas de comprobación, Otorgar acceso a otras plataformas o métodos de acceso a los sistemas KNX.

2.6.2.6 Aplicaciónes

En la actualidad este tipo de instalaciones se realizan principalmente en edificios de oficinas e industriales para una gestión de energías y automatización de sistemas y en las viviendas para el confort y como tecnología asistiva para ancianos y discapacitados. A pesar que en los últimos años ha bajado de precio de sus elementos, el sistema encarece el precio final de la instalación, aunque a la larga puede otorgar una reducción de consumo eléctrico si la configuración del sistema es eficiente.

Se están diseñando las bases para posiblemente cambiar el bus o encapsularlo dentro del protocolo TCP/IP esto es: KNX/IP. Esto es debido a que el citado protocolo ha ido estandarizándose y absorbiendo buses y protocolos de otros sistemas (CCTV, VOZ ANALÓGICA, etc.). También el crecimiento, implantación y estandarización de TCP/IP hace que esta opción se pueda convertir en el diseño final de KNX, siendo un elemento más cada equipo de este sistema dentro de las redes IP, es decir, conectaríamos los equipos directamente a ethernet y a través del enrutador típico de conexión a Internet podríamos gestionar y monitorizar externamente los sistemas, también nuestros equipos WiFi instalados en el edificio nos darían acceso al sistema de una manera cómoda y con dispositivos estándar (móviles, tabletas, ordenadores, etc.). A diferencia del sistema actual que necesita pasarelas IP, los propios equipos "hablarían" directamente en IP, simplificando las instalaciones pues el cableado más usado hoy es el cableado UTP dado a la implantación ya estandarizada de TCP/IP desde hogares pequeños hasta grandes empresas.

2.6.3 DMX512

DMX512, a menudo abreviado como DMX (Digital MultipleX), es un protocolo electrónico utilizado en luminotecnia para el control de la iluminación de espectáculos, permitiendo la comunicación entre los equipos de control de luces y las propias fuentes de luz.

Desarrollado por la comisión de Ingeniería de USITT, el estándar comenzó en 1986, con posteriores revisiones en 1990 que dieron paso al USITT DMX512/1990. ESTA tomo el control del estándar en 1998 y empezó el proceso de revisión. El nuevo estándar, conocido oficialmente como "Entertainment Technology — USITT DMX512–A — Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories", fue aprobado por ANSI en noviembre del 2004. El actual estándar es también conocido como "E1.11, USITT DMX512–A", o solo "DMX512-A", y es mantenido por la ESTA.

DMX aparece como la solución al problema de la incompatibilidad que existía entre marcas por la utilización de protocolos propietarios, lo cual obligaba a tener un control de manejo por cada marca de luces que se tenía.

DMX fue originalmente pensado para usarlo en controladores de enlace y dimmers de diferentes fabricantes, un protocolo que sería usado como último recurso después de probar otros métodos más en propiedad, no GNU. Sin embargo, pronto se convirtió en el protocolo preferido no sólo para controladores de enlace y dimmers, sino también para controlar aparatos de iluminación como scanners y cabezas móviles, y dispositivos de efectos especiales como máquinas de humo. Como DMX512 es un sistema de transmisión de datos poco fiable, no debe ser usado para controlar Pirotecnia, para esta tarea se usa a veces controladoresMIDI. El protocolo DMX funciona bajo licencia GNU. Las aplicaciones de este protocolo son el control de luces en teatros, museos, discotecas iluminación espectacular.

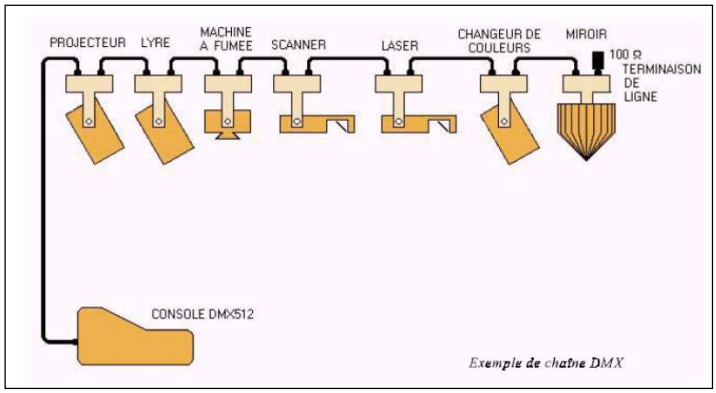


Figura 17: Esquema conexionado DMX512

2.7 Aplicaciones de los sistemas de control y regulación. Control inteligente.

La combinación de lámparas LED y sistemas de control permite conseguir un ahorro energético de hasta el 85% en comparación con instalaciones equipadas con fuentes de luz tradicionales. Por eso es muy importante usar herramientas modernas para gestionar una instalación de iluminación del modo más eficiente.

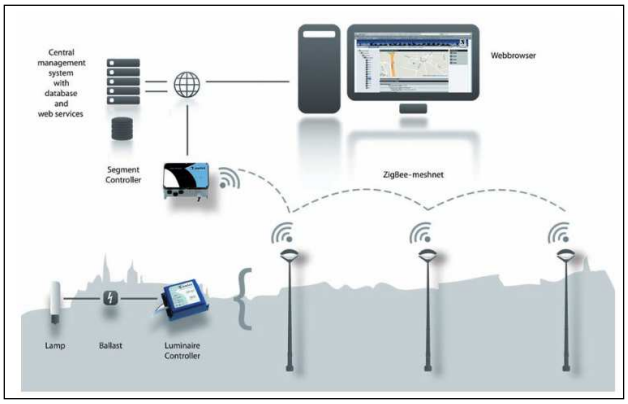


Figura 18: Sistemas de iluminación. Control inteligente

Aplicaciones prácticas de las lámparas, protocolos y herramientas descritos anteriormente, hacen que la gestión de infraestructuras sea de lo más sencillo y eficiente posible.

Figura 19: Sistemas de control inteligente. Aplicaciones prácticas

Los sistemas de Control Inteligente están conectados con las grandes redes de datos de las “smart cities” gracias a las normas de protocolo abiertas y con enfoques de inteligencia distribuida, los nuevos sistemas de iluminación no tienen necesariamente una sola estación de control, sino que pueden programarse como una colonia de hormigas, donde cada miembro trabaja de forma independiente, siguiendo los objetivos generales de la comunidad. De este modo, las estaciones de control pueden ser más pequeñas y numerosas, y las luminarias obtener la capacidad de tomar decisiones individuales, gracias a los sensores y controles integrados.

Figura 20: Sistemas de Control inteligente. Smart Cities

3. Protocolo/Norma Dali

3.1 Introducción a DALI.

DALI es un estándar internacional (IEC 62386) en los sistemas de control de iluminación que proporciona un único interfaz de control electrónico para fuentes de luz y dispositivos como controladores de iluminación.

El estándar DALI permite conectar una amplia variedad de dispositivos como: balastros regulables, transformadores, módulos de relés, accesorios de emergencia y controladores de todos ellos de diferentes fabricantes y combinados en un solo sistema de control. Un sistema DALI proporciona a los diseñadores, instaladores, y usuarios finales gestionar un sistema de iluminación digital de gran alcance y flexible con la seguridad de interconexionado de diferentes fabricantes.

El estándar DALI es supervisado por el grupo de actividad "AG-DALI", que comprende: ingenieros, fabricantes, e instituciones que trabajan en el campo de control e iluminación digital.

Más información sobre el estándar DALI se puede encontrar en los siguientes documentos:
• IEC 62386[1]
• Norma NEMA 243-2004[2]

Las siguientes secciones proporcionan una visión general del protocolo/norma DALI y Describen los principios básicos de su interfaz.

3.2 Propósito y propiedades.

El protocolo DALI fue diseñado para controlar fuentes de luz utilizando un ordenador.
Las funciones incluyen:

- atenuación
- Conexión / desconexión
- La agrupación de luces a un control común
- almacenamiento y selección de escena

Las propiedades de diseño DALI incluyen:

- Cableado simple usando cables de instalación eléctrica estándar
- No topología de cableado especial (como con los cables eléctricos de potencia)
- Instalación simple dar la independencia polaridad del cable
- automatizada fuente de luz frente
- El uso de microcontroladores de bajo coste en el lado de la fuente de luz para minimizar el costo de la fuente de luz
- El uso de un simple protocolo para controlar la luz y oscurecimiento de conmutación.

[1] https://domino.iec.ch/preview/info_iec62386-103%7Bed1.0%7Db.pdf
[2] http://www.energy.ca.gov/2005publications/CEC-500-2005-141/CEC-500-2005-141-A23.PDF

3.3 Especificaciones

A continuación describimos brevemente sus especificaciones, para hacer su posterior implementación:

3.3.1 Capa Física.

La interfaz DALI (bus) consiste en una capa física a partir de dos cables. La polaridad es independiente.

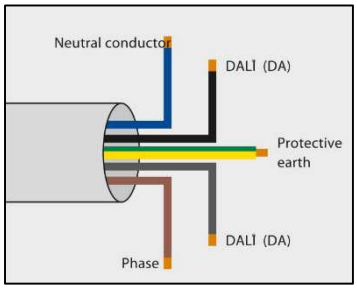


Figura 21: Dali. Capa Física

Los niveles de tensión en los cables de comunicación presentes DALI son más altos que los niveles que se utilizan normalmente usados en lógica (TTL). Esto es debido a una mejor inmunidad al ruido debido a una mayor interferencia presente en los cables de la instalación eléctrica cercanas. Los niveles de tensión se definen como sigue:

Estado de bajo nivel

- -4.5 A la de 4,5 V (transmisor)
- -6.5 A la 6,5 V (receptor)

Estado de alto nivel

- 11,5 a 20,5 V (transmisor)
- 9,5 a 22,5 V (receptor)

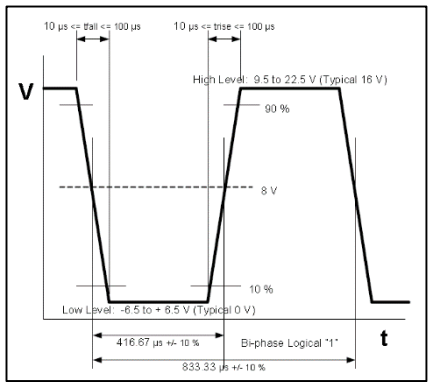


Figura 22:Dali. Niveles de Tensión

Con una codificación Manchester se utiliza para una mejor resincronización. Cada flanco ascendente es un 1 lógico y un flanco descendente es 0 lógico.

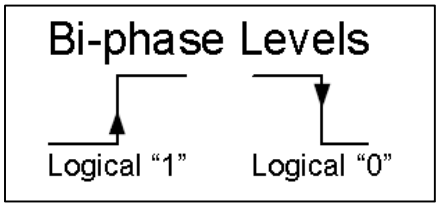


Figura 23:Dali. Codificación Manchester

El protocolo utilizado en estos cables es un protocolo serie estándar. Consta de 1 bit de inicio, 8 de datos 4 bits de parada. La velocidad de comunicación en bits es fija de 1200 Bps.

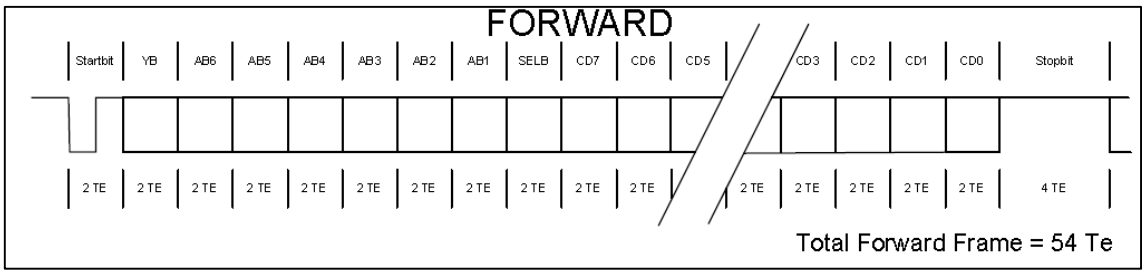


Figura 24: Dali. Trama

Los Bytes se agrupan en tramas. Una trama generalmente consiste de 1 o 2 bytes, los cuales pueden ser solo datos o dirección + datos (comandos hacia los dispositivos). Para el tiempo de bit, tenemos:

$$Te = \frac{1}{2 * 1200} s = 416{,}67 \mu s$$

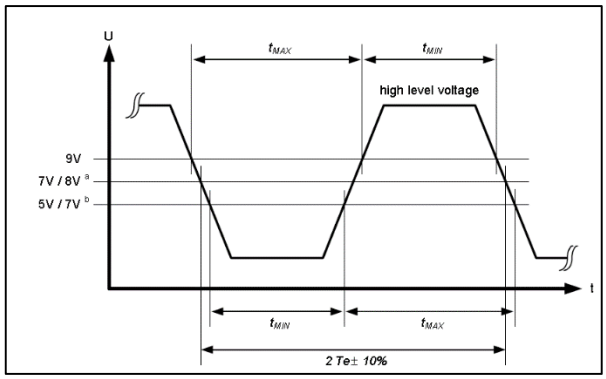


Figura 25: Dali. Diagramade tiempos

Con unas tolerancias del 10%, los límites de los tiempos máximos y mínimos son de:

$$t_{MAX} < 500 \mu s \; y \; t_{MIN} > 334 \mu s$$

Por consiguiente una trama desde el master a los clientes (forward), indicando dirección y dato, tendrá un valor sin contar tolerancias de:

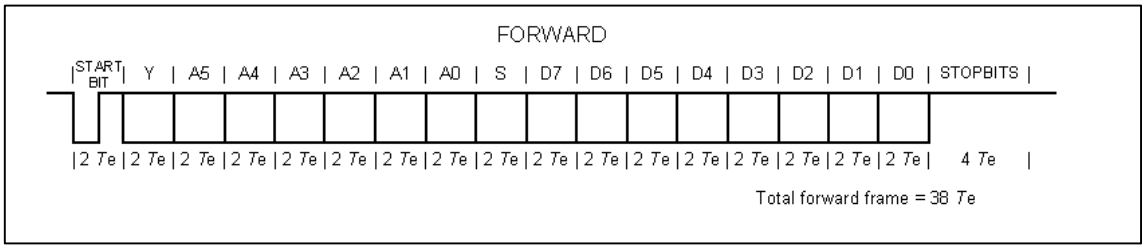


Figura 26. Dali. Trama Forward

$$38Te = 38 * 416{,}67 \mu s = 15{,}83 \, ms$$

Las tramas de respuesta desde los clientes hacia el master (backward) indicando solo el dato solicitado, tendrá un valor de:

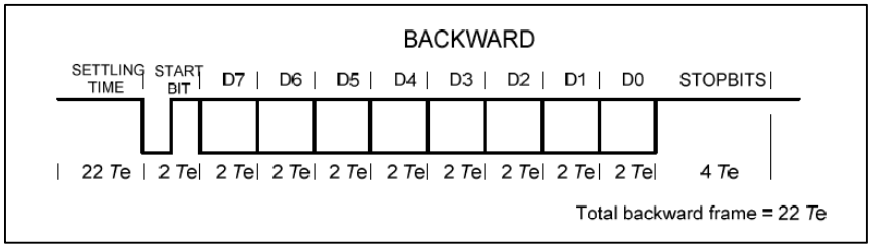


Figura 27. Dali. Trama Backward

$$22Te = 22 * 416{,}67\mu s = 9{,}17\ ms$$

El tiempo entre dos tramas consecutivas se consideran entre el último bit de stop de la trama y el bit de start de la siguiente trama. (Recordamos que hay 4 bits de stop). Los requerimientos de transmisión son:

- El tiempo entre dos tramas consecutivas forward (master -> slave) debe der al menos de 22 Te
- El tiempo entre una trama forward (master -> slave) y una backward (slave -> master) debe der al menos de 7 Te - 22 Te
- El tiempo entre una trama backward (slave -> master) y una trama forward (master -> slave) debe der al menos de 22 Te.

El tiempo entre bits (settling time) de dos tramas consecutivas se considera entre el último flaco de subida de la trama y el primer flanco de bajada de la siguiente trama. (Recordamos que hay 4 bits de stop). Los requerimientos de transmisión son:

- El settling time entre dos tramas consecutivas forward (master -> slave) debe der al menos de 27 Te
- El settling time entre una trama forward (master -> slave) y una backward (slave -> master) debe der al menos de 11 Te - 27 Te
- El settling time entre una trama backward (slave -> master) y una trama forward (master -> slave) debe der al menos de 27 Te.

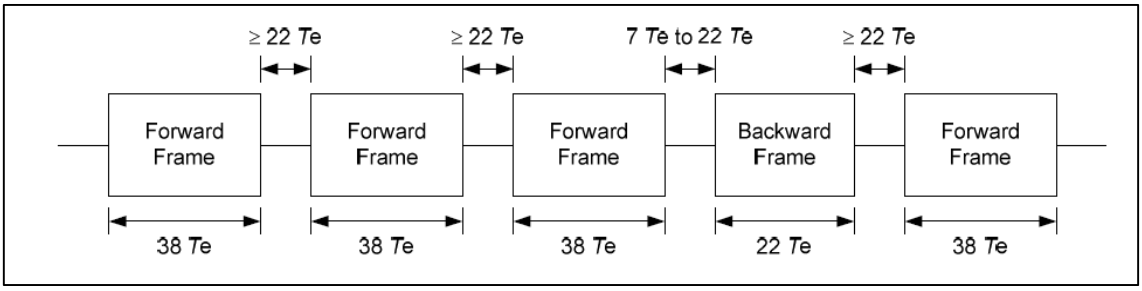


Figura 28. Dali Tiempos de tramas.

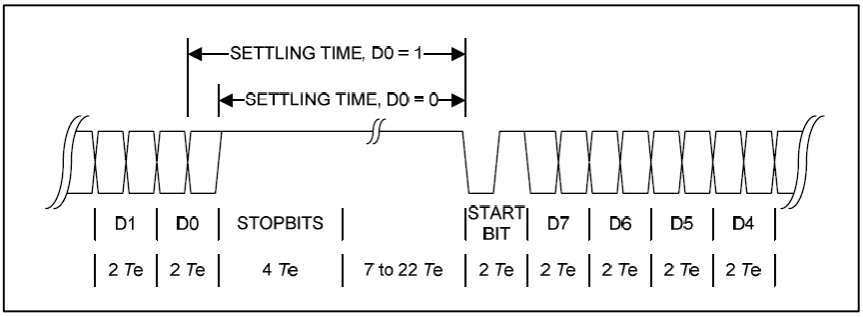


Figura 29: Dali. Settling Time entre tramas forward, backward

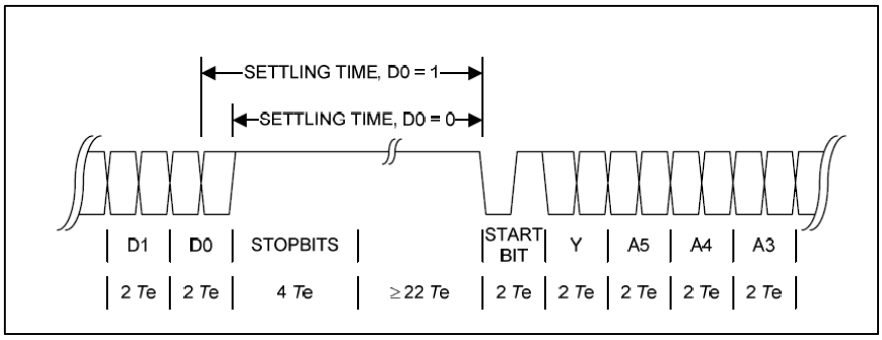


Figura 30: Dali. Settling Time entre tramas backward, forward

Hay dos tipos de dispositivos:

1. Dispositivo DALI (maestro o esclavo). Consumen un máximo de 2 mA en reposo para recibir estado de nivel alto y en cortocircuito un mínimo de 250 mA para transmitir el estado de bajo nivel. La corriente es la corriente de cortocircuito que proporciona el bus DALI.

2. Dispositivo DALI de fuente de alimentación. Estos dispositivos alimentan el bus DALI. El suministro es limitado a un máximo de 250 mA.

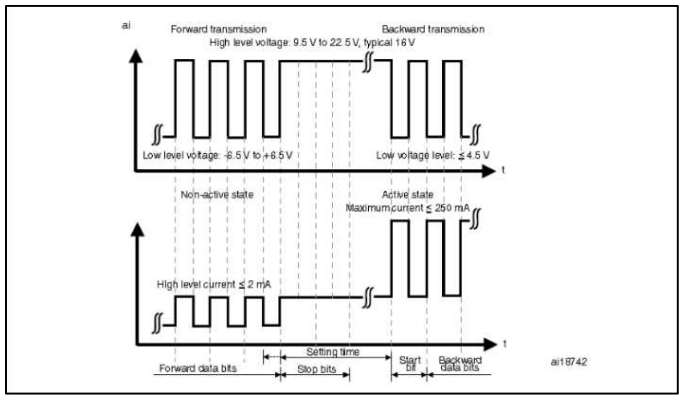


Figura 31: Dali. Diagrama de corriente en el bus

El bus de datos es alimentado por al menos una fuente de alimentación de bucle. Los mensajes se envían por un cortocircuito y la liberación de los conductores de bucle para crear una señal digital.

La longitud máxima del bus DALI depende de los cables que se utilizan para la instalación eléctrica. Por ejemplo, un cable de 1,5 mm^2 de sección permite una longitud máxima del bus de hasta 300 m. La longitud es linealmente dependiente de la sección transversal del conductor.

Las colisiones entre varios dispositivos master en el bus DALI se resuelven sobre la base de prioridades de tiempo. Cuando se detecta una colisión (el master DALI debe comprobar los datos enviados), la comunicación se silencia durante un período de tiempo de acuerdo con la prioridad asignada a cada master. Hay cinco niveles de prioridad definidos: 12 ms, 13 ms, 14 ms, 15 ms y 16 ms. La espera más larga tiempo tiene menos prioridad.

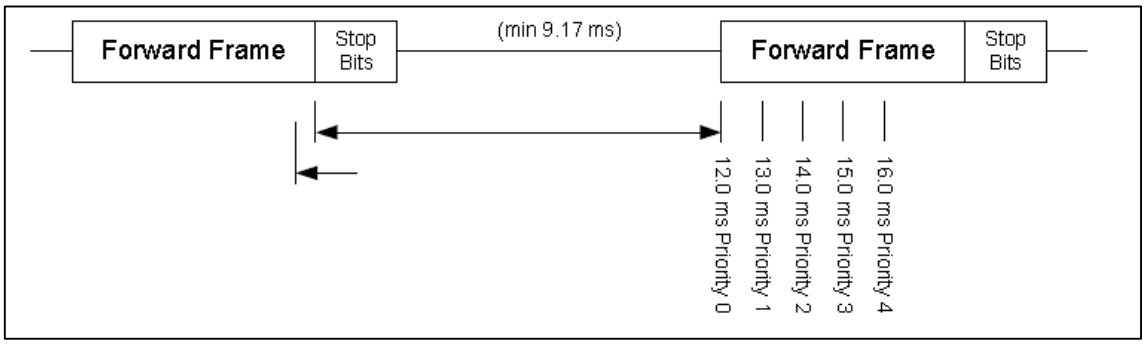


Figura 32. Dali. Colisiones entre tramas

Descripción general de niveles de prioridad.
Los comandos se asignan a los niveles de prioridad. El tipo de la orden y el tipo de aplicación determinar la asignación, como sigue:

a) Prioridad 0: prioridad temporal para los mensajes que se debe repetir
b) Prioridad 1: Los comandos de control de potencia iniciados por el usuario y los comandos que tienen un impacto en ajuste manual de la potencia (por ejemplo, regular hacia máxima luminosidad con un tiempo especial de fundido)
c) Prioridad 2: Configuración en general
d) Prioridad 3: Los comandos de control de potencia automáticos
e) Prioridad 4: Consultas.

3.3.2 Capa Stack.

La capa stack DALI es la capa superior que implementa en el microcontrolador los comandos, las estructuras, el control del tiempo y el control de errores conforme a las especificaciones. La característica más importante de la capa stack DALI son los comandos. Estos comandos se utilizan para:

- Control de la luz directa, como oscurecimiento, encendido / apagado, y la selección de escenas.
- Configuración de dispositivos para establecer las variables DALI, las propiedades del dispositivo, la dirección asignaciones y el estado del dispositivo consulta.

3.3.2.1 Tipos de comandos.

DALI tiene tres tipos de direccionamiento con el fin de proporcionar la resolución de control deseada. En todos los tipos de comandos, el control se logra mediante comandos DALI de 2 bytes estos enviados al bus. Los comandos son recibidos por los balastros u otros dispositivos DALI de salida conectados en ese mismo bus.

3.3.2.1.1 Comandos DALI de difusión

Todos los balastros conectados a un circuito deben responder a una orden emitida a nivel de difusión igualmente todos deben responden al mismo tiempo y de la misma manera a un mismo comando de difusión. Los comandos de difusión se utilizan a menudo en los arranques o donde hay una necesidad de ajustar todos los balastros con el mismo nivel de luz digital.

3.3.2.1.2 Comandos DALI de Grupo

Un balastro puede pertenecer a uno o varios de los grupos de 16 (0-15). Se requiere la puesta en marcha para inicializar el balastro y consiste en colocar en la memoria del balastro a que grupos pertenecerá y por consiguiente responderá el balasto. Para poder asignar grupos a los balastros, las unidades deben ser accesible mediante su dirección. Los comandos a los balastros en el bus pueden ser enviados a una dirección específica de grupo de balastos mediante el envío de un comando de grupo. Todos los balastros que pertenecen a este grupo deberán responder al mismo tiempo y de la misma manera a este comando. Los comandos de grupo a menudo se utilizan para controlar las áreas específicas de una habitación, por ejemplo, iluminarias de la misma pared, o para controlar varias habitaciones en un mismo bus donde cada habitación se le asigna un grupo diferente.

3.3.2.1.3 Comandos DALI individuales

A los Balastros se les pueden asignar una dirección individual única. Hay 64 direcciones posibles (0-63). Dos balastos no deberían tener la misma dirección en un circuito DALI. Se requiere la puesta en marcha del balastro para asignarle una dirección inicial, como en el grupo también se graba en la memoria del dispositivo. El envío de un comando de direcciones individual se utiliza para direccionar balastros específicos. El balastro con esta dirección deberá responder a la dirección individual del comando. Los comandos de dirección individuales se utilizan a menudo para proporcionar un control personal para luminarias individuales por ejemplo apagar una luminaria en una oficina.

3.3.2.1.4 Comandos DALI de Escena

Los Balastros pueden haber almacenado escenas en la memoria. Una escena se asocia a niveles de luz digitales predeterminado. Hay 16 escenas posibles (0-15), y cada escena almacena un nivel de luz digital (0-254). Se requiere la puesta en servicio del balastro para almacenar un nivel en escena. Las escenas se pueden recuperar usando los tres tipos de direcciones (Broadcast, Grupo DALI, y Comandos de direcciones individual).

3.3.2.2 Ejemplos de comandos.

Recordamos que le dispositivo master, envía tramas forward frame hacia los diversos dispositivos conectados al bus.

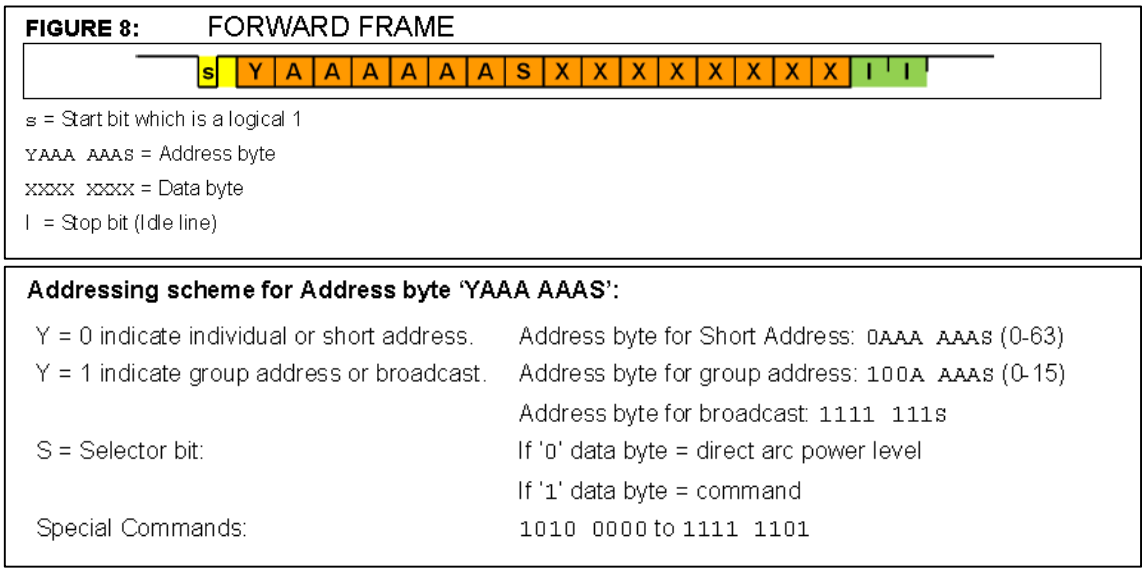


Figura 33. Dali. Especificación Forward Frame

Ejemplos de comandos:

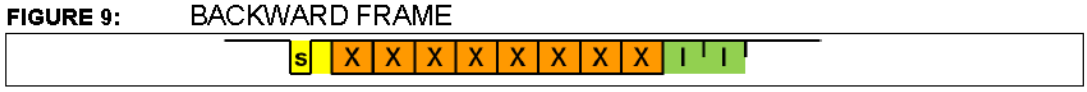
DALI Commands (Request)

Output Data						
No.	D0 Command Code	D1 DALI Address	D2 =0	D3 =0	D4 =0	Function
0	0000 0000	YAAA AAA1	0	0	0000 0000	Off
1	0000 0001	YAAA AAA1	0	0	0000 0000	Up
2	0000 0010	YAAA AAA1	0	0	0000 0000	Down
3	0000 0011	YAAA AAA1	0	0	0000 0000	Step up
4	0000 0100	YAAA AAA1	0	0	0000 0000	Step down
5	0000 0101	YAAA AAA1	0	0	0000 0000	Recall Max – Level
6	0000 0110	YAAA AAA1	0	0	0000 0000	Recall Min – Level
7	0000 0111	YAAA AAA1	0	0	0000 0000	Step down and off
8	0000 1000	YAAA AAA1	0	0	0000 0000	On and step up
9-15	0000 1XXX	YAAA AAA1	0	0	0000 0000	Reserved
16-31	0001 XXXX	YAAA AAA1	0	0	0000 0000	Go to scene 1 - 16
32 *)	0010 0000	YAAA AAA1	0	0	0000 0000	Reset
33 *)	0010 0001	YAAA AAA1	0	0	0000 0000	Store actual Level in the DTR **)
34–41 *)	0010 XXXX	YAAA AAA1	0	0	0000 0000	Reserved
42 *)	0010 1010	YAAA AAA1	0	0	0000 0000	Store the DTR **) as Max – Level
43 *)	0010 1011	YAAA AAA1	0	0	0000 0000	Store the DTR **) as Min – Level
44 *)	0010 1100	YAAA AAA1	0	0	0000 0000	Store the DTR **) as System Failure – Level
45 *)	0010 1101	YAAA AAA1	0	0	0000 0000	Store the DTR **) as Power On Level
46 *)	0010 1110	YAAA AAA1	0	0	0000 0000	Store the DTR **) as Fade Time
47 *)	0010 1111	YAAA AAA1	0	0	0000 0000	Store the DTR **) as Fade Rate
48–63 *)	0011 XXXX	YAAA AAA1	0	0	0000 0000	Reserved
64-79 *)	0100 XXXX	YAAA AAA1	0	0	0000 0000	Store the DTR **) as Scene 1 - 16
80-95 *)	0101 XXXX	YAAA AAA1	0	0	0000 0000	Remove from Scene 1 - 16
96-111 *)	0110 XXXX	YAAA AAA1	0	0	0000 0000	Add to Group 1 - 16
112 *)	0111 XXXX	YAAA AAA1	0	0	0000 0000	Remove from Group 1 - 16
128 *)	1000 0000	YAAA AAA1	0	0	0000 0000	Store the DTR **) as Short Address
129–143	1000 XXXX	YAAA AAA1	0	0	0000 0000	Reserved
144	1001 0000	YAAA AAA1	0	0	0000 0000	Query Status
145	1001 0001	YAAA AAA1	0	0	0000 0000	Query Ballast
146	1001 0010	YAAA AAA1	0	0	0000 0000	Query Lamp Failure
147	1001 0011	YAAA AAA1	0	0	0000 0000	Query Lamp Power On
148	1001 0100	YAAA AAA1	0	0	0000 0000	Query Limit Error

Figura 34. Dali. ejemplo de comandos

Las tramas backward son enviadas desde los distintos dispositivos conectados al bus hacia el master, normalmente con respuestas a las peticiones.

FIGURE 9: BACKWARD FRAME



s = Start bit, which is a logical 1
xxxx xxxx = Data byte
I = Stop bit (Idle line)

Response

Response	DALI-Code
„Yes“	1111 1111
„No“	No Answer
8-Bit Information	XXXX XXXX

Figura 35. Dali. Especificación Backward Frame

Input Data						
No.	D0 DALI Response	D1 DALI Address	D2 Mes-sage 3	D3 Mes-sage 2	D4 Mes-sage 1	Function
0	-	-	-	-	-	Off
1	-	-	-	-	-	Up
2	-	-	-	-	-	Down
3	-	-	-	-	-	Step up
4	-	-	-	-	-	Step down
5	-	-	-	-	-	Recall Max – Level
6	-	-	-	-	-	Recall Min – Level
7	-	-	-	-	-	Step down and off
8	-	-	-	-	-	On and step up
9-15	-	-	-	-	-	Reserved
16-31	-	-	-	-	-	Go to scene 1 - 16
32	-	-	-	-	-	Reset
33	-	-	-	-	-	Store actual Level in the DTR **)
34–41	-	-	-	-	-	Reserved
42	-	-	-	-	-	Store the DTR **) as Max – Level
43	-	-	-	-	-	Store the DTR **) as Min – Level
44	-	-	-	-	-	Store the DTR **) as System Failure – Level
45	-	-	-	-	-	Store the DTR **) as Power On Level
46	-	-	-	-	-	Store the DTR **) as Fade Time
47	-	-	-	-	-	Store the DTR **) as Fade Rate
48–63	-	-	-	-	-	Reserved
64-79	-	-	-	-	-	Store the DTR **) as Scene1 -16
80-95	-	-	-	-	-	Remove from Scene 1 - 16
96-111	-	-	-	-	-	Add to Group 1 - 16
144	Bit 0=„0“=OK, Ballast Bit 1=„0“=OK, Lamp Failure Bit 2=„0“=Off, Lamp Power On Bit 3=„0“=Off, Limit Error Bit 4=„0“=Terminate Fading, Bit 5=„0“=No, Reset State Bit 6=„0“=No, Missing Short Address Bit 7=„0“=No, Power Failure	YAAA AAA1	-	-	-	Query Status
145	„Yes“ or „No“	YAAA AAA1	-	-	-	Query Ballast
146	„Yes“ or „No“	YAAA AAA1	-	-	-	Query Lamp Failure
147	„Yes“ or „No“	YAAA AAA1	-	-	-	Query Lamp Power On
148	„Yes“ or „No“	YAAA AAA1	-	-	-	Query Limit Error
149	„Yes“ or „No“	YAAA AAA1	-	-	-	Query Reset State
150	„Yes“ or „No“	YAAA AAA1	-	-	-	Query Missing Short Address

Figura 36. Dali. Ejemplos respuestas

3.3.2.3 Conexionado y funcionamiento inicial.

Inicialmente conectamos al bus de datos todos los dispositivos DALI. Un bus conecta hasta dispositivos de salida 64 de iluminación (es decir, balastos, transformadores, relés y otros).

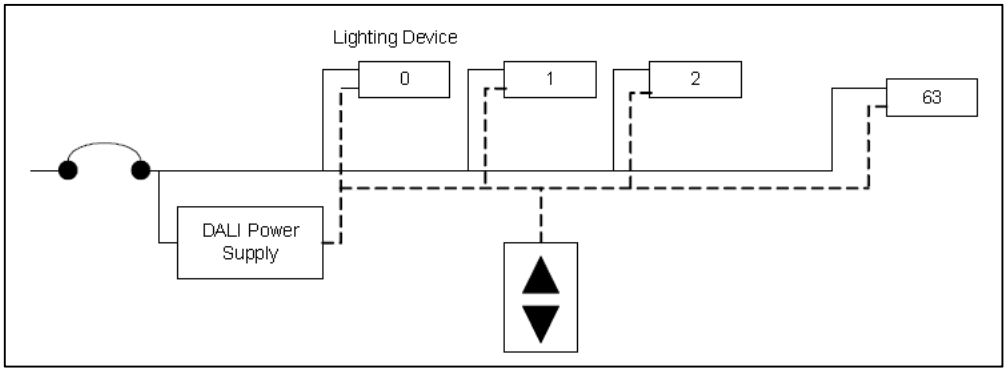


Figura 37. Dali conexionado

Otros dispositivos de control (hasta un total de 64) podrían añadirse a la configuración anterior. En este caso, el bucle podría ser de hasta un total de 128 dispositivos direccionables (64 dispositivos de salida y 64 dispositivos de control). Los dispositivos de control DALI se utilizan para enviar comandos a los balastros DALI y otros componentes DALI.

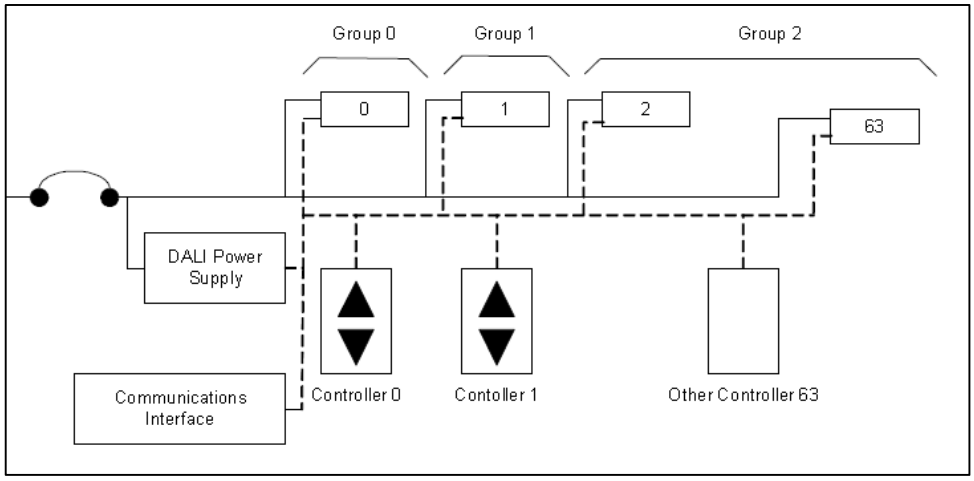


Figura 38: Dali. Grupos de luminarias

Una vez todos los dispositivos están conectados al bus y conexionados a red eléctrica, no hay ningún tipo inicial de comunicación. Será siempre es el master quien inicia la comunicación mediante los diversos tipos de comandos. En este caso por ejemplo el master puede entablar una conversación con todos los dispositivos asignándoles una dirección, un grupo, escenas….esto se realizaría mediante un protocolo similar al dhcp mediante polling. También puede utilizar comandos de broadcast para apagar inicialmente todas las luminarias y posteriormente mediante polling interrogar a cada dispositivo para signar direcciones, grupos, escenas…Para todo ello hace falta una herramienta de programación o que el master lo programemos con las opciones que creamos convenientes.

4. Diseño e Implementación

Una vez definas las especificaciones del protocolo y algunos aspectos básicos relacionados con la norma, procedemos a la implementación práctica. Para ello, tendremos en cuenta el diagrama propuesto. Para cada apartado explicaremos los detalles de la implementación así como todos los elementos utilizados.

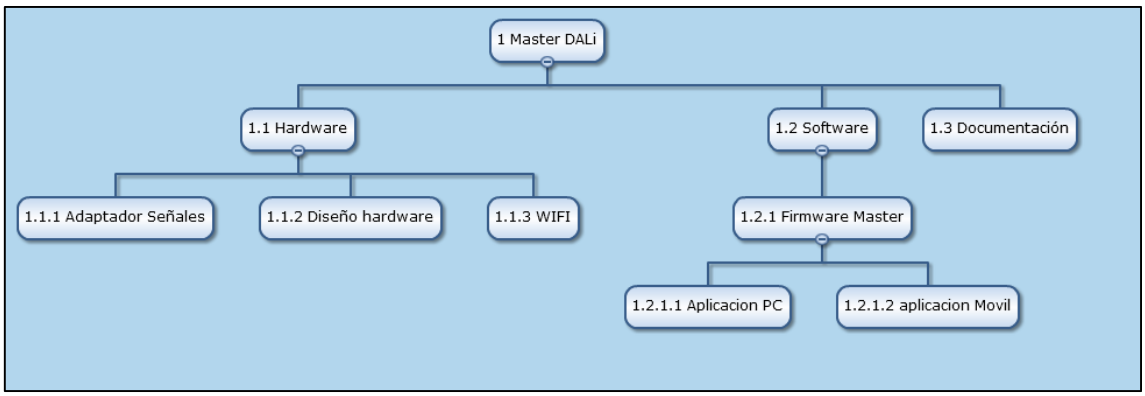


Figura 39. Plan de trabajo

Los pasos a seguir serán los siguientes.

1. Enumerar los elementos disponibles para la realización del proyecto.
2. Selección de la plataforma de desarrollo.
3. Selección entorno y lenguaje de desarrollo.
4. Diseño del hardware para la comunicación entre los elementos del sistema. Adaptador de señales.
5. Diseño del hardware del dispositivo completo.
6. Implementación del protocolo en el hardware. (Firmware)
7. Maqueta de sistema completo.
8. Encender apagar luminaria con un pulsador. Envío comando on/off.
9. Conexión Exterior (wifi).
10. Interfaz con el PC y desarrollo de la aplicación del PC
11. Aplicación para móvil.
12. Control de Varias luminarias.
13. Ampliación de comandos: Grupos, escenas, etc…

4.1 Elementos disponibles para la realización del proyecto.

Disponemos inicialmente de:

- Una Placa Slave Dali certificada. Al ser certificada, sabemos que cumplen tanto el protocolo como la noma al completo, incluso tienen implementadas funciones avanzadas pertenecientes a la norma no documentas en este proyecto. Estás placas pertenece a un producto más complejo que es un driver para luminarias led. La placa se corresponde con el adaptador de señales para el bus así como la implementación de la capa Stack anteriormente descrita pero para un cliente. La implementación está realizada mediante un microcontrolador Freescale MKE02 Z32VLC2. Las placa está modificada y simula una lámpara mediante un led el control de regulación de luminosidad lo hacen mediante PWM.

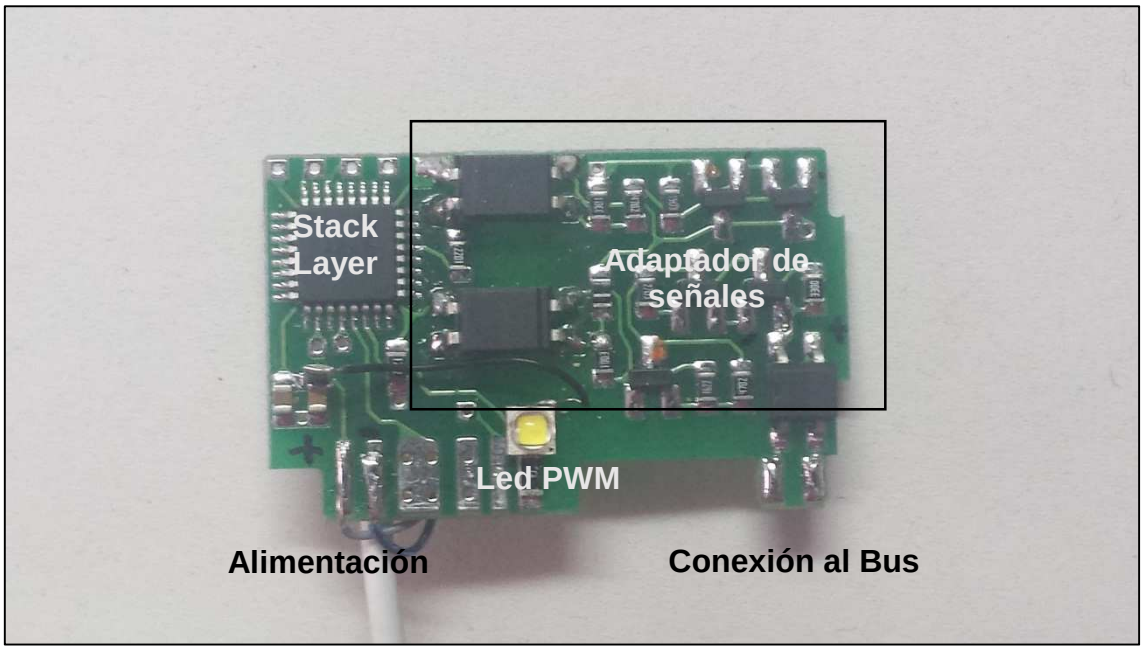


Figura 40: Placa Dali

- Varias fuentes de alimentación. Para la alimentación de los diversos elementos. Una para la placa slave y otra para la alimentación del bus. Los esquemas de las fuentes de alimentación son desarrollados y comentados posteriormente.

4.2 Selección del hardware (plataforma de desarrollo).

No son muchos los requisitos de hardware relacionados con el microcontrolador pero hemos valorado una selección de los siguientes fabricantes, por su popularidad y basados en la utilización industrial:

- Microchip. PIC18F2550. Entrono de programación muy bueno y gratuito MPLAB. El programador siempre va a parte. Mucho soporte disponible.

- Freescale (actualmente NXP). Cortex-M0+ MCU Freedom Board. Entorno de programación gratuito. Existen placas con programador integrado y otras no.

- ST microelectronics. Cortex-M4 MCU Placa desarrollo STM32F407VG MCU. Entorno gratuito Cocoox. Programador integrado en placa.

Todos los microcontroladores son actuales y fáciles de programar pero también son menos potentes que el STM32F407. Aun así para los requisitos del proyecto a realizar son más que suficientes.

La selección de la plataforma es la placa desarrollo STM32F407VG MCU. Vemos que el entorno de programación es gratuito o comerciales: IAR, Netbeans, Visual Studio express, Cocoox. El programador viene incorporado en la placa. Es el más potente de las tres posibles opciones pensando en usos futuros.

Para la parte WIFI.

- Paltaforma Arduino. Arduino Shield YUM.
- ST microelectronics. WiFi Board for STM32F4 Discovery Kit

La selección de la plataforma WIFI, es la WiFi Board for STM32F4 Discovery Kit. Dado que es el complemento perfecto para la plataforma de desarrollo, ya que es incluso del mismo fabricante.

La placa de la plataforma de desarrollo seleccionada *Discovery kit with STM32F407VG MCU*. Está basada en un microcontrolador ARM cortex- M4 de la firma ST microelectronics.

Figura 41. Plataforma desarrollo hardware

Dispone de las siguientes características:

- Microcontrolador STM32F407VGT6 que ofrece 32 bits ARM Cortex® -M4 con núcleo de FPU, 1 Mbyte de memoria flash, memoria RAM de 192 Kbytes en un paquete LQFP100
- Interfaz ST-LINK / V2 en STM32F4DISCOVERY o ST-LINK / V2-A en STM32F407G-DISC1
- ARM mbed ™ habilitado para (http://mbed.org) con ST-Link sólo V2-A
- ST-LINK USB con capacidad de re-enumeración y tres interfaces diferentes:
    - puerto com virtual (con ST-LINK / V2-A)
    - almacenamiento masivo (con ST-LINK / V2-A)
    - puerto de depuración
- Alimentación de la tarjeta: a través del bus USB o desde una tensión externa de alimentación de 5 V
- Conector para conexión externa de energía: 3 V y 5 V
- LIS302DL o LIS3DSH ST MEMS acelerómetro de 3 ejes
- MP45DT02 ST MEMS sensor de audio de micrófono digital omnidireccional
- DAC de audio con CS43L22 integrado de clase D controlador del altavoz
- Ocho LEDs:
    - LD1 (rojo / verde) para la comunicación USB
    - LD2 (rojo) de 3,3 V de encendido
    - Cuatro LED de usuario, LD3 (naranja), LD4 (verde), LD5 (rojo) y LD6 (azul)
    - 2 USB OTG LED LD7 (verde) VBUS y LD8 (rojo) sobrecorriente
- Dos pulsadores (usuario y reset)
- FS USB OTG con micro-AB conector
- Puerto de extensión para todos LQFP100 E / S para la conexión rápida de placa de prototipo y fácil de sondeo
- Software libre integral que incluye una variedad de ejemplos, parte del paquete STM32CubeF4 o STSW-STM32068 para el uso de las bibliotecas estándar.

La placa para WIFI es la WiFi Board for STM32F4 Discovery Kit.

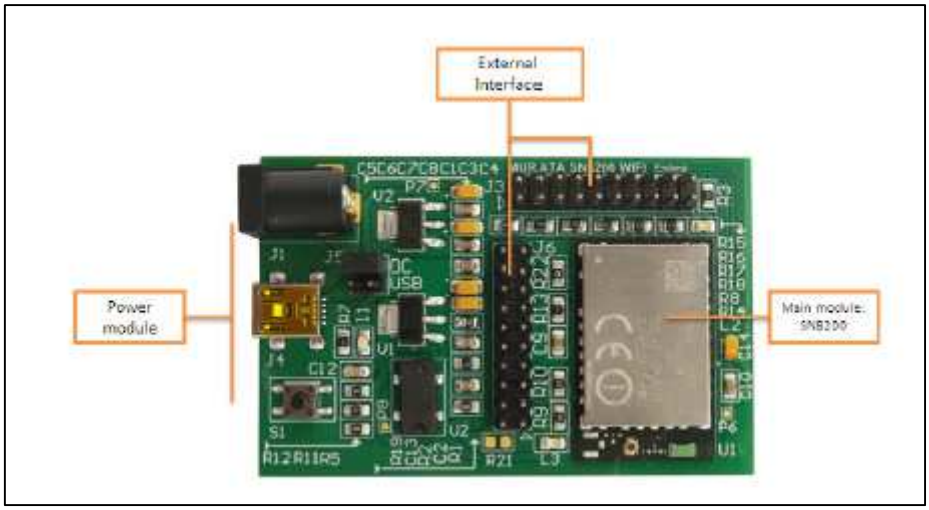


Figura 42: Plataforma desarrollo WIFI

Descripción de Bloques:

- Wi-Fi Module: SN8200 Wi-Fi module
- Power Section (J1, J4): The board is powered by Mini-USB or 5V, 2A DC.
- Switch and LEDs: One reset switch and two signal LEDs.
- External Interface
- JTAG interface (J3): Standard 20 pin interface, used for Module Firmware Loading.
- User interface (J6): External interface for users.

Wi-Fi Module - SN8200 Características:

- 2.4GHz IEEE 802.11b/g/n RadioTechnology
- Wi-Fi Chip - Broadcom BCM43362
- MCU - STM32 ARM Cortex-M3
- Dimensions: 30.5 x 19.4 x 2.8 mm
- Package: LGA
- On-Board Antenna
- Max Receive Sensitivity: -96dbm @ b mode/11Mbps
- Transmit Power: +18 dBm
- Host Interfaces: UART & SPI
- Other Interfaces: ADC, DAC, I2C, GPIO
- Operating Temperature Range: -30ºC to 85ºC
- ROHS Compliant
- FCC/IC certified, CE compliant
- PN 88-00151-00
- EVK/SDK P/N 88-00151-85

4.3 Selección entorno y lenguaje de programación

Para implementar el protocolo (capa de stack), necesitamos seleccionar un lenguaje y un entorno de programación. La selección de lenguajes es "sencilla" ya que normalmente para sistemas donde existen microcontroladores usamos C, como es nuestro caso. El entorno de programación es la herramienta que nos facilita el acceso a las funcionalidades de la plataforma (API, integración de librerías…). Tenemos disponibles una amplia variedad de entornos de programación tanto gratuitos como comerciales con versiones reducidas. Entre ellos podemos destacar: IAR, Netbeans, Visual Studio express, Cocoox.

En nuestro caso y dado que hemos escogido una plataforma de desarrollo, el mismo fabricante ST microelectronics facilita unas herramientas (librerías, API…) y códigos fuentes para diversos entornos. Incluso facilita plantillas "vacías" con todos los drivers, librerías…preconfigurados. **La selección del entorno ha sido el entorno IAR**. A pesar de ser una herramienta comercial nos ofrece una versión con una funcional reducida de memoria de hasta 32K.

4.4 Diseño de las fuentes de alimentación.

Vamos a utilizar fuentes de alimentación estándar con tensiones normalizadas 5, 12 Voltios. Por ello podemos optar por usar reguladores tipo 78XX, LMXXX etc…

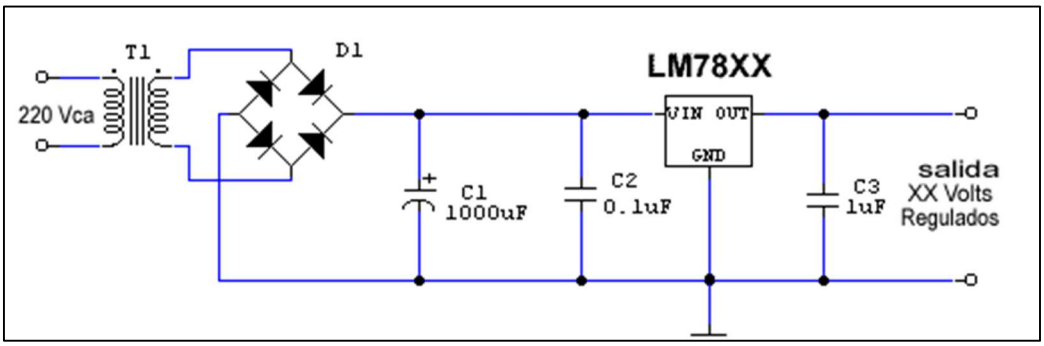


Figura 43. Diseño fuente alimentación general basada en 78XX

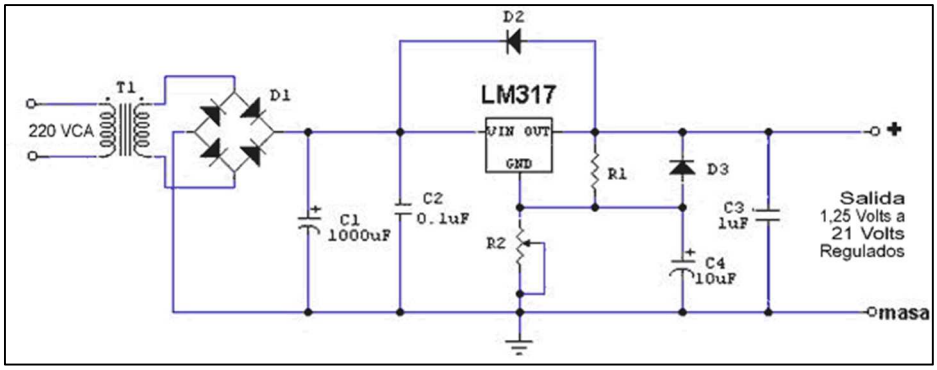


Figura 44: Diseño fuente de alimentación basada en LM317

La única consideración es la fuente de alimentación para el bus**, indican que debe suministrar una corriente de cortocircuito máxima de aproximadamente de 250 mA y una tensión de salida de aproximadamente 20 Voltios.**

Como la tensión de salida es de 20 Voltios, se puede usar un regulador tipo LM317 regulable en salida desde 1,2 V hasta 37 Voltios, este proporciona una corriente máxima de 1,5 Amperios. Características más que suficientes según lo especificado, solo se necesitan 250 mA.

Para la limitación de corriente existen varias soluciones, de las cuales comentaremos dos. Una de ellas es colocar una resistencia en serie con la salida y conectarla al bus. Recordamos que para hacer la señal digital de comunicación entre los dispositivos se hace un cortocircuito "temporal" al bus, por lo tanto la corriente de salida queda limitada por esa resistencia de cortocircuito. Para que no haya desequilibrios podemos dividir el valor calculado de la resistencia para dos y colocar una resistencia tanto en positivo como en negativo.

$$R = \frac{V}{I} = \frac{1{,}4V}{250\ mA} = 5{,}6\ \Omega$$

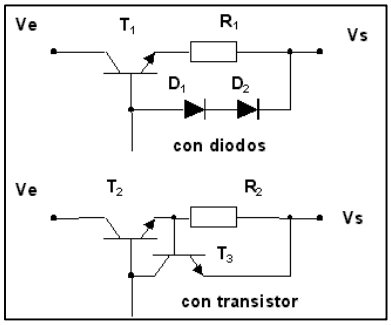


Figura 45. Limitador de corriente mediante resistencia

Otra de las soluciones es colocar a la salida de la fuente de alimentación una fuente de corriente constante limitada a 250 mA. Se puede realizar con un regulador LM317.

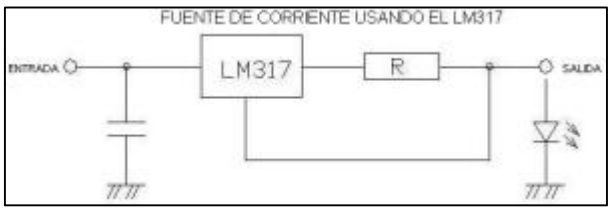


Figura 46. LM317 como fuente de corriente constante.

La solución adoptada es esta última para el diseño de la fuente de alimentación del bus. Ya que el regulador, además de regular la corriente de salida, también nos la mantiene constante a pesar que la carga nos pida más corriente.

Para los cálculos de la tensión de salida, tenemos:

En el regulador U2 (Tensión alimentación del bus). Dejamos fija R2 en 220 ohmios según fabricante, además de:

$$V_s = 1{,}2V\left(1 + \frac{R3}{R2}\right)$$

$$R3 = \frac{(V_s - 1{,}2V) * R2}{1{,}2V} = \frac{(20V - 1{,}2V) * 220\Omega}{1{,}2V} = 3k5\Omega$$

En el regulador U1. Actúa de fuente de corriente constante limitando esta a la corriente de cortocircuito, se limita mediante la resistencia R1.

$$R1 = \frac{1{,}2V}{250mA} = 4{,}7\Omega$$

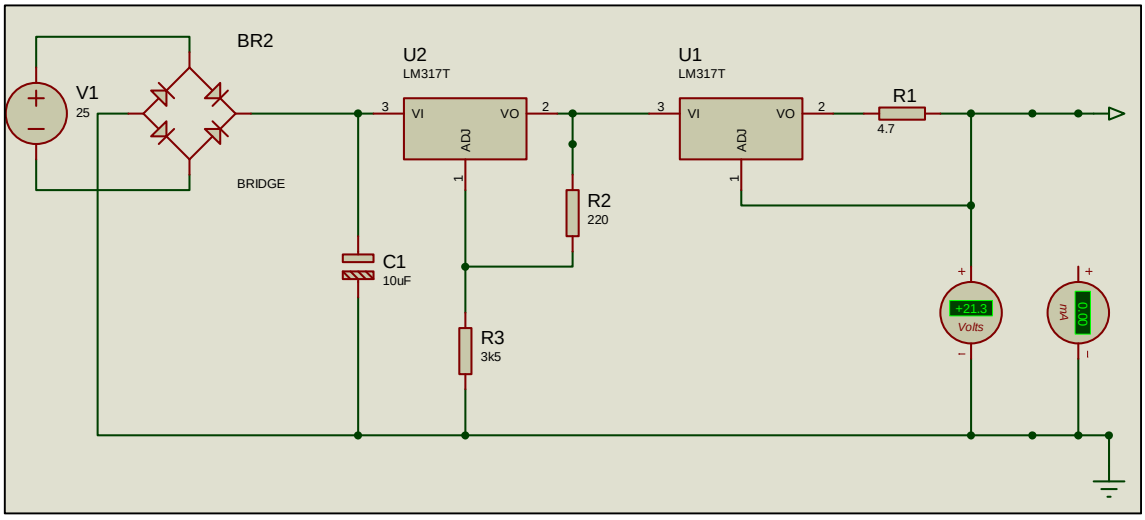


Figura 47. Diseño fuente de alimentación Bus. Tensión de salida

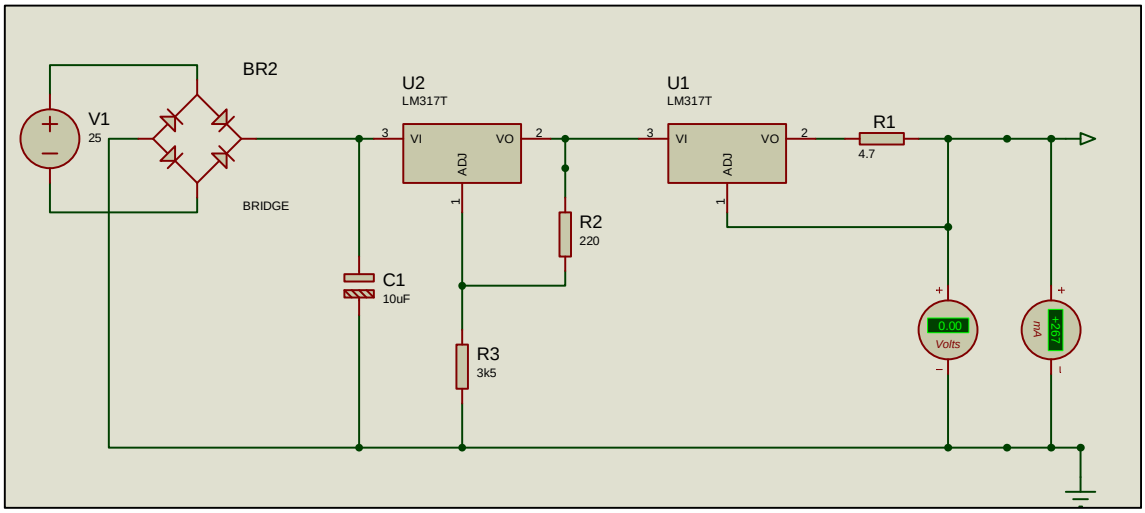


Figura 48. Diseño fuente de alimentación Bus. Limitación de corriente

En ambas capturas de la simulación de la fuente de alimentación, se aprecia como la tensión de salida nos da un valor de 21,3 Voltios y una corriente de cortocircuito de 267 mA.

En proyectos reales habría que utilizar fuentes de alimentación conmutadas, ya que su eficiencia energética es mucho mayor que las clásicas de transformador además de tener un menor tamaño.

4.5 Diseño del hardware. Adaptador de señales del bus.

Las consideraciones y soluciones a tener en cuenta para el desarrollo del adaptador de señales al bus son:

- La interfaz DALI (bus) consiste en una capa física a partir de dos cables**. La polaridad es independiente.** Esto nos obliga al uso de un puente rectificador de entrada al adaptador ya que esto hará independiente al dispositivo de la polaridad.

- Dispositivo DALI (maestro o esclavo). **Consumen un máximo de 2 mA en reposo para recibir estado de nivel alto y en cortocircuito un mínimo de 250 mA** para transmitir el estado de bajo nivel. La corriente es la corriente de cortocircuito que proporciona el bus DALI. Debemos limitar la corriente de entrada y tener en cuenta que el elemento que cortocircuita el bus debe soportar una corriente de 250mA. Casi Cualquier transistor genérico tipo BC457 soporta esa corriente y la tensión de bus (20 voltios) entre colector-emisor.

- Tenemos un **bus** de dos hilos que hace la conexión a todos los elementos que intervienen en la instalación. El bus podemos conectarlo directamente al dispositivo o aislarlo. La opción escogida es **opto-aislarlo**, ya que así cualquier subida de tensión no nos afectaría al dispositivo completo. Lo elementos de entrada opto-acopladores nos realizan la función de “fusible”. Además de poder utilizar tensiones independientes en cada elemento (dispositivos, bus) sin conexión común de masa entre ellos.

- Cuando recibimos un 1 lógico, estamos recibiendo una tensión aproximada de 11,5 a 20,5 V el 0 lógico se define entre -6.5 A la 6,5 V. Podemos utilizar un diodo zener para que cuando la tensión sea mayor de la tensión zener utilizarlo como 1 lógico.

Con estas consideraciones para el esquema del adaptador de señales tenemos un esquema básico:

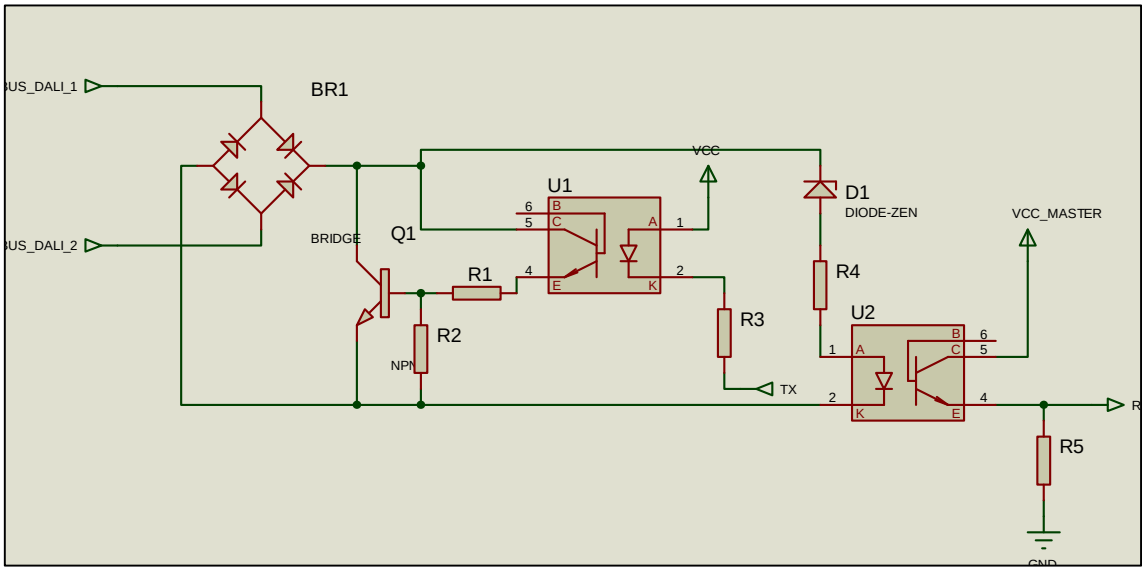


Figura 49. Adaptador de señales esquema básico

En la entrada tenemos un puente rectificador que hace que sea independiente la polaridad. Además de los dos opto-acopladores que nos aíslan del bus y marcan los dos circuitos el de transmisión y el de recepción. En el circuito de transmisión observamos que cuando haya un 0 lógico el opto-acoplador U1 estará saturado, como la base de Q1 estará a positivo, este conducirá, por lo tanto está haciendo un cortocircuito al bus haciendo un 0 físico. Cuando esté a 1 lógico, el transistor Q1 estará bloqueado y habrá un 1 físico en el bus. El circuito de recepción está formado por un diodo zener, la resistencia limitadora de corriente y el led del opto-acoplador. Cuando hay un 1 físico en el bus, el transistor del opto-acoplador U2 estará saturado por consiguiente en R5 tendremos la tensión de alimentación 1 lógico. Cuando este a 0 físico el bus, el transistor del opto-acoplador U2 estará bloqueado tendremos entonces un 0 lógico.

Para los cálculos en recepción, nos dice que no se pueden superar los 2 mA en espera. Esto limita la corriente para el circuito de recepción formado por la R4, el diodo zener y el led del opto-acoplador. Primero escogemos el diodo zener, lo fijamos en 5v1. Es un valor cercano al valor de tensión asociado a al 0 por debajo de 6 voltios. Ahora calculamos el valor de la resistencia limitadora, esta vendrá dada por la tensión que cae en ella dividida para la corriente máxima de 2mA.

$$R4 = \frac{V_{bus} - V_z - V_{led}}{2\ mA} = \frac{22 - 5{,}1 - 0{,}3}{2\ mA} = 8k2\Omega$$

Para el Valor de R5, lo fijamos en 10K. La corriente por el transistor del opto-acoplador será de:

$$I = \frac{V_{cc}}{R5} = \frac{5}{10K} = 0{,}5mA$$

En el circuito de transmisión, sabemos que el transistor de cortocircutio, debe soportar por lo menos los 250 mA. Escogemos un transistor genérico como es el BC337. Soporta corriente de colector de 0,5 A con una ganancia de 60. Además su ancho de banda es superior a lo que necesitamos. Para el cálculo de las resistencias de polarización:

La corriente por la base de transistor, será de

$$I_b = \frac{I_c}{\beta_{min}} = \frac{250mA}{60} = 4{,}1\ mA$$

Vemos que es un poco elevada, ya que por el divisor de polarización, tendrá que circular por lo menos 10 veces. Optamos por una configuración Darlington de un bc337 y un bc547.

$$I_b = \frac{I_c}{\beta_{min}} = \frac{250mA}{60 * 110} = 38\ \mu A$$

La corriente por el divisor normalmente es 10 veces la intensidad de base mínimo $380\ \mu A$. La fijamos en $500\ \mu A$. Tenemos que tener en cuenta que al tener un Darlington, nos aseguramos que las tensiones base emisor la fijamos en 2 voltios.

$$R2 = \frac{V_{be}}{500\ \mu A} = \frac{2}{500\ \mu A} = 4k \cong 3k9$$

Para el cálculo de R1 hay que pensar que cuando cortocircuitamos el bus, tenemos que fijar la tensión de 1 físico por debajo de 6 voltios. Para asegurar la fijaremos en 3 voltios. Que será la tensión que caerá además en los diodos de los puentes rectificadores. Además, tenemos un diodo Zener en el circuito de recepción de 5v1, que no detectará los 3 voltios fijados. Por lo tanto:

$$R1 = \frac{V_{cc} - V_{be}}{500\ \mu A} = \frac{3 - 2}{500\ \mu A} = 2k \cong 2k2$$

Finalmente partiremos del siguiente esquema para el adaptador de señales para su construcción.

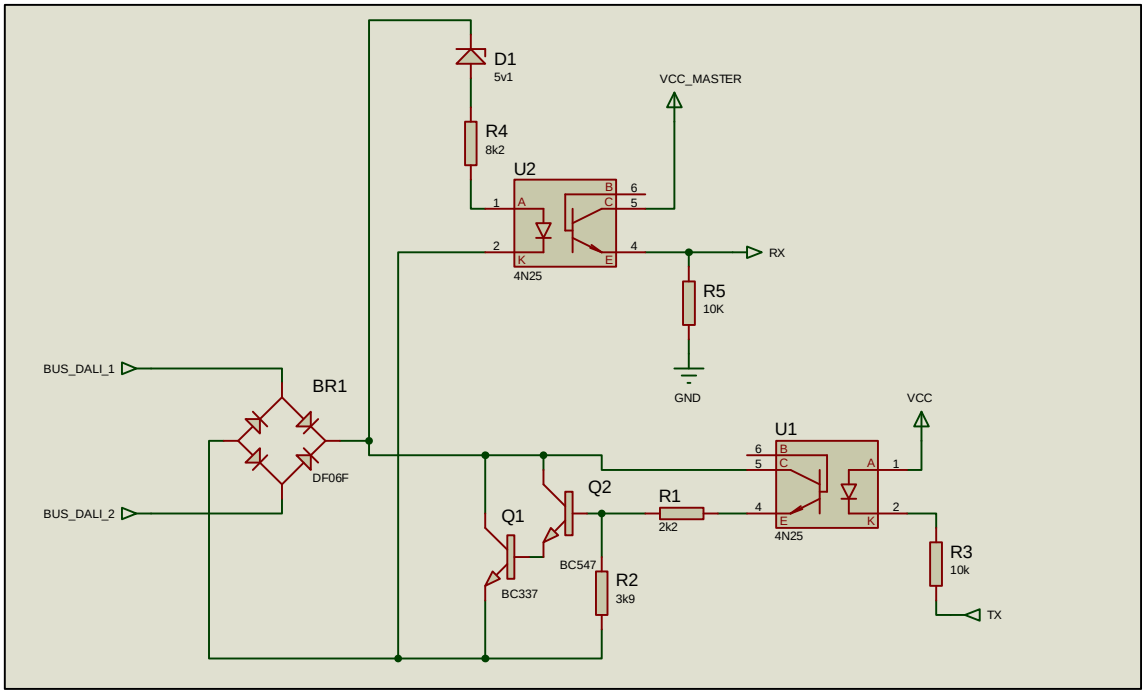


Figura 50. Adaptador de señales. Montaje final

Listado de componentes.

Componente	Descripción	Valor
BR1	Puente rectificador	DF06F
D1	Diodo Zener	5v1
R4	Resistencia	8k2
R5	Resistencia	10k
U2	Opto-acoplador	TLP151
Q1	Transistor	BC337
Q2	Transistor	BC547
R1	Resistencia	2k2
R2	Resistencia	3k9
R3	Resistencia	10k
U1	Opto- acoplador	TLP151

Una vez realizado el circuito básico, hacemos una simulación del canal de comunicaciones. En la parte derecha se ve el inyector de señal y lo fijamos en 5Khz como margen de seguridad, ya que la velocidad de transmisión es de 1k2Hz.

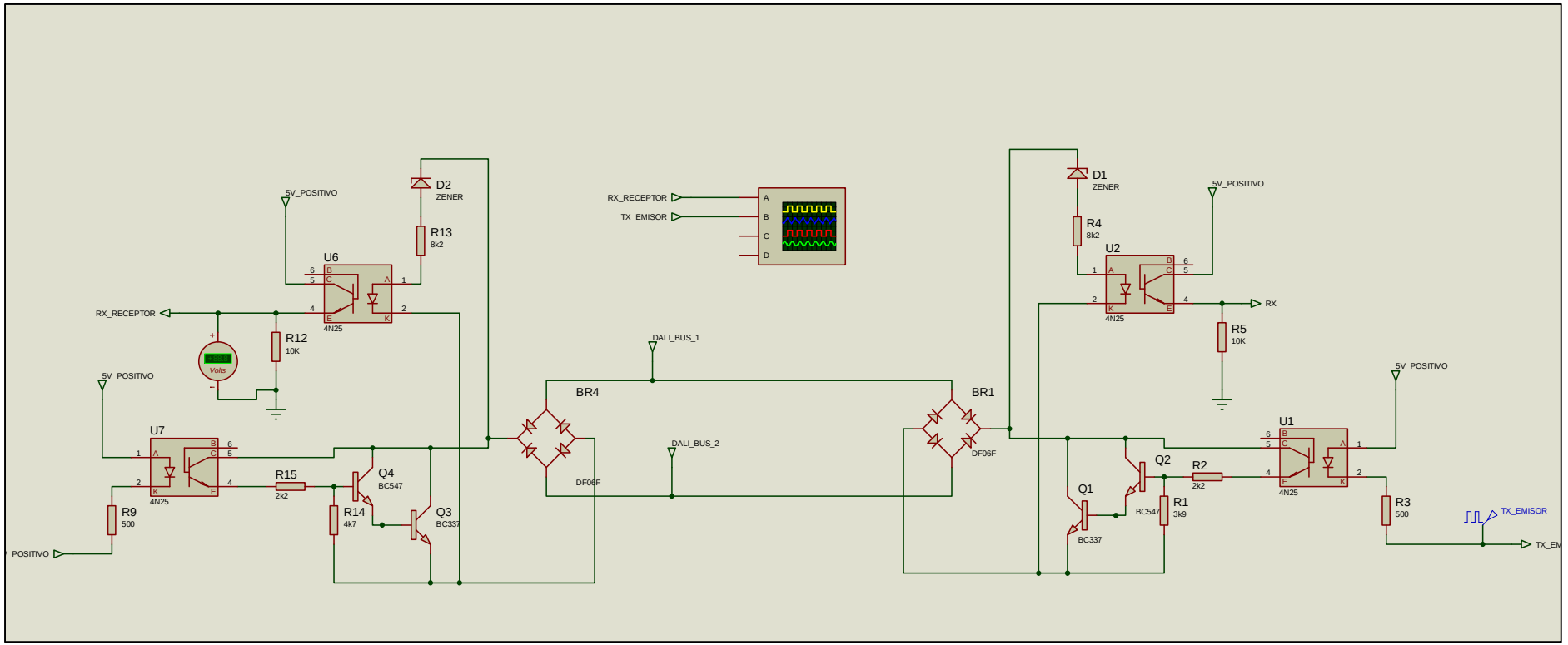


Figura 51. Adaptador de señales. Simulación capa física.

Capturamos una pantalla del osciloscopio virtual y comparamos las dos señales la del emisor y la del receptor:

Frecuencia de 4 KHz

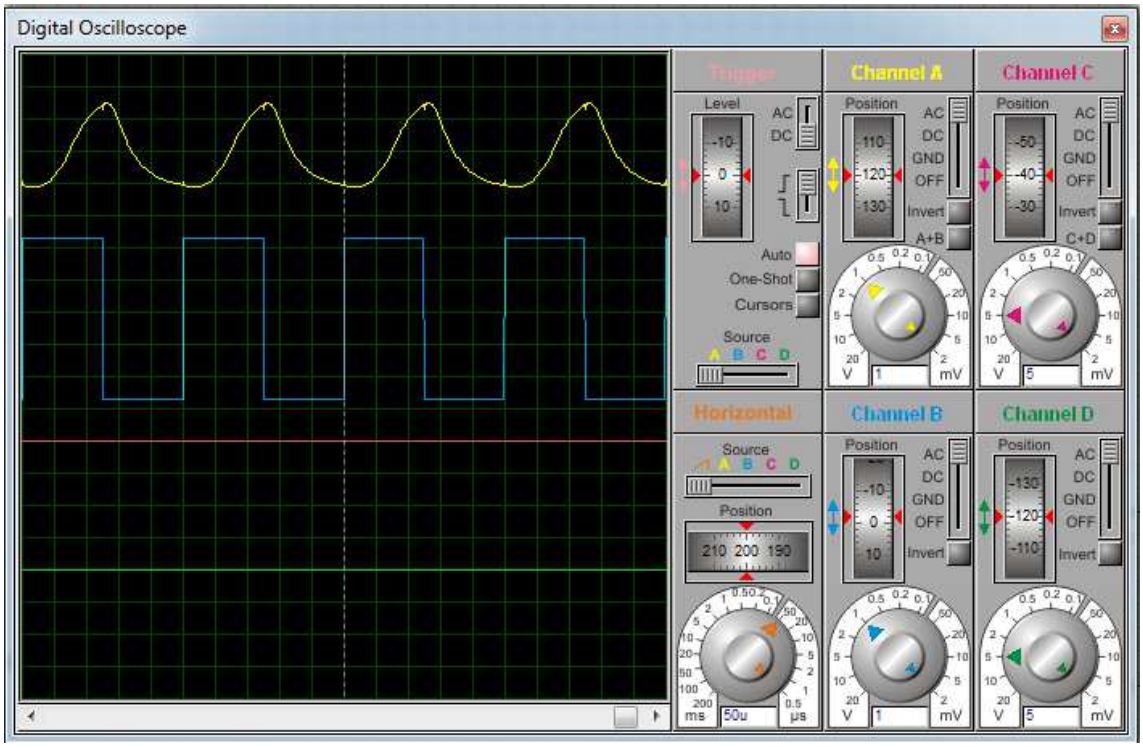


Figura 52. Adaptador de señales. Simulación frecuencia 4KHz

Frecuencia de 1K2 Hz

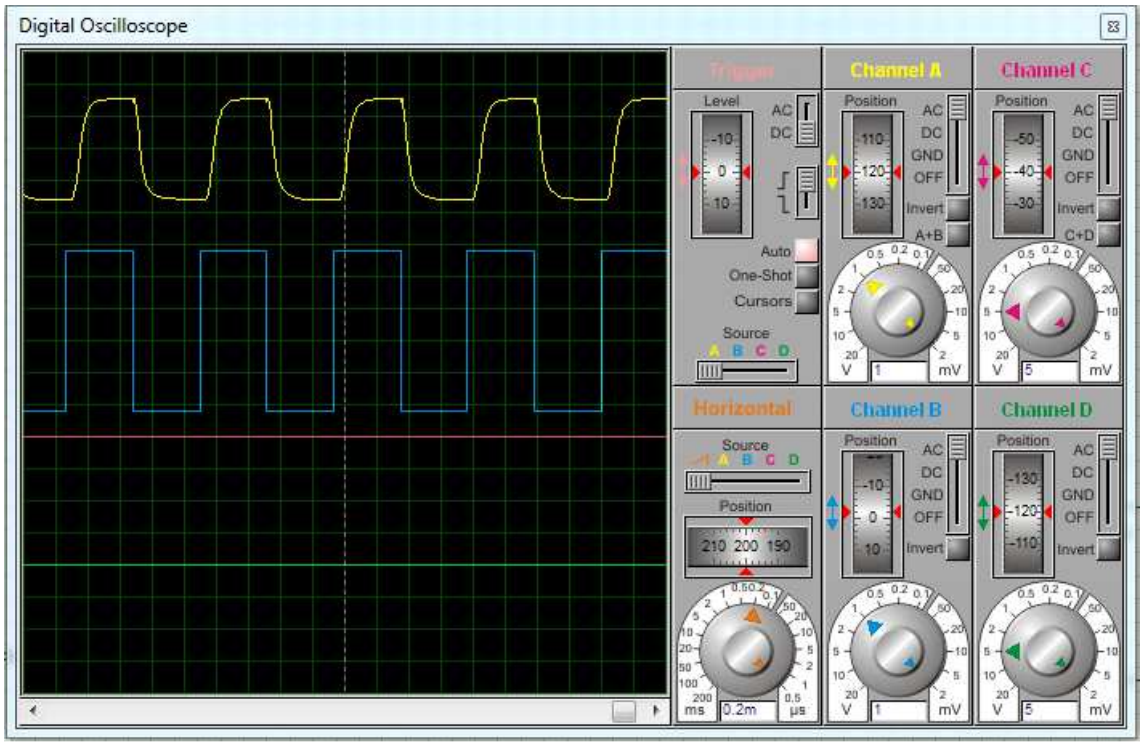


Figura 53 Adaptador de señales. Simulación frecuencia 1200 Hz

4.6 Mejoras para adaptador de señales[3].

4.6.1 Limitador de corriente.

Se puede apreciar en el esquema de la figura adjunta, que el bloque formado por los transistores Q1 y Q3 además de las resistencias R2, R3, R4 y R5, forman un limitador de corriente. Al estar R5 entre base y emisor del transistor Q1, este conducirá cuando en dicha resistencia haya una diferencia de potencial de 0,7 voltios, que es lo que nos limita la corriente. Derivando el resto a través de las resistencias R2, R3, y R4 a masa. Por lo tanto:

$$I_{MAX} = \frac{0{,}7}{390\ \Omega} = 1{,}8\ mA$$

Se puede pensar en un módulo de protección, de tal modo que si un componente se cortocircuita y aparece un consumo excesivo de corriente, este quedara mitigado por el circuito de protección (limitador) además de no dejar deshabilitado el bus para la comunicación entre otros dispositivos. También nos independiza de la tensión aplicada al bus, ya que una resistencia limitadora depende de esa tensión.

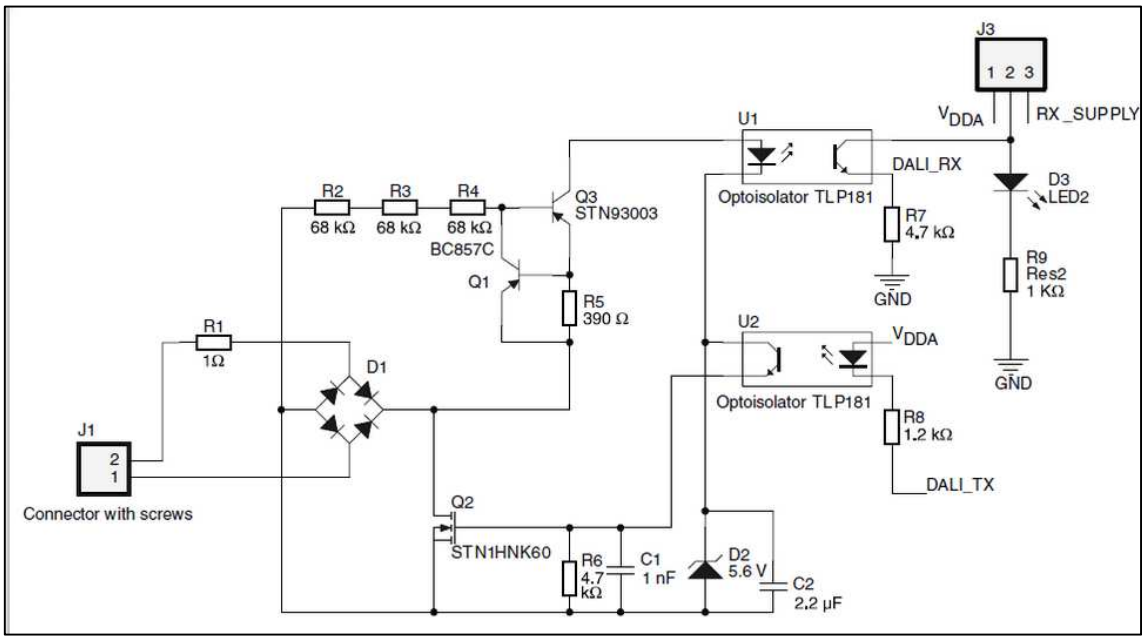


Figura 54. Adaptador de señales. Limitado de corriente

[3] http://www.mikrocontroller.net/topic/261045

4.6.2 Selección de alimentación integrada en el master

Esta modificación permite integrar la alimentación del bus en el master, siendo este quien alimenta el bus. Esta activación la seleccionamos por programa siendo el propio microprocesador el que activa esta opción. Al activarla mediante el opto-acoplador 0K2, el transistor de salida coloca a positivo la base de un MOSFET, haciendo que conduzca. Al conducir, une las masas de las alimentaciones dotando de una referencia al bus respecto a la fuente de alimentación integrada en el. Los positivos, están siempre conectados. El resto de componentes pertenecientes a este bloque son para polarizar el MOSFET.

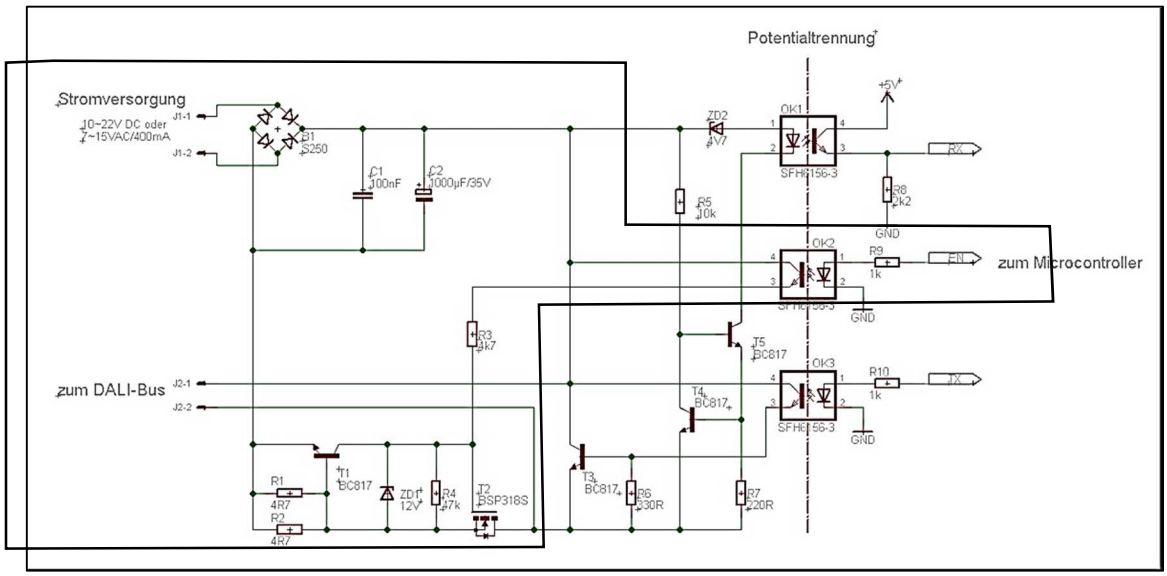


Figura 55. Adaptador de señales. Selección de Alimentación

4.7 Diseño del software

Implementación del protocolo en el hardware. (Firmware).

Para el desarrollo de la implementación, el problema lo podemos seguir dividiendo en dos partes, la parte de transmisión y la parte de recepción.

4.7.1 Transmisión.

Inicialmente, debemos escoger un pin de la plataforma de desarrollo que ira conectado a la parte de transmisión de todos los disponibles, escogemos el PD1. Escogemos este pin, ya que no soporta ninguna otra función dentro del microcontrolador excepto servir como entrada/salida.

La transmisión consiste en el envío por parte del master de una trama hacia el bus. La trama, está formada por una dirección de destino, indica quien debe realizar la acción y un comando que indica que acción debe realizar el destino. Acompañada de 3 bits de señalización, un bit de start y 2 bits de stop (consisten en mantener un tiempo el bus en estado alto). La longitud de la trama es en total de 19 bits. Como sabemos la velocidad de transmisión es de 1200 Hz, por lo tanto el tiempo de bit es de 416 microsegundos. Teniendo en cuenta que debemos codificar la trama con una codificación Manchester.

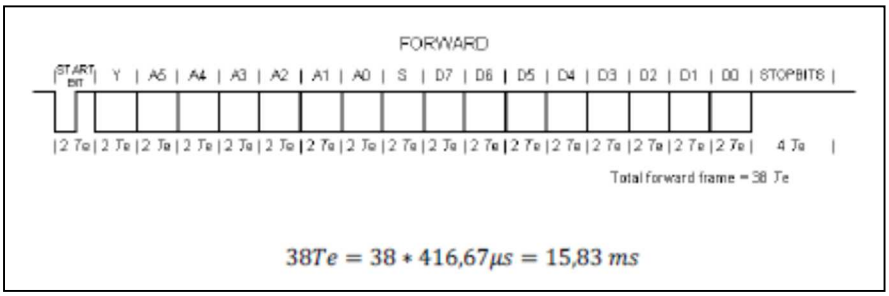


Figura 56. Tiempos de trama

$$Te = \frac{1}{2 * 1200} s = 416{,}67\mu s$$

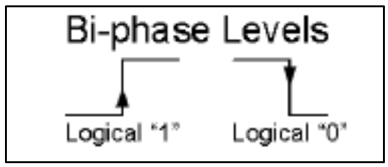


Figura 57. Codificación Manchester

Para realizar los cambios de estado marcados por la codificación y la temporalidad, necesitamos una base te tiempos la cual nos permitirá mantener durante el tiempo necesario el estado alto o bajo de cada bit. Para ello tomamos como referencia de base de tiempos Te, 416 microsegundos, ya que es medio bit. Por lo tanto cada 416 microsegundos haremos un cambio de estado.

Dentro de la plataforma de desarrollo disponemos de varios temporizadores que nos pueden ayudar, pero escogemos el sysTick.

4.7.1.1 SysTick

Se trata de un temporizador de 24 bits de cuenta descendente, que produce una interrupción cuando el registro interno llega a cero desde el valor de recarga inicial.

Offset	Registro	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	STK_CTRL *Valor de reset*	Reservado															COUNTFLAG 0	Reservado													CLKSOURCE 1	TICKINT 0	ENABLE 0
0x04	STK_LOAD *Valor de reset*	Reservado															RELOAD [23:0] 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x08	STK_VAL *Valor de reset*	Reservado															CURRENT [23:0] 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	STK_CALIB *Valor de reset*	Reservado															TENMS [23:0] 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 58. Registros SysTick

El flag COUNTFLAG se pone a 1 cuando el contador pasa del valor de cuenta 1a 0. Para que vuelva a 0 se puede leer el registro de control de SysTick, o poner a 0 la cuenta escribiendo cualquier valor en el registro de cuenta.

Su mayor ventaja es su sencillez, dado que no hay que configurar nada excepto el valor de recarga citado anteriormente. El inconveniente es que al no poder configurar preescalas ni otros parámetros las temporizaciones que se pueden realizar son bastante básicas.

4.7.1.1.1 Configuración del SysTick

Para configurar el SysTick deben seguirse los siguientes pasos:

- Desactivar el contador. ENABLE=0.
- Cargar el valor de RELOAD.
- Escribir cualquier valor en la cuenta para que se ponga a 0.
- Configurar los registros de control y estado, incluyendo la activación.

Esta funcionalidad es proporcionada por la función SysTick_Config(), que configura y pone en marcha el temporizador SysTick.
Concretamente, esta función CMSIS (Cortex Microcontroller Software Interface Standard) configura y realiza las siguientes funciones:

- Configura el registro de recarga del SysTick con el valor pasado como parámetro
- Configura la prioridad de la interrupción del SysTick al valor más bajo (IRQ priority).
- Configura la fuente de reloj para el contador del SysTick a HCLK - Core Clock Source.
- Habilita la interrupción del timer.
- Inicia el contador.

Para ajustar el tiempo base del SysTick se utiliza la siguiente fórmula:

Valor de recarga = SysTick Counter Clock (Hz) x Temporización deseada (s)

El SysTick Counter Clock es el reloj que le llega al temporizador SysTick, que en el STM32F4 es 168 Mhz. El valor de recarga es el parámetro que le pasamos a la función SysTick_Config(), que no debe exceder 0xFFFFFF. Si fuera necesario, se puede cambiar la prioridad del temporizador usando la función NVIC_SetPriority(SysTick_IRQn,...) después de invocar la función SysTick_Config(). La función NVIC_SetPriority() está definida en el fichero core_cm4.h

Ahora veremos cómo medir el tiempo, sabiendo la velocidad del reloj y realizando los cálculos indicados.

Reloj de 168 MHz y queremos medir 416 microsegundos.
Valor de recarga = 168 Mhz * 416 microsegundos = 70000

La instrucción para configurarlo es: SysTick_Config(Valor de recarga);

Normalmente el valor de recarga debe ser independiente del clock del micro, ya que si cambiamos la velocidad debemos cambiar este valor. Por lo tanto lo indicamos en función de la velocidad del reloj de la siguiente manera:

$$70000 = \frac{\text{SystemCoreClock}}{x}; x = 2400$$

Donde SystemCoreClock indica la velocidad del micro. En nuestro caso 168 MHz

Finalmente:
SysTick_Config(SystemCoreClock / 2400);

4.7.1.1.2 Implementación del manejador de interrupción asociado

Una vez configurado el temporizador, solo nos queda implementar el manejador de interrupción asociado. Dicho manejador es el void SysTick_Handler(void):

```
void SysTick_Handler(void)
{
        // aquí el código de la rutina de servicio del timer
}
```

Debemos pensar que esta rutina se ejecuta cada medio bit y programaremos aquello que deseemos ejecutar cuando termine la temporización configurada del SysTick. En principio podemos pensar en los siguientes pasos.

- Envío de bit start.
- Leer dirección destino bit a bit mediante rotaciones de bits. Envío cambio de estado pin de salida.
- Leer comando bit a bit mediante rotaciones de bits. Envío cambio de estado pin de salida.
- Mantener alto el bus durante dos bits de stop.

4.7.2 Recepción

Al igual que en transmisión, debemos escoger un pin de la plataforma de desarrollo que ira conectado a la parte de recepción de todos los disponibles, escogemos el PD3. Este pin tampoco tiene ninguna otra función asociada al microcontrolador.

La recepción consiste en una vez enviado un comando a un destino, escuchar una posible respuesta por parte del slave. Existe una serie de comando que tienen respuestas asociadas, como son las consultas sobre el estado de la lámpara, el master pregunta si la lámpara de un slave está encendida y el slave debe reponder con un SI o un NO. También disponemos de consultas sobre niveles de luminosidad….

Además en la recepción tenemos un caso similar a la transmisión respecto a los tiempos, necesitamos una base de tiempos, podemos utilizar la misma base. Usaremos el systick como base tiempos y su manejador asociado para la recepción de las tramas de respuesta por parte de los slaves.

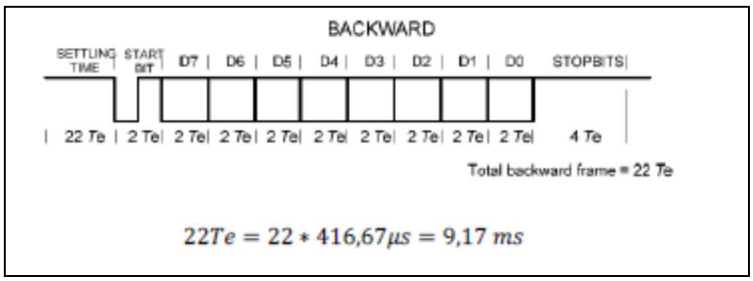


Figura 59. Trama Recepción

Debemos pensar que esta rutina se ejecuta cada medio bit y programaremos aquello que deseemos ejecutar cuando termine la temporización configurada del SysTick. En principio podemos pensar en los siguientes pasos.

- Escuchar el bus si hay primer cambio de estado (bit start),
    - seguir escuchando el bus y por cada cambio de estado será un bit a almacenar.
    - Seguir así hasta la recepción total de la trama o time out.

Hemos comentado en el primer paso, escuchar el bus para verificar si hay un primer cambio de estado (bit start). Pero podemos optar por varias soluciones. La primera es tomar muestras repetitivamente del bus y comprobar si hay cambio de estado. Otra soluciones es tener una “base de datos” de los comandos indicando si es un comando con respuesta o no para evitar tener que comprobar si hay siempre respuesta escuchando el bus. Otra solución, es utilizar las interrupciones y cuando se detecta un cambio de estado se ejecutar el manejador de interrupciones asociado al igual que en systick, y allí tendremos nuestra recepción del comando.

Para evitar estar siempre escuchado el bus, utilizaremos las interrupciones. Es decir asociaremos una interrupción al pin de recepción en nuestro caso PD3 y cuando se de ese primer cambio de estado lo notificaremos al micro y ejecutaremos la rutina de interrupción asociada a ese pin. Exactamente la misma casuística que en el sistyck, pero esta vez el responsable será el pin de recepción.

Debemos tener claros ahora varios aspectos:

- Hemos escogido 2 pines uno para transmisión PD1 y otro para recepción PD3.
- Tenemos una base de tiempo que se ejecuta cada 416 microsegundos que hay que compartir entre transmisión y recepción. SysTick
- Ejecutaremos una interrupción al recibir el primer bit.
- Esperaremos respuesta de todos los comandos activando la interrupción de recepción, en caso de no activarla pasado un tiempo time_out. Caso de activarse gestionaremos la interrupción.

4.7.3 Maquina de estados.

Ya tenemos definidas las estrategias tanto para transmisión como para recepción. Ahora necesitamos tener un control sobre estas.

Se denomina máquina de estados a un modelo de comportamiento de un sistema con entradas y salidas, en donde las salidas dependen no sólo de las señales de entradas actuales sino también de las anteriores. Las máquinas de estados se definen como un conjunto de estados que sirve de intermediario en esta relación de entradas y salidas, haciendo que el historial de señales de entrada determine, para cada instante, un estado para la máquina, de forma tal que la salida depende únicamente del estado y las entradas actuales. Nuestra máquina de estados quedará definida de la siguiente manera:

Tenemos tres estados, transmisión, recepción y espera. Inicialmente estaremos en espera. Si llega una interrupción por parte de recepción pasaremos al estado de recepción. Mantendremos en ese estado hasta que recibamos toda la trama o exista algún error, que volveremos al estado de espera. Cuando tengamos algo que enviar cambiaremos de estado a transmisión. Una vez finalizada la transmisión, pasaremos a espera de recepción, en caso que el comando no responda saltara el temporizador de Time_out, regresando al estado de espera inicial. Si hay respuesta, se activará la interrupción generada en PD3, para finalmente acabar siempre en espera inicial. Las máquinas de estados las solemos implementar con estructuras de control if then o switch case.

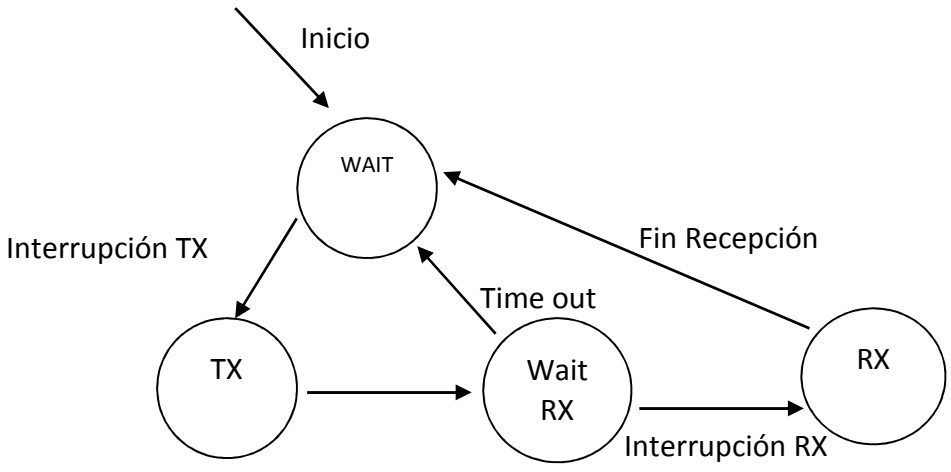


Figura 60. Maquina de estados.

4.7.4 Implementación del software.

4.7.4.1 Main.c

Tenemos un programa **main.c** que es el punto de entrada a la ejecución del firmware. Antes de realizar cualquier acción en un microprocesador o plataforma de desarrollo, debemos inicializarla por eso la primera instrucción que ejecutamos es una llamada a un procedimiento que nos inicializa la placa. Acto seguido inicializa la plataforma para convertirla en un Master Dali y finalmente entre en el bucle infinito donde está implementada la máquina de estados.

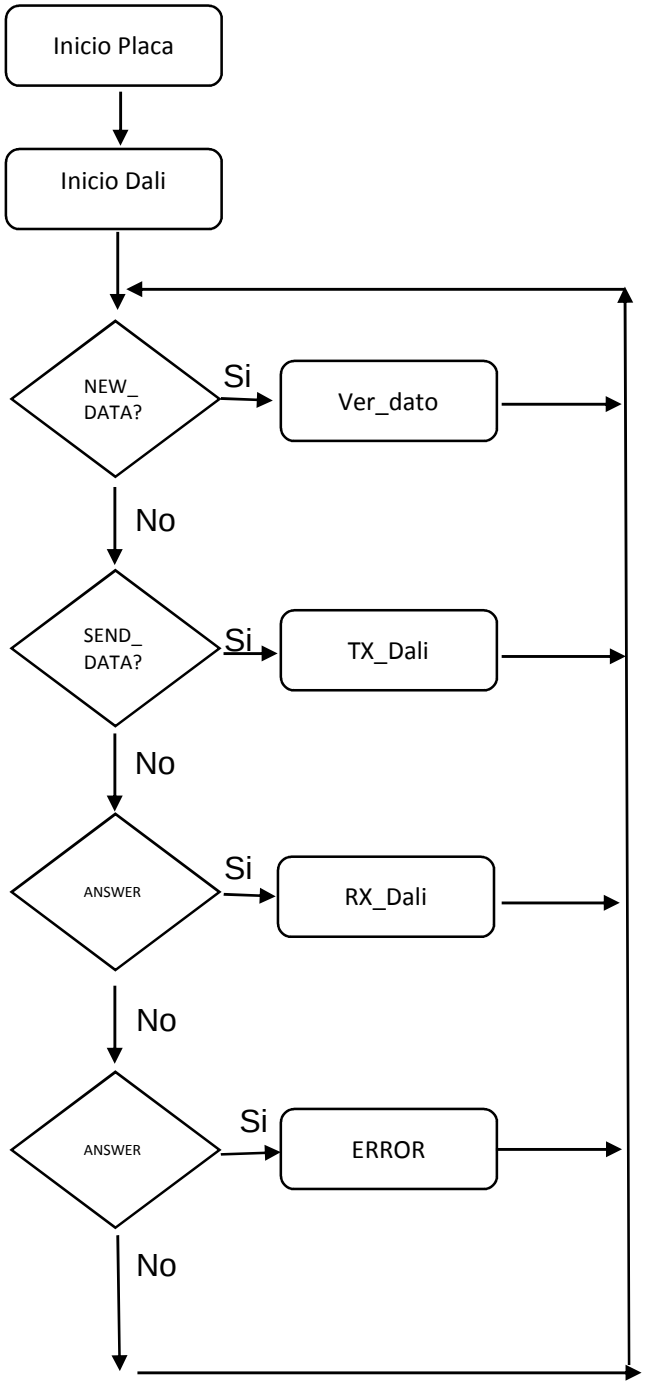


Figura 61. Software diagrama Main.c

4.7.4.2 InicioPlaca.c

Este programa solo nos inicializa la placa de la plataforma **Inicio_Placa().** Hace 3 acciones básicas de Inicialización, la capa HAL de programación, los leds, el reloj del sistema, configura la base de tiempos del sistema anteriormente comentada a 4800. Posteriormente en dificultades encontradas, comentaremos esta configuración ya que habíamos calculado su inicialización en 2400, doble de la frecuencia de bus. En este caso es 4 veces superior.

4.7.4.3 Dali.c

En primer lugar encontramos el procedimiento **Inicio_Dali(),** el cual inicializa los elementos necesarios para la comunicación. Configura el pin PD3 como entrada, el pin PD1 Como salida, habilita los relojes para los puertos utilizados. Configura también el pind PA0 como entrada, que es donde está conectado el pulsador de la placa. Y asocia interrupciones de entrada tanto a la línea PA0, como a la línea de entrada PD3.

```
GPIO_InitTypeDef GPIO_InitStructure;
 //Pin out DALI  PD1
 /* Pull-up Vdd pin for data output */
 GPIO_InitStructure.Pin =  OUT_DALI_PIN ;
 GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_HIGH;
 HAL_GPIO_Init(OUT_DALI_PORT, &GPIO_InitStructure);

  //Mantener el bus a 1  OBLIGATORIO
 //HAL_GPIO_WritePin( OUT_DALI_PORT, OUT_DALI_PIN, GPIO_PIN_SET);
 busReposo();

 //PIN IN DALI PD3
 GPIO_InitStructure.Pin =  IN_DALI_PIN;
 GPIO_InitStructure.Mode  =  GPIO_MODE_IT_FALLING;//GPIO_MODE_IT_RISING_FALLING;//  //Flanco
bajada
 //GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_HIGH;
 GPIO_InitStructure.Pull= GPIO_NOPULL;
 HAL_GPIO_Init(IN_DALI_PORT, &GPIO_InitStructure);

/* Enable and set Button EXTI Interrupt to the lowest priority */

 HAL_NVIC_SetPriority(EXTI0_IRQn, 3, 0);
 HAL_NVIC_EnableIRQ(EXTI0_IRQn);

 HAL_NVIC_SetPriority(EXTI3_IRQn, 3, 0);
 HAL_NVIC_DisableIRQ(EXTI3_IRQn);
```

TX_Datos_Dali() y RX_Datos_Dali(). Corresponden con la llamadas desde el bucle principal en el programa main.c cuando se produce un cambio de estado tanto de emisión de un comando como de recepción de respuesta. En ellas se inicializan las variables que integran los procesos relacionados. En La parte de transmisión, encontramos la dirección y el comando a eviar.

```
void TX_Datos_Dali(void)
{

   direccion=DIRECT_ARC_POWER;
   comando=OFF;  //Apagado
   //comando=0xFE;  //TOPE
   //comando=0xAB;

   //direccion=0xFF;
   //comando=QUERY_ACTUAL_LEVEL;

   //inicializa variables
   send_position=0;
   send_value=0;
   send_active=TRUE;

}

void RX_Datos_Dali(void)
{

  rec_start=TRUE;

}
```

HAL_GPIO_EXTI_Callback(). Este procedimiento es la retrollamada de los manejadores de interrupciones, correspondientes a las líneas 3 y 0. Es la agrupación de los manejadores de interrupción. Según se haya activado en una línea u otra, cambiamos de estado a enviar datos o recibir datos.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  //Envio Comando
  if (KEY_BUTTON_PIN == GPIO_Pin)
  {
   //Si es el pin del pulsador
    flag |=SEND_DATA;
}
//Recibo comando
if (IN_DALI_PIN == GPIO_Pin)
{
    flag |=ANSWER_EVENT;
}
}// Fin HAL_GPIO_EXTI_Callback
```

SysTick_Handler(void). Podemos decir que este es el procedimiento principal de toda la implementación, es nuestra base de tiempos configurada con el valor de 4800 es decir se ejecuta cada 208 microsegundos. Recordamos que esta base de tiempos, es compartida por transmisión y recepción y es donde tenemos la emisión y la recepción propiamente dicha.

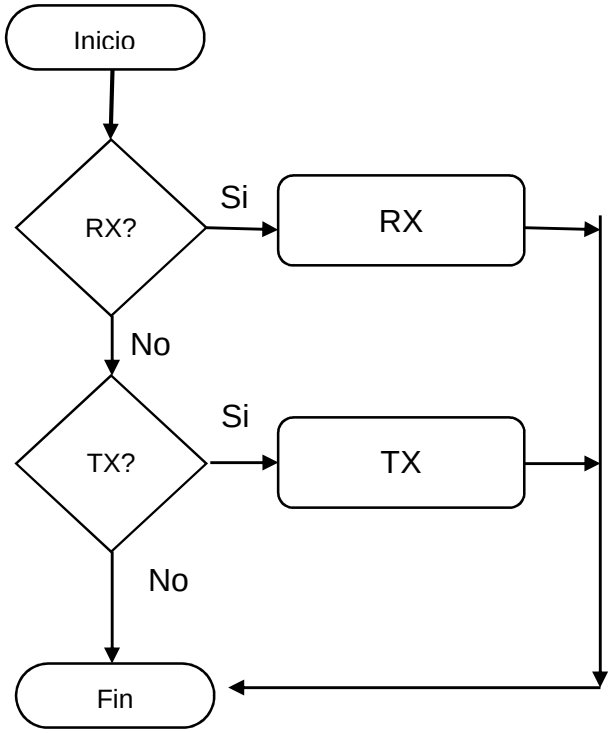


Figura 62: Software Diagrama SysTick

Describimos inicialmente la transmisión:

Sabemos que tenemos que enviar 19 bits en total, bit start + dirección +comando + 2 bits stop. La norma nos indica exactamente los tiempos de cada bit. Podemos decir que por cada bit tenemos dos medias partes si lo configuramos a velocidad normal 2400, en nuestro caso por cada bit tenemos 4 medias partes, ya que hemos duplicado por 2 la velocidad. Por consiguiente, tenemos que "rellenar" un total de 19 * 4 = 76 partes. Con esta premisa podemos hacer la siguiente gráfica. Además según la codificación manchester las dos primera partes del bit es el dato negado y las dos segundas partes el dato que queremos enviar. Se puede ver en el bit de start cuyo valor es 1.

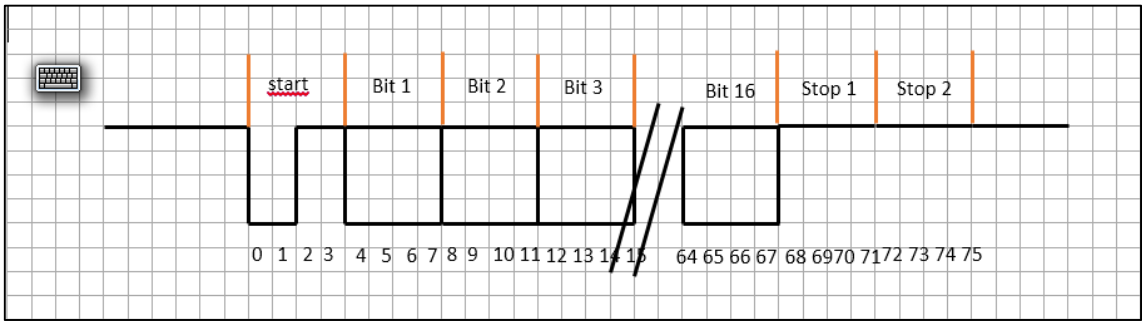


Figura 63: Software transmisión

Ahora definimos como enviar la trama. Una vez activada la interrupción de enviar comando, inicializamos una variable posición=0. Vamos contando la cantidad de veces que ejecutamos este procedimiento y acumularlo en esa variable. Esto nos indica la posición en que estamos situados, como si de un cursor se tratase. Vamos tratando la posición a tratar y finalmente mantendremos el bus en alto en el tiempo correspondiente a los dos bits de stop.

```
//EVIAR COMANDO
 if (send_active==TRUE)
 {
    //Esta el bus a 1
   //bit inicio
    if (send_position<=3)
    {
     enviaDato(send_position,1);
    }
    //direccion y dato
    else if (send_position>=4 && send_position<=67)
    {

     if(send_position<=35)
     {
      //direccion 4 35
      temp_value = (direccion>>((35-send_position)/4)) & 0x01;//35 pos maxima 4 muetras por bit

     }
     else
     {
      //comando 36 67
      temp_value = (comando>>((67-send_position)/4)) & 0x01;//67 pos maxima 4 muetras por bit
     }

     enviaDato(send_position,temp_value);

    }
    //bit 1 final
    //else if (send_position<=68 && send_position<=71)
    else if (send_position==68)
    {
      busReposo();
    }
    //bit 1 final
    /*else if (send_position<=72 && send_position<=75)
    {
      busReposo();
    }*/
    //Salir
    else if (send_position>=76)//76
    {

     //Habilitamos escucha por interrupcion e inicializamos
     //Fin comando
     send_active = FALSE;

     //Inicializo escucha
     rec_position=0;
     rec_bit=0;
     rec_value=0x00;
     temp_value=0x00;
     respuesta=0x00;

     //Activo int ext 3 y activo la escucha
     //Me pongo a escuchar
     rec_active=TRUE;
     rec_start=FALSE;

     HAL_NVIC_EnableIRQ(EXTI3_IRQn);
     HAL_NVIC_DisableIRQ(EXTI0_IRQn);

//HAL_NVIC_ClearPendingIRQ(EXTI3_IRQn);
    }


    send_position++;
  }//Fin envio datos send_active=TRUE
```

Ahora según la posición y el bit a enviar sea 1 o 0 (comando o dirección), mantenemos la salida a 1 o 0 del pin de salida el tiempo necesario teniendo en cuenta la codificación manchester descrita. Esto queda resumido en la siguiente rutina:

```
void enviaDato(unsigned char posicion,unsigned char dato)
{
 //deteccion en que parte del bit estamos
    switch (posicion % 4)
    {
       //primera mitad siempre inverso
       case 0:
       case 1:
       {
         dato=!dato;
         break;
       }
       //Segunda mitad mismo dato
       case 2:
       case 3:
       {
         dato=dato;
         break;
       }

     }//Fin switch

    if (dato==0)
    {
      //envio 0  //Bus a 5 voltios
       HAL_GPIO_WritePin( OUT_DALI_PORT, OUT_DALI_PIN, GPIO_PIN_RESET);
       led_Verde_OFF();
       //led_Rojo_OFF();
    }
    else
    {
      //envio 1//Bus a 20 voltios
      HAL_GPIO_WritePin( OUT_DALI_PORT, OUT_DALI_PIN, GPIO_PIN_SET);
      led_Verde_ON();
      //led_Verde_OFF();
    }

} //Fin envia datoa
```

Para la recepción:

Utiliza la misma técnica de las posiciones y el tiempo. La única diferencia ahora es que una vez activada la recepción el sincronismo depende del primer flanco de bajada del bit de start, a partir de ese flanco se van tomando 4 muestras por cado bit recibido. Lo que hacemos a continuación es quedarnos solo con la muestra que nos interesa, el resto se desecha. Volviendo a la gráfica anterior se ve concretamente las muestras que necesitamos.

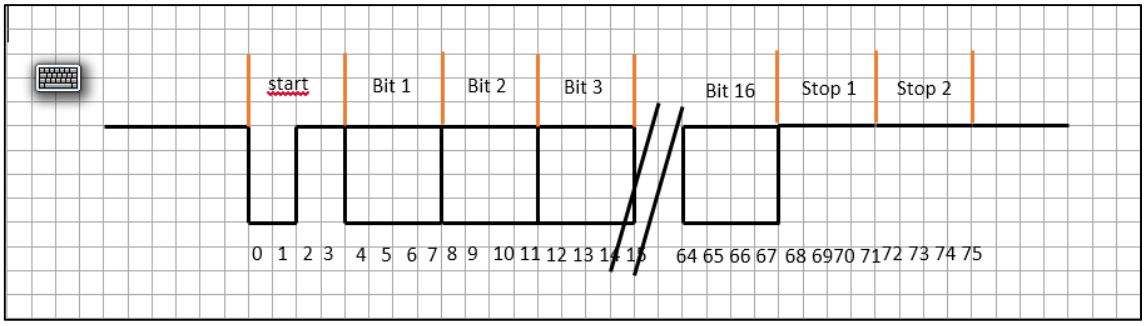


Figura 64. Software. Trama recepción

Sabemos por la codificación manchester que la según parte del bit nos da el dato, tenemos: Para el bit de stop, el valor lo da la muestra 2 (1), Para el bit 1 su valor lo marca la muestra 6 (X), para el bit 2 su valor lo marca la muestra 10 (X)…..Es decir tomamos como referencia la muestra 2 y después cada 4 muestras tenemos el valor del bit que nos responde el slave Dali. Al final habilitamos interrupciones. Vemos en el código que no son las muetras descritas en el texto, esto es debido al tiempo de espera en que nos responde el slave.

```
if (rec_active==TRUE)
 {
  //Time out lo marca la norma cuanto hay que escuchar
  if( rec_start==FALSE && rec_position>=44)
  {
    //Para salir
    rec_active=FALSE;
    rec_start=FALSE;
     rec_position=0;

      HAL_NVIC_DisableIRQ(EXTI3_IRQn);
     //habilito la interrupcion del pulsador
     HAL_NVIC_EnableIRQ(EXTI0_IRQn);
     //deshabilito la interupcion de escuchar
     //no hay nada que escuchar

    //Indicar no hay respuesta
    //el numero de bits recibidos es cero

  }//Fin rec_start  FALSE

  else if (rec_start==TRUE)
  {
   //leo puerto cada tick en las muestras necesarias calculadas

    switch (rec_position)
    {
     //Son posiciones que corresponden con cada tick de respuesta
      case 31:
      case 35:
      case 39:
      case 43:
      case 47:
      case 51:
      case 55:
      case 59:

      {
       //leo el puerto de entrada segun muestra
       //Parpadeo del bit azul veo qeu leo y recibo algo
```

```
      rec_bit=HAL_GPIO_ReadPin(
IN_DALI_PORT, IN_DALI_PIN);
       if (rec_bit==0)
          led_Azul_OFF();
       if (rec_bit==1)
         led_Azul_ON();
       //Acumulo los bits recibidos
       rec_value = (rec_value << 1) |rec_bit;

      break;
     }
   case 68:
    {
     //fin de recepcion
     led_Azul_OFF();
     rec_start=FALSE;
     rec_active=FALSE;
     rec_position=0;
     HAL_NVIC_DisableIRQ(EXTI3_IRQn);
     HAL_NVIC_EnableIRQ(EXTI0_IRQn);
     break;
    }
   }//Fin switch

   }//fin rec_start
   rec_position++;

}//Fin rec_active
```

4.7.4.4 stm32f4xx_it.c.

En realidad los manejadores de interrupciones tanto de la línea PD3 y PA0 se encuentran en este programa. Lo que hacen ambos manejadores es deshabilitar las interrupciones para evitar reentradas y hacer una llamada a la función de retrollamada de las interrupciones, pasándole como argumento el pin en el cual se ha producido la interrupción. Por ello somos capaces de discernir donde se ha producido la interrupción.

```
/**
  * @brief  This function handles External line 0 interrupt request.
  * @param  None
  * @retval None
  */
void EXTI0_IRQHandler(void)
{
 HAL_NVIC_DisableIRQ(EXTI0_IRQn);
 HAL_NVIC_DisableIRQ(EXTI3_IRQn);
 HAL_GPIO_EXTI_IRQHandler(KEY_BUTTON_PIN);

}

void EXTI3_IRQHandler(void)
{
 HAL_NVIC_DisableIRQ(EXTI0_IRQn);
 HAL_NVIC_DisableIRQ(EXTI3_IRQn);
 HAL_GPIO_EXTI_IRQHandler(IN_DALI_PIN);
}
```

4.7.4.5 UtilidadesPlaca.c

Se han implementado unos procedimientos de apoyo que controlan los 4 leds, así nos quedan referenciados por colores y no por pines. También podemos encender y apagar todos con un solo comando.

4.7.4.6 Dali_Comandos.h

Se define los comandos que componen la norma dali. No están definidos todos ellos, pero si varios de los que hemos probado y utilizado

```
//comandos de control lampara
#define OFF 0x00
#define UP 0x01
#define DOWN 0x02
#define STEP_UP 0x03
#define STEP_DOWN 0x04
```

4.7.4.7 Dali_Config.h

Se define los pines y los puertos de entrada y salida. Es una referencia rápida, ya que los podemos cambiar y no hay que tocar ninguna otra parte de código.

```
#define OUT_DALI_PORT     GPIOD //PB1 = TX
#define OUT_DALI_PIN      GPIO_PIN_1
//#define INVERT_OUT_DALI   0

#define IN_DALI_PORT      GPIOD //PB0 = RX
#define IN_DALI_PIN       GPIO_PIN_3
```

4.7.4.8 DatosComunes.h

Se definen las variables globales del sistema, asi como los estados marcados anteriormente marcados por la variable flag

```
#define TRUE 1
#define FALSE 0

// Global definition
extern unsigned char direccion; // direccion DALI
extern unsigned char comando; // comando DALI
extern unsigned char respuesta; // respuesta DALI
extern unsigned char error; // codigo error
extern unsigned int flag; // Nos indica los eventos

#define NEW_DATA 0x01 // Nuevos datos
#define SEND_DATA 0x02 // Para el envio de datos
#define ERROR_EVENT 0x04 // Si hay error
#define ANSWER_EVENT 0x08 // Respuesta de comandos
```

4.7.5 Maqueta del sistema Completo.

Se ha descrito con detalle todas las partes de las que consta el proyecto tanto hardware como software. Solo resta unir todas las partes y construir el sistema completo. Se aprecia en la figura la maqueta del sistema construido con sus partes diferencias,

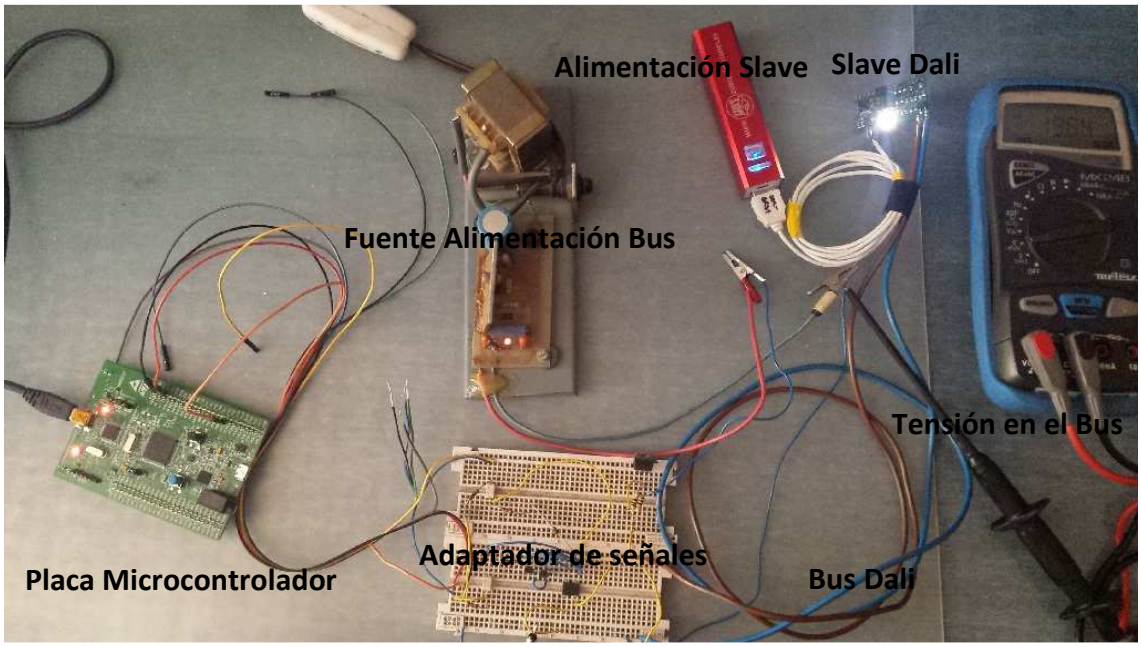


Figura 65 :Maqueta Sistema Completo

Prototipo Adaptador de señales

Aquí se puede apreciar en detalle del adaptador de señales con sus partes descritas y calculadas anteriormente.

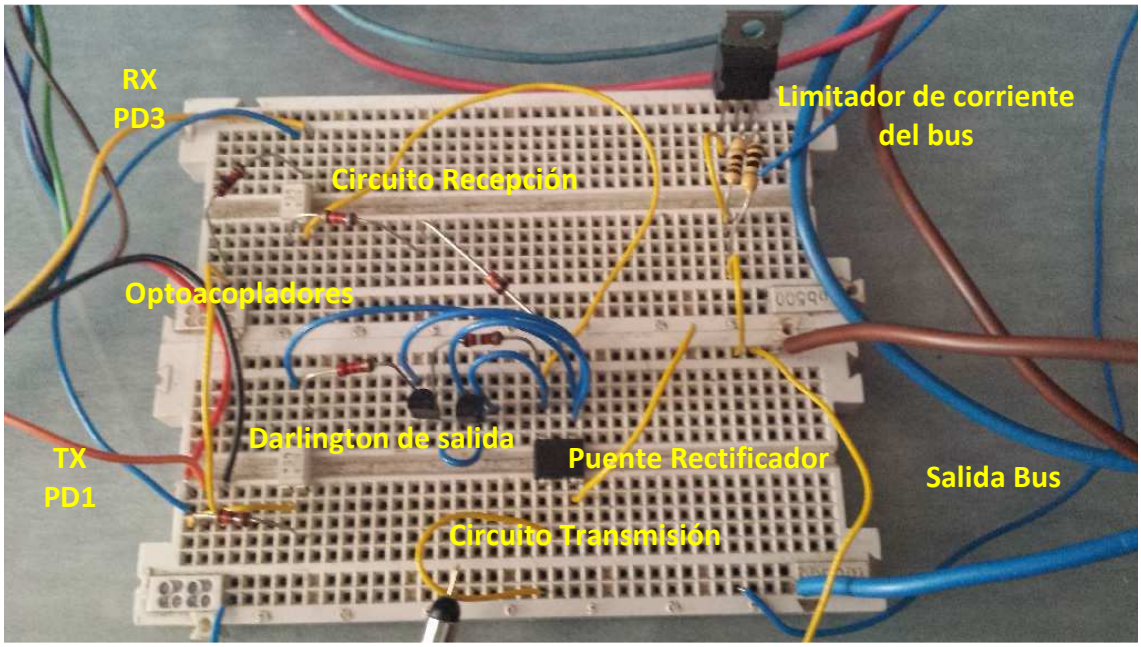


Figura 66: Prototipo Adaptador de señales

4.7.6 Encender apagar luminaria con un pulsador. Envío comando off/on.

Nuestro objetivo en este punto es usando el pulsador que lleva la placa de desarrollo apagar y/o encender el led. Todo el dialogo se hará sobre una dirección de broadcast especial, es la "direct arc power level". Ya que no contemplamos por ahora el uso del direccionamiento individual de las luminarias. Este comando lo que hace es acceso directo al registro donde se almacena el valor de la luminosidad y escribe el valor indicado. Su valor puede variar desde 0 a 254, la norma indica que convertirlo en lúmenes sigue una curva logarítmica. El comando de apagado consiste en enviar 00.

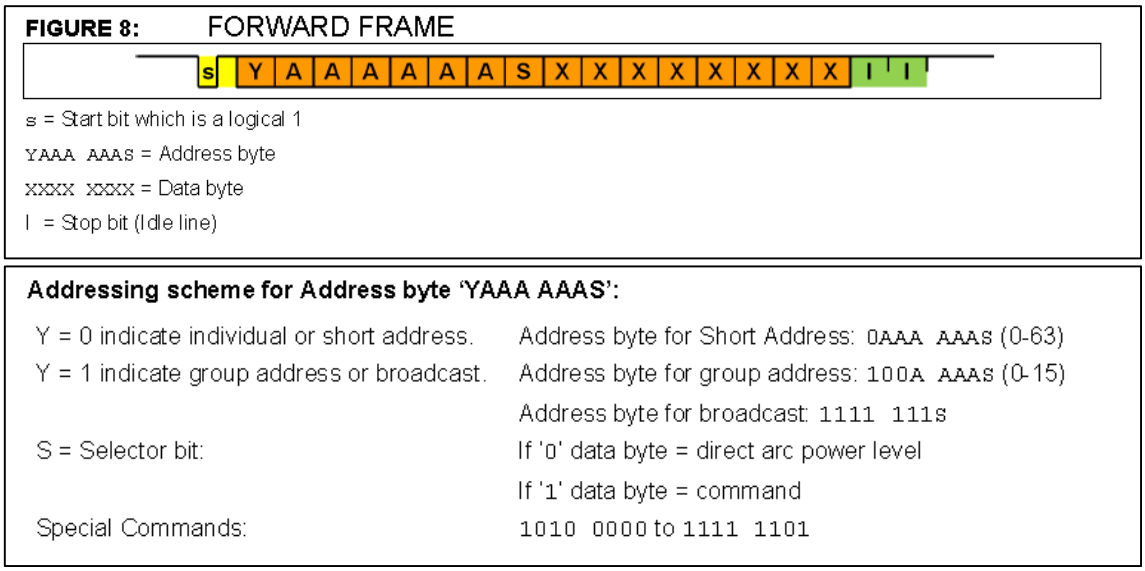


Figura 67. Enviar Trama apagar Led.

La trama a enviar es: 1 1111 1110 0000 0000 1 1

Para modificar el comando a enviar debemos ir al programa Dali.c al procedimiento TX_Datos_Dali() y modificar manualmente las variables dirección y comando con los valores siguientes. Hay que cargar nuevamente el programa en el microcontrolador cada vez que queramos modificar un comando

```
void TX_Datos_Dali(void)
{

  //Comandos a eviar
  //broadcast OFF
  direccion=DIRECT_ARC_POWER;
  //comando=0xDC; //Mitad
  comando=OFF; //Apagado
  //comando=0xFE; //TOPE
  //comando=0xAB;
```

Aquí vemos la captura con un analizador de la trama enviada que apaga la luz de la luminaria. Y después la trama que enciende la lámpara.

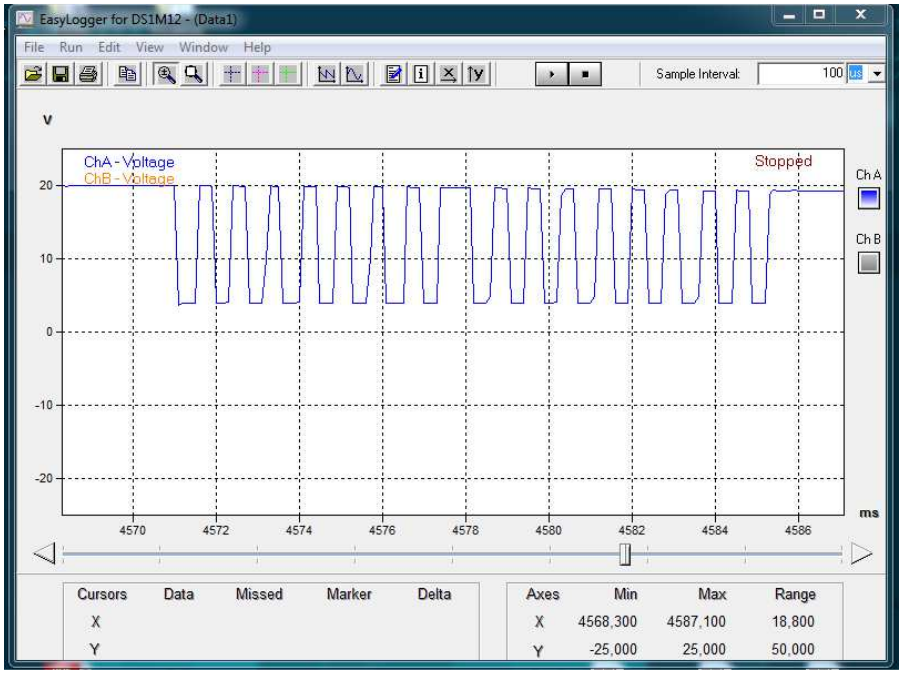


Figura 68: Captura trama apagar luminaria

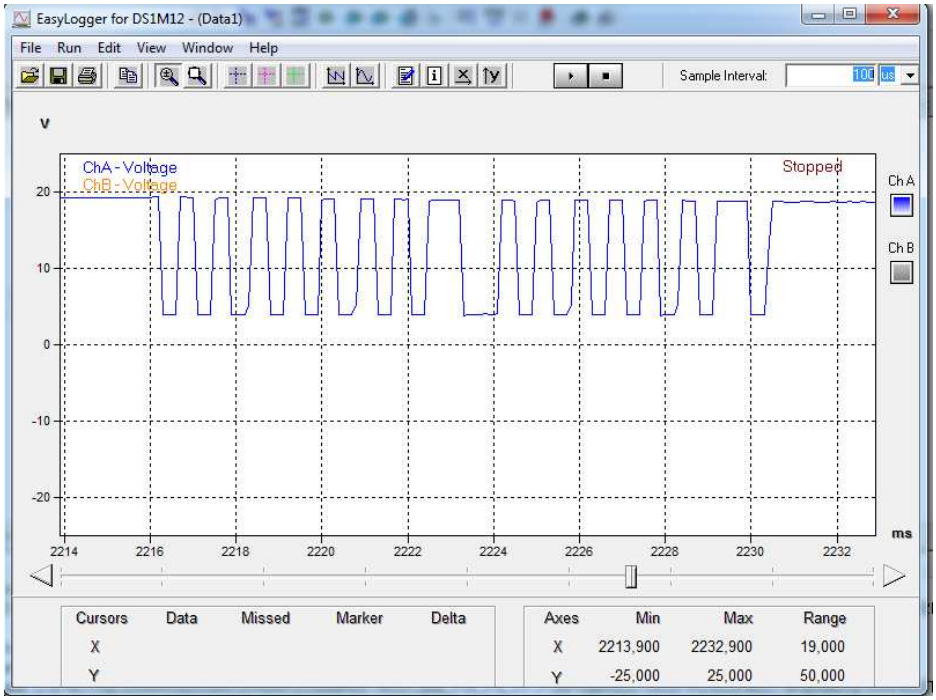


Figura 69: Captura trama encender luminaria

4.7.7 Envió de comandos con respuesta.

Una vez comprobado en transmisión, debemos verificar que el dispositivo realmente responde a los comandos enviados y ver si podemos leer las respuestas y coincidan con lo esperado. Para comprobar la respuesta vamos a realizarla en dos pasos.

En primer lugar utilizaremos la dirección DIRECT_ARC_POWER con un valor de luminosidad precargado. En este primer comando no hay respuesta por parte del dispositivo slave. Después le preguntaremos al slave que valor de luminosidad tiene cargado. Ahora el salve nos responderá con el valor que le habíamos enviado con el anterior comando.

Para modificar el comando a enviar debemos ir al programa Dali.c al procedimiento TX_Datos_Dali() y modificar manualmente las variables dirección y comando con los valores siguientes.

```
 void TX_Datos_Dali(void)
{

   //Comandos a eviar
   //broadcast OFF
   direccion=DIRECT_ARC_POWER;
   //comando=0xDC;  //Mitad
   //comando=OFF;  //Apagado
   //comando=0xFE;  //TOPE
   comando=0xAB;
```

Una vez enviado, paramos el sistema, modificamos el comando y se lo lanzamos al slave.

```
void TX_Datos_Dali(void)
{

   //Comandos a eviar
   //broadcast OFF
   //direccion=DIRECT_ARC_POWER;
   //comando=0xDC;  //Mitad
   //comando=OFF;  //Apagado
   //comando=0xFE;  //TOPE
   //comando=0xAB;

   direccion=0xFF;
   comando=QUERY_ACTUAL_LEVEL;
```

Captura de tramas en ambos comandos.

Poner el led con un nivel de luz de 0xAB

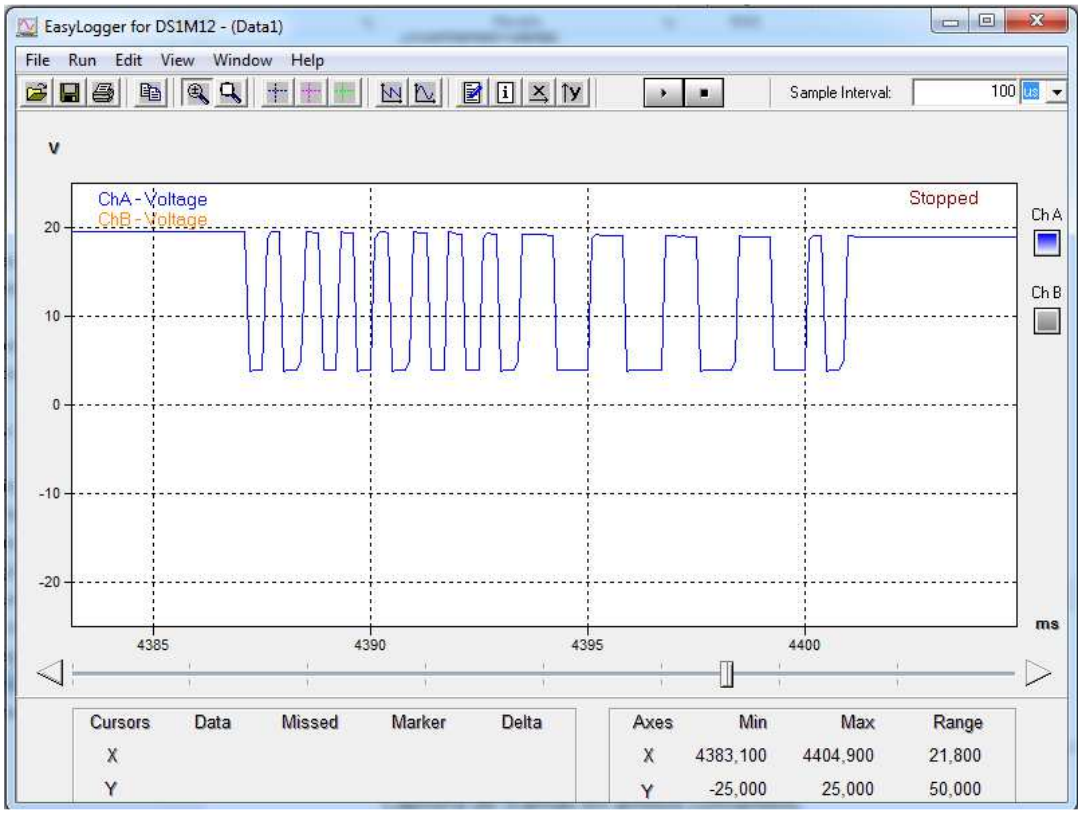


Figura 70: Trama Envio luminosidad 0xAB

Una vez enviado este comando la luz del led no es tan brillante como inicialmente ni en estado apagado.

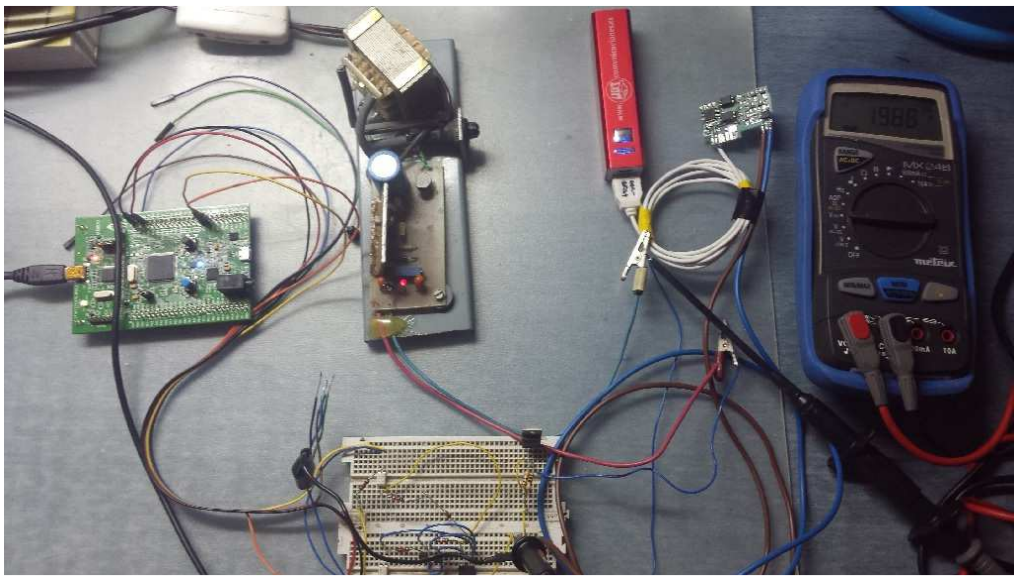


Figura 71: .Foto Maqueta Led con nivel luminosidad 0xAB

Ahora le preguntamos que nivel de luminosidad tiene, vemos el comando y la respuesta del salve. Para comprobar la respeusta debemos colocar un punto de ruptura en el codigo donde justo acabemos de leer la respuesta al comando.

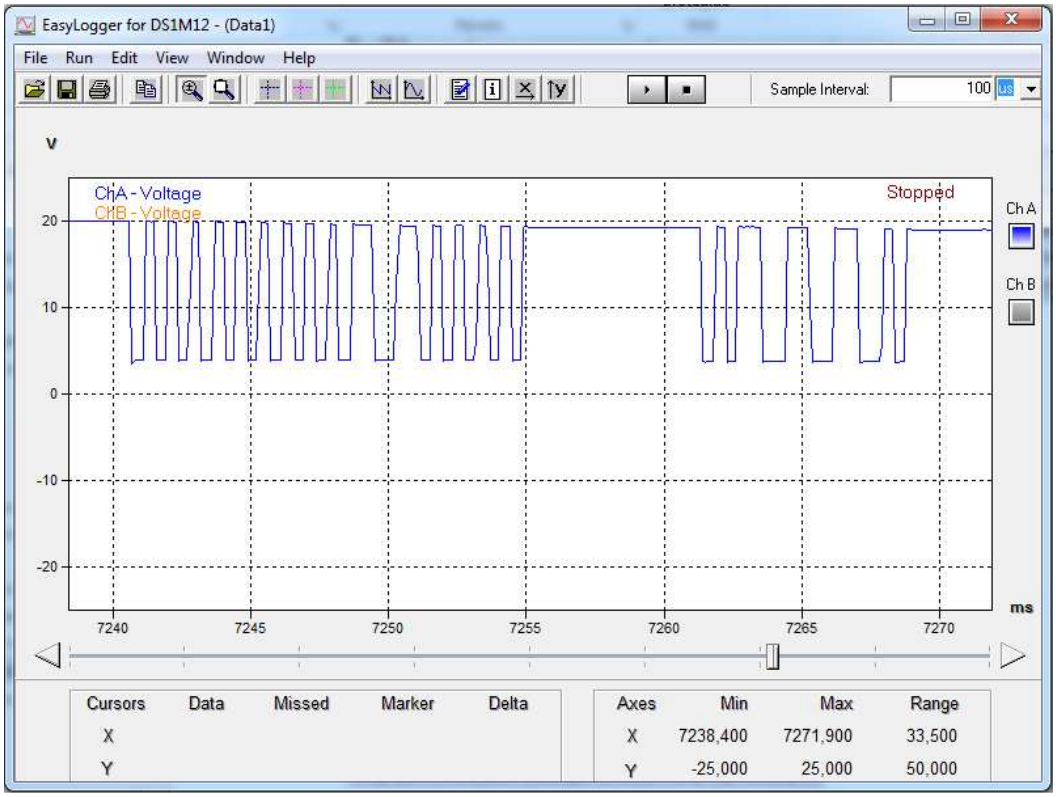


Figura 72: Trama que pregunta cuanta luminosidad y la respuesta al comando

Apreciamos en la captura el valor obtenido en decimal 171, corresponde al valor 0xAB en hexadecimal. Punto de ruptura en el programa.

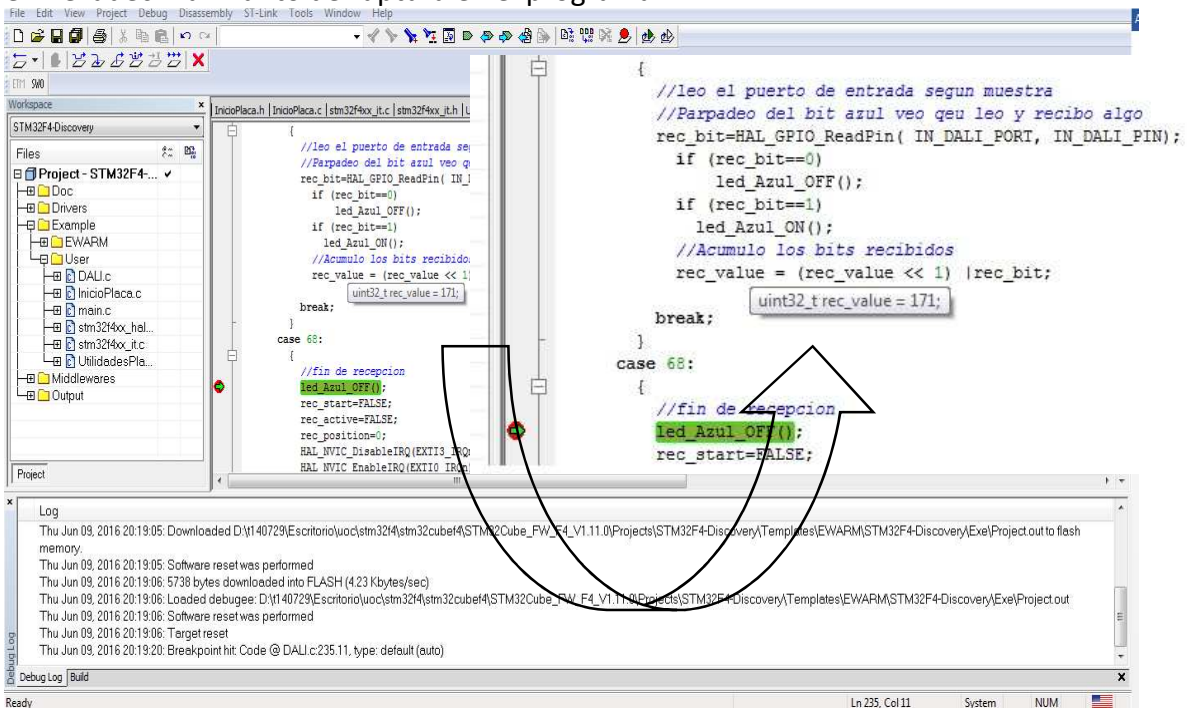


Figura 73: Respuesta al comando. Luminosidad

5. Conclusiones

El proyecto es interesante tanto en cuanto se ha realizado una maqueta de un sistema de iluminación real, que puede estar instalado en cualquiera de los ámbitos comentados en el estado del arte. Además de todos los conocimientos adquiridos en el desarrollo del mismo. Un aspecto a destacar ha sido el cálculo del hardware asociado con el adaptador de señales y después de haberlo construido ha funcionado sin ningún tipo de ajuste.

No ha estado exento de problemas para los cuales se ha buscado soluciones sencillas para su implementación. El más problemático ha estado relacionado con la base de tiempos del sistema, ya que una vez alcanzado el objetivo principal de poder establecer comunicación con la luminaria el objetivo no alcanzado es dotarle de comunicación externa mediante WIFI ni mediante puerto USB, emulando un puerto serie (VCP).

El problema es que se ha modificado la base de tiempos del sistema y esta base de tiempos, también es usada por las mediciones de tiempo de las uart, por consiguiente los cálculos internos para las transmisiones y recepciones mediante las uart no se podían calcular correctamente y no han funcionado los programas probados. Por defecto el sistema crea una base de tiempos de 1 milisegundo, se referencias muchas cosas a ella como por ejemplo los cálculos de los retardos y nosotros la hemos modificado a 208 microsegundos.

La razones básicas de haber modificado este valor que inicialmente estaba en 416 microsegundos es que para poder componer la señal de salida es necesario mínimo el doble de muestras de la velocidad, como la velocidad es de 1200 Hz, pero un bit es la composición de dos mitades la velocidad es de 2400 Hz que todo el código se había ajustado a este valor pero para recomponerla la señal de entrada son necesarios 4800 Hz. Este error de cálculo, ha derivado en volver a rehacer el código para ajustarlo a esa velocidad superior.

La solución a adoptar para evitar modificar la base de tiempo del sistema, es crear una propia con un temporizador de propósito general. Y cada vez que haya que enviar o recibir un dato usarla. Así nos queda libre para comunicaciones con uart.

También se ha escogido la solución a la hora de recibir datos de tomar muestras del puerto de entrada en base a una interrupción y quedarnos con las que nos interesan en función de la posición leída. Una técnica más depura es utilizar dos timers y verificar el tiempo entre dos transiciones con la medición de ese tiempo, se puede saber si el bit leído es 1 o 0.

Al no poderlo dotar de la comunicación esperada, no se ha podido programar secuencias de comandos para que se vayan ejecutando secuencialmente, lo que se llama programar una escena. Por ejemplo variaciones de luminosidad en función de la hora, etc…

6. Bibliografía

[1] Título del recurso. Discovering the STM32 Microcontroller
Autor/s. Año: Geoffrey Brown 2012
Fuente de información http://www.cs.indiana.edu/~geobrown/book.pdf

[2]Título del recurso. Embedded software in c for ARM Cortex
Autor/s. Jonathan W. Valvano and Ramesh Yerraballi. 2015
Fuente de información.
https://www.inonu.edu.tr/uploads/contentfile/405/files/Embedded%20Software%20in%20C%20for%20ARM%20Cortex%20M.pdf

[3]Título del recurso. The Definitive Guide to the ARM Cortex-M3
Autor/s. Año Joseph Yiu. 2010
Fuente de información. Editorial: Elsevier ISBN 978-1-85617-963-8

[4] Título del recurso. Web del fabricante de la plataforma de hardware
Autor/s. ST microelectronics
Fuente de información.
Discovery kit with STM32F407VG MCU
http://www2.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/stm32f4discovery.html

STM32F4DIS-WIFI
http://www2.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-3rd-party-evaluation-tools/stm32f4dis-ext.html

[5]Título del recurso. Web sobre el protocolo DALI
Autor/s. Grupo DALI
Fuente de información. http://www.dali-ag.org/

[6]Título del recurso. Diseño de Master y slave DALI
Autor/s. NXP Semiconductors y Motorola
Fuente de información.
DALI slave using the LPC111x
http://www.mouser.com/pdfdocs/AN11174.pdf
DALI master using the LPC2141
http://www.nxp.com/documents/application_note/AN10760.pdf
Digitally Addressable Designer Reference Manual Lighting Interface (DALI) Unit Using the
MC68HC908KX8
http://www.nxp.com/files/microcontrollers/doc/ref_manual/DRM004.pdf

[7]Título del recurso. *Entornos de programación*
Autor/s. *Varios*
Fuente de información.
IAR
https://www.iar.com/
Coocox
http://www1.coocox.org/index.html
Uso de netbeans desarrollo embebido
http://elvioladordesegmentos.net/es/node/65
Uso de visual Studio express desarrollo embebido
http://visualgdb.com/tutorials/arm/stm32/

[8]Título del recurso. *Conexiones mediante WIFI*
Autor/s. *Varios*
Fuente de información.
Usando el acelerometro y envio de datos por wifi
https://www.element14.com/community/community/designcenter/single-board-computers/blog/2015/06/21/sending-a-signal-over-wifi-based-on-an-accelerometer
Placa desarrollo WIFI STM32
http://datasheet.octopart.com/STM32F4DIS-WIFI-STMicroelectronics-datasheet-17455206.pdf

[9]Título del recurso. *Conexiones mediante USB*
Autor/s. *Varios*
Fuente de información.
http://stm32f4-discovery.net/2014/09/library-34-stm32f4-usb-hid-device/
http://www.pezzino.ch/stm32f3-discovery-usb-pc-communication/

[10]Título del recurso. *Decodificacion Manchester*
Autor/s. *Varios*
Fuente de información.
http://www.st.com/content/ccc/resource/technical/document/application_note/03/9c/4f/86/78/12/46/25/DM00053084.pdf/files/DM00053084.pdf/jcr:content/translations/en.DM00053084.pdf

[11]Título del recurso. *Uso de Timmers*
Autor/s. *Varios*
Fuente de información.
http://electronicatk.blogspot.com.es/2016/01/timer-stm32f4-discovery.html

[12]Título del recurso. *Uso de entrades y salidas*
Autor/s. *Varios*
Fuente de información.
http://aimagin.com/blog/how-to-use-digital-input-output/

7. Anexos

Se añade el código fuente de los programas utilizados.

Dali.h

```
/**
 ******************************************************************************
 * @file    DALI_Stack.h
 * @author  Pascual Martinez Perez
 * @version V1.0
 * @date    12-Mayo-2016
 * @brief   Header for DALI_Stack.c module
 ******************************************************************************
 *
 *
 ******************************************************************************
 */

 /* Define to prevent recursive inclusion -------------------------------------*/
#ifndef DALI_STACK
#define DALI_STACK

 /* Includes ------------------------------------------------------------------*/


/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void Inicio_Dali(void);
void Inicio_Master_Dali(void);
void TX_Datos_Dali(void);
void RX_Datos_Dali(void);
void enviaDato(unsigned char posicion,unsigned char dato);
void busReposo(void);



/**
// Initialize
extern void dali_Init(void);
// Sends the DALI address and command on the dali port
extern void dali_SendData(void);
// Interrupt handler for incoming data on the dali port
extern @interrupt void dali_Start(void);
// Interrupt handler for timebase module
extern @interrupt void dali_Tick(void);
 **/
#endif /* DALI_STACK */
```

Dali_comandos.h

```
/**
  ******************************************************************************
  * @file    DALI_Comandos.h
  * @author  Pascual mMrtinez Perez
  * @version V1.0
  * @date    12-Mayo-2016
  * @brief
  ******************************************************************************
  * Comandos DALI
  *
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef DALI_COMANDOS
#define DALI_COMANDOS

/* Includes ------------------------------------------------------------------*/
/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/

//comandos de control lampara
#define OFF 0x00
#define UP 0x01
#define DOWN 0x02
#define STEP_UP 0x03
#define STEP_DOWN 0x04
#define RECALL_MAX_LEVEL 0x05
#define RECALL_MIN_LEVEL 0x06
#define STEP_DOWN_AND_OFF 0x07
#define ON_AND_STEP_UP 0x08
#define GO_TO_SCENE 0x10
// Comandos generales
#define RESET 0x20
#define STORE_ACTUAL_LEVEL_IN_THE_DTR 0x21
// Comandos parametros
#define STORE_THE_DTR_AS_MAX_LEVEL 0x2A
#define STORE_THE_DTR_AS_MIN_LEVEL 0x2B
#define STORE_THE_DTR_AS_SYSTEM_FAILURE_LEVEL 0x2C
#define STORE_THE_DTR_AS_POWER_ON_LEVEL 0x2D
#define STORE_THE_DTR_AS_FADE_TIME 0x2E
#define STORE_THE_DTR_AS_FADE_RATE 0x2F
#define STORE_THE_DTR_AS_SCENE 0x40
// Comandos parametros sistema
#define REMOVE_FROM_SCENE 0x50
#define ADD_TO_GROUP 0x60
#define REMOVE_FROM_GROUP 0x70
#define STORE_DTR_AS_SHORT_ADDRESS 0x80
// Comandos de consultas
#define QUERY_STATUS 0x90
#define QUERY_BALLAST 0x91
#define QUERY_LAMP_FAILURE 0x92
#define QUERY_LAMP_POWER_ON 0x93
#define QUERY_LIMIT_ERROR 0x94
#define QUERY_RESET_STATE 0x95
#define QUERY_MISSING_SHORT_ADDRESS 0x96
#define QUERY_VERSION_NUMBER 0x97
#define QUERY_CONTENT_DTR 0x98
#define QUERY_DEVICE_TYPE 0x99
#define QUERY_PHYSICAL_MINIMUM_LEVEL 0x9A
#define QUERY_POWER_FAILURE 0x9B
// comandos consultas Lamparas parametros
#define QUERY_ACTUAL_LEVEL 0xA0
#define QUERY_MAX_LEVEL 0xA1
#define QUERY_MIN_LEVEL 0xA2
#define QUERY_POWER_ON_LEVEL 0xA3
#define QUERY_SYSTEM_FAILURE_LEVEL 0xA4
#define QUERY_FADE 0xA5
```

```
// Comandos consulta sistema parametros
#define QUERY_SCENE_LEVEL 0xB0
#define QUERY_GROUPS_0_7 0xC0
#define QUERY_GROUPS_8_15 0xC1
#define QUERY_RANDOM_ADDRESS_H 0xC2
#define QUERY_RANDOM_ADDRESS_M 0xC3
#define QUERY_RANDOM_ADDRESS_L 0xC4
// Comando de fin proceso
#define TERMINATE 0xA1
// Descarga informacion
#define DATA_TRANSFER_REGISTER 0xA3
// Comandos de direccionamiento
#define INITIALISE 0xA5
#define RANDOMISE 0xA7
#define COMPARE 0xA9
#define WITHDRAW 0xAB
#define SEARCHADDRH 0xB1
#define SEARCHADDRM 0xB3
#define SEARCHADDRL 0xB5
#define PROGRAM_SHORT_ADDRESS 0xB7
#define VERIFY_SHORT_ADDRESS 0xB9
#define QUERY_SHORT_ADDRESS 0xBB
#define PHYSICAL_SELECTION 0xBD

#define DIRECT_ARC_POWER 0xFE

/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */

#endif /*  DALI_COMANDOS*/
```

DALI_CONFIG.h

```
/**
  ******************************************************************************
  * @file    DALI_Config.h
  * @author  Pascual Martinez Perez
  * @version V1.0
  * @date    12-Mayo-2016
  * @brief   Header for DALI_Config.c module
  ******************************************************************************
  *
  *
  ******************************************************************************
  */
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef DALI_Config
#define DALI_Config

/* Includes ------------------------------------------------------------------*/
#include "stm32f4_discovery.h"
#include "stm32f4xx.h"

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* --- HARDWARE definitions --- */
/* IO pins for DALI in and DALI out signals */
#define OUT_DALI_PORT      GPIOD //PD1 = TX
#define OUT_DALI_PIN       GPIO_PIN_1
//#define INVERT_OUT_DALI    0

#define IN_DALI_PORT       GPIOD //PD3 = RX
#define IN_DALI_PIN        GPIO_PIN_3
//#define INVERT_IN_DALI     0

/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */


#endif /* DALI_Config */
```

DatosComunes.h

```
/**
 ******************************************************************************
 * @file    DatosComunes.h
 * @author  Pascual Martinez Perez
 * @version V1.0
 * @date    12-Mayo-2016
 * @brief
 ******************************************************************************
 *
 * Contiene definiciones comunes
 *
 *
 ******************************************************************************
 */
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef DATOS_COMUNES
#define DATOS_COMUNES

/* Includes ------------------------------------------------------------------*/

/* Exported types ------------------------------------------------------------*/

/* Exported constants --------------------------------------------------------*/
#define TRUE 1
#define FALSE 0

// Global definition
extern unsigned char direccion; // direccion DALI
extern unsigned char comando; // comando DALI
extern unsigned char respuesta; // respuesta DALI
extern unsigned char error; // codigo error
extern unsigned int flag; // Nos indica los eventos

#define NEW_DATA 0x01 // Nuevos datos
#define SEND_DATA 0x02 // Para el envio de datos
#define ERROR_EVENT 0x04 // Si hay error
#define ANSWER_EVENT 0x08 // Respuesta de comandos

/* Exported macro ------------------------------------------------------------*/
#define enableInt() _asm("cli") // Enable interrupt
#define disableInt() _asm("sei") // Disable interrupt

/* Exported functions ------------------------------------------------------- */

#endif /* DATOS_COMUNES */
```

InicioPlaca.h

```
/**
  ******************************************************************************
  * @file    IncioPlaca.h
  * @author  Pascual Martinez Perez
  * @version V1.0
  * @date    12-Mayo-2016
  * @brief   Header for InicioPlaca.c module
  ******************************************************************************
  *
  *
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef INICIO_PLACA
#define INICIO_PLACA

/* Includes ------------------------------------------------------------------*/
#include "stm32f4_discovery.h"

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void Inicio_Placa(void);
static void SystemClock_Config(void);
void Error_Handler(void);


#endif /* INICIO_PLACA */
```

Main.h

```
/**
  ******************************************************************************
  * @file   main.h
  * @author  Pascualn mrtinez Perez
  * @version V1.0
  * @date    12-Mayo-2016
  * @brief   Header for main.c module
  ******************************************************************************
  *
  *
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

/* Includes ------------------------------------------------------------------*/
#include "DatosComunes.h"
#include "InicioPlaca.h"
#include "UtilidadesPlaca.h"
#include "DALI_Master.h"


/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */


#endif /* __MAIN_H */
```

stm32f4xx_hal_conf.h

```
/**
  ******************************************************************************
  * @file    stm32f4xx_hal_conf.h
  * @author  MCD Application Team
  * @version V1.2.3
  * @date    29-January-2016
  * @brief   HAL configuration file
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT(c) 2016 STMicroelectronics</center></h2>
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __STM32F4xx_HAL_CONF_H
#define __STM32F4xx_HAL_CONF_H

#ifdef __cplusplus
 extern "C" {
#endif

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/

/* ########################## Module Selection ############################## */
/**
  * @brief This is the list of modules to be used in the HAL driver
  */
#define HAL_MODULE_ENABLED
#define HAL_ADC_MODULE_ENABLED
#define HAL_CAN_MODULE_ENABLED
#define HAL_CRC_MODULE_ENABLED
#define HAL_CRYP_MODULE_ENABLED
#define HAL_DAC_MODULE_ENABLED
#define HAL_DCMI_MODULE_ENABLED
#define HAL_DMA_MODULE_ENABLED
/* #define HAL_DMA2D_MODULE_ENABLED */
#define HAL_ETH_MODULE_ENABLED
#define HAL_FLASH_MODULE_ENABLED
#define HAL_NAND_MODULE_ENABLED
#define HAL_NOR_MODULE_ENABLED
#define HAL_PCCARD_MODULE_ENABLED
#define HAL_SRAM_MODULE_ENABLED
/* #define HAL_SDRAM_MODULE_ENABLED */
```

```
#define HAL_HASH_MODULE_ENABLED
#define HAL_GPIO_MODULE_ENABLED
#define HAL_I2C_MODULE_ENABLED
#define HAL_I2S_MODULE_ENABLED
#define HAL_IWDG_MODULE_ENABLED
#define HAL_LTDC_MODULE_ENABLED
#define HAL_PWR_MODULE_ENABLED
#define HAL_RCC_MODULE_ENABLED
#define HAL_RNG_MODULE_ENABLED
#define HAL_RTC_MODULE_ENABLED
/* #define HAL_SAI_MODULE_ENABLED */
#define HAL_SD_MODULE_ENABLED
#define HAL_SPI_MODULE_ENABLED
#define HAL_TIM_MODULE_ENABLED
#define HAL_UART_MODULE_ENABLED
#define HAL_USART_MODULE_ENABLED
#define HAL_IRDA_MODULE_ENABLED
#define HAL_SMARTCARD_MODULE_ENABLED
#define HAL_WWDG_MODULE_ENABLED
#define HAL_CORTEX_MODULE_ENABLED
#define HAL_PCD_MODULE_ENABLED
#define HAL_HCD_MODULE_ENABLED


/* ########################## HSE/HSI Values adaptation ###################### */
/**
  * @brief Adjust the value of External High Speed oscillator (HSE) used in your application.
  *       This value is used by the RCC HAL module to compute the system frequency
  *       (when HSE is used as system clock source, directly or through the PLL).
  */
#if !defined  (HSE_VALUE)
  #define HSE_VALUE    ((uint32_t)8000000) /*!< Value of the External oscillator in Hz */
#endif /* HSE_VALUE */

#if !defined  (HSE_STARTUP_TIMEOUT)
  #define HSE_STARTUP_TIMEOUT    ((uint32_t)100)   /*!< Time out for HSE start up, in ms */
#endif /* HSE_STARTUP_TIMEOUT */

/**
  * @brief Internal High Speed oscillator (HSI) value.
  *       This value is used by the RCC HAL module to compute the system frequency
  *       (when HSI is used as system clock source, directly or through the PLL).
  */
#if !defined  (HSI_VALUE)
  #define HSI_VALUE    ((uint32_t)16000000) /*!< Value of the Internal oscillator in Hz*/
#endif /* HSI_VALUE */

/**
  * @brief Internal Low Speed oscillator (LSI) value.
  */
#if !defined  (LSI_VALUE)
 #define LSI_VALUE  ((uint32_t)32000)
#endif /* LSI_VALUE */                 /*!< Value of the Internal Low Speed oscillator in Hz
                                         The real value may vary depending on the variations
                                         in voltage and temperature.  */
/**
  * @brief External Low Speed oscillator (LSE) value.
  */
#if !defined  (LSE_VALUE)
 #define LSE_VALUE  ((uint32_t)32768)    /*!< Value of the External Low Speed oscillator in Hz */
#endif /* LSE_VALUE */

#if !defined  (LSE_STARTUP_TIMEOUT)
  #define LSE_STARTUP_TIMEOUT    ((uint32_t)5000)   /*!< Time out for LSE start up, in ms */
#endif /* LSE_STARTUP_TIMEOUT */

/**
  * @brief External clock source for I2S peripheral
  *       This value is used by the I2S HAL module to compute the I2S clock source
  *       frequency, this source is inserted directly through I2S_CKIN pad.
```

```
 */
#if !defined  (EXTERNAL_CLOCK_VALUE)
 #define EXTERNAL_CLOCK_VALUE    ((uint32_t)12288000) /*!< Value of the Internal oscillator in Hz*/
#endif /* EXTERNAL_CLOCK_VALUE */

/* Tip: To avoid modifying this file each time you need to use different HSE,
  === you can define the HSE value in your toolchain compiler preprocessor. */

/* ########################### System Configuration ######################### */
/**
 * @brief This is the HAL system configuration section
 */
#define  VDD_VALUE                ((uint32_t)3300) /*!< Value of VDD in mv */
#define  TICK_INT_PRIORITY          ((uint32_t)0x0F) /*!< tick interrupt priority */
#define  USE_RTOS                 0
#define  PREFETCH_ENABLE           0 /* The prefetch will be enabled in SystemClock_Config(), depending on the used
                                 STM32F405/415/07/417 device: RevA (prefetch must be off) or RevZ (prefetch can be on/off) */
#define  INSTRUCTION_CACHE_ENABLE    1
#define  DATA_CACHE_ENABLE          1

/* ########################### Assert Selection ############################## */
/**
 * @brief Uncomment the line below to expanse the "assert_param" macro in the
 *       HAL drivers code
 */
/* #define USE_FULL_ASSERT    1 */

/* ################### Ethernet peripheral configuration ##################### */

/* Section 1 : Ethernet peripheral configuration */

/* MAC ADDRESS: MAC_ADDR0:MAC_ADDR1:MAC_ADDR2:MAC_ADDR3:MAC_ADDR4:MAC_ADDR5 */
#define MAC_ADDR0   2
#define MAC_ADDR1   0
#define MAC_ADDR2   0
#define MAC_ADDR3   0
#define MAC_ADDR4   0
#define MAC_ADDR5   0

/* Definition of the Ethernet driver buffers size and count */
#define ETH_RX_BUF_SIZE             ETH_MAX_PACKET_SIZE /* buffer size for receive            */
#define ETH_TX_BUF_SIZE             ETH_MAX_PACKET_SIZE /* buffer size for transmit           */
#define ETH_RXBUFNB                 ((uint32_t)4)      /* 4 Rx buffers of size ETH_RX_BUF_SIZE  */
#define ETH_TXBUFNB                 ((uint32_t)4)      /* 4 Tx buffers of size ETH_TX_BUF_SIZE  */

/* Section 2: PHY configuration section */

/* DP83848 PHY Address*/
#define DP83848_PHY_ADDRESS           0x01
/* PHY Reset delay these values are based on a 1 ms Systick interrupt*/
#define PHY_RESET_DELAY               ((uint32_t)0x000000FF)
/* PHY Configuration delay */
#define PHY_CONFIG_DELAY              ((uint32_t)0x00000FFF)

#define PHY_READ_TO                   ((uint32_t)0x0000FFFF)
#define PHY_WRITE_TO                  ((uint32_t)0x0000FFFF)

/* Section 3: Common PHY Registers */

#define PHY_BCR                     ((uint16_t)0x00)    /*!< Transceiver Basic Control Register   */
#define PHY_BSR                     ((uint16_t)0x01)    /*!< Transceiver Basic Status Register    */

#define PHY_RESET                    ((uint16_t)0x8000)  /*!< PHY Reset */
#define PHY_LOOPBACK                 ((uint16_t)0x4000)  /*!< Select loop-back mode */
#define PHY_FULLDUPLEX_100M           ((uint16_t)0x2100)  /*!< Set the full-duplex mode at 100 Mb/s */
#define PHY_HALFDUPLEX_100M           ((uint16_t)0x2000)  /*!< Set the half-duplex mode at 100 Mb/s */
#define PHY_FULLDUPLEX_10M            ((uint16_t)0x0100)  /*!< Set the full-duplex mode at 10 Mb/s  */
#define PHY_HALFDUPLEX_10M            ((uint16_t)0x0000)  /*!< Set the half-duplex mode at 10 Mb/s  */
#define PHY_AUTONEGOTIATION           ((uint16_t)0x1000)  /*!< Enable auto-negotiation function     */
#define PHY_RESTART_AUTONEGOTIATION     ((uint16_t)0x0200)  /*!< Restart auto-negotiation function    */
```

```
#define PHY_POWERDOWN              ((uint16_t)0x0800)  /*!< Select the power down mode        */
#define PHY_ISOLATE                ((uint16_t)0x0400)  /*!< Isolate PHY from MII              */

#define PHY_AUTONEGO_COMPLETE      ((uint16_t)0x0020)  /*!< Auto-Negotiation process completed  */
#define PHY_LINKED_STATUS          ((uint16_t)0x0004)  /*!< Valid link established              */
#define PHY_JABBER_DETECTION       ((uint16_t)0x0002)  /*!< Jabber condition detected           */

/* Section 4: Extended PHY Registers */

#define PHY_SR                     ((uint16_t)0x10)    /*!< PHY status register Offset                     */
#define PHY_MICR                   ((uint16_t)0x11)    /*!< MII Interrupt Control Register                 */
#define PHY_MISR                   ((uint16_t)0x12)    /*!< MII Interrupt Status and Misc. Control Register */

#define PHY_LINK_STATUS            ((uint16_t)0x0001)  /*!< PHY Link mask                                  */
#define PHY_SPEED_STATUS           ((uint16_t)0x0002)  /*!< PHY Speed mask                                 */
#define PHY_DUPLEX_STATUS          ((uint16_t)0x0004)  /*!< PHY Duplex mask                                */

#define PHY_MICR_INT_EN            ((uint16_t)0x0002)  /*!< PHY Enable interrupts                          */
#define PHY_MICR_INT_OE            ((uint16_t)0x0001)  /*!< PHY Enable output interrupt events             */

#define PHY_MISR_LINK_INT_EN       ((uint16_t)0x0020)  /*!< Enable Interrupt on change of link status      */
#define PHY_LINK_INTERRUPT         ((uint16_t)0x2000)  /*!< PHY link status interrupt mask                 */

/* Includes ------------------------------------------------------------------*/
/**
  * @brief Include module's header file
  */

#ifdef HAL_RCC_MODULE_ENABLED
  #include "stm32f4xx_hal_rcc.h"
#endif /* HAL_RCC_MODULE_ENABLED */

#ifdef HAL_GPIO_MODULE_ENABLED
  #include "stm32f4xx_hal_gpio.h"
#endif /* HAL_GPIO_MODULE_ENABLED */

#ifdef HAL_DMA_MODULE_ENABLED
  #include "stm32f4xx_hal_dma.h"
#endif /* HAL_DMA_MODULE_ENABLED */

#ifdef HAL_CORTEX_MODULE_ENABLED
  #include "stm32f4xx_hal_cortex.h"
#endif /* HAL_CORTEX_MODULE_ENABLED */

#ifdef HAL_ADC_MODULE_ENABLED
  #include "stm32f4xx_hal_adc.h"
#endif /* HAL_ADC_MODULE_ENABLED */

#ifdef HAL_CAN_MODULE_ENABLED
  #include "stm32f4xx_hal_can.h"
#endif /* HAL_CAN_MODULE_ENABLED */

#ifdef HAL_CRC_MODULE_ENABLED
  #include "stm32f4xx_hal_crc.h"
#endif /* HAL_CRC_MODULE_ENABLED */

#ifdef HAL_CRYP_MODULE_ENABLED
  #include "stm32f4xx_hal_cryp.h"
#endif /* HAL_CRYP_MODULE_ENABLED */

#ifdef HAL_DMA2D_MODULE_ENABLED
  #include "stm32f4xx_hal_dma2d.h"
#endif /* HAL_DMA2D_MODULE_ENABLED */

#ifdef HAL_DAC_MODULE_ENABLED
  #include "stm32f4xx_hal_dac.h"
#endif /* HAL_DAC_MODULE_ENABLED */

#ifdef HAL_DCMI_MODULE_ENABLED
  #include "stm32f4xx_hal_dcmi.h"
```

```
#endif /* HAL_DCMI_MODULE_ENABLED */

#ifdef HAL_ETH_MODULE_ENABLED
  #include "stm32f4xx_hal_eth.h"
#endif /* HAL_ETH_MODULE_ENABLED */

#ifdef HAL_FLASH_MODULE_ENABLED
  #include "stm32f4xx_hal_flash.h"
#endif /* HAL_FLASH_MODULE_ENABLED */

#ifdef HAL_SRAM_MODULE_ENABLED
  #include "stm32f4xx_hal_sram.h"
#endif /* HAL_SRAM_MODULE_ENABLED */

#ifdef HAL_NOR_MODULE_ENABLED
  #include "stm32f4xx_hal_nor.h"
#endif /* HAL_NOR_MODULE_ENABLED */

#ifdef HAL_NAND_MODULE_ENABLED
  #include "stm32f4xx_hal_nand.h"
#endif /* HAL_NAND_MODULE_ENABLED */

#ifdef HAL_PCCARD_MODULE_ENABLED
  #include "stm32f4xx_hal_pccard.h"
#endif /* HAL_PCCARD_MODULE_ENABLED */

#ifdef HAL_SDRAM_MODULE_ENABLED
  #include "stm32f4xx_hal_sdram.h"
#endif /* HAL_SDRAM_MODULE_ENABLED */

#ifdef HAL_HASH_MODULE_ENABLED
 #include "stm32f4xx_hal_hash.h"
#endif /* HAL_HASH_MODULE_ENABLED */

#ifdef HAL_I2C_MODULE_ENABLED
 #include "stm32f4xx_hal_i2c.h"
#endif /* HAL_I2C_MODULE_ENABLED */

#ifdef HAL_I2S_MODULE_ENABLED
 #include "stm32f4xx_hal_i2s.h"
#endif /* HAL_I2S_MODULE_ENABLED */

#ifdef HAL_IWDG_MODULE_ENABLED
 #include "stm32f4xx_hal_iwdg.h"
#endif /* HAL_IWDG_MODULE_ENABLED */

#ifdef HAL_LTDC_MODULE_ENABLED
 #include "stm32f4xx_hal_ltdc.h"
#endif /* HAL_LTDC_MODULE_ENABLED */

#ifdef HAL_PWR_MODULE_ENABLED
 #include "stm32f4xx_hal_pwr.h"
#endif /* HAL_PWR_MODULE_ENABLED */

#ifdef HAL_RNG_MODULE_ENABLED
 #include "stm32f4xx_hal_rng.h"
#endif /* HAL_RNG_MODULE_ENABLED */

#ifdef HAL_RTC_MODULE_ENABLED
 #include "stm32f4xx_hal_rtc.h"
#endif /* HAL_RTC_MODULE_ENABLED */

#ifdef HAL_SAI_MODULE_ENABLED
 #include "stm32f4xx_hal_sai.h"
#endif /* HAL_SAI_MODULE_ENABLED */

#ifdef HAL_SD_MODULE_ENABLED
 #include "stm32f4xx_hal_sd.h"
#endif /* HAL_SD_MODULE_ENABLED */
```

```
#ifdef HAL_SPI_MODULE_ENABLED
 #include "stm32f4xx_hal_spi.h"
#endif /* HAL_SPI_MODULE_ENABLED */

#ifdef HAL_TIM_MODULE_ENABLED
 #include "stm32f4xx_hal_tim.h"
#endif /* HAL_TIM_MODULE_ENABLED */

#ifdef HAL_UART_MODULE_ENABLED
 #include "stm32f4xx_hal_uart.h"
#endif /* HAL_UART_MODULE_ENABLED */

#ifdef HAL_USART_MODULE_ENABLED
 #include "stm32f4xx_hal_usart.h"
#endif /* HAL_USART_MODULE_ENABLED */

#ifdef HAL_IRDA_MODULE_ENABLED
 #include "stm32f4xx_hal_irda.h"
#endif /* HAL_IRDA_MODULE_ENABLED */

#ifdef HAL_SMARTCARD_MODULE_ENABLED
 #include "stm32f4xx_hal_smartcard.h"
#endif /* HAL_SMARTCARD_MODULE_ENABLED */

#ifdef HAL_WWDG_MODULE_ENABLED
 #include "stm32f4xx_hal_wwdg.h"
#endif /* HAL_WWDG_MODULE_ENABLED */

#ifdef HAL_PCD_MODULE_ENABLED
 #include "stm32f4xx_hal_pcd.h"
#endif /* HAL_PCD_MODULE_ENABLED */

#ifdef HAL_HCD_MODULE_ENABLED
 #include "stm32f4xx_hal_hcd.h"
#endif /* HAL_HCD_MODULE_ENABLED */

/* Exported macro ------------------------------------------------------------*/
#ifdef  USE_FULL_ASSERT
/**
  * @brief  The assert_param macro is used for function's parameters check.
  * @param  expr: If expr is false, it calls assert_failed function
  *         which reports the name of the source file and the source
  *         line number of the call that failed.
  *         If expr is true, it returns no value.
  * @retval None
  */
  #define assert_param(expr) ((expr) ? (void)0 : assert_failed((uint8_t *)__FILE__, __LINE__))
/* Exported functions ------------------------------------------------------- */
  void assert_failed(uint8_t* file, uint32_t line);
#else
  #define assert_param(expr) ((void)0)
#endif /* USE_FULL_ASSERT */



#ifdef __cplusplus
}
#endif

#endif /* __STM32F4xx_HAL_CONF_H */


/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

stm32f4xx_it.h

```
/**
  ******************************************************************************
  * @file    Templates/Inc/stm32f4xx_it.h
  * @author  MCD Application Team
  * @version V1.2.3
  * @date    29-January-2016
  * @brief   This file contains the headers of the interrupt handlers.
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT(c) 2016 STMicroelectronics</center></h2>
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __STM32F4xx_IT_H
#define __STM32F4xx_IT_H
#ifdef __cplusplus
 extern "C" {
#endif
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4_discovery.h"

  /* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */

void NMI_Handler(void);
void HardFault_Handler(void);
void MemManage_Handler(void);
void BusFault_Handler(void);
void UsageFault_Handler(void);
void SVC_Handler(void);
void DebugMon_Handler(void);
void PendSV_Handler(void);
void SysTick_Handler(void);
void EXTI0_IRQHandler(void);
void EXTI3_IRQHandler(void);

#ifdef __cplusplus
}
#endif

#endif /* __STM32F4xx_IT_H */
```

UtilidadesPlaca.h

```
/**
 ******************************************************************************
 * @file    UtilidadesPlaca.h
 * @author  Pascual Martinez Perez
 * @version V1.0
 * @date    12-Mayo-2016
 * @brief   Header for UtilidadesPlaca.c module
 ******************************************************************************
 *
 *
 ******************************************************************************
 */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef UTILIDADES_PLACA
#define UTILIDADES_PLACA

/* Includes ------------------------------------------------------------------*/
/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void leds_Saludo(void);
void leds_OFF(void);
void leds_ON(void);
void led_Naranja_ON(void);
void led_Naranja_OFF(void);
void led_Azul_ON(void);
void led_Azul_OFF(void);
void led_Rojo_ON(void);
void led_Rojo_OFF(void);
void led_Verde_ON(void);
void led_Verde_OFF(void);


#endif /* UTILIDADES_PLACA */
```

DALI.c

```
/**
  ******************************************************************************
  * @file    DALI_Stack.c
  * @author  Pascual Martinez Perez
  * @version V1.0
  * @date    12-Mayo-2016
  * @brief
  ******************************************************************************
  *
  * Implementacion de la capa Stack de DALI
  *
  ******************************************************************************
  */


/* Includes ------------------------------------------------------------------*/

#include "DALI.h"
#include "DALI_Config.h"
#include "stm32f4_discovery.h"
#include "stm32f4xx.h"
#include "UtilidadesPlaca.h"
#include "DatosComunes.h"
#include "stm32f4xx_it.h"
#include "DALI_Comandos.h"




/* Private typedef -----------------------------------------------------------*/

/* Private define ------------------------------------------------------------*/


/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
static unsigned char send_position; // Posicion de los datos a trensferir
static unsigned char send_active; // True si TX activo
static unsigned char send_value; // EL valor del bit a transmitir

static unsigned char rec_position; // Posicion de recepcion de bit
static unsigned char rec_active; // True si la recepcion ha comenzado
static uint32_t rec_value; // mantiene el bit recibido
static unsigned char rec_bit; // numero de bits recibidos
static unsigned char rec_start;  //inicio recepcion;
static unsigned char temp_value;  //valor temporal a transmitir

static unsigned char contadorComandos;  //contador de comandos para pruebas
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

void Inicio_Dali()
{
  contadorComandos=0;
  send_active=FALSE;
  rec_active=FALSE;

  /* Configure USER Button Interrupcion PA0*/
  BSP_PB_Init(BUTTON_KEY, BUTTON_MODE_EXTI);


  //Habilito clks del puerto a usar
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();


   GPIO_InitTypeDef GPIO_InitStructure;
  //Pin out DALI  PD1
```

```
  /* Pull-up Vdd pin for data output */
  GPIO_InitStructure.Pin =  OUT_DALI_PIN ;
  GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(OUT_DALI_PORT, &GPIO_InitStructure);

   //Mantener el bus a 1  OBLIGATORIO
  //HAL_GPIO_WritePin( OUT_DALI_PORT, OUT_DALI_PIN, GPIO_PIN_SET);
  busReposo();

  //PIN IN DALI PD3
  GPIO_InitStructure.Pin =  IN_DALI_PIN;
  GPIO_InitStructure.Mode = GPIO_MODE_IT_FALLING;//GPIO_MODE_IT_RISING_FALLING;//  //Flanco bajada
  //GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_HIGH;
  GPIO_InitStructure.Pull= GPIO_NOPULL;
  HAL_GPIO_Init(IN_DALI_PORT, &GPIO_InitStructure);


  /* Enable and set Button EXTI Interrupt to the lowest priority */

  HAL_NVIC_SetPriority(EXTI0_IRQn, 3, 0);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);

  HAL_NVIC_SetPriority(EXTI3_IRQn, 3, 0);
  HAL_NVIC_DisableIRQ(EXTI3_IRQn);


}//fin inicio DALI


void TX_Datos_Dali(void)
{

    //Comandos a eviar
    //broadcast OFF
    //direccion=DIRECT_ARC_POWER;
    //comando=0xDC;  //Mitad
    //comando=OFF;  //Apagado
    //comando=0xFE;  //TOPE
    //comando=0xAB;

    direccion=0xFF;
    comando=QUERY_ACTUAL_LEVEL;

    //inicializa variables
    send_position=0;
    send_value=0;
    send_active=TRUE;
    /*if (contadorComandos % 2==0 )

      comando=STEP_DOWN;
    else
      comando=STEP_DOWN;
    contadorComandos++;*/


}

void RX_Datos_Dali(void)
{

  /*rec_position=0;
  rec_active=TRUE;
  rec_bit=0;
  rec_value=0x00;
  respuesta=0x00;*/


  rec_start=TRUE;
  //rec_int=0;
```

```
}



/**
  * @brief  EXTI line detection callbacks.
  * @param  GPIO_Pin: Specifies the pins connected EXTI line
  * @retval None
  */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{

  //Envio Comando
  if (KEY_BUTTON_PIN == GPIO_Pin)
  {

    //Si es el pin del pulsador
    flag |=SEND_DATA;
}
//Recibo comando
if (IN_DALI_PIN == GPIO_Pin)
{

     flag |=ANSWER_EVENT;

}


}// Fin HAL_GPIO_EXTI_Callback

/**
  * @brief  This function handles SysTick Handler.
  * @param  None
  * @retval None
  */
void SysTick_Handler(void)
{

  //Escuchar respuesta


  if (rec_active==TRUE)
  {
    //Time out lo marca la norma cuanto hay que escuchar
    if( rec_start==FALSE && rec_position>=44)
    {
       //Para salir
       rec_active=FALSE;
       rec_start=FALSE;
         rec_position=0;

          HAL_NVIC_DisableIRQ(EXTI3_IRQn);
         //habilito la interrupcion del pulsador
         HAL_NVIC_EnableIRQ(EXTI0_IRQn);
         //deshabilito la interupcion de escuchar
         //no hay nada que escuchar

      //Indicar no hay respuesta
      //el numero de bits recibidos es cero

    }//Fin rec_start  FALSE


     else if (rec_start==TRUE)
    {
      //leo puerto cada tick en las muestras necesarias calculadas

      switch (rec_position)
      {
```

```
    //Son posiciones que corresponden con cada tick de respuesta
     case 31:
     case 35:
     case 39:
     case 43:
     case 47:
     case 51:
     case 55:
     case 59:

     {
       //leo el puerto de entrada segun muestra
       //Parpadeo del bit azul veo qeu leo y recibo algo
       rec_bit=HAL_GPIO_ReadPin( IN_DALI_PORT, IN_DALI_PIN);
         if (rec_bit==0)
            led_Azul_OFF();
         if (rec_bit==1)
           led_Azul_ON();
         //Acumulo los bits recibidos
         rec_value = (rec_value << 1) |rec_bit;

       break;
     }
    case 68:
     {
       //fin de recepcion
       led_Azul_OFF();
       rec_start=FALSE;
       rec_active=FALSE;
       rec_position=0;
       HAL_NVIC_DisableIRQ(EXTI3_IRQn);
       HAL_NVIC_EnableIRQ(EXTI0_IRQn);
       break;
     }
    }//Fin switch

    }//fin rec_start
    rec_position++;

}//Fin rec_active


//EVIAR COMANDO
if (send_active==TRUE)
{

    //Esta el bus a 1

    //bit inicio
    if (send_position<=3)
    {
      enviaDato(send_position,1);
    }
    //direccion y dato
    else if (send_position>=4 && send_position<=67)
    {

      if(send_position<=35)
      {
        //direccion 4 35
        temp_value = (direccion>>((35-send_position)/4)) & 0x01;//35 pos maxima 4 muetras por bit

      }
      else
      {
        //comando 36 67
        temp_value = (comando>>((67-send_position)/4)) & 0x01;//67 pos maxima 4 muetras por bit
      }

      enviaDato(send_position,temp_value);
```

```
    }
    //bit 1 final
    //else if (send_position<=68 && send_position<=71)
    else if (send_position==68)
    {
       busReposo();
    }
    //bit 1 final
    /*else if (send_position<=72 && send_position<=75)
    {
       busReposo();
    }*/
    //Salir
    else if (send_position>=76)//76
    {

      //Habilitamos escucha por interrupcion e inicializamos
      //Fin comando
      send_active = FALSE;

       //Inicializo escucha
       rec_position=0;
      rec_bit=0;
      rec_value=0x00;
      temp_value=0x00;
      respuesta=0x00;

      //Activo int ext 3 y activo la escucha
       //Me pongo a escuchar
      rec_active=TRUE;
      rec_start=FALSE;

      HAL_NVIC_EnableIRQ(EXTI3_IRQn);
      HAL_NVIC_DisableIRQ(EXTI0_IRQn);
      //HAL_NVIC_ClearPendingIRQ(EXTI3_IRQn);
    }


   send_position++;
  }//Fin envio datos send_active=TRUE

}//Fin systick  ***

/* Esta rutina cambia cambia de estado el dato de salida
* codifica el manchester en funcion de en que posicion estoy de la trama
* y el dato qeu tengo que enviar */
void enviaDato(unsigned char posicion,unsigned char dato)
{
 //deteccion en que parte del bit estamos
    switch (posicion % 4)
    {
       //primera mitad siempre inverso
       case 0:
       case 1:
       {
         dato=!dato;
         break;
       }


       //Segunda mitad mismo dato
       case 2:
       case 3:
       {
        dato=dato;
        break;
       }

     }//Fin switch
```

```
    if (dato==0)
    {
     //envio 0  //Bus a 5 voltios
      HAL_GPIO_WritePin( OUT_DALI_PORT, OUT_DALI_PIN, GPIO_PIN_RESET);
      led_Verde_OFF();
      //led_Rojo_OFF();
    }
    else
    {
     //envio 1//Bus a 20 voltios
     HAL_GPIO_WritePin( OUT_DALI_PORT, OUT_DALI_PIN, GPIO_PIN_SET);
     led_Verde_ON();
     //led_Verde_OFF();
    }

} //Fin envia datoa

// DEja el bus en estado alto
void busReposo()
{
      //envio 1//Bus a 20 voltios
      HAL_GPIO_WritePin( OUT_DALI_PORT, OUT_DALI_PIN, GPIO_PIN_SET);
}
```

InicioPlaca.c

```
/**
 ******************************************************************************
 * @file    IncioPlaca.c
 * @author  Pascual Martinez Perez
 * @version V1.0
 * @date    12-Mayo-2016
 * @brief
 ******************************************************************************
 *  Inicializacion de los elementos necesarios del Placa.
 *
 ******************************************************************************
 */
/* Includes ------------------------------------------------------------------*/
#include "InicioPlaca.h"


/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/


void Inicio_Placa()
{




  /* STM32F4xx HAL library initialization:
      - Configure the Flash prefetch, instruction and Data caches
      - Configure the Systick to generate an interrupt each 1 msec
      - Set NVIC Group Priority to 4
      - Global MSP (MCU Support Package) initialization
    */
  if(HAL_Init()!= HAL_OK)
  {
    /* Start Conversation Error */
    Error_Handler();
```

```
  }

  /* Configure the system clock to 168 MHz */
  SystemClock_Config();

  //Configuramos el sysTick a Te
  SysTick_Config(SystemCoreClock/4800);//2400



  //Configuro los leds
  BSP_LED_Init(LED3);
  BSP_LED_Init(LED4);
  BSP_LED_Init(LED5);
  BSP_LED_Init(LED6);


}//Fin InicioPlaca

/**
  * @brief  System Clock Configuration
  *         The system Clock is configured as follow :
  *           System Clock source          = PLL (HSE)
  *           SYSCLK(Hz)                   = 168000000
  *           HCLK(Hz)                     = 168000000
  *           AHB Prescaler                = 1
  *           APB1 Prescaler               = 4
  *           APB2 Prescaler               = 2
  *           HSE Frequency(Hz)            = 8000000
  *           PLL_M                        = 8
  *           PLL_N                        = 336
  *           PLL_P                        = 2
  *           PLL_Q                        = 7
  *           VDD(V)                       = 3.3
  *           Main regulator output voltage  = Scale1 mode
  *           Flash Latency(WS)            = 5
  * @param  None
  * @retval None
  */
static void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;

  /* Enable Power Control clock */
  __HAL_RCC_PWR_CLK_ENABLE();

  /* The voltage scaling allows optimizing the power consumption when the device is
     clocked below the maximum system frequency, to update the voltage scaling value
     regarding system frequency refer to product datasheet.  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /* Enable HSE Oscillator and activate PLL with HSE as source */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  HAL_RCC_OscConfig(&RCC_OscInitStruct);

  /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
     clocks dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_PCLK1 |
RCC_CLOCKTYPE_PCLK2);
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
```

```
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
 HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);

 /* STM32F405x/407x/415x/417x Revision Z devices: prefetch is supported  */
 if (HAL_GetREVID() == 0x1001)
 {
  /* Enable the Flash prefetch */
  __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
 }
}

/**
 * @brief  This function is executed in case of error occurrence.
 * @param  None
 * @retval None
 */
void Error_Handler(void)
{
 /* Turn LED5 on */
 //BSP_LED_Init(LED5);
 BSP_LED_On(LED5);

 while(1)
 {
 }
}//Fin Error_Handler
```

Main.c

```
/**
 ******************************************************************************
 * @file    main.c
 * @author  Pascual Martinez Perez
 * @version V1.0
 * @date    12-Mayo-2016
 * @brief
 ******************************************************************************
 *
 *
 ******************************************************************************
 */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "DALI_Master.h"
#include "DALI.h"



/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
unsigned char direccion; // direccion DALI
unsigned char comando; // comando DALI
unsigned char respuesta; // respuesta DALI
unsigned char error; // codigo error
unsigned int flag; // Nos indica los eventos

/* Private function prototypes -----------------------------------------------*/

/* Private functions ---------------------------------------------------------*/


/**
 * @brief  Main program
 * @param  None
 * @retval None
 */
```

```
int main(void)
{

  //inicio variables
  direccion = 0x00;
  comando = 0x00;
  respuesta = 0x00;
  error = 0x00;
  flag = 0x00;

  //inicio de elementos
  Inicio_Placa();
  Inicio_Dali();


  //Encendemos los leds y los apagamos.
  leds_Saludo();

  //Bucle  infinito
  while(1)
  {

    // NEW_DATA flag
    if (flag & NEW_DATA)
    {
       flag &= ~NEW_DATA; // Clear NEW_DATA flag
       //Pensado para llamada para visualizar el resultado
    }

    // SEND_DATA flag
    if (flag & SEND_DATA)
    {
     flag &= ~SEND_DATA; // Clear SEND_DATA flag
     TX_Datos_Dali();  // Envio de comando
    }

    // ANSWER_EVENT flag
    if (flag & ANSWER_EVENT)
    {
       flag &= ~ANSWER_EVENT; // Clear ANSWER_EVENT flag
       RX_Datos_Dali();  //Recepcion de la respuesta

    }
    // ERROR_EVENT flag
    if (flag & ERROR_EVENT)
    {
     flag &= ~ERROR_EVENT; // Clear ERROR_EVENT flag
                  //Control de errores
     }

  }//Fin while infinito
}//Fin main



#ifdef USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *       where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
```

```
 /* Infinite loop */
 while (1)
 {
 }
}
#endif

/**
 * @}
 */

/**
 * @}
 */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

stm32f4xx_hal_msp.c

```
/**
 ******************************************************************************
 * @file    Templates/Src/stm32f4xx_hal_msp.c
 * @author  MCD Application Team
 * @version V1.2.3
 * @date    29-January-2016
 * @brief   HAL MSP module.
 *
 @verbatim
 ==============================================================================
              ##### How to use this driver #####
 ==============================================================================
  [..]
  This file is generated automatically by STM32CubeMX and eventually modified
  by the user

 @endverbatim
 ******************************************************************************
 * @attention
 *
 * <h2><center>&copy; COPYRIGHT(c) 2016 STMicroelectronics</center></h2>
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *   1. Redistributions of source code must retain the above copyright notice,
 *      this list of conditions and the following disclaimer.
 *   2. Redistributions in binary form must reproduce the above copyright notice,
 *      this list of conditions and the following disclaimer in the documentation
 *      and/or other materials provided with the distribution.
 *   3. Neither the name of STMicroelectronics nor the names of its contributors
 *      may be used to endorse or promote products derived from this software
 *      without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
 * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 ******************************************************************************
 */

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"

/** @addtogroup STM32F4xx_HAL_Driver
```

```
  * @{
  */

/** @defgroup HAL_MSP
  * @brief HAL MSP module.
  * @{
  */

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/** @defgroup HAL_MSP_Private_Functions
  * @{
  */

/**
  * @brief  Initializes the Global MSP.
  * @param  None
  * @retval None
  */
void HAL_MspInit(void)
{
  /* NOTE : This function is generated automatically by STM32CubeMX and eventually
        modified by the user
   */
}

/**
  * @brief  DeInitializes the Global MSP.
  * @param  None
  * @retval None
  */
void HAL_MspDeInit(void)
{
  /* NOTE : This function is generated automatically by STM32CubeMX and eventually
        modified by the user
   */
}

/**
  * @}
  */

/**
  * @}
  */

/**
  * @}
  */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

stm32f4xx_it.c

```
/**
  ******************************************************************************
  * @file    Templates/Src/stm32f4xx_it.c
  * @author  MCD Application Team
  * @version V1.2.3
  * @date    29-January-2016
  * @brief   Main Interrupt Service Routines.
  *          This file provides template for all exceptions handler and
  *          peripherals interrupt service routine.
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT(c) 2016 STMicroelectronics</center></h2>
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx_it.h"
#include "DatosComunes.h"


/** @addtogroup STM32F4xx_HAL_Examples
  * @{
  */

/** @addtogroup Templates
  * @{
  */

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/******************************************************************************/
/*          Cortex-M4 Processor Exceptions Handlers                           */
/******************************************************************************/

/**
  * @brief   This function handles NMI exception.
  * @param  None
```

```
  * @retval None
  */
void NMI_Handler(void)
{
}

/**
  * @brief  This function handles Hard Fault exception.
  * @param  None
  * @retval None
  */
void HardFault_Handler(void)
{
  /* Go to infinite loop when Hard Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Memory Manage exception.
  * @param  None
  * @retval None
  */
void MemManage_Handler(void)
{
  /* Go to infinite loop when Memory Manage exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Bus Fault exception.
  * @param  None
  * @retval None
  */
void BusFault_Handler(void)
{
  /* Go to infinite loop when Bus Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Usage Fault exception.
  * @param  None
  * @retval None
  */
void UsageFault_Handler(void)
{
  /* Go to infinite loop when Usage Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles SVCall exception.
  * @param  None
  * @retval None
  */
void SVC_Handler(void)
{
}

/**
  * @brief  This function handles Debug Monitor exception.
  * @param  None
```

```
  * @retval None
  */
void DebugMon_Handler(void)
{
}

/**
  * @brief  This function handles PendSVC exception.
  * @param  None
  * @retval None
  */
void PendSV_Handler(void)
{
}


/**
  * @brief  This function handles SysTick Handler.
  * @param  None
  * @retval None
  */
/*void SysTick_Handler(void)
{
  HAL_IncTick();

}*/



/******************************************************************************/
/*            STM32F4xx Peripherals Interrupt Handlers                        */
/*  Add here the Interrupt Handler for the used peripheral(s) (PPP), for the  */
/*  available peripheral interrupt handler's name please refer to the startup */
/*  file (startup_stm32f4xx.s).                                               */
/******************************************************************************/

/**
  * @brief  This function handles PPP interrupt request.
  * @param  None
  * @retval None
  */
/*void PPP_IRQHandler(void)
{
}*/


/**
  * @}
  */

/**
  * @}
  */


/**
  * @brief  This function handles External line 0 interrupt request.
  * @param  None
  * @retval None
  */
void EXTI0_IRQHandler(void)
{
  HAL_NVIC_DisableIRQ(EXTI0_IRQn);
  HAL_NVIC_DisableIRQ(EXTI3_IRQn);
  HAL_GPIO_EXTI_IRQHandler(KEY_BUTTON_PIN);

}

void EXTI3_IRQHandler(void)
{
```

```
 HAL_NVIC_DisableIRQ(EXTI0_IRQn);
 HAL_NVIC_DisableIRQ(EXTI3_IRQn);
 HAL_GPIO_EXTI_IRQHandler(IN_DALI_PIN);
}


/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

system_stm32f4xx.c

```
/**
  ******************************************************************************
  * @file    system_stm32f4xx.c
  * @author  MCD Application Team
  * @version V1.2.3
  * @date    29-January-2016
  * @brief   CMSIS Cortex-M4 Device Peripheral Access Layer System Source File.
  *
  *   This file provides two functions and one global variable to be called from
  *   user application:
  *      - SystemInit(): This function is called at startup just after reset and
  *                      before branch to main program. This call is made inside
  *                      the "startup_stm32f4xx.s" file.
  *
  *      - SystemCoreClock variable: Contains the core clock (HCLK), it can be used
  *                                  by the user application to setup the SysTick
  *                                  timer or configure other parameters.
  *
  *      - SystemCoreClockUpdate(): Updates the variable SystemCoreClock and must
  *                                 be called whenever the core clock is changed
  *                                 during program execution.
  *
  *
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; COPYRIGHT 2016 STMicroelectronics</center></h2>
  *
  * Redistribution and use in source and binary forms, with or without modification,
  * are permitted provided that the following conditions are met:
  *   1. Redistributions of source code must retain the above copyright notice,
  *      this list of conditions and the following disclaimer.
  *   2. Redistributions in binary form must reproduce the above copyright notice,
  *      this list of conditions and the following disclaimer in the documentation
  *      and/or other materials provided with the distribution.
  *   3. Neither the name of STMicroelectronics nor the names of its contributors
  *      may be used to endorse or promote products derived from this software
  *      without specific prior written permission.
  *
  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
  *
  ******************************************************************************
  */

/** @addtogroup CMSIS
  * @{
  */
```

```
/** @addtogroup stm32f4xx_system
  * @{
  */

/** @addtogroup STM32F4xx_System_Private_Includes
  * @{
  */

#include "stm32f4xx.h"

#if !defined  (HSE_VALUE)
  #define HSE_VALUE    ((uint32_t)8000000) /*!< Default value of the External oscillator in Hz */
#endif /* HSE_VALUE */

#if !defined  (HSI_VALUE)
  #define HSI_VALUE    ((uint32_t)16000000) /*!< Value of the Internal oscillator in Hz*/
#endif /* HSI_VALUE */

/**
  * @}
  */

/** @addtogroup STM32F4xx_System_Private_TypesDefinitions
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32F4xx_System_Private_Defines
  * @{
  */

/************************* Miscellaneous Configuration ************************/

/*!< Uncomment the following line if you need to relocate your vector Table in
     Internal SRAM. */
/* #define VECT_TAB_SRAM */
#define VECT_TAB_OFFSET  0x00 /*!< Vector Table base offset field.
                                  This value must be a multiple of 0x200. */
/******************************************************************************/

/**
  * @}
  */

/** @addtogroup STM32F4xx_System_Private_Macros
  * @{
  */

/**
  * @}
  */

/** @addtogroup STM32F4xx_System_Private_Variables
  * @{
  */
  /* This variable is updated in three ways:
      1) by calling CMSIS function SystemCoreClockUpdate()
      2) by calling HAL API function HAL_RCC_GetHCLKFreq()
      3) each time HAL_RCC_ClockConfig() is called to configure the system clock frequency
         Note: If you use this function to configure the system clock; then there
               is no need to call the 2 first functions listed above, since SystemCoreClock
               variable is updated automatically.
  */
  uint32_t SystemCoreClock = 16000000;
  __I uint8_t AHBPrescTable[16] = {0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 6, 7, 8, 9};

/**
```

```
 * @}
 */

/** @addtogroup STM32F4xx_System_Private_FunctionPrototypes
 * @{
 */

/**
 * @}
 */

/** @addtogroup STM32F4xx_System_Private_Functions
 * @{
 */

/**
 * @brief  Setup the microcontroller system
 *       Initialize the FPU setting, vector table location and External memory
 *       configuration.
 * @param  None
 * @retval None
 */
void SystemInit(void)
{
 /* FPU settings ------------------------------------------------------------*/
 #if (__FPU_PRESENT == 1) && (__FPU_USED == 1)
   SCB->CPACR |= ((3UL << 10*2)|(3UL << 11*2));  /* set CP10 and CP11 Full Access */
 #endif
 /* Reset the RCC clock configuration to the default reset state ------------*/
 /* Set HSION bit */
 RCC->CR |= (uint32_t)0x00000001;

 /* Reset CFGR register */
 RCC->CFGR = 0x00000000;

 /* Reset HSEON, CSSON and PLLON bits */
 RCC->CR &= (uint32_t)0xFEF6FFFF;

 /* Reset PLLCFGR register */
 RCC->PLLCFGR = 0x24003010;

 /* Reset HSEBYP bit */
 RCC->CR &= (uint32_t)0xFFFBFFFF;

 /* Disable all interrupts */
 RCC->CIR = 0x00000000;

 /* Configure the Vector Table location add offset address ------------------*/
#ifdef VECT_TAB_SRAM
 SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM */
#else
 SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal FLASH */
#endif
}

/**
  * @brief  Update SystemCoreClock variable according to Clock Register Values.
 *       The SystemCoreClock variable contains the core clock (HCLK), it can
 *       be used by the user application to setup the SysTick timer or configure
 *       other parameters.
 *
 * @note   Each time the core clock (HCLK) changes, this function must be called
 *       to update SystemCoreClock variable value. Otherwise, any configuration
 *       based on this variable will be incorrect.
 *
 * @note   - The system frequency computed by this function is not the real
 *         frequency in the chip. It is calculated based on the predefined
 *         constant and the selected clock source:
 *
 *         - If SYSCLK source is HSI, SystemCoreClock will contain the HSI_VALUE(*)
```

```
  *
  *         - If SYSCLK source is HSE, SystemCoreClock will contain the HSE_VALUE(**)
  *
  *         - If SYSCLK source is PLL, SystemCoreClock will contain the HSE_VALUE(**)
  *           or HSI_VALUE(*) multiplied/divided by the PLL factors.
  *
  *       (*) HSI_VALUE is a constant defined in stm32f4xx_hal_conf.h file (default value
  *           16 MHz) but the real value may vary depending on the variations
  *           in voltage and temperature.
  *
  *       (**) HSE_VALUE is a constant defined in stm32f4xx_hal_conf.h file (its value
  *            depends on the application requirements), user has to ensure that HSE_VALUE
  *            is same as the real frequency of the crystal used. Otherwise, this function
  *            may have wrong result.
  *
  *       - The result of this function could be not correct when using fractional
  *         value for HSE crystal.
  *
  * @param  None
  * @retval None
  */
void SystemCoreClockUpdate(void)
{
  uint32_t tmp = 0, pllvco = 0, pllp = 2, pllsource = 0, pllm = 2;

  /* Get SYSCLK source -------------------------------------------------------*/
  tmp = RCC->CFGR & RCC_CFGR_SWS;

  switch (tmp)
  {
    case 0x00:  /* HSI used as system clock source */
      SystemCoreClock = HSI_VALUE;
      break;
    case 0x04:  /* HSE used as system clock source */
      SystemCoreClock = HSE_VALUE;
      break;
    case 0x08:  /* PLL used as system clock source */

      /* PLL_VCO = (HSE_VALUE or HSI_VALUE / PLL_M) * PLL_N
         SYSCLK = PLL_VCO / PLL_P
         */
      pllsource = (RCC->PLLCFGR & RCC_PLLCFGR_PLLSRC) >> 22;
      pllm = RCC->PLLCFGR & RCC_PLLCFGR_PLLM;

      if (pllsource != 0)
      {
        /* HSE used as PLL clock source */
        pllvco = (HSE_VALUE / pllm) * ((RCC->PLLCFGR & RCC_PLLCFGR_PLLN) >> 6);
      }
      else
      {
        /* HSI used as PLL clock source */
        pllvco = (HSI_VALUE / pllm) * ((RCC->PLLCFGR & RCC_PLLCFGR_PLLN) >> 6);
      }

      pllp = (((RCC->PLLCFGR & RCC_PLLCFGR_PLLP) >>16) + 1 ) *2;
      SystemCoreClock = pllvco/pllp;
      break;
    default:
      SystemCoreClock = HSI_VALUE;
      break;
  }
  /* Compute HCLK frequency --------------------------------------------------*/
  /* Get HCLK prescaler */
  tmp = AHBPrescTable[((RCC->CFGR & RCC_CFGR_HPRE) >> 4)];
  /* HCLK frequency */
  SystemCoreClock >>= tmp;
}

/**
```

```
 * @}
 */

/**
 * @}
 */

/**
 * @}
 */
/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

UtilidadesPlaca.c

```
/**
 ******************************************************************************
 * @file    UtilidadesPlaca.c
 * @author  Pascual Martinez Perez
 * @version V1.0
 * @date    12-Mayo-2016
 * @brief
 ******************************************************************************
 *
 *
 ******************************************************************************
 */



/* Includes ------------------------------------------------------------------*/
#include "UtilidadesPlaca.h"
#include "stm32f4_discovery.h"

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */

/**
 * @brief  leds_Saludo
 * @param  None
 * @retval None
 */
void leds_Saludo(void)
{
 leds_ON();
 //HAL_Delay(500);
 leds_OFF();

 /*BSP_LED_Toggle(LED3);
 HAL_Delay(500);
 BSP_LED_Toggle(LED4);
 HAL_Delay(500);
 BSP_LED_Toggle(LED5);
 HAL_Delay(500);
 BSP_LED_Toggle(LED6);
 HAL_Delay(500);*/
}

/**
 * @brief  leds_OFF
 * @param  None
 * @retval None
 */
void leds_OFF(void)
{
 BSP_LED_Off(LED3);
 BSP_LED_Off(LED4);
 BSP_LED_Off(LED5);
 BSP_LED_Off(LED6);
```

```
}


/**
  * @brief  leds_ON
  * @param  None
  * @retval None
  */
void leds_ON(void)
{
  BSP_LED_On(LED3);
  BSP_LED_On(LED4);
  BSP_LED_On(LED5);
  BSP_LED_On(LED6);
}

/**
  * @brief  led_Naranja
  * @param  None
  * @retval None
  */

void led_Naranja_ON(void)
{
  BSP_LED_On(LED3);
}

void led_Naranja_OFF(void)
{
  BSP_LED_Off(LED3);
}

/**
  * @brief  led_Verde
  * @param  None
  * @retval None
  */

void led_Verde_ON(void)
{
  BSP_LED_On(LED4);
}

void led_Verde_OFF(void)
{
  BSP_LED_Off(LED4);
}

/**
  * @brief  led_Rojo
  * @param  None
  * @retval None
  */

void led_Rojo_ON(void)
{
  BSP_LED_On(LED5);
}

void led_Rojo_OFF(void)
{
  BSP_LED_Off(LED5);
}

/**
  * @brief  led_Azul
  * @param  None
  * @retval None
  */
```

```
void led_Azul_ON(void)
{
  BSP_LED_On(LED6);
}

void led_Azul_OFF(void)
{
  BSP_LED_Off(LED6);
}
```