Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

ÁREA DE TECNOLOGÍA ELECTRÓNICA

TRABAJO FIN DE GRADO Nº 18010181

DISEÑO DE UN DRIVER DE LED DE 200W DESDE LA RED ELÉCTRICA CON CONTROL DALI

AUTOR: D. MIGUEL GARNELO RODRÍGUEZ

TUTOR: D. ANTONIO JAVIER CALLEJA RODRÍGUEZ
COTUTOR: D. PABLO JOSÉ QUINTANA BARCIA

FECHA: JULIO 2018

Agradecimientos

En primer lugar, me gustaría mostrar mi agradecimiento a todos los componentes del grupo de investigación de Conversión Eficiente de Energía, Electrónica Industrial e Iluminación (CE^3I^2), en especial a Manolo y a Calleja por haberme brindado la oportunidad de trabajar codo con codo con ellos. Haber formado parte de sus proyectos de investigación ha supuesto una experiencia inolvidable que, no solamente me ha hecho mejorar en el ámbito académico y profesional, sino también en el personal.

Quisiera hacer también una mención especial a mi cotutor Pablo Quintana, quien ha dedicado numerosas horas de su tiempo a que este proyecto lograra salir adelante. Y quien considero que debe llevarse parte del mérito de los grandes resultados obtenidos.

Gracias a todas aquellas personas que en menor o mayor medida han estado ahí a lo largo de estos años, sobre todo, a las que han decidido seguir a mi lado con el paso del tiempo. Sin vuestra ayuda, el camino no habría sido igual de fácil.

Por último, me gustaría dedicar este trabajo a mis padres y a mi hermana, en agradecimiento a los valores que me han inculcado y a todo el cariño, apoyo y consejo que he recibido por su parte en los momentos más difíciles. Todo lo que pueda decir de vosotros siempre se quedará corto.

Índice

1.- Introducción

1.1.- MOTIVACIÓN

En los últimos años, con la rápida evolución de la electrónica y las comunicaciones ha surgido una nueva tendencia que se está comenzando a implantar a pasos agigantados: la Industria 4.0. Este término se emplea para describir una visión de la fábrica inteligente, en la que todos sus procesos están interconectados por el denominado Internet de las cosas (IoT). Y es que hoy en día, es absurdo no reconocer que todo y todos estamos conectados.

Por otra parte, el cambio climático y el calentamiento global son un tema de actualidad de los que tampoco se puede hacer caso omiso, siendo necesaria una transición a modelos de consumos de energía más eficientes y sostenibles. Por ejemplo, el consumo de energía en España el pasado año 2017, alcanzó los 268.5 TWh [1] (un 1.3% más que en 2016) y, en 2015, el consumo energético en iluminación y electrodomésticos (sector residencial) se cuantificó en 51.5 GWh (fuente IDAE).

Observando el gran porcentaje que supone el consumo de energía para sistemas de iluminación y el cambio que está experimentando la industria y el mundo en general, es interesante estudiar la posibilidad de un encontrar un sistema de iluminación que reduzca estas cifras.

Desde hace unos años, el empleo de la tecnología LED en sistemas de iluminación ha aumentado notablemente (Figura 1.1) debido a sus grandes prestaciones y ventajas con respecto a las tecnologías tradicionales (lámparas de incandescencia, de descarga, etc.).

La principal ventaja de los LED es su alta eficiencia energética: con un consumo de energía muy bajo son capaces de proporcionar un flujo de luz elevado. Por otra parte, son elementos que cuentan con una larga vida útil, hasta 50.000 horas, no precisando además de ningún tipo de mantenimiento, puesto que se trata de elementos semiconductores que no posee ningún elemento mecánico como filamentos. Su composición los convierte

también en un elemento muy poco contaminante, ya que no contiene elementos metálicos como mercurio o wolframio (a diferencia de las lámparas de incandescencia o de descarga).

Otra gran ventaja es que son fácilmente *dimmables* (capacidad de regular su salida lumínica), característica muy difícil de conseguir con las lámparas de incandescencia y muy poco viable en las lámparas de descarga. Esta cualidad reduce aún más todavía su consumo energético y aumenta su vida útil al trabajar con corrientes medias más bajas.

A la vista de las numerosas ventajas que proporciona la tecnología LED resulta interesante combinarla con un protocolo de comunicaciones, desarrollando así luminarias con la capacidad de incluirse en ambientes totalmente conectados y automatizados. Esta combinación proporciona un producto de aún mejores prestaciones, ya que se puede, entre otras funcionalidades, saber en todo momento qué luminarias están en funcionamiento, gestionando a distancia su supervisión, o se podrían crear diferentes escenas dentro de una misma estancia, dependiendo de los niveles de iluminación de cada zona.

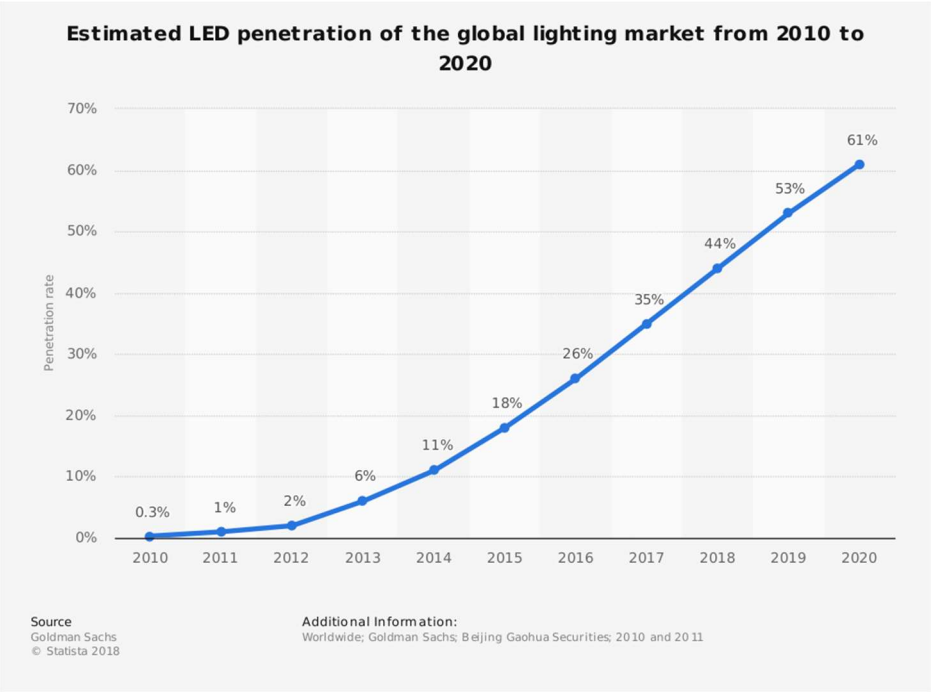


Figura 1.1.- Penetración de la tecnología LED en el mercado de la iluminación (fuente Goldman Sachs, gráfico obtenido de la web Statista).

1.2.- OBJETIVO

El objetivo general de este trabajo es presentar al lector una revisión detallada de los sistemas de iluminación, enfocándose particularmente en los basados en tecnología LED y la interconexión de estos sistemas con buses de comunicación para su manejo, así como la presentación de un prototipo que cumpla todos estos requisitos.

Concretamente, se detallará el proceso de diseño, implementación y testeo de un *driver* para una luminaria LED de 200 W controlada mediante el protocolo DALI, desarrollado dentro de un proyecto de investigación en colaboración con la empresa Normagrup.

Este proyecto surge para expandir el uso de la tecnología LED a ámbitos que no solamente sean de índole doméstica, buscando así nueva línea de mercado.

Al tratarse de una luminaria de tanta potencia, está pensada para destinarse a grandes espacios, tales como instalaciones industriales, centros comerciales, salas de exposiciones o estadios deportivos, en los que se requiere una fuente de luz que sea, en primer lugar, de gran intensidad lumínica con un consumo bajo de energía y, además, perfectamente integrable en una red automatizada, logrando de esta forma un control central y a distancia de la operación de la lámpara.

El diseño del sistema completo supone un gran reto, ya que, combina el desarrollo de la parte hardware (*driver*) y de la parte software (implementación del protocolo DALI en un microcontrolador). Además, una vez diseñadas, deberán ser capaces de funcionar en conjunto integradas en un mismo dispositivo.

2.- Estado del arte

Antes de sumergirse en el diseño físico del sistema, es importante establecer una breve base teórica, en la cual se detallarán conceptos fundamentales relacionados con la iluminación, la tecnología LED y las diferentes formas de control y regulación de luminarias, centrándose más en detalle en el protocolo DALI.

2.1.- ILUMINACIÓN Y ANTECEDENTES A LA TECNOLOGÍA LED

2.1.1.- ¿Qué es la luz? El espectro electromagnético

La luz es una parte de la radiación electromagnética que porta una cantidad de energía perceptible por el ojo humano [2]. Se corresponde con una banda de longitudes de onda determinadas, denominada espectro visible, dentro del espectro electromagnético. Como se puede observar en la Figura 2.1, dichas longitudes están comprendidas entre los 400 y los 750 nm.

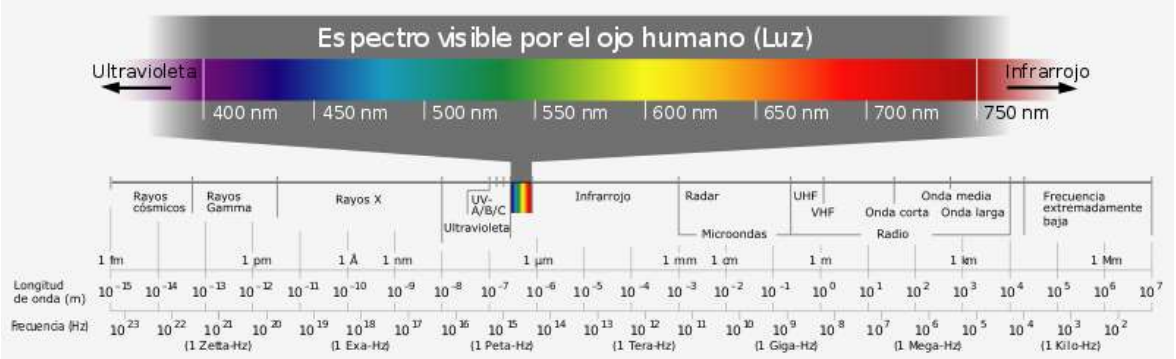


Figura 2.1.- Espectro electromagnético con detalle de la banda visible por el ojo humano (imagen bajo la licencia Attribution-Share Alike 3.0 Unported de Creative Commons).

Al comportarse como una radiación electromagnética, existe una relación entre la energía de la partícula (fotones en este caso) y la longitud de onda de la onda electromagnética asociada, según la conocida ecuación de Planck (2.1):

$$E = h \cdot \frac{c}{\lambda} = hv \qquad (2.1)$$

Donde: h es la constante de Planck, c es la velocidad de propagación de la partícula en el vacío, λ la longitud de onda asociada a la onda electromagnética (luz en este caso) y v es la velocidad de propagación en un determinado medio.

2.1.2.- Propagación de la luz

La luz puede emitirse mediante diferentes mecanismos, clasificados según la forma de excitar a los electrones [3]. Dentro de todos ellos, cabe destacar dos mecanismos principales:

- **Radiación térmica**: puede definirse como la emisión de radiación electromagnética debida al movimiento de electrones causado por la temperatura del material. Si esa emisión de energía se encuentra dentro del espectro visible, entonces es denominada incandescencia [4]. A bajas temperaturas, la energía máxima irradiada se encuentra en el entorno de la región infrarroja; a medida que aumenta la temperatura, el máximo cambia a una longitud de onda cada vez más corta, alcanzando teóricamente las regiones visibles o ultravioleta. De esta forma, se puede establecer una relación directa entre la temperatura y la energía emitida, como se puede observar en la Figura 2.2.
- **Luminiscencia**: se define como la emisión de energía electromagnética irradiada de un cuerpo debido a la excitación de electrones producida por un agente externo. Dependiendo de la fuente de excitación, el fenómeno de luminiscencia puede clasificarse en los siguientes tipos (no siendo los únicos):

  1.- Electroluminiscencia: generada por la acción de un campo o corriente eléctrica al aplicarse sobre un determinado material o sustancia.

  2.- Fotoluminiscencia: generada por la absorción de fotones irradiados con diferentes longitudes de onda. Puede dividirse en subcategorías: fluorescencia, fosforescencia, láser, etc.

  3.- Quimioluminiscencia: debida a reacciones químicas.

  4.- Bioluminiscencia: producida por organismos vivos.

5.- Triboluminiscencia: generada al ejercer una acción mecánica sobre un cuerpo.

6.- Sonoluminiscencia: debida a ondas sonoras.

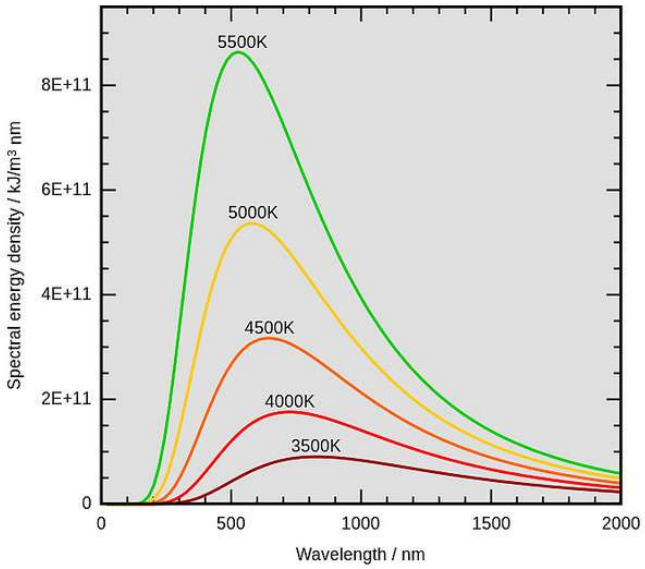


Figura 2.2.-Curvas de energía para radiaciones de un cuerpo negro a diferentes temperaturas (imagen permitida para ser mostrada al dominio público).

2.1.3.- Tecnologías previas al LED

En el caso particular de los diodos LED, la luz es producida por electroluminiscencia; pero antes de la inclusión de esta alternativa en la fabricación de lámparas, otros tipos de tecnologías eran empleadas. A continuación, se describen en función del mecanismo que emplean para la emisión de luz [2], [3]:

- **Lámparas incandescentes**: como su nombre indica, generan la luz mediante radiación térmica al hacer circular una determinada corriente eléctrica a través de un filamento de tungsteno. Generan una gran cantidad de energía fuera del espectro visible (banda infrarroja exactamente), por lo que su eficiencia es baja, entre 8 y 20 lm/W. Su ciclo de vida comprende de 1000 a 2000 horas.
- **Lámparas incandescentes de halógeno**: son básicamente lámparas incandescentes, a diferencia que cuentan con un aditivo basado en halógeno en el gas de relleno: yodo o bromo por lo general. Dicho aditivo reacciona con el vapor

de tungsteno del filamento, surgiendo así un efecto regenerativo que previene la deposición de tungsteno en el bulbo y permite al filamento trabajar a temperaturas más altas, lo cual se materializa en una mayor vida media de la lámpara (entre 3000 y 4000 horas) y en un aumento de la salida lumínica y, por tanto, de su eficiencia la cual puede alcanzar los 25 lm/W.

- **Lámparas de descarga de gas a baja presión**: emiten luz producida por el mecanismo de luminiscencia, la cual tiene lugar dentro de un plasma debido a la acción de un campo eléctrico externo que excita los átomos de dicho plasma. Están caracterizadas por la emisión en espectro discontinuo, fenómeno que depende de la estructura atómica del plasma y de la presión dentro del bulbo (generalmente en torno a 1 Pa).

  A cada lado del bulbo hay dos (o más) electrodos, responsables de generar el campo eléctrico (aunque se pueden encontrar en el mercado lámparas de este tipo sin electrodos) [5]. Dicho campo eléctrico causa la ionización del gas hasta que se alcanza la descarga disruptiva. En este punto, la corriente aumenta rápidamente, así que es necesario emplear un balasto limitador de corriente.

  Las lámparas de baja presión más comunes son: **lámparas de mercurio** (conocidas como fluorescentes) caracterizadas por la emisión de luz UV y que cuentan con una eficacia de 100 lm/W, y las **lámparas de sodio** (LPS) las cuales emiten átomos de sodio y se caracterizan por una eficiencia que rondan los 180 lm/W y porque su luz es bastante monocromática, por lo que no son adecuadas para todas las aplicaciones.

- **Lámparas de descarga de gas a alta presión (HID)**: son un tipo de lámparas que producen luz mediante la formación de un arco eléctrico entre electrodos de tungsteno situados dentro de un tubo lleno tanto con gas como con sales metálicas. La presión del gas de relleno es un parámetro clave en la emisión de luz.

  Al principio, la formación del arco eléctrico se ve facilitada por el gas y, una vez se ha producido, calienta y evapora las sales, incrementando la intensidad de luz

producida por el arco. Sin embargo, como consecuencia de la alta presión en el interior, se alcanzan temperaturas más altas, afectando al rendimiento lumínico de la lámpara, ya que parte de la energía se disipa en forma de calor.

Al igual que en las lámparas de baja presión, es necesario incorporar un balasto que controle y limite la corriente [6].

Las lámparas de gas a alta presión más comunes son: **lámparas de vapor de mercurio** (HPM-HME) [7], la presión en su interior comprende valores entre 200 y 2000 kPa y fueron las primeras lámparas de alta presión en ser comercializadas, pero debido a su baja eficiencia (alrededor de 50 lm/W), pronto fueron reemplazadas por otras opciones; **lámparas de sodio** (HPS-SON), emplean sodio en estado excitado para producir la luz y cuentan con una eficiencia que alcanza los 120 lm/W [8]; y, por último, las **lámparas de haluro metálico** (MH), provienen de las lámparas de mercurio de alta presión, con la adición de haluros metálicos y sales de tierras raras, los cuales irradian dentro del espectro visible, con el objetivo de mejorar tanto la calidad de la luz producida como la eficiencia, la cual supera los 100 lm/W [9], [10].

En la Figura 2.3, se puede observar la evolución de las eficiencias lumínicas de los diferentes tipos de lámparas a lo largo del tiempo, desde la aparición de la bombilla incandescente de Thomas A. Edison en 1879.

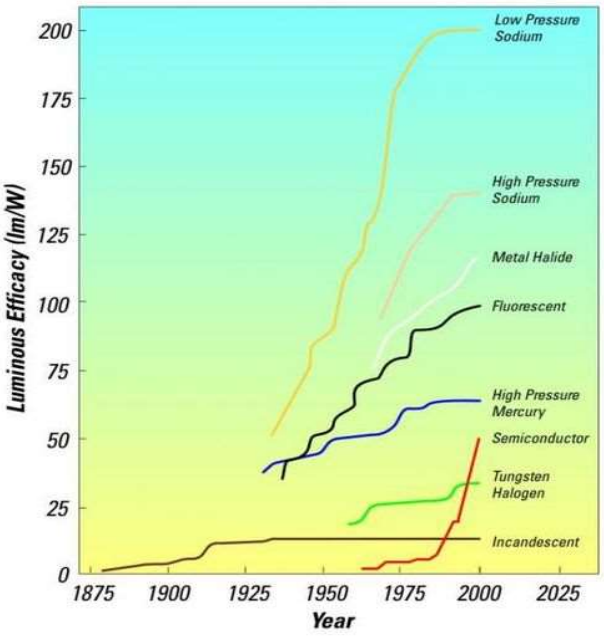


Figura 2.3.- Eficiencia lumínica (lm/W) frente al tiempo (años) (imagen permitida para ser mostrada al dominio público).

2.2.- DIODOS EMISORES DE LUZ

Como se anticipó en el apartado de introducción, se ha decidido optar por la tecnología LED para el desarrollo de la lámpara, incluida en la parte hardware de este proyecto.

2.2.1.- Principio de funcionamiento

Las siglas LED provienen de su nombre en inglés *Light-Emitting Diode*. Este tipo de diodos emplean el mecanismo de electroluminiscencia para emitir luz. La primera vez que alguien notificó este fenómeno, fue Henry Joseph Round en 1907, cuando observó la emisión de luz amarilla al aplicar una determinada tensión sobre un cristal de carburo de silicio (SiC) [11]. Esto es debido a la recombinación de los pares agujero-electrón a través de la banda de separación de un semiconductor cuando la unión p-n es excitada por un campo eléctrico externo.

Los electrones y agujeros constituyen los diferentes portadores de carga en una red semiconductora. Al aplicar un determinado campo eléctrico, los electrones son capaces de "saltar" de la banda de valencia a los niveles de energía libres en la banda de

conducción si cuentan con la energía, creando así huecos en la banda de valencia. La energía ganada (consecuencia del campo eléctrico) se convierte en energía cinética, ya que los electrones están siempre en movimiento. Este aporte de energía les permite alcanzar los niveles de energía más bajos de la banda de conducción; cuando esto sucede es necesario un proceso de recombinación, puesto que es la única posibilidad de mantener el balance de energía (no son posibles estados de energía en la banda de separación). En este proceso, un electrón desciende a la capa de valencia, ocupando alguno de los agujeros disponibles. Si esta liberación de energía se produce por una recombinación radiactiva (emisión de un fotón), que además se encuentra dentro del espectro visible, se producirá la correspondiente emisión de luz. Este fenómeno se puede observar de una manera más ilustrativa en la Figura 2.4.

Una vez visto el principio de funcionamiento de los diodos LED, a continuación, se detallarán sus parámetros más característicos y su modelo equivalente.

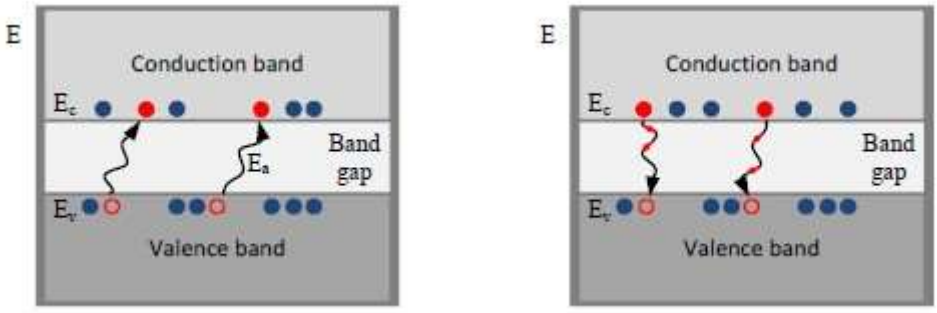


Figura 2.4.- Bandas de energía de un semiconductor. Salto de un electrón a la banda de conducción (izquierda) y vuelta a la banda de valencia (derecha) (imagen reproducida con permiso del autor [3]*).*

2.2.2.- Parámetros característicos y modelo equivalente.

Como su nombre indica, los LEDs son diodos y, por lo tanto, se comportan eléctricamente igual que ellos. Este comportamiento corriente-tensión característico sigue la ecuación de Shockley (2.2) [11]:

$$i = I_S \cdot \left(e^{\frac{v}{\frac{nkT}{q}}} - 1 \right) \qquad (2.2)$$

Donde I_S es la corriente inversa, v es la caída de tensión entre los terminales del diodo, n es un factor de idealidad, *k* es la constante de Boltzmann, T es la temperatura (en Kelvin) y *q* la carga del electrón.

Aplicando la ecuación anterior (2.2), se puede obtener la curva característica V-I del LED. Esta curva suele ser dada por el fabricante para facilitar al usuario el modelado de su diodo. Sin embargo, esta ecuación nos proporcionaría una curva V-I solamente para diodos ideales, ya que en los diodos reales hay elementos no ideales y parásitos. Los más típicos son los efectos resistivos, capacitivos y térmicos. Cabe destacar que esta curva se ve muy influenciada sobre todo por la temperatura en la unión (T_j). Este parámetro es crítico en el comportamiento de los LED, debido a su gran influencia sobre otras de sus características como son la eficiencia, el comportamiento eléctrico y la vida útil de los mismos.

Como ejemplo ilustrativo, en la Figura 2.5 se muestra la curva corriente directa vs. tensión directa del diodo LED empleado en la lámpara diseñada en este proyecto, el diodo NFSL757D-V1 del fabricante Nichia [12], el cual se detallará más en el siguiente apartado de este informe.

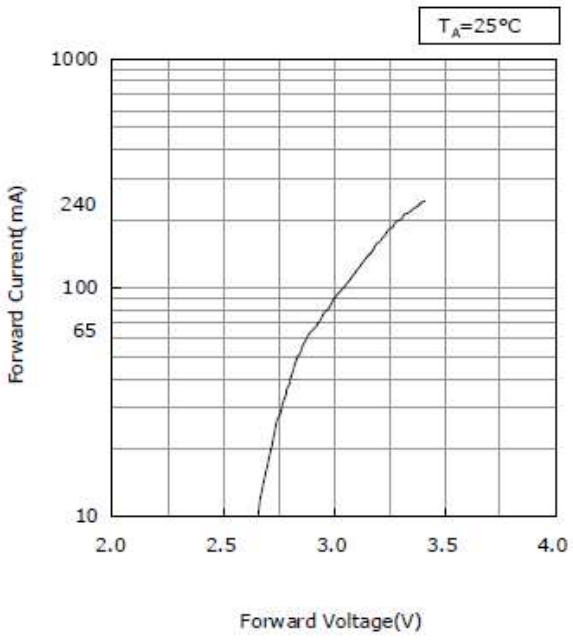


Figura 2.5.- Curva tensión directa vs. corriente directa del diodo LED NFSL757D-V1 [12].

Aunque bien es cierto que hay modelos más complejos teniendo en cuenta todos los efectos anteriormente descritos [13], el comportamiento de un diodo LED se puede modelar como una resistencia colocada en serie con una fuente de tensión y un diodo ideal, como se muestra en la figura 2.6 [3].

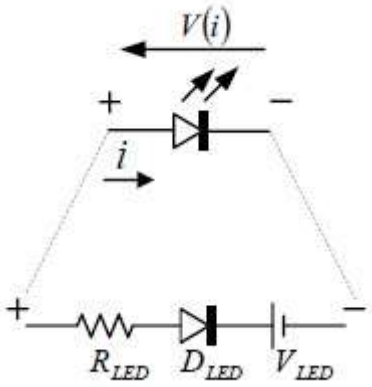


Figura 2.6.- Modelo eléctrico simplificado de un diodo LED (imagen reproducida con permiso del autor [3]*).*

2.2.3.- Principales tipos de LED existentes

Una vez definidas las características constructivas fundamentales de los LED, es conveniente reflejar una clasificación de los diferentes tipos que existen y sus aplicaciones más habituales.

En primer lugar, se puede hacer una enumeración de los distintos tipos de LEDs disponibles en el mercado, basándose en su potencia nominal o encapsulado [2]:

- **LEDs de baja y media potencia**: también conocidos como LEDs de señalización. Están caracterizados por niveles de corriente del orden de las decenas de miliamperios, pudiendo alcanzar los de media potencia unos pocos cientos de miliamperios (150 mA típicamente). Esto, conlleva a un poder de disipación térmica también bajo. En cuanto a sus eficiencias, pueden comprender entre los 80 y los 100 lm/W.
- **LEDs de 1 W**: denominados también LEDs de potencia (*Power-LEDs o P-LEDs* en inglés) o de alta luminosidad (*High-Brightness LEDs o HB-LEDs*). Caracterizados por

un encapsulado de una superficie aproximada de 1 mm^2, su corriente nominal suele ser de unos 350 mA, disipando 1 W de potencia (de ahí su nombre). Aunque estos dispositivos normalmente operan bajo corrientes de entre 1000 y 1500 mA (correspondiendo con disipaciones del orden de los 5 W). Este tipo de diodos fueron la primera alternativa comercial en sobrepasar los 100 lm/W.

- **LEDs de alta potencia**: este tipo de LEDs ha sido desarrollado con el objetivo de cubrir las necesidades que no se consigue satisfacer con los P-LEDs: principalmente, el coste relativo por lumen, debido al gran número de emisores que hay que colocar para satisfacer las especificaciones de lúmenes. Normalmente, están compuestos por asociaciones en paralelo, en serie o, incluso las dos anteriores combinadas, de varios P-LED, formando así un multi-chip o los conocidos diodos CoB (Chip-on-Board). Estos dispositivos han llegado a alcanzar, e incluso sobrepasar, la barrera de los 2 A, consiguiendo altas eficiencias lumínicas.
- Por último, hay un tipo de LEDs que no se pueden encajar en ninguno de los grupos anteriores debido a que cuentan con características especiales como, por ejemplo, diodos LED de corriente alterna alimentados directamente desde la red eléctrica, o LED de alta tensión, que operan al contrario que los LED anteriores: tensiones del orden de decenas de voltios y corrientes muy bajas (no superando apenas los 40 mA), disipando entre 1 y 4 W y alcanzando eficiencias de 100 lm/W.

En la siguiente figura, 2.7, puede apreciarse un ejemplo de cada uno de los anteriores tipos de diodos LED citados.

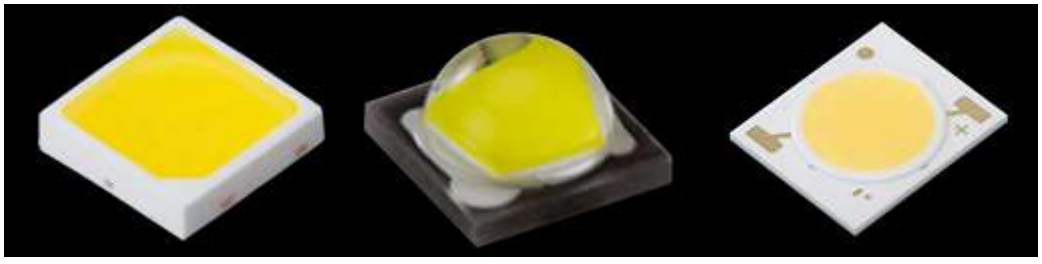


Figura 2.7.- LED de baja potencia (izda.), LED de 1 W (centro) y LED de alta potencia tipo CoB (dcha.) (imágenes del fabricante Nichia con permiso de publicación).

2.2.4.- Aplicaciones

En cuanto a las diferentes aplicaciones de los LED, nos podemos encontrar con una gran variedad, ya que son elementos que demuestran una gran versatilidad y flexibilidad a la hora de abordar un diseño [14]:

- **Iluminación general**: se emplean diodos para aplicaciones que necesitan alta reproducción cromática, gran eficacia y una larga vida útil. En ellas se engloban las bombillas, apliques, paneles informativos, iluminación industrial, farolas, focos, luminarias modulares, etc.
- **Iluminación especial**: diodos con amplia variedad de colores, estilos de encapsulado y directividad. Son los empleados en semáforos, señales de tráfico, linternas, luces decorativas, efectos especiales, paneles de información variable, …
- **Displays**: matrices de LEDs, vallas publicitarias, rótulos, marcadores deportivos.
- **Automoción**: empleo de LEDs de alta fiabilidad y que cumplen con los estándares y normativas de seguridad. Empleados en intermitentes, luces de posición, faros normales y antiniebla, display interno del vehículo (cuentakilómetros, cuentarrevoluciones, …), iluminación interior del habitáculo, indicadores o interruptores en el salpicadero u ordenador de a bordo, etc.
- **Luz de fondo (backlighting)**: retroiluminación en teléfonos móviles, ordenadores portátiles, monitores, televisiones, …
- **LEDs ultravioleta**: de alta potencia y eficacia, empleados para el secado de tintas, tratamiento de resinas y otros materiales, detección de falsificaciones, litografía o grabado de superficies (insoladoras).

Figura 2.8.- Ejemplos de diferentes aplicaciones de la tecnología LED (imágenes bajo la licencia Attribution-Share Alike 3.0 Unported de Creative Commons).

2.3.- REGULACIÓN Y CONTROL DE LEDS Y LUMINARIAS

La variable empleada para el control de los LEDs es la corriente que circula por los mismos, pudiendo trabajar a diferentes valores de la corriente nominal, aunque reduciendo su eficiencia.

Controlar la salida lumínica de una determinada lámpara o luminaria es una característica muy interesante, no sólo por el claro ahorro de energía que ello supone, sino también porque, dependiendo de la aplicación, permite crear diferentes escenarios o ambientes variando el nivel de luminosidad u otros parámetros.

2.3.1.- *Dimming*

Esta técnica, conocida como *dimming*, consiste generalmente en regular la cantidad de corriente que circula a través de la luminaria correspondiente. En el caso particular de los diodos LED, éstos son altamente *dimmables.*

A continuación, se explican las técnicas de *dimming* más conocidas [2][15]:

- ***Dimming* analógico**: también conocido como *dimming* de modulación de amplitud (AM). Es la técnica más simple para controlar la salida lumínica y está basado en la relación lineal existente entre la corriente directa y el flujo luminoso (Figura 2.9). De este modo, variando la corriente continua que alimenta a los LEDs, se va regulando el flujo luminoso hasta el valor deseado.

  Las principales ventajas de este método son su simplicidad y su rentabilidad, siendo aplicable a cualquier driver de LED con solamente variar la referencia de corriente de salida deseada. Otra ventaja es la ausencia de perturbaciones electromagnéticas (EMI). La principal desventaja es que se produce un cambio en las coordenadas cromáticas y en el índice de reproducción cromática (CRI), debido a su dependencia con la corriente de trabajo.

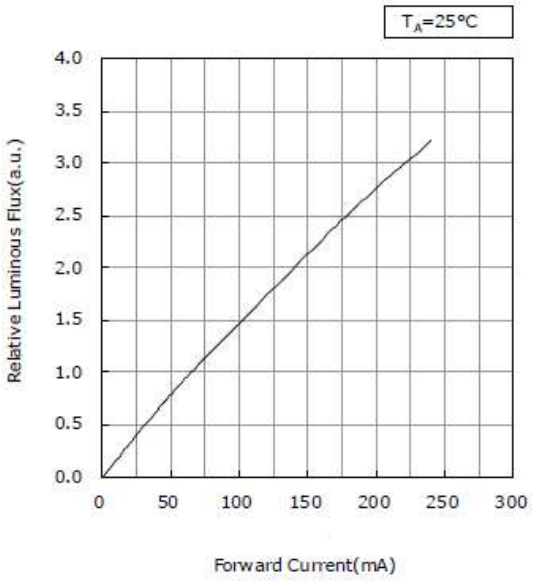


Figura 2.9.- Curva flujo luminoso vs. corriente directa del diodo LED NFSL757D-V1[12].

- ***Dimming* con modulación de ancho de pulso (PWM)**: esta técnica consiste en ir variando el valor medio de la corriente, pero manteniendo constante el valor de pico, con la intención de evitar así el cambio en las coordenadas cromáticas. De esta forma, la lámpara es alimentada con una forma de onda de corriente cuadrada cuyo valor de pico es igual al nivel de corriente continua, pero varía el tiempo en el cual los diodos LED están encendidos con relación al período de la

señal cuadrada de control, consiguiendo modificar el valor medio de corriente de salida [16].

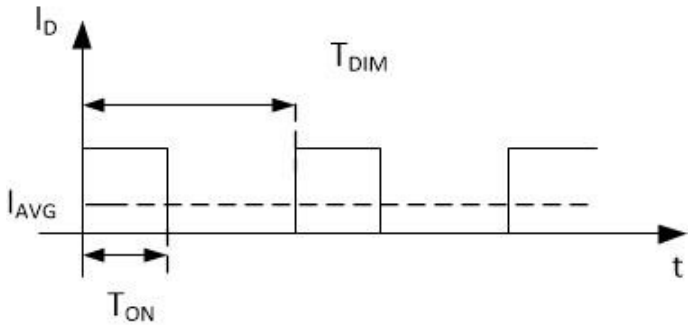


Figura 2.10.- Señal PWM genérica (imagen reproducida con permiso del autor [2]).

Es importante tener en cuenta la frecuencia de dicha señal de control, puesto que si es mayor que la denominada frecuencia crítica de fusión del *flicker* (CFFF) [17], el ojo humano percibirá un decremento de la intensidad lumínica, proporcional al valor medio de la señal cuadrada. El valor de la CFFF está considerado normalmente entre 35 y 40 Hz. Sin embargo, la mínima frecuencia recomendada es de 120 Hz, para evitar, o al menos reducir, así posibles efectos estroboscópicos. Diferentes autores también recomiendan otros valores mayores: 200, 300 o incluso 400 Hz [18]. También cabe destacar que frecuencias superiores a 1 kHz son recomendables para evitar así efectos indeseados de baja frecuencia como el *flicker* no visible, efectos estroboscópicos o ruido audible [19].

En cuanto a las principales ventajas de esta técnica se puede destacar que, al mantener el valor de la corriente de pico constante, se consigue minimizar el cambio en las coordenadas cromáticas (aun no evitándose del todo, ya que se ve afectado también por la temperatura en la unión) [20]. Otra ventaja es la alta linealidad entre la referencia del *dimming* y la salida de luz conseguida [21].

2.3.2.- Protocolos para el control de sistemas de iluminación

En cuanto a los protocolos de control de luminarias, se pueden destacar cuatro principalmente [22], [23]. Tres de ellos se describirán brevemente a continuación, y el

cuarto de ellos se describirá en mayor detalle en el siguiente subapartado, ya que es el que se empleará en el desarrollo de este proyecto: el DALI.

- **Protocolo 1-10 V**: permite la regulación del flujo lumínico entre el 1 y el 100% mediante una señal analógica que llega a los dispositivos a través de una línea de control formada por dos hilos, que cuentan con polaridad. El mínimo nivel lumínico, se consigue cuando la diferencia de tensión es de 1 V entre los hilos del bus de control o bien cortocircuitando los mismos; mientras que el máximo nivel se consigue con una diferencia de 10 V entre los dos hilos o dejándolos en circuito abierto. Cabe destacar que estas líneas de control se emplean únicamente para regular el flujo luminoso, no permitiendo el encendido o apagado de la luminaria; para ello, sería necesario colocar un interruptor en la línea de alimentación del equipo. Las líneas de control y alimentación se encuentran separadas eléctricamente entre sí. Este sistema de regulación es unidireccional, es decir, la información solamente fluye en único sentido (del controlador a las luminarias). La máxima longitud del cableado está limitada por la caída de tensión que se produce a lo largo de la misma, dependiendo a su vez del número de equipos conectados al bus de control.
- **DMX**: acrónimo de Digital MultipleX. Es un protocolo de comunicaciones empleado generalmente en la iluminación arquitectónica, desarrollado en los años 80 por la comisión de ingeniería del Instituto de Tecnología Teatral de los Estados Unidos (USITT). Es un estándar de transmisión de datos digitales a controladores de luminarias, capaz de controlar 512 canales de *dimming*. Los datos se envían en paquetes, que actualizan todos los dispositivos conectados. La selección del dato a transmitir se realiza mediante interruptores de dirección y las tramas de datos no llevan ningún identificador, por lo que se identifican mediante el orden en el que aparecen en el paquete transmitido, siendo la primera trama para el primer *dimmer* que se encuentra, la segunda para el siguiente y así sucesivamente. También es un protocolo unidireccional, solamente envía instrucciones del emisor a uno o más receptores. El cableado de transmisión que emplea tiene una interfaz de 3 o 5 pines.

- **KNX**: acrónimo de Konnex. Es un protocolo de comunicaciones de red definido en el estándar IEC-14543. Nace en 1999 de la fusión de tres estándares previos pertenecientes a European Installation Bus (EIB), a European Home Standards Association (EHSA) y a Batibus [24], desarrollado para el control y gestión de edificios inteligentes (domótica), pudiendo controlar luminarias, pero también otros elementos como persianas, electroválvulas, motores, etc. Las comunicaciones se realizan principalmente a través de un bus formado por un cableado de par trenzado a una tasa de transmisión de entre 4800 y 9600 baudios, aunque el estándar define otros medios de transmisión como la red eléctrica (PLC), radiofrecuencia (RF) o IP (empleando Ethernet o WiFi). Los datos transmitidos están basados en una pila de comunicación desarrollada por EIB completada con los mecanismos de configuración y medios físicos nuevos originalmente desarrollados por Batibus y EHSA. En cada línea KNX se pueden conectar hasta 64 dispositivos como máximo, quedando todos ellos comunicados entre sí, por lo que en este caso la comunicación es bidireccional y todos los equipos pueden comportarse como maestro.

2.4.- PROTOCOLO DALI

Del inglés *Digital Addressable Lighting Interface*. DALI es un protocolo de comunicaciones especificado en los diferentes capítulos del estándar IEC 62386, en el que se define el manejo de forma sencilla e inteligente de equipos de iluminación (balastos, luces de emergencia, equipos LED, etc.). El estándar cumple con el modelo de referencia OSI de ISO.

Figura 2.11.- Logotipo DALI.

La normativa contiene diferentes partes: la parte 101 contiene las especificaciones generales que deben cumplir los componentes del sistema [25], la parte 102 detalla los requerimientos para el caso de los equipos de control de luminarias (esclavos)[26], y la parte 103 define las especificaciones relativas a los dispositivos de control del bus (maestros) [27].

Entre las diferentes funciones que nos permite el protocolo, las principales son: encendido/apagado de luminarias, regulación directa (*dimming*) o progresiva (*fading*) de la salida lumínica, obtención de información de las lámparas (estado de conexión, nivel de luz actual, etc.), creación de grupos de luminarias y almacenamiento de escenas o ambientes de luz determinados, entre otras.

2.4.1.- Capa física. Especificaciones eléctricas.

En cuanto a la interfaz de conexión DALI, consiste en una capa física que sigue una topología de bus formada por dos cables que no cuentan con polaridad, lo cual hace que la instalación DALI sea una tarea relativamente sencilla.

Entre las especificaciones eléctricas del bus DALI, encontramos:

- **Rangos de tensión**:

Emisor	**Mín.**	**Máx.**
Nivel alto (V)	10	22.5
Nivel bajo (V)	0	4.5
Receptor	**Mín.**	**Máx.**
Nivel alto (V)	9	22.5
Nivel bajo (V)	-6.5	9.5
Fuente alim. bus	**Mín.**	**Máx.**
Salida (V)	12	20.5

Tabla 2.1.- Rangos de tensión de los elementos que intervienen en el bus DALI [25].

- **Rangos de corriente**: el máximo valor admisible en el bus es de 250 mA. Mientras no se está transmitiendo nada (bus en reposo), el consumo mínimo debe de ser de 10 µA, necesario para descargar las capacidades del cableado y los condensadores de entrada de los elementos conectados al bus.

2.4.2.- Formato de transmisión/recepción de información

En lo relativo a la estructura que sigue el protocolo para la transmisión de información, DALI emplea codificación Manchester, lo que significa que cada bit está codificado en dos partes (codificación bi-fase). De esta forma, un '0' lógico contiene un flanco de bajada dentro del bit y un '1' lógico se representa como un flanco de subida, como se muestra en la Figura 2.12.

Figura 2.12.- Codificación Manchester [25].

La tasa de transmisión empleada es de 1200 baudios. Esto supone que el tiempo de duración de un bit, $\mathbf{t_{bit}}$, es de:

$$t_{bit} = \frac{1}{1200} = 833.3\ \mu s$$

Aunque al presentar una codificación bi-fase y estar el bit formado por dos niveles claramente diferenciados, el tiempo que se suele emplear como referencia en las comunicaciones DALI es el tiempo que dura medio bit, $\mathbf{t_{half\text{-}bit}}$:

$$t_{half-bit} = \frac{1}{2 \cdot 1200} = 416.7\ \mu s$$

Si dichos tiempos no son cumplidos, las tramas deben ser descartadas. Cabe destacar que existe una tolerancia en los tiempos del ±10%.

Una vez conocido como se codifican los bits, veamos cómo se agrupan los mismos para conformar una trama DALI. Como se puede apreciar en la Figura 2.13, una trama está siempre compuesta por los siguientes elementos:

- **Bit de start**: codificado como un '1' lógico.
- **n bits de datos**.
- **Condición de stop**: generalmente de dos bits de duración. Se pone el bus en estado de reposo.

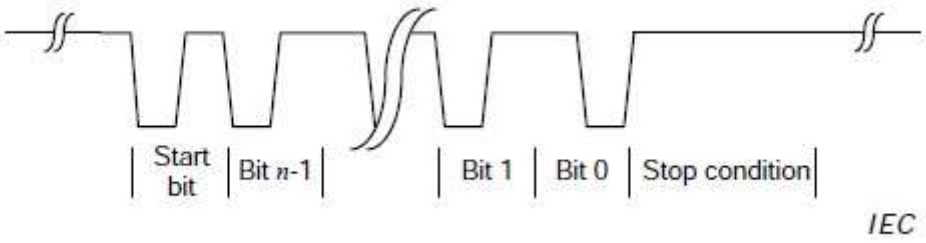


Figura 2.13.- Ejemplo genérico de trama DALI [25].

En el protocolo DALI se pueden encontrar cuatro tipos de tramas principales:

- **Tramas *forward***: son las tramas enviadas por un dispositivo maestro. Pueden ser de 16 bits de datos (si el destinatario es un esclavo) o de 24 bits de datos (destinatario otro maestro).

- **Tramas *forward* de doble envío (*send-twice*)**: son tramas *forward* que, por norma (exclusivas de algunos comandos DALI), deben ser enviadas dos veces consecutivas al esclavo.

- **Tramas *backward***: son las enviadas como respuesta a una trama *forward*. Contienen siempre 8 bits de datos.

- **Tramas reservadas**: no deben ser usadas. Compuestas por 20 o 32 bits de datos.

Estas tramas no pueden ser enviadas libremente, sino que la norma define un tiempo de establecimiento que debe cumplirse entre trama y trama, para evitar confusiones. De esta forma, el mínimo tiempo de establecimiento entre dos tramas especificado es de 2.4

ms. Los máximos dependen del tipo de trama que se esté enviando/recibiendo. Para más información acerca de las tramas, consultar el estándar [25]–[27]

2.4.3.- Redes DALI

En una red DALI puede haber hasta 64 dispositivos, como máximo, conectados simultáneamente al bus de comunicaciones; aunque se pueden emplear pasarelas con el fin de ampliar dicha red con más elementos. Estos dispositivos pueden agruparse generalmente en cuatro tipos:

- **Dispositivos de control del bus**: se comportan como dispositivos maestros enviando los comandos correspondientes.
- **Dispositivos de control de luminarias**: se comportan como esclavos, actuando sobre la lámpara cuando lo indique un maestro y enviando las correspondientes respuestas cuando sea necesario.
- **Fuentes de alimentación del bus**: proporcionan alimentación al bus para que el flujo de información a través sea posible.
- **Pasarelas**: como ya se anticipó, permiten ampliar una red DALI si se alcanza el máximo número de dispositivos conectados al bus (64). En una red doméstica no suelen ser empleadas, pero si la red se encuentra en un lugar amplio, como un edificio de oficinas, un centro comercial o una nave industrial, por ejemplo, su uso puede ser necesario.

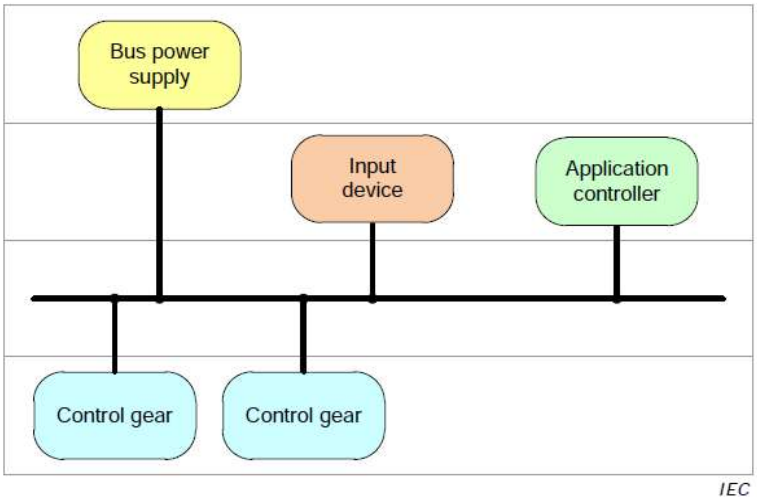


Figura 2.14.- Ejemplo de una red DALI básica [25].

2.4.4.- Comandos DALI

Por último, se puede añadir que todas las funciones que nos ofrece DALI, comentadas unas líneas más arriba, se realizan mediante el envío de unos determinados comandos (y sus correspondientes respuestas si las hay). Dichos comandos también están definidos en el estándar [26] y cada uno tiene un código hexadecimal (*opcode)* que lo identifica. Se pueden agrupar en cinco grupos claramente diferenciados:

- **Comandos de nivel**: establecen el nivel de salida lumínica de la lámpara. Pueden ser de dos tipos: sin *fade*, actualizan de forma inmediata el nuevo valor de luz a fijar; o iniciar un *fade*, la salida lumínica cambia de manera gradual hasta llegar al valor indicado por el comando.
- **Comandos de configuración**: se emplean para modificar alguna propiedad del controlador de la lámpara (fijar los límites de salida, resetear las variables, etc.)
- **Consultas**: su objetivo es obtener el valor de alguna de las variables del controlador o saber en qué estado se encuentra la lámpara.
- **Comandos especiales**: todos los controladores deben interpretarlos. Se emplean sobre todo para el direccionamiento de los esclavos.
- **Comandos de aplicación extendida**: reservados para dispositivos o características especiales. Todavía no han sido definidos en la norma.

3.- Diseño y solución propuesta

Una vez se ha establecido la base teórica, es posible estudiar de manera fiable las diversas alternativas para desarrollar el producto final. En este apartado se explican detalladamente los criterios de diseño utilizados para seleccionar la topología del convertidor DC-DC para el driver y el resto de los componentes del sistema.

El proceso de diseño se ha divido en dos partes fundamentales: por una parte, se ha llevado a cabo el diseño del hardware y elementos de potencia que componen el sistema final y, por otra, el desarrollo de la parte software, desarrollando un código que satisfaga y cumpla todas las especificaciones impuestas por el protocolo DALI. La metodología de trabajo que se ha planteado para esta etapa se puede apreciar en la Figura 3.1. Se ha optado por una división en pequeños bloques claramente diferenciados entre sí; de esta forma, a la hora de abordar el diseño resultará todo más sencillo y finalmente, tras probar cada bloque por separado, sólo será necesario juntarlos para conformar el producto final.

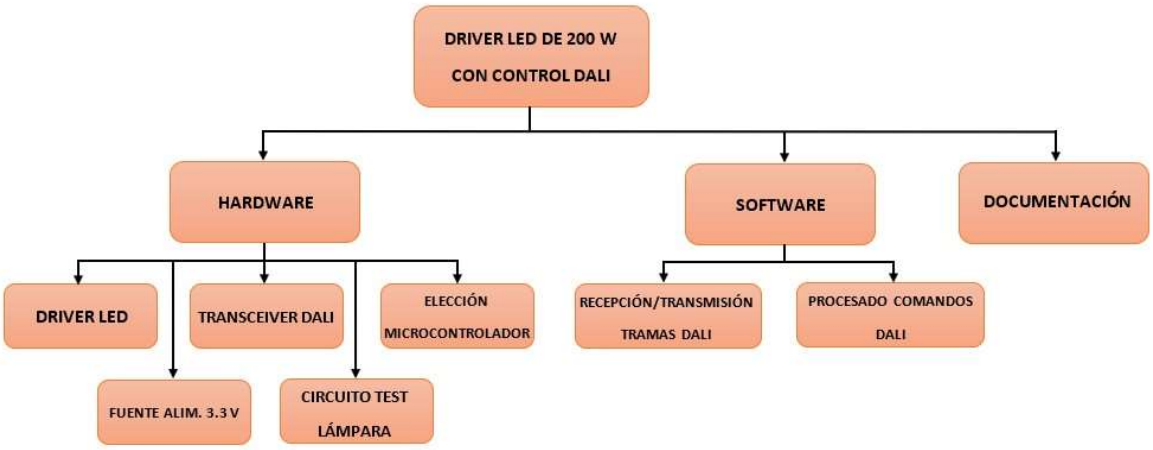


Figura 3.1.- Esquema de la metodología de trabajo seguida.

3.1.- DISEÑO HARDWARE

En este apartado se detalla el cálculo y la selección de los distintos componentes electrónicos que constituirán la implementación física de nuestro diseño.

Es necesario destacar que, a la hora de enfrentarse al diseño de un dispositivo de este tipo, se cuenta con unas restricciones de partida: la tecnología empleada como fuente de luz de la luminaria y las especificaciones eléctricas que define la normativa DALI, las cuales condicionarán las decisiones a tomar.

La tecnología elegida para este proyecto, como ya se ha anticipado, es la tecnología LED. Como punto de partida, se cuenta con unas tiras de LEDs cedidas por la empresa Normagrup (Figura 3.2).

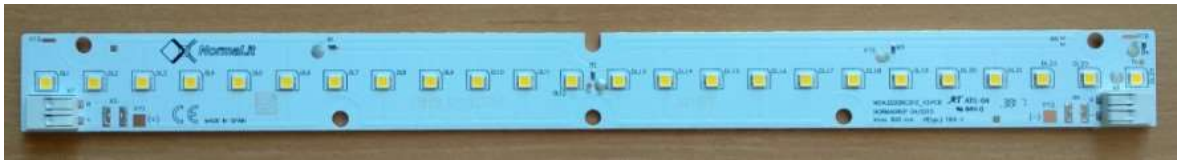


Figura 3.2.- Tira de LED empleada.

El diodo elegido para su fabricación es el LED NFSL757D del fabricante Nichia Corporation [12]. De su hoja de características es posible extraer algunas de sus características más destacables:

- **Tensión directa, V_F** = 2.9 V.
- **Máxima corriente de trabajo, I_F** = 180 mA.
- **Flujo lumínico, ϕ_v** = 23.3 lm.
- **Intensidad lumínica, I_v** = 7.8 cd.

Las tiras están compuestas por cuatro ramas en paralelo que cuentan a su vez con seis diodos LED asociados en serie, es decir, un total de 24 LEDs (Figura 3.3). La forma constructiva de estas tiras cumple con la normativa Zhaga [28], [29], la cual busca estandarizar la fabricación de este tipo de elementos, permitiendo así la inclusión o sustitución por un producto de otro fabricante. En cuanto a las características eléctricas de estas tiras de LED, cuentan con una tensión directa máxima, $V_{F\ máx}$, de 18.6 V y una corriente máxima, $I_{máx}$, de 600 mA.

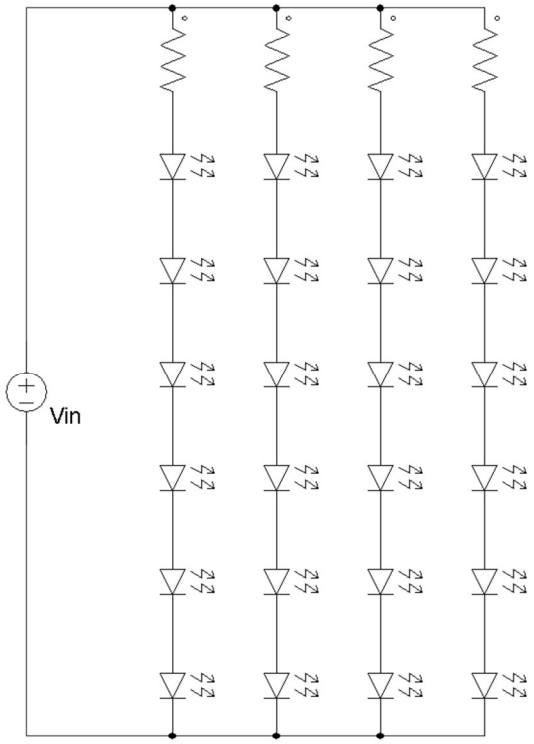


Figura 3.3.- Esquema eléctrico de la tira de LED usada.

La potencia máxima consumida por cada tira empleada puede calcularse de la siguiente forma:

$$P_{máx,\ tira} = 18.6\,V\ \cdot 0.6\,A \approx 11.2\,W$$

De esta forma, para conseguir la potencia buscada (200 W) serían necesarias 18 tiras de LED. Sin embargo, no es recomendable trabajar en el valor de corriente límite, ya que podría tener consecuencias muy negativas para los diodos LED.

Considerando la especificación anterior, se ha decidido optar por una estrategia de diseño modular. Cada módulo contará con dos tiras de LED (asociadas en serie) con su correspondiente *driver*, el cual se encargará de controlar la corriente que circula a través de los diodos LED, limitando el máximo en 500 mA (esta estrategia de diseño se justificará en el apartado de Experimentos y resultados este informe). Por lo tanto, la potencia entregada por cada módulo será de:

$$P_{módulo} = 2 \cdot 18.6\,V\ \cdot 0.5\,A \approx 20\,W$$

De este modo, para conseguir una luminaria de aproximadamente 200 W, serán necesarios 10 módulos como los descritos unas líneas más arriba, lo que supone un total de 20 tiras de LED a emplear, con sus correspondientes *drivers* asociados. Esto se puede ver ilustrado en la Figura 3.4.

Este tipo de estrategia supone numerosas ventajas, en comparación a diseñar un único convertidor electrónico que controle todo el sistema de potencia. Por ejemplo, en caso de fallo de uno de los módulos, el resto podrá seguir operando con total normalidad y sólo será preciso sustituir el defectuoso; además, si se desea una luminaria de menor potencia, simplemente habrá que conectar las tiras de LEDs a los *drivers* necesarios, consiguiendo una variedad de sistemas de iluminación con potencias múltiplo de 20 W.

Aunque también supone alguna desventaja como el empleo de más componentes, lo cual se traduce en un mayor peso y coste del diseño.

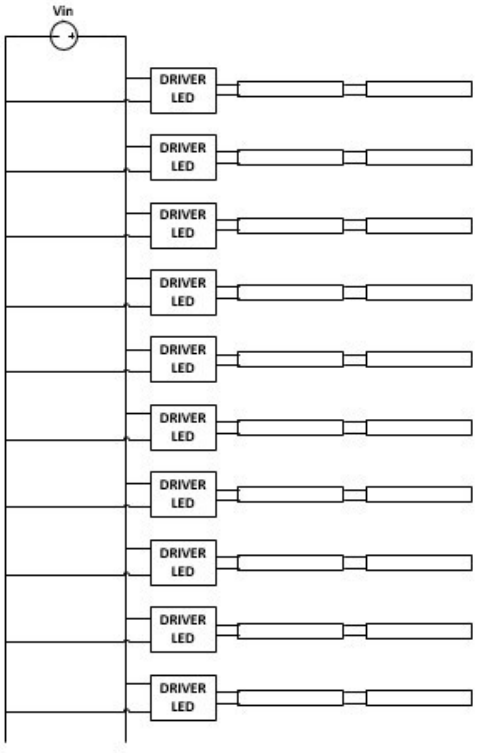


Figura 3.4.- Esquema de potencia de la luminaria.

Como se puede apreciar tanto en la Figura 3.3 como en la Figura 3.4, es necesario contar con una fuente de tensión continua que proporcione alimentación a todos los elementos

del sistema. Al optar por un diseño modular y asociar dos tiras de LED en serie, la tensión máxima de trabajo de los LED será de 37.2 V, por lo que la fuente de alimentación elegida deberá proporcionar, al menos, dicha tensión. Para el desarrollo de este proyecto se ha decidido emplear una fuente de alimentación comercial, cuyo valor es de 48 V.

La fuente seleccionada es del fabricante Mean Well, modelo GS220A48-R7B [30]. Esta fuente se alimenta desde la red (entrada universal de 110 o 230 V) y proporciona 48 V y hasta 4.6 A en la salida. Además, cumple con todas las normativas de seguridad y EMC, lo cual la convierte en un elemento de gran fiabilidad a la hora de seleccionarla para la realización de este trabajo.

Figura 3.5.- Imagen de la fuente de alimentación GS220A48-R7B (proporcionada por el propio fabricante)

Una vez presentados los elementos con los que contamos al inicio, se puede proceder a detallar el diseño del resto de elementos hardware que forman parte del sistema completo.

3.1.1.- Driver LED

Este elemento será el encargado de regular la corriente que circula a través de los LED, lo cual afecta directamente a la salida lumínica que producen los mismos.

Para conseguirlo, es necesario diseñar un convertidor electrónico que cumpla con las siguientes especificaciones de partida:

- Tensión de entrada, V_{in} = 48 V.
- Tensión de salida, V_{out} = 38.2 V.
- Corriente de salida, I_{out} = 500 mA.

Observando dichas condiciones, se puede ver que, a la salida de la topología de potencia necesaria, la tensión es menor que a la entrada del convertidor. Como el sistema ya cuenta con aislamiento galvánico (proporcionado por la fuente de alimentación de 48 V), se ha optado por emplear un convertidor DC-DC reductor (*BUCK converter*), el cual cuenta con un gran rendimiento y su diseño es relativamente sencillo. En la Figura 3.6, se puede apreciar el esquema de un convertidor de este tipo.

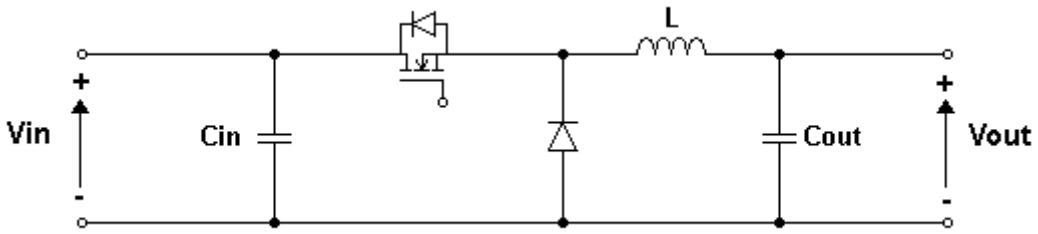


Figura 3.6.- Esquema de un convertidor reductor.

Para su implementación, se ha decidido emplear un circuito integrado fundamentalmente por dos razones: en primer lugar, facilita el diseño del convertidor ya que posee un control analógico interno que regula la corriente de salida; y, en segundo lugar, supone un ahorro económico en componentes electrónicos debido a que reemplaza al transistor MOSFET, su correspondiente *driver* de puerta y el microcontrolador o circuito analógico correspondiente para realizar el control del convertidor.

De esta forma, se ha elegido el circuito integrado Zetex ZXLD1362 (Figura 3.7) del fabricante Diodes Incorporated [31]. Este componente soporta una tensión máxima de alimentación de hasta 60 V y es capaz de proporcionar una corriente de salida de hasta 1 A, características más que suficientes para las condiciones de diseño impuestas.

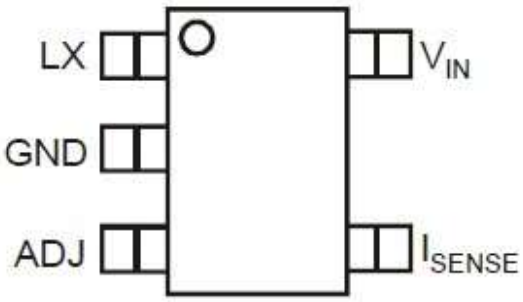


Figura 3.7.- Driver LED integrado ZXLD1362 [31].

Para regular la corriente de salida del convertidor, el ZXLD1362 emplea un control analógico en modo corriente, siguiendo una estrategia de control denominada $I_{máx}$ - $I_{mín}$. Esta técnica consiste en mantener el transistor MOSFET cerrado mientras la corriente es menor que la corriente máxima de la bobina; y, una vez alcanzado dicho valor máximo, se abre el transistor MOSFET hasta que el valor de corriente alcance ahora el valor mínimo de corriente por la bobina. Después el proceso se repite. Este algoritmo se puede ver de una forma más grafica en la Figura 3.8.

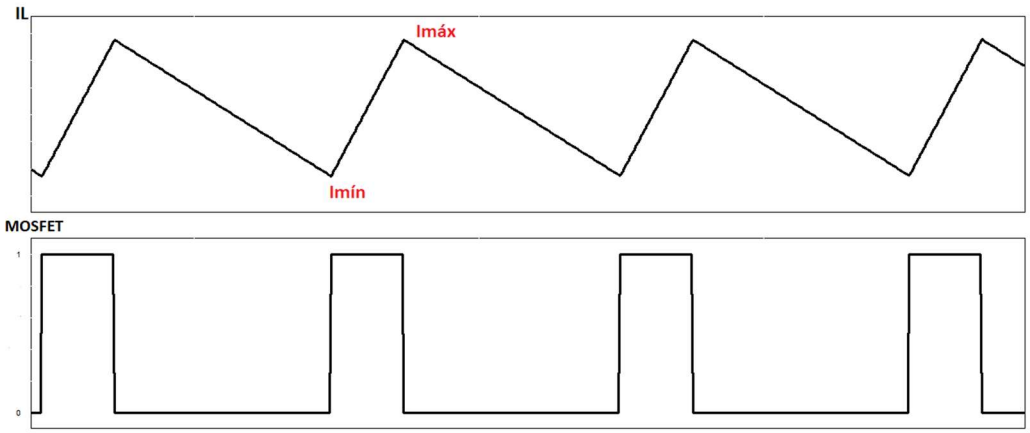


Figura 3.8.- Relación entre la corriente por la bobina y el estado del MOSFET en el algoritmo $I_{máx} – I_{mín}$.

Para que el integrado pueda determinar la señal de control necesaria para regular la corriente, es necesario colocar una resistencia de sensado, R_S, entre las patillas V_{IN} e I_{SENSE} del integrado (Figura 3.7). Según el *datasheet* del componente (*ref ds*), la corriente de salida está relacionada con la resistencia de sensado según la expresión:

$$I_{out} = \frac{0.1}{R_S} \tag{3.1}$$

Siguiendo la ecuación 3.1, para establecer una corriente máxima de 500 mA a la salida del convertidor, el valor de R_S deberá ser de:

$$R_S = \frac{0.1}{0.5} = 0.2\ \Omega$$

Este valor se conseguirá colocando cinco resistencias de 1 Ω en paralelo.

Para completar el diseño del *driver*, es necesario colocar una bobina, un diodo y los respectivos condensadores de filtrado a la entrada y a la salida del convertidor, como indica la Figura 3.6.

Para el caso de la bobina, la hoja de datos del integrado proporciona unas gráficas mostrando el comportamiento del integrado con diferentes valores posibles de bobinas: 68, 100, 150, 220 y 2200 µH. Finalmente, se ha decidido emplear un valor de 150 µH, puesto que ofrece buenos resultados en cuanto a rizado de corriente de salida y, además, ya se contaba en el laboratorio con bobinas de este valor. El modelo de bobina empleado es el ELC08D151E del fabricante Panasonic [32], la cual soporta una corriente de hasta 700 mA.

En lo que respecta al diodo, el fabricante recomienda emplear un diodo de tipo Schottky que sea rápido, de baja capacidad y buen comportamiento en inversa. Siguiendo estas recomendaciones, se ha seleccionado el diodo B260A del fabricante Diodes Incorporated [33]. Este diodo soporta una tensión en inversa de hasta 60 V de pico y una corriente media de 2 A. Además, cuenta con una tensión de codo de 0.7 V y una capacidad parásita de 200 pF.

Por último, la hoja de características del ZXLD1362 aconseja emplear un condensador de desacoplo a la entrada, con el fin de evitar transitorios. Por lo tanto, se ha decidido colocar una capacidad de 4 µF, compuesta por la asociación en paralelo de cuatro

condensadores de 1 µF. Este valor se ha empleado también para el condensador de salida, con el fin de suavizar el rizado de corriente que circula por las tiras de LED.

Otra característica destacable del circuito integrado elegido es que cuenta con una patilla adicional, ADJ, la cual permite regular la corriente de salida a través de una señal externa (analógica o PWM). De esta forma, se podrá emplear una técnica de *dimming* para regular la intensidad lumínica de la luminaria.

Este diseño contará con un microcontrolador encargado de controlar los distintos procesos del sistema; por ello, es conveniente realizar una regulación de la salida lumínica con una señal PWM. Para este caso, el fabricante sugiere el montaje que se muestra en la Figura 3.9, para asegurar que se opera de forma segura sobre el pin ADJ del integrado.

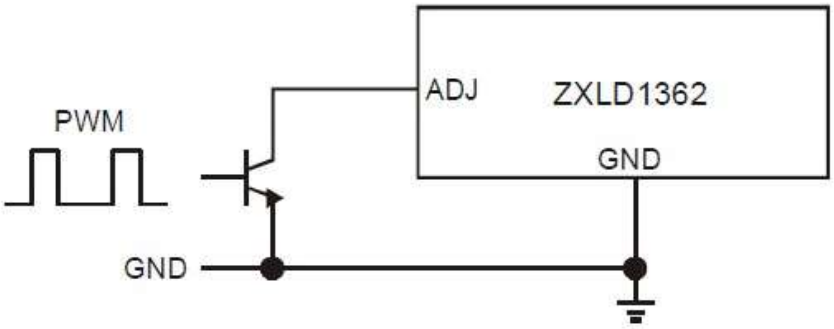


Figura 3.9.- Inclusión de una señal PWM en el integrado ZXLD1362 [31].

De este modo, se ha seleccionado el transistor bipolar (de tipo NPN) BC847B del fabricante Nexperia [34]. Se ha establecido la resistencia de base en un valor de 390 Ω, suficiente para hacer saturar la base del emisor. Se ha conectado una resistencia del mismo valor entre la base del transistor y masa, encargada de limitar la corriente cuando se hace saturar el transistor y se produce un cortocircuito.

Cabe destacar que esta estrategia de *dimming* es "inversa". Cuando el microcontrolador genera una señal PWM de *duty* = 100%, el transistor satura y se cortocircuita el pin ADJ. Esto se traduce en que la corriente de salida del integrado es de 0 A, produciendo el apagado de los LED, en lugar de hacerlos brillar a máxima intensidad como es de esperar para un *duty* del 100%. Por el contrario, para una

señal de *duty* = 0%, el transistor está en corte, quedando el pin ADJ en estado de alta impedancia. Cuando esto ocurre, el control del integrado interpreta que tiene no hay ningún tipo de regulación por lo que la corriente de salida es máxima (los 500 mA fijados), brillando así los LED a máxima intensidad.

De esta forma, para realizar el *dimming* de la luminaria, el *duty* que debe tener la señal PWM generada es en realidad de valor (aspecto que habrá que tener en cuenta a la hora de programar la generación de esta señal):

$$duty_{ADJ} = 100 - duty_{teórico} \tag{3.2}$$

Para finalizar con el diseño del *driver* de LED es necesario realizar dos anotaciones. En primer lugar, el fabricante especifica que la frecuencia de la señal PWM empleada no debe superar los 400 Hz. Por último, es necesario colocar un condensador en paralelo al circuito de control (conectado entre el pin ADJ y masa) para filtrar posibles ruidos que afectan al funcionamiento interno del integrado (esta decisión se detallará en el apartado de Experimentos y resultados). El valor seleccionado es de 100 pF.

3.1.2.- Fuente de alimentación de 3.3 V

Además de los elementos de potencia alimentados a partir de la fuente de 48 V, se precisa contar con una tensión de 3.3 V para alimentar otros elementos como el microcontrolador, el transceiver DALI o el circuito de test de presencia de luminaria, por ejemplo.

Aprovechando que contamos con la tensión de 48 V, la decisión óptima es obtener los 3.3 V a partir de ella por medio de un convertidor. En este caso, sería necesario implementar un convertidor *BUCK* o *FLYBACK*, ya que ambos reducen la tensión de salida respecto a la de entrada. De nuevo, se ha optado por una topología *BUCK*, empleando un circuito integrado que sustituya al transistor MOSFET y su respectivo circuito de control. El componente elegido es el regulador de tensión LM2576HV-ADJ (Figura 3.10) del fabricante Texas Instruments [35]. Este circuito integrado permite una tensión de

alimentación de entre 6 y 60 V y es capaz de proporcionar una corriente a la carga de entre 0.5 y 3 A.

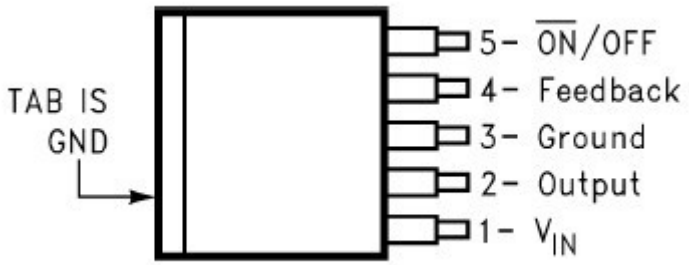


Figura 3.10.- Regulador de tensión LM2576 (ref ds).

El circuito que es necesario diseñar se detalla en la siguiente figura:

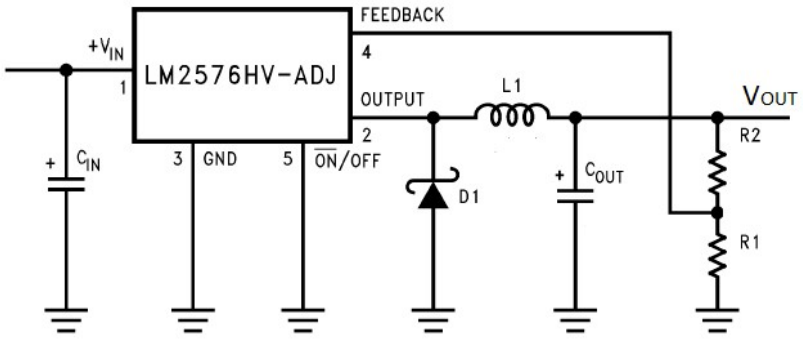


Figura 3.11.- Circuito genérico a partir del LM2576 [35].

Para proceder al diseño de la fuente, se parte de las siguientes condiciones:

- Tensión de entrada, V_{IN} = 48 V.
- Tensión de salida, V_{OUT} = 3.3 V.
- Corriente de salida máxima para los integrados, I_{OUT} = 500 mA.
- Frecuencia de conmutación fija, f_S = 52 kHz.

Las resistencias R1 y R2 se utilizan para medir y realimentar la tensión de salida; de esta forma se cierra el lazo de control. Para determinar sus valores se puede utilizar la siguiente expresión facilitada en el *datasheet*:

$$V_{out} = V_{REF} \cdot \left(1 + \frac{R_2}{R_1}\right) \quad (3.3)$$

Donde V_{REF} = 1.23 V (proporcionado por el fabricante) y R_1 debe tener un valor entre 1 y 5 kΩ.

Por tanto, seleccionando un valor de R_1 de 1 kΩ, el valor de R_2, despejando en la ecuación 3.3, resultará:

$$R_2 = 1000 \cdot \left(\frac{3.3}{1.23} - 1\right) = 1.7\ k\Omega$$

Este valor se ha obtenido, por ejemplo, asociando en serie dos resistencias de montaje superficial (SMD) de valores estándar 1K6 y 100 Ω.

En cuanto al cálculo de la bobina, el fabricante también detalla el procedimiento a seguir en la hoja de características. En primer lugar, es necesario calcular el valor E·T, según la ecuación 3.4.

$$E \cdot T = (V_{IN} - V_{OUT}) \cdot \frac{V_{OUT}}{V_{IN}} \cdot \frac{1000}{f_S\ (kHz)} \quad [V \cdot \mu s] \quad (3.4)$$

Sustituyendo por los valores de diseño en la ecuación 3.4, se obtiene un valor de E·T ≈ 60 V·µs. Con el valor máximo de corriente de salida deseado (500 mA) y el valor de E·T recién calculado, determinamos el valor de la bobina necesaria para la fuente a partir de la siguiente gráfica (Figura 3.12), proporcionada en el *datasheet*.

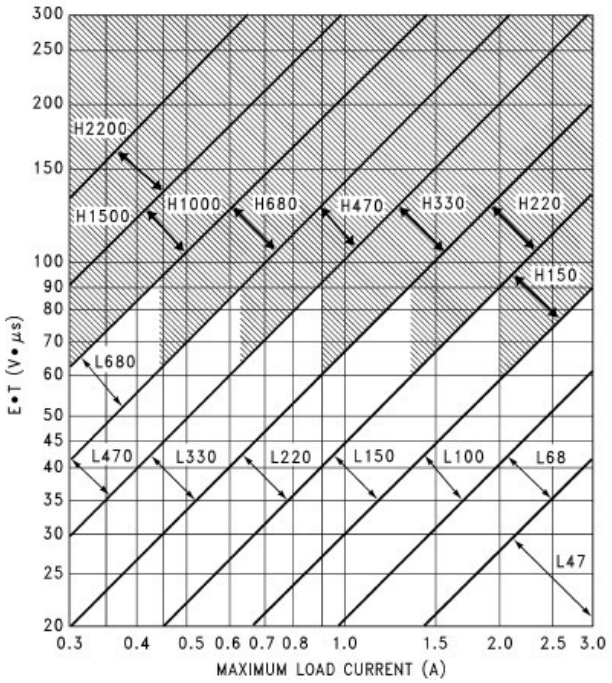


Figura 3.12.- Relación corriente de salida - E·T para la selección de la bobina necesaria para el integrado LM2576 [35].

Al consultar esta gráfica con los valores correspondientes, se determina que nos encontramos en la región L470. Con este código, se consulta la tabla 4 de la hoja de características [35],la cual determina que el valor de bobina necesario es de 470 µH. Además, la bobina elegida debe soportar una corriente 1.15 veces superior la de diseño (500 mA) y una frecuencia de conmutación de 52 kHz. Teniendo en cuenta todos estos requisitos, la bobina elegida ha sido el modelo DO5010H-474MLD de Coilcraft [36].

En cuanto al diodo a utilizar, el fabricante especifica que sea de tipo Schottky y que, además, soporte una corriente al menos 1.2 veces mayor que la de salida (500 mA) y una tensión en inversa 1.25 veces superior a la tensión de entrada (48 V). Con estas condiciones, se ha elegido el diodo PDU620 del fabricante Diodes Incorporated [37]. Es capaz de trabajar con una tensión en inversa de 200 V de pico y una corriente máxima de 6 A, características más que suficientes para el diseño.

Para el condensador de entrada, se especifica en la hoja de características que colocando un condensador electrolítico de 100 µF es suficiente. Se ha colocado un condensador con una tensión máxima de trabajo de 100 V. En paralelo con él, se ha colocado un

condensador cerámico de 100 nF que sirva como filtro para posibles ruidos de alta frecuencia.

Por último, para dimensionar el condensador de salida, el fabricante proporciona la siguiente expresión:

$$C_{OUT} \geq 13300 \cdot \frac{V_{IN}}{V_{OUT} \cdot L\ (\mu H)} \quad [\mu F] \tag{3.5}$$

Sustituyendo en 3.5 con nuestros valores, se obtiene un valor de condensador de al menos 412 µF. Dicho condensador, debe cumplir además que su tensión máxima sea al menos 1.5 veces superior al valor de tensión de salida (3.3 V). A partir de estas condiciones, se ha elegido un condensador de 470, µF (valor estándar inmediatamente superior a 412 µF), que soporta 35 V. En este caso, también se ha colocado un condensador cerámico de 100 nF en paralelo con él.

3.1.3.- Circuito de test de lámpara

Algunos comandos DALI tienen como propósito saber si hay algún elemento que genere luz conectado al sistema o, en otras ocasiones, realizar alguna operación en función de esta información. Por ello, se ha diseñado un circuito electrónico capaz de determinar si las tiras de LED están conectadas, o no, a su correspondiente *driver*.

El circuito consiste en colocar una resistencia de sensado, R_S, en la masa de los *drivers* de LED. De esta forma, si las tiras están conectadas circulará la corriente y producirá una caída de tensión en dicha resistencia. Por el contrario, sino hay tiras presentes, no circulará corriente a través del *driver* y la caída de tensión en la resistencia de sensado será de 0 V.

La caída de tensión producida en R_S se comparará con una tensión estable de referencia empleando una topología de comparación tradicional por medio de un amplificador operacional. La salida del comparador se conectará a un pin de entrada/salida (GPIO) del microcontrolador y a partir del valor leído en él se podrá determinar la presencia de LEDs.

El circuito implementado se representa en la Figura 3.13:

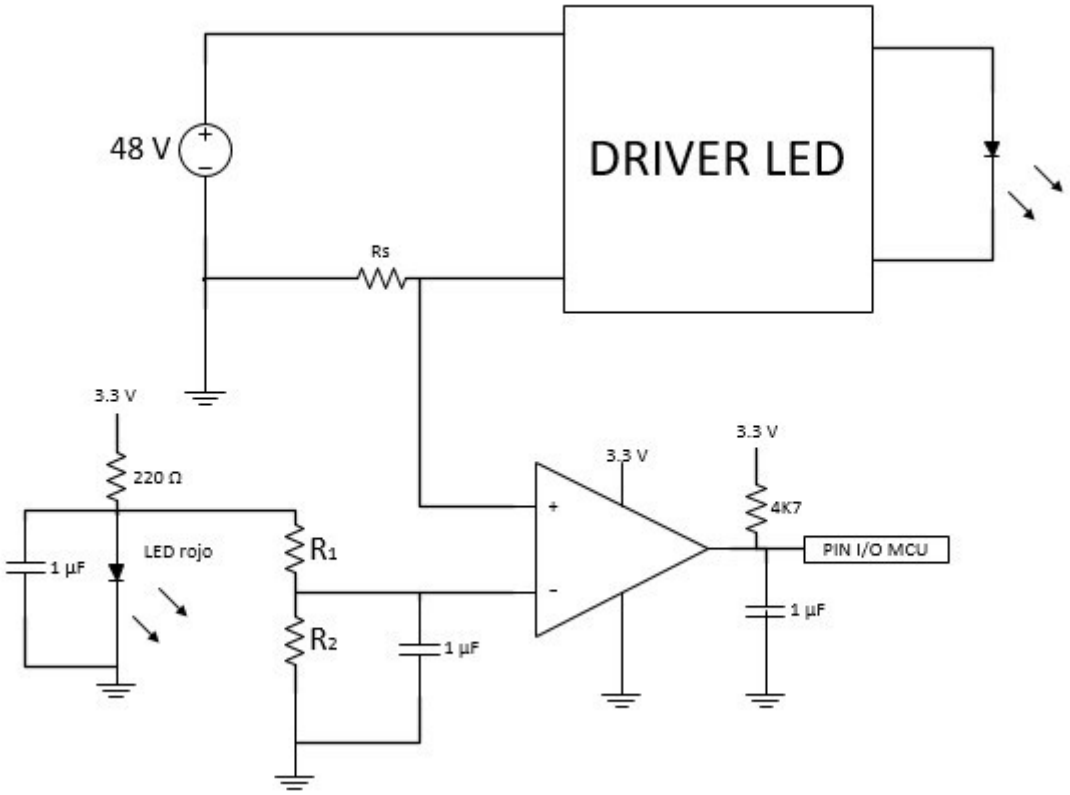


Figura 3.13.- Esquema del circuito empleado para determinar el estado de conexión de una luminaria al driver.

Lo primero que se ha de seleccionar es el valor de la resistencia de sensado. Dicho valor ha de ser muy bajo ya que, debido a la posición donde se está situando, está modificando ligeramente la referencia de masa del *driver*. Además, se sabe que en un convertidor electrónico existe balance de potencias, es decir, la potencia de entrada es igual a la potencia de salida. Aplicándolo, se puede determinar el valor de la corriente de entrada del *driver*, según:

$$V_{IN} \cdot i_{IN} = V_{OUT} \cdot i_{OUT} \tag{3.6}$$

De apartados anteriores, se sabe que la tensión de entrada es de 48 V, la tensión de salida 37.2 V y la corriente de salida se ha fijado en un máximo de 500 mA (cabe destacar que, en el momento de realizar este test, se especifica en la norma DALI que hay que situar la

lámpara al máximo de intensidad lumínica). De esta forma, el valor de la corriente de entrada (despejando en 3.6) es de 386 mA aproximadamente.

Eligiendo uno de los valores mínimos de resistencia que existen, 1.1 Ω, la caída de tensión que se produce en R_S es de 0.4 V. Este valor será la referencia de tensión que se introduce al comparador cuando sí hay tira de LED conectada.

Pero no se podrá hacer la comparación sino se dispone de una referencia estable con la que comparar. Para producir dicha referencia se emplea la tensión de codo de un diodo LED SMD de color rojo, cuyo valor está en torno a los 1.8 V. El diodo LED empleado presenta una tensión de codo de 1.78 V (presenta una desviación del 1%, lo cual no resulta crítico). Para limitar la corriente por dicho LED se coloca una resistencia de 220 Ω en serie con él.

A partir de esos 1.78 V de referencia, se va a generar, por medio de un divisor resistivo, una tensión de 0.1 V, que será la otra tensión para introducir en el comparador. Fijando el valor de R_1 (Figura 3.13) en 10 kΩ, se tiene un valor de R_2 de:

$$R_2 = \frac{0.1 \cdot R_1}{1.78 - 0.1} = \frac{0.1 \cdot 10000}{1.68} = 595\ \Omega$$

Colocando una resistencia del valor normalizado inmediatamente superior a 595 Ω (680 Ω), la caída de tensión que se produce es de 0.11 V (perfectamente admisible como referencia).

Por último, el amplificador operacional (empleado como comparador) elegido es el LM393 (Figura 3.14) del fabricante Texas Instruments [38]. Se trata de un amplificador *Rail-to-Rail Input Output* (RRIO), lo cual es clave a la hora de conectarlo a la entrada de un microcontrolador, ya que nos proporcionará dos niveles lógicos claramente definidos.

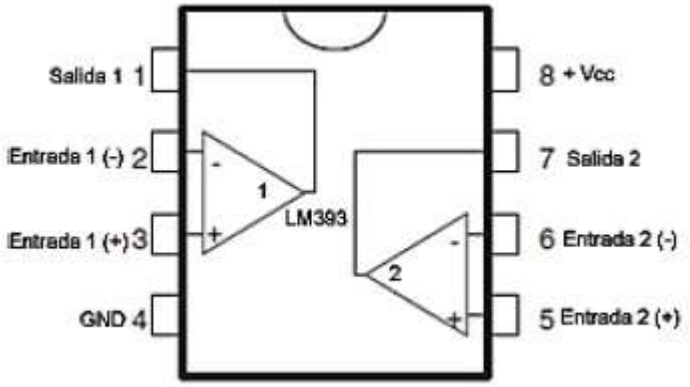


Figura 3.14.- Esquema del circuito integrado LM393, detalle de su distribución interna[38].

De esta forma, cuando se realice el test de presencia de lámpara y las tiras de LED se encuentren conectadas, caerá una tensión de 0.4 V en R_S que, al ser superiores a los 0.1 V de referencia, hará que el microcontrolador lea un '1' lógico a la salida del operacional. Sin embargo, si no hay tiras conectadas a la patilla "+" del comparador llegarán 0 V, los cuales son inferiores a los 0.1 V que llegan a la patilla "-" y el microcontrolador leerá un '0' lógico en el pin correspondiente.

Por último, cabe destacar que el diseño se ha completado con una resistencia de *pull-up* de 4.7 kΩ conectada a la salida del comparador, debido a que esta es de colector abierto. Además, se han añadido varios condensadores cerámicos de desacoplo de 1 µF para filtrar posibles señales no deseadas en el circuito.

NOTA: en la práctica, solamente se ha implementado el circuito en uno de los con el fin de ahorrar componentes, puesto que cada uno de los nueve *drivers* restantes llevaría un circuito idéntico asociado. Como se trata de un proyecto de investigación lo que se pretendía era demostrar su funcionalidad (lo cual se ha conseguido), pero la inclusión de los nueve circuitos de test restantes sólo supondría un pequeño cambio en el código del microcontrolador y en el aspecto de la placa de circuito impreso diseñada, que se comentará unas líneas más abajo.

3.1.4.- *Transceiver* DALI

En el apartado 2.4 de este informe, se presentaron los principios básicos del protocolo DALI, así como las especificaciones eléctricas que tiene el bus empleado en las comunicaciones DALI (Tabla 2.1).

Los valores de tensión presentados superan la tensión de alimentación del microcontrolador (3.3 V), por ello, es necesario diseñar un interfaz que sirva como adaptador de los rangos de tensión del bus a los rangos de tensión manejados por el microcontrolador. Este circuito se conoce como *transceiver* (transceptor o transductor en castellano).

Aunque hay numerosas formas distintas de implementar este circuito, la más común [39], [40] es la que se muestra en la Figura 3.15:

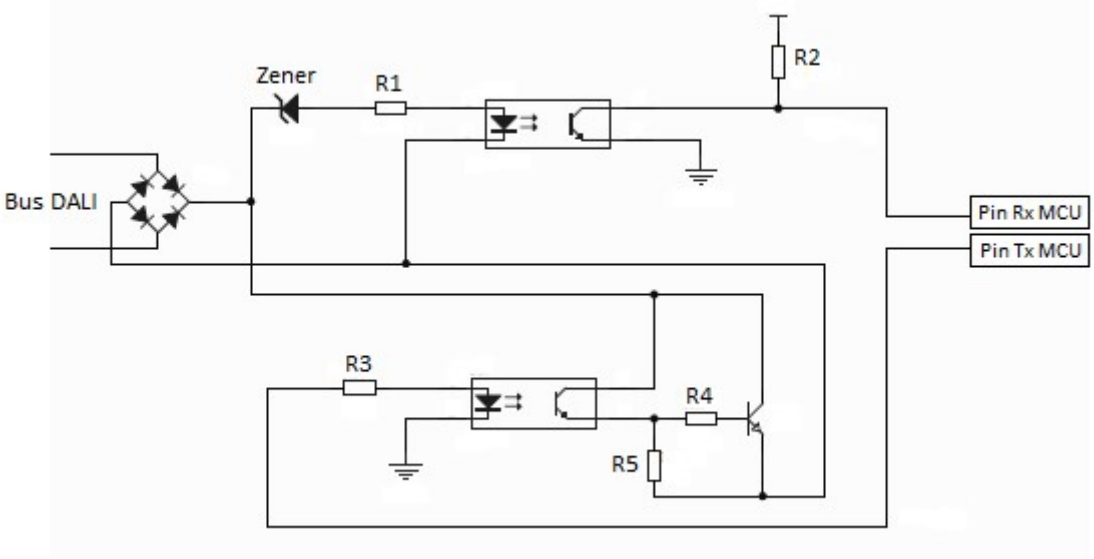


Figura 3.15.- Esquema del circuito de un transceiver DALI.

Según esta implementación, nos encontramos ante una lógica inversa de nuevo. Cuando por el bus DALI no circula información este debe permanecer en estado de reposo (‘1’ lógico según la norma), por lo que al llegar ese ‘1’ lógico al circuito de recepción (parte superior del esquema de la Figura 3.15), el LED del optoacoplador produce luz, saturando así el fototransistor asociado a él. De esta forma, cuando en el bus hay un ‘1’, al pin Rx del

microcontrolador está llegando en realidad un ‘0’ lógico. Del mismo modo, cuando se recibe un ‘0’ en el bus, el diodo LED no luce y al microcontrolador llega un ‘1’.

Lo mismo pasa con la parte de transmisión (parte inferior del esquema de la Figura 3.15). Como el bus DALI se encuentra en estado alto cuando está en reposo, la forma de transmitir información que sigue el protocolo DALI se realiza cortocircuitando el propio bus para cambiar así el nivel de tensión. De este modo, cuando por el pin Tx del microcontrolador se envía un ‘1’, el LED del optoacoplador asociado produce luz, haciendo saturar a su fototransistor, que a su vez hace saturar al transistor NPN conectado a la salida, cortocircuitando así al bus, es decir, hay un ‘0’ en él. Sin embargo, si el microcontrolador envía un ‘0’ por Tx, el optoacoplador no conduce y el bus sigue estando en estado alto (‘1’).

Esto será de gran importancia a la hora de programar el proceso de recepción/transmisión de tramas DALI.

Una vez aclarado esto, se puede comenzar con el diseño hardware del adaptador de señales.

En primer lugar, se puede apreciar que a la entrada del *transceiver* hay un puente rectificador. Esto se debe a que el bus DALI no tiene polaridad, por lo que al colocar este elemento se consigue que la conexión con el bus se pueda realizar de manera indiferente. El puente rectificador elegido es el S380 del fabricante Diotec Semiconductor [41]. Este componente soporta una tensión alterna máxima de entrada de hasta 560 V_{RMS} y los diodos que lo componen, una tensión en inversa de 800 V de pico y una corriente de 800 mA.

Para proteger al microcontrolador de los valores tan altos que maneja el bus DALI, se decide emplear optoacopladores, aislando de esta forma ambas partes. Los componentes elegidos para esta tarea son los optoacopladores TLP181 del fabricante Toshiba [42], los cuales soportan los posibles rangos de tensiones posibles en el bus.

El diodo Zener empleado se usa para bajar la tensión que llega a la entrada del optoacoplador. En relación con los valores del bus, el valor típico empleado en este diodo es de 5.1 V.

A partir de la tensión de alimentación de nuestro bus DALI que proporciona una tensión de alimentación de 17 V, los 5.1 V que caen en el Zener, la tensión de codo del diodo LED del optoacoplador (1.3 V) y la corriente que por dicho LED (recomendable entre 16 y 20 mA según el fabricante), se puede determinar el valor de R_1 según:

$$R_1 = \frac{17 - 5.1 - 1.3}{0.018} = 589\ \Omega$$

Escogiendo un valor estándar de 620 Ω para R_1 (valor inmediatamente superior a 589), la corriente por el LED sería de:

$$i_{LED} = \frac{17 - 5.1 - 1.3}{620} = 17\ mA$$

El valor es totalmente válido, por lo tanto, R_1 = 620 Ω.

A la salida del optoacoplador empleado para la recepción, se coloca una resistencia de pull-up para mantener un valor estable en el pin Rx del microcontrolador cuando no se está recibiendo información a través del bus. Con un valor de R_2 = 1 kΩ es suficiente.

Para el caso de R_3, la tensión de un puerto digital del microcontrolador es de 3.3 V cuando se envía un '1'. De este modo, fijando una corriente por el LED del optoacoplador en cuestión de 20 mA, se obtiene:

$$R_3 = \frac{3.3}{0.02} = 150\ \Omega$$

Por último, en cuanto al circuito empleado para cortocircuitar el bus DALI se emplea una topología idéntica a la utilizada para el control mediante una señal PWM del circuito integrado usado en el diseño del *driver* LED. El transistor bipolar (tipo NPN) es el BC847B del fabricante Nexperia [34]. La resistencia R_4 encargada de saturar la base del transistor

se ha fijado en 68 Ω y R_5, encargada de limitar la corriente cuando se produce el cortocircuito del bus, se ha fijado en 1 kΩ.

3.1.5.- Elección del microcontrolador

El microcontrolador se encargará de controlar varias funcionalidades dentro del sistema tales como la generación de la señal PWM que regulará la corriente de los *drivers* para conseguir una salida de luz con la intensidad deseada en los LED, controlar las comunicaciones con el maestro DALI a través del bus (recepción y transmisión de tramas), así como procesar los diferentes comandos recibidos para generar la acción que indiquen.

Según lo expuesto hasta ahora, es necesario disponer (como mínimo) de:

- Un pin capaz de generar una señal PWM para la regulación de los *drivers*.
- Un pin digital de entrada, capaz de generar una interrupción, para detectar la llegada de una nueva trama y recibirla correctamente.
- Un pin digital de salida para la transmisión de tramas cuando sea necesario (respuestas al maestro).
- Un pin digital de entrada para leer la señal generada por el comparador del circuito que comprueba si la lámpara está conectada.
- Un oscilador capaz de generar una base de tiempos que permita la recepción de tramas DALI de forma correcta. Se recuerda que la velocidad de transmisión empleada por el protocolo DALI es de 1200 baudios.

De acuerdo con las especificaciones anteriores, el microcontrolador elegido es el STM32F051C6T6 del fabricante STMicroelectronics [43]. Este microcontrolador cuenta con un núcleo ARM Cortex-M0 de 32 bits y en su hoja de características se pueden consultar sus características principales, las cuales demuestran que nos encontramos ante un componente de grandes prestaciones y calidad. Entre ellas encontramos algunas como:

- Reloj interno de 8 MHz (capaz de llegar hasta 48 MHz mediante un PLL). Posibilidad de conectar osciladores externos de entre 4 y 32 MHz o de 32 kHz.

- Convertidor analógico/digital de 12 bits que cuenta con hasta 16 canales de conversión.
- Hasta 11 módulos *timer* de diferentes características.
- Memoria flash de hasta 64 Kbytes y SRAM de 8 Kbytes.
- Interfaces de comunicación USART, SPI e I^2C.
- Gran número de pines de entrada/salida digital disponibles.

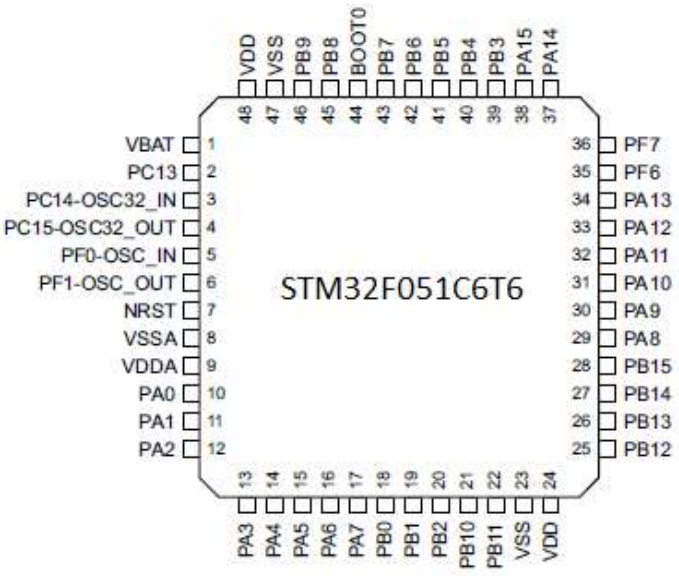


Figura 3.16.- Pinout del microcontrolador STM32F051C6T6 [43].

Aparte de las grandes prestaciones del núcleo ARM, otra ventaja de los microcontroladores del fabricante STMicroelectronics son los drivers HAL [44].

El HAL (Hardware Abstraction Layer) proporciona un conjunto genérico de varias instancias de API (interfaces de programación de aplicaciones) para interactuar con la capa superior (aplicaciones, bibliotecas y pilas). Las API del controlador HAL se dividen en dos categorías: API genéricas (proporcionan funciones comunes y genéricas para todas las series STM32) y API de extensión (incluyen funciones específicas y personalizadas para una determinada línea, por ejemplo, para conectar módulos WiFi o Bluetooth, bus CAN, etc.).

Los controladores HAL incluyen un conjunto completo de API listas para usar que simplifican la implementación de la aplicación del usuario: como ejemplo, los periféricos contienen API para inicializar y configurar el periférico, administrar transferencias de datos, manejar interrupciones o DMA (Direct Memory Access), y administrar errores de comunicación.

Para programar el microcontrolador, será necesario incluir un interfaz de tipo JTAG (Joint Test Action Group) (Figura 3.17) a través del cual se cargará el programa correspondiente en la memoria flash.

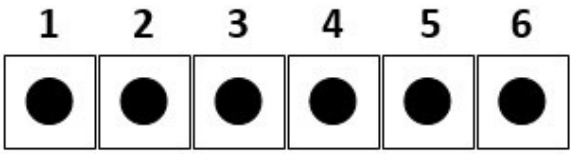


Figura 3.17.- Conector JTAG para programa el microcontrolador.

La conexión del programador con el microcontrolador se realizará de la siguiente forma:

- **Pin 1**: conectado a 3.3 V.
- **Pin 2**: conectado a SWCLK (pin PA14).
- **Pin 3**: conectado a masa (GND).
- **Pin 4**: conectado a SWDAT (pin PA13).
- **Pin 5**: conectado a NRST.
- **Pin 6**: conectado a SWO (pin PB3).

3.1.6.- Otros elementos hardware

En este apartado se recogen otros elementos que se ha precisado incluir en el diseño final.

Es necesario colocar un pulsador de RESET para el microcontrolador. Como el pin NRST al que va conectado es de lógica negada, será necesario una resistencia de pull-up en serie con el pulsador. Su valor se ha fijado en 4.7 kΩ.

Según la hoja de características, se recomienda conectar el pin BOOT0 a masa a través de una resistencia de 100 kΩ. Entre todos los pines de alimentación y masa del microcontrolador se ha colocado un condensador cerámico de 1 µF para filtrar así posibles ruidos.

Además, se han colocado dos LEDs verdes en paralelo a las salidas de ambas fuentes de alimentación, para indicar que están conectadas y funcionando correctamente. En el caso de la fuente de 48 V, se ha colocado una resistencia en serie con el diodo LED de valor 2.2 kΩ (así se limita su corriente a unos 20 mA aproximadamente). Para la fuente de 3.3 V, se ha empleado una resistencia de 220 Ω, circulando entonces una corriente de 15 mA por el LED.

También se han colocado tres diodos LED para realizar indicaciones relacionadas con el proceso de comunicación. El LED azul está asociado a la recepción de tramas (se enciende cada vez que se recibe un '1'), el LED verde está asociado al proceso de transmisión (se enciende cada vez que se envía un '1') y, por último, el LED rojo está asociado a fallos en el bus DALI (se enciende si el bus permanece en estado bajo durante medio segundo). Estos LED están conectados a un pin digital del microcontrolador a través de una resistencia de 220 Ω cada uno.

En lo que respecta a la parte de diseño hardware, solamente queda añadir que todos los elementos descritos en este apartado se han diseñado en una placa de circuito impreso (PCB) diseñada con el programa Altium Designer. Para la parte experimental se ha hecho una placa individual de cada elemento; pero para un futuro diseño final, todos los elementos se integrarían una única PCB.

NOTA: Todos los esquemáticos realizados se pueden consultar en el Anexo I.

3.2.- DISEÑO SOFTWARE

En este apartado se explicará la implementación del protocolo DALI en el microcontrolador [22], [45].

Como se especificó en el esquema que refleja la metodología a seguir, este parte del proceso de diseño se puede dividir en dos bloques diferentes:

- La recepción o transmisión de las tramas correspondientes a través del bus DALI.
- El procesamiento de los comandos enviados por el dispositivo maestro.

3.2.1.- Proceso de comunicaciones a través del bus

Como especifica la normativa DALI en su capítulo de aspectos generales [25], la tasa de transmisión empleada es de 1200 baudios (como se vio en el apartado 2.4). Esto implica que el tiempo de transmisión de un bit es de 833.3 µs; pero al transmitir la información codificada en Manchester, se concretó que el tiempo característico a tener en cuenta era el tiempo que dura el envío de medio bit (416.7 µs).

Tomando este tiempo como referencia, será necesario crear una base de tiempos para las operaciones de recepción y transmisión que asegure que se cumplen los rangos de tiempo especificados en la norma y, además, no afecte a la integridad de las tramas.

Basándose en el teorema del muestreo de Nyquist-Shannon, la frecuencia de muestreo empleada deberá ser, al menos, el doble de la frecuencia de transmisión. Para el caso del medio bit DALI, la frecuencia de transmisión es de:

$$f_{half-bit} = \frac{1}{416.7 \cdot 10^{-6}} = 2400\ Hz$$

Por lo tanto, la frecuencia empleada en la base de tiempo definida deberá ser al menos de 4800 Hz. No obstante, en los procesos de comunicaciones es común realizar cuatro muestras por bit para asegurar que la trama se recibe (o transmite) de forma adecuada. De este modo, aplicando dicha estrategia al caso que se describe, se tomará un número de cuatro muestras por cada medio bit. Esta decisión implica que la frecuencia empleada deberá ser de 9600 Hz, lo que supone que el *timer* empleado para generar la base de tiempos deberá de temporizar:

$$t_{muestreo} = \frac{1}{9600} = 104.17\ \mu s$$

Para esta tarea, se ha decidido emplear el *timer* denominado SysTick. Se trata de un temporizador de 24 bits de cuenta descendente, que produce una interrupción cuando su registro cuenta llega desde el valor de precarga inicial hasta 0. Además, una vez se produce la interrupción, se autorecarga empezando una nueva temporización desde el mismo valor de precarga (sino se indica uno nuevo). Estas características lo hacen óptimo para el proceso de comunicaciones, ya que en cada interrupción se podrá ejecutar la acción correspondiente a la recepción o transmisión de tramas si procede.

Por ello, es necesario determinar el valor de precarga que genera una temporización de la duración deseada. Del *datasheet* del microcontrolador se obtiene la siguiente expresión [43]:

$$Precarga = SysTick\ Counter\ Clock\ (Hz) \cdot Temporización\ deseada\ (s) \qquad (3.7)$$

Donde, en este caso la frecuencia del reloj se ha configurado para la máxima del microcontrolador (48 MHz) y la temporización deseada es de 104.17 µs. Sustituyendo en 3.7, el valor de precarga resulta:

$$Precarga = 48 \cdot 10^6 \cdot 104.17 \cdot 10^{-6} = 5000$$

Este valor será necesario cargarlo en el registro correspondiente al realizar la configuración del SysTick.

Esta base de tiempos será compartida por los procesos de recepción y transmisión, por lo tanto, se emplearán unas variables que indiquen en qué proceso nos encontramos, evitando así la interferencia entre ambos.

3.2.1.1.- Proceso de recepción

El microcontrolador debe recibir las tramas enviadas por el dispositivo maestro, las cuales contienen la instrucción que se deberá ejecutar.

Las tramas enviadas por el dispositivo maestro son del tipo *forward***.** Cuentan con 16 bits de datos, de los cuales los ocho más altos representan la dirección del esclavo

correspondiente y, los ocho más bajos, el código hexadecimal del comando enviado. El aspecto de una trama de este tipo puede observarse en la Figura 3.19:

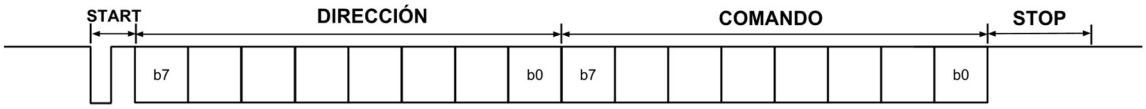


Figura 3.18.- Ejemplo de trama forward.

En total se van a recibir 19 bits (1 bit de start + 8 bits de dirección + 8 bits de comando + 2 bits de stop).

Como se comentó al explicar la base de tiempos, se va a realizar la toma de 4 muestras por cada medio bit. Por lo que, de esta forma, la trama se puede dividir en un total de 152 muestras:

$$4\frac{muestras}{medio\ bit}\cdot 2\frac{medio\ bit}{bit}\cdot 19\ bits = 152\ muestras$$

Empleando entonces una variable que se incremente cada vez que se produzca una interrupción del SysTick, se puede recorrer toda la trama, leyendo la información del pin Rx y almacenándola en la variable que corresponda. Además, el protocolo DALI emplea codificación Manchester y, en este tipo de codificación, el valor lógico del bit recibido se corresponde con el nivel lógico de la segunda mitad del bit. Por lo tanto, sabiendo esto, eligiendo las posiciones de la trama clave de cada bit, se podrá almacenar la trama enviada por el maestro (recordando que debido al diseño del *transceiver*, el nivel lógico recibido en realidad es el complementario del valor leído en Rx). Las posiciones en las que se leerá el valor del pin Rx se marcan en la Figura 3.19:

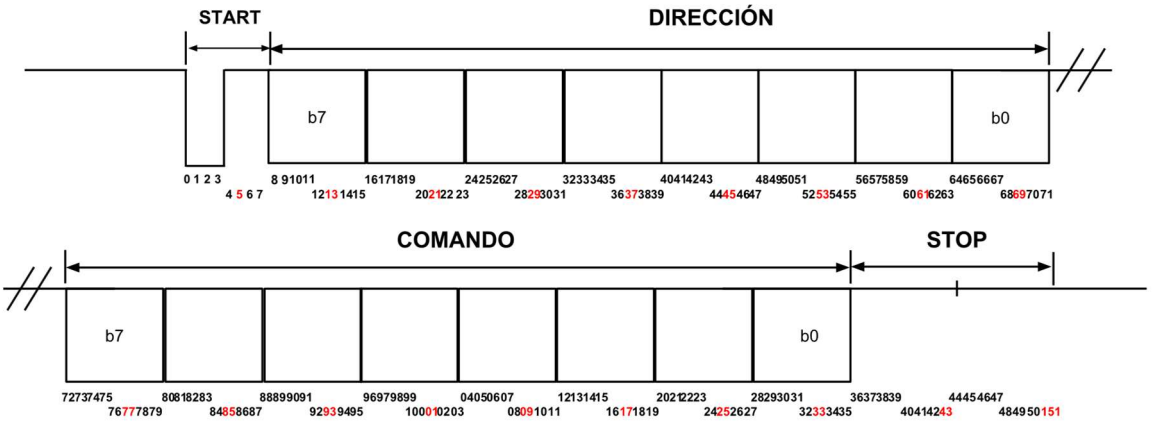


Figura 3.19.- Trama forward con el número total de muestras tomadas, resaltando en rojo en las que se realizará la lectura del valor en el pin Rx.

Una vez que se sabe en qué momento se deben tomar las muestras de la trama para recibirla de forma satisfactoria, es necesario emplear un mecanismo que nos permita detectar en qué instante se produce el inicio de la recepción.

Observando la apariencia de la trama *forward*, se puede observar que en el momento en que llega el bit de start, el bus DALI pasa de estar en estado de reposo (‘1’) a tener un nivel lógico de ‘0’. Por lo tanto, se puede emplear ese flanco de bajada para sincronizarse e indicar que se ha producido el inicio del proceso de recepción. Por ello, será necesario configurar el pin Rx para que genere una interrupción debido a un evento externo caracterizado por un flanco de bajada (en este caso, debe configurarse para un flanco de subida debido a la arquitectura del *transceiver*).

El pin del microcontrolador seleccionado como Rx es el pin PB1. El cual, según lo expuesto debe configurarse como entrada digital capaz de generar una interrupción si se detecta un flanco de subida.

Es importante reflejar que, una vez detectado el primer flanco correspondiente al bit de start, se debe desactivar la interrupción externa asociada al pin Rx hasta que se complete todo el proceso de recepción. Esto se debe a que, de otro modo, los flancos de subida internos en la trama, debidos al cambio de valor de los bits (más acusado en el caso de la codificación Manchester ya que hay flancos dentro de cada propio bit), harían que se

generase dicha interrupción, haciendo entender al microcontrolador que se está comenzando un nuevo proceso de recepción.

También se ha implementado un mecanismo de detección de fallos en la codificación de las tramas, por lo que, si no se recibe lo que debería, se descarta esa trama. Tras el descarte de la trama se habilita de nuevo la interrupción de Rx para que el microcontrolador quede a la espera de una nueva trama.

3.2.1.2.- Proceso de transmisión

Siguiendo un proceso similar al de recepción, el microcontrolador de la luminaria (esclavo) deberá responder a las peticiones del maestro cuando corresponda.

En este caso, las tramas enviadas son del tipo *backward* (Figura 3.20). Cuentan con 8 bits de datos, los cuales contienen la respuesta que se enviará al dispositivo maestro.

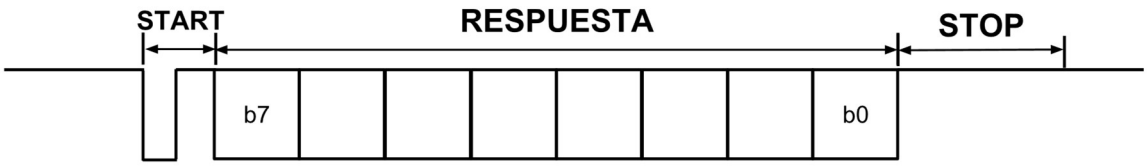


Figura 3.20.- Ejemplo de trama backward.

En total se van a enviar 11 bits (1 bit de start + 8 bits de respuesta + 2 bits de stop). Lo que se traduce en un número de muestras de:

$$4\frac{muestras}{medio\ bit}\cdot 2\frac{medio\ bit}{bit}\cdot 11\ bits = 88\ muestras$$

Como se especificó en el apartado 2.4 (Protocolo DALI) de este informe, debe existir un tiempo de establecimiento mínimo entre las tramas de 2.4 ms. Por lo que una vez se recibe una trama enviada por el maestro, el dispositivo esclavo debe esperar el tiempo de establecimiento necesario antes de enviar la respuesta. Como no todos los comandos

DALI requieren de respuesta, la espera del tiempo de establecimiento se realizará dentro del proceso de transmisión.

Realizando una espera equivalente a 3 bits, el tiempo transcurrido sería de 2.5 ms (válido como tiempo de establecimiento). Este cálculo se detalla a continuación:

$$3\ bits \cdot 8\frac{muestras}{bit} \cdot 104.17\frac{\mu s}{muestra} = 2.5\ ms$$

Por lo tanto, a la vista de estos cálculos habrá que esperar a que transcurran 24 interrupciones del SysTick para empezar a enviar la correspondiente rama *backward* de respuesta. Sumadas a las 88 muestras de la trama, se tendrá un total de 112 muestras o lo que es lo mismo, de interrupciones de la base de tiempos.

Siguiendo una estrategia idéntica a la del proceso de recepción. Empleando una variable que incremente su valor cada vez que se produzca una interrupción del SysTick, se podrá recorrer la trama, cambiando el nivel enviado por Tx en el instante que sea necesario (recordando que la lógica del transceiver es inversa).

El proceso de transmisión comenzará cuando el valor de la variable usada sea igual a 23 (se inicializa en 0). A partir de este momento, cada cuatro interrupciones del SysTick (duración del medio bit) se irá cambiando el valor lógico de salida en el pin Tx según el valor del bit que se quiera enviar. Este proceso se ilustra de forma más gráfica en la Figura 3.21. Como se trata de codificación Manchester, el nivel a colocar en el pin Tx se extraerá mediante operaciones lógicas entre el bit de la respuesta que toca enviar y la variable empleada para recorrer la trama.

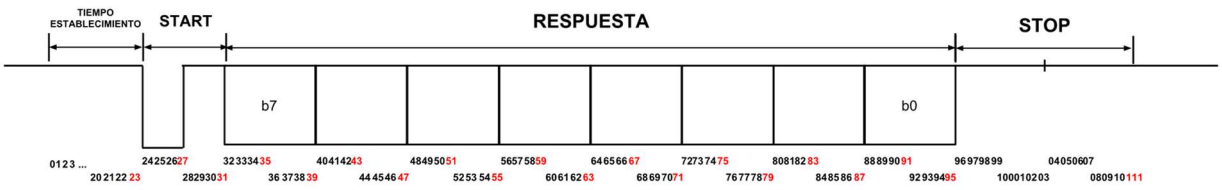


Figura 3.21.- Trama backward con el número de interrupciones necesarias para enviar toda la respuesta, destacando en rojo las posiciones en las que se cambia el valor enviado por el pin Tx.

El pin del microcontrolador seleccionado como Tx es el pin PB0, el cual debe configurarse como salida digital.

Es interesante reflejar que cuando no se está recibiendo ni enviando ningún dato, el microcontrolador sigue produciendo interrupciones del SysTick cada 104.17 µs. La norma especifica que, si el bus DALI permanece en estado bajo durante un período de tiempo superior a 500 ms, el sistema deberá ponerse en estado de fallo, realizando las acciones pertinentes que conlleva esta situación. Por ello, se puede aprovechar esas interrupciones en las que no hay operaciones ligadas al proceso comunicativo para leer el valor del bus y almacenar en una variable las veces seguidas que permanece a '0'. Si dicha variable llega al valor equivalente al transcurso de 500 ms, el sistema se pondrá en estado de fallo.

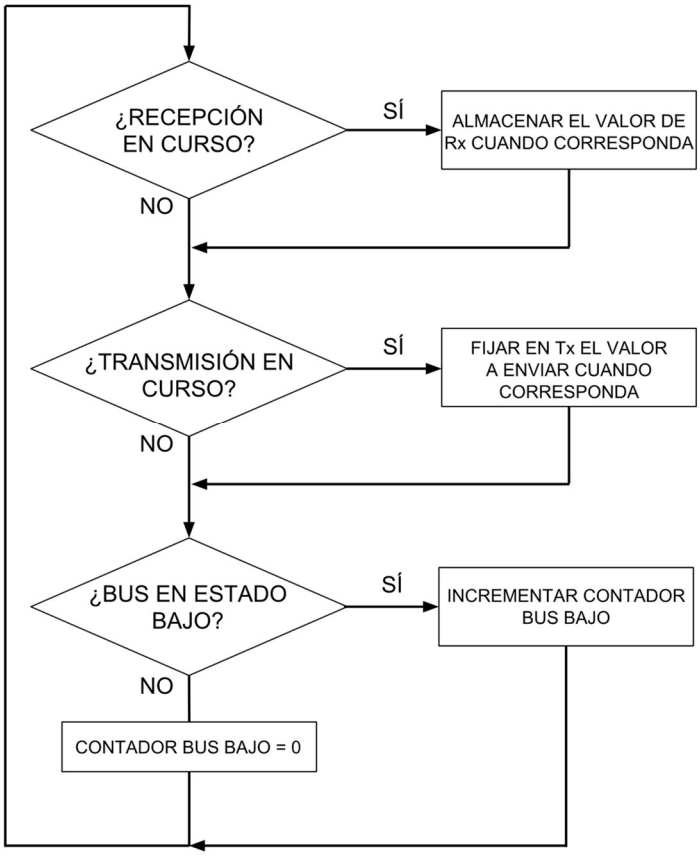


Figura 3.22.- Flujograma que refleja las distintas operaciones dependientes de las interrupciones de tiempo generadas por el SysTick.

3.2.2.- Procesado de los comandos DALI recibidos

El papel que desempeña la luminaria diseñada dentro de la red DALI es el de dispositivo esclavo. De esta manera, se recibirán órdenes enviadas por parte de un dispositivo maestro, que deberán ser procesados por el microcontrolador para realizar la acción asociada a cada comando que se especifica en la norma [26].

El código implementado para la interpretación de comandos se ha realizado siguiendo las indicaciones para cada comando que proporciona la normativa IEC62386-102 en su capítulo 11: Definición de comandos [26].

Aunque gran parte del código ha consistido en la implementación de las especificaciones del protocolo en el microcontrolador, otra parte precisa ser comentada para su mejor comprensión.

3.2.2.1.- Configuración PWM

Cuando se reciban comandos de nivel, el microcontrolador deberá generar una señal PWM para regular la corriente que circulará por las tiras de LED.

Para generar dicha señal, se emplea el canal 1 del *timer* TIM1 del STM32F051C6T6 (correspondiente al pin PA8) [43]. Se trata de un *timer* de control avanzadode 16 bits que cuenta con hasta seis canales para operaciones CCP (Captura, Comparación y PWM), aunque también puede ser usado como un temporizador de propósito general.

La frecuencia del PWM se ha establecido en 200 Hz (valor típico empleado en *dimming* como se vio en el apartado 2.3.1 y admitido por el integrado ZXLD1362). Entonces, será necesario configurar el TIM1 del microcontrolador con los valores correspondientes para producir la señal de dicha frecuencia deseada. Del *datasheet* obtenemos la siguiente expresión:

$$f_{timer} = \frac{f_{CLK}}{(Prescaler - 1)} \tag{3.8}$$

Donde f_{CLK} es la frecuencia de reloj, configurada en 48 MHz. Fijando un Prescaler de valor igual a 4, se obtiene que la frecuencia del *timer* es la quinta parte de la frecuencia del reloj interno, es decir, 9.6 MHz. Es necesario emplear el Prescaler, ya que la frecuencia de PWM deseada es demasiado baja en comparación a la del reloj interno del microcontrolador; entonces, al calcular el siguiente valor necesario, TIMPeriod, se obtendría un valor superior a los 16 bits del *timer*.

Se calcula ahora el valor necesario a cargar en el registro de configuración del PWM (TIMPeriod). Este valor se obtiene con la expresión:

$$f_{PWM} = \frac{f_{timer}}{(TIMPeriod + 1)} \tag{3.9}$$

Donde f_{PWM} es 200 Hz y f_{timer} es 9.6 MHz. Despejando TIMPeriod de la ecuación 3.9, se obtiene un valor de 48000.

De este modo, se está indicando al microcontrolador que el 100% de *duty* se corresponde con un valor en el contador del *timer* de 48000. Para generar señales de un porcentaje de *duty* determinado, habrá que cargar en el registro CCR1 (registro de comparación) un valor comprendido entre 0 y 48000 [46]. Esto se explicará en mayor detalle en el siguiente apartado (3.2.2.2).

Los valores calculados anteriormente para el Prescaler y el TIMPeriod sería necesario cargarlos en los registros correspondientes del TIM1, pero como se empleará un software específico para la configuración del microcontrolador (como se explicará más adelante en esta memoria) esta tarea resultará muy sencilla al disponer de una interfaz gráfica para realizar su configuración.

3.2.2.2.- Curva de *dimming*

La normativa DALI permite realizar la reguación del nivel de luz de las luminarias mediante el comando DAPC (Direct Arc Power Control). Cuando se emplea este comando, el maestro envía un valor entre 0 y 254, los cuales se corresponden con el 0 y el 100% de salida lumínica respectivamente.

No obstante, en lugar de hacer una variación lineal de la salida de luz, la norma especifica una expresión (3.10) para realizar un *dimming* logarítmico, pensado para compensar la curva de sensibilidad hacia la luz del ojo humano.

$$Salida\ lumínica\ (\%) = 10^{\frac{nivel\ enviado-1}{253/3}} - 1 \qquad (3.10)$$

Se puede comprobar que si, en 3.10, se sustituye nivel enviado por 0 y 254 respectivamente, se obtienen los valores de 0 y 100%. Sin embargo, esta expresión presenta un problema: cuando el maestro envía niveles de luz bajos, se obtienen valores flotantes menores que 1 que, al cargarlos en el registro de 16 bits del TIM1, se redondearían a 0, lo cual supone que no se produzca salida de luz (cuando si la debiera haber, aunque sea muy tenue).

Para solventar este problema, se ha implementado en el microcontrolador una tabla con en el resultado de sustituir en 3.10 los 255 valores de nivel posibles enviados por el maestro. El valor obtenido de la fórmula se ha multiplicado por 480, para disponer de valores enteros que cargar en el registro correspondiente del TIM1. De este modo, los valores estarán comprendidos entre 0 y 48000, los cuales se corresponden respectivamente con 0 y 100% de salida de luz. Por esta razón, se configuró el TIM1 de la forma en la que se hizo.

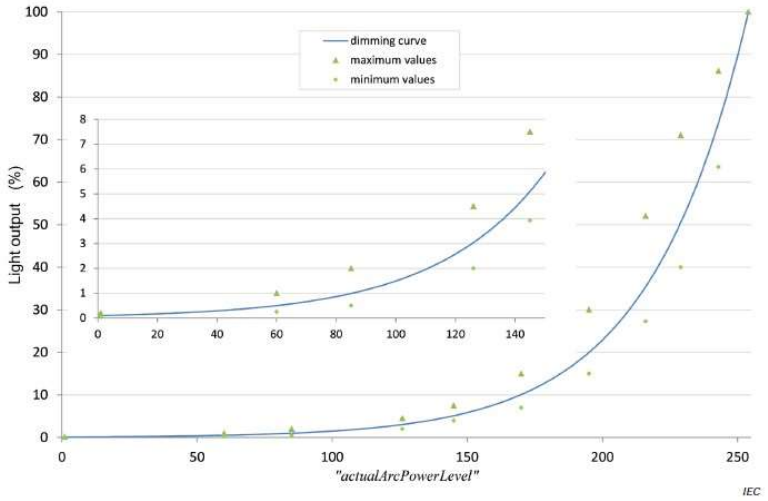


Figura 3.23.- Curva de dimming exponencial. Los puntos se obtienen aplicando la ecuación 3.10 [26].

3.2.2.3.- Configuración de la base de tiempos para las operaciones de la luminaria

Algunas acciones especificadas en la normativa DALI necesitan ejecutarse en un determinado tiempo. Por ejemplo:

- El proceso de inicialización del sistema debe durar 600 ms.
- Cuando se realiza un *fade* (regulación progresiva de la salida lumínica) este debe durar 200 ms como máximo si se emplea la tasa de *fade* (fadeRate) o el tiempo que indica la norma si se emplea el tiempo de *fade* (fadeTime) [26], el cual se expresará en función de la base de tiempos empleada.
- El tiempo de establecimiento entre tramas de doble envío (*send-twice*), tiempo que transcurre desde que se envía la trama por primera vez y el inicio del envío de la misma trama una segunda vez, está definido en 100 ms por la norma.

Por lo tanto, para poder realizar estas acciones (entre otras), es preciso contar con una base de tiempos destinada exclusivamente al procesado de comandos.

De ninguna forma debe usarse la base de tiempos empleada en el proceso de comunicaciones, ya que se solaparían las tareas de recepción/transmisión con las de la luminaria. Esto produciría la pérdida de información transmitida a través del bus.

De este modo, se empleará uno de los *timers* disponibles en el microcontrolador. El elegido, es el TIM2, el cual es un *timer* de propósito general basado en un contador de 32 bits y en un prescaler de 16 bits.

A la vista de las temporizaciones que hay que realizar (100, 200, 600 ms, …), se ha decidido configurar el TIM2 para que produzca una interrupción cada 20 ms (50 Hz). De esta forma se emplearán contadores del número de interrupciones producidas y, cuando hayan transcurrido las correspondientes a alguna de las operaciones que dependen de la base de tiempos, se ejecutará la acción asociada a ella.

Los valores característicos para configurar el *timer* se obtienen de las mismas ecuaciones empleadas para configurar el TIM1 (3.8 y 3.9).

Como se desea realizar una temporización de una frecuencia bastante alta (50 Hz), es preciso emplear un prescaler. El valor elegido es de 19. De esta forma, el valor de TIMPeriod para el TIM2 será de 47999.

Cuando se esté ejecutando una operación de recepción o transmisión, es recomendable desactivar momentáneamente las interrupciones del TIM2 hasta que se hayan completado dichas operaciones, para asegurarse que no se interfiera en el proceso comunicativo.

Por último, cabe destacar que todas las funcionalidades implementadas en el microcontrolador (comunicaciones y manejo de la lámpara) se controlarán desde el bucle de ejecución continua, while (1), del *main*. El siguiente flujograma ilustra este proceso:

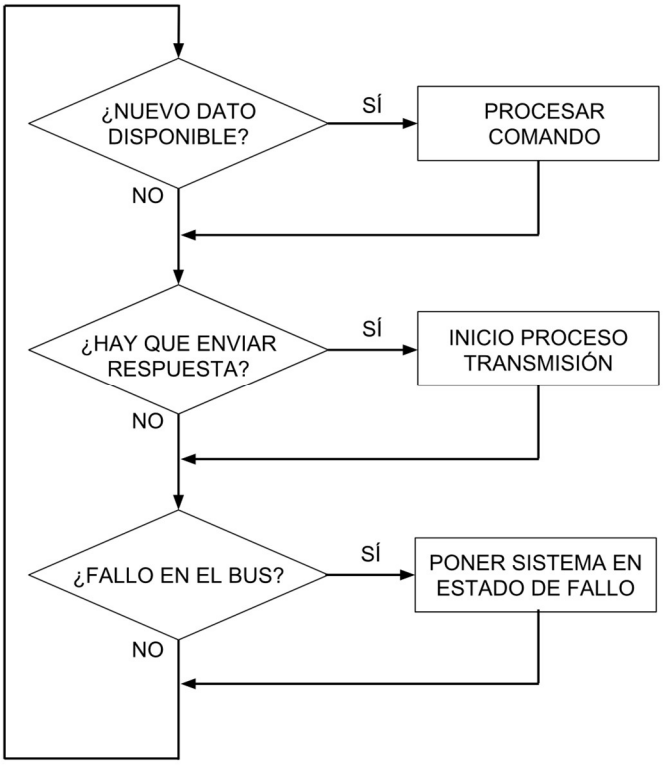


Figura 3.24.- Flujograma del bucle infinito implementado en el main.

3.3.- SOFTWARE EMPLEADO EN LA IMPLEMENTACIÓN DEL CÓDIGO FUENTE

Para la programación del protocolo DALI y de las acciones de control sobre la luminaria en el microcontrolador se han empleado dos programas.

En primer lugar, se ha empleado el programa STM32CubeMX, desarrollado por el propio fabricante del microcontrolador, STMicroelectronics [47].

Este programa permite realizar la configuración de los diferentes pines, relojes y periféricos de todos los microcontroladores desarrollados por STMicroelectronics a través de una interfaz gráfica y muy intuitiva (Figura 3.26).

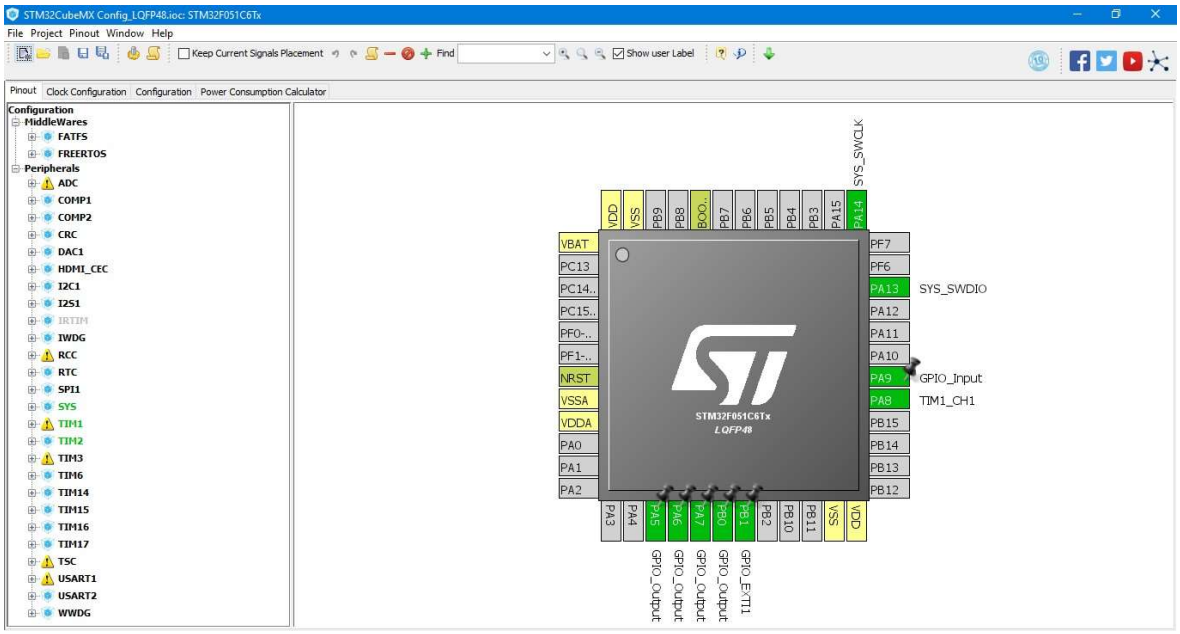


Figura 3.25.- Interfaz de usuario del programa STMCubeMX.

Una vez realizada la configuración de todos los elementos del microcontrolador que se van a usar, el programa genera un archivo de código (.c) que incluye las instrucciones necesarias para la configuración de todos los registros del microcontrolador, ahorrando esta tarea al usuario. Además, adjunta los ficheros de las diferentes librerías HAL [44].

Dicho archivo se puede abrir con el siguiente programa, el cual se emplea para programar todo el código que se va a implementar en el microcontrolador.

Para esta última labor, será necesario un programa a modo de entorno de desarrollo (IDE). El software elegido para esta tarea es el Keil µVision5, desarollado por ARM [48]. Este programa ha sido diseñado para la programación de microcontroladores basados en una arquitectura ARM Cortex-M.

Keil µVision5 permite la programación, compilación, depuración (*debug*) y transferencia al microcontrolador de nuestro programa (grabándolo en su memoria flash). Además, este entorno de desarrollo es compatible con ANSI-C y permite incluir las librerías de C estándar para sistemas embebidos. Otra de sus ventajas es que permite el manejo de los drivers HAL desarrollados por STMicroelectronics.

Todas estas características hacen que la labor de programación del microcontrolador relativamente sencilla.

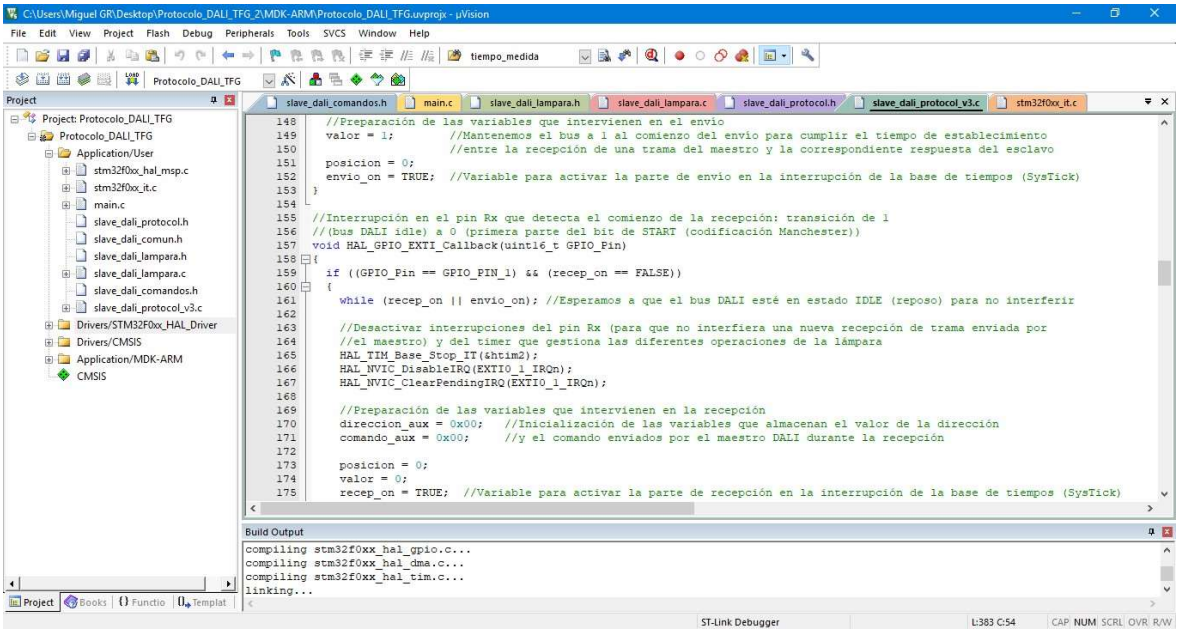


Figura 3.26.- Interfaz de usuario de Keil µVision5.

Ambos programas son de licencia libre, pudiéndose descargar en las páginas web de sus respectivos fabricantes.

NOTA: en el Anexo II se adjunta el código fuente desarrollado para este proyecto, por si hubiera alguna duda con lo descrito acerca del diseño software.

4.- Resultados experimentales

4.1.- PLACA DE DESARROLLO BASADA EN STM32F0DISCOVERY

A la hora de comenzar con el desarrollo de este proyecto, no se contaba con experiencia en la programación de microcontroladores de STMicroelectronics y, aunque en algunos aspectos se asemeja a dispositivos de otros fabricantes (como Microchip, por ejemplo), cuentan con algunas características propias y funciones propias que es necesario conocer.

Para familiarizarse con su programación y uso de los distintos periféricos que contiene (*timers*, módulo PWM, conversor A/D, etc.), se ha diseñado una placa de desarrollo (Figura 4.1) que incluye diferentes elementos hardware para facilitar el aprendizaje. Por ejemplo, se cuenta con varios pulsadores, *microswitches*, LEDs indicadores, potenciómetros, posibilidad de conectar un display LCD, sensores de temperatura resistivos (tipo NTC o Pt100) y, hasta cinco, de los *drivers* LED diseñados.

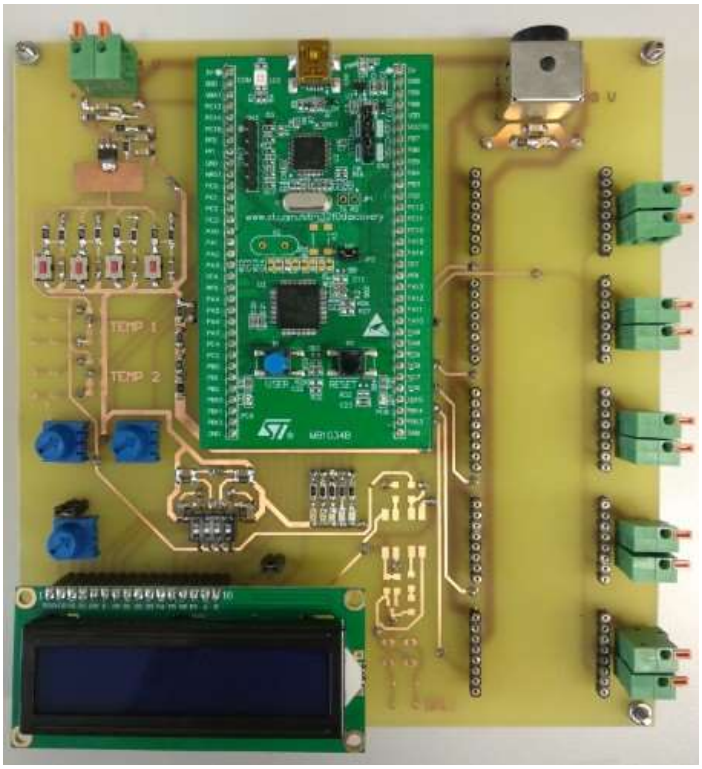


Figura 4.1.- Placa de desarrollo para la STM32F0Discovery.

También se dispone de un zócalo para conectar una placa STM32F0Discovery, la cual incluye el microcontrolador. Cuenta con un microcontrolador STM32F051R8T6, el cual es idéntico en términos funcionales y de programación al elegido para el proyecto, el STM32F051C6T6. La única diferencia entre ellos es que el primero presenta un encapsulado LQFP64 (64 pines) mientras que el encapsulado del segundo es un LQFP48, lo que se traduce en que el primero cuenta con un número mayor de puertos GPIO.

4.2.- PRUEBAS DE LOS DRIVERS DE LED

Antes de llegar a un diseño óptimo de los *drivers* de LED empleados, se tuvo que lidiar con numerosos contratiempos, que resultaron en varios rediseños de este.

Como se trata de un elemento de potencia, antes de integrarlo en el sistema completo se realizaron pruebas individuales para comprobar su funcionamiento.

En primer lugar, se pensó en un diseño basado en cinco *drivers*, asociando cuatro tiras de LED (dos tiras en serie que, a su vez, se colocaban en paralelo con otras dos tiras en serie). De este modo, habría que configurar el integrado ZXLD1362 para que entregase a su salida 1 A (el máximo de corriente que el fabricante asegura), distribuyéndose en 500 mA por cada rama en paralelo (valor admisible por las tiras de LED).

Para ello, era necesario una resistencia de sensado de 0.1 Ω (conseguida asociando diez resistencias de 1 Ω en paralelo). Sin embargo, al poner el *driver* en funcionamiento al máximo de corriente (todavía sin control PWM), el circuito integrado no era capaz de disipar la potencia (en forma de calor) producida en su interior y se destruía al cabo de apenas un minuto de tiempo.

A la vista de estos resultados catastróficos, se pensó el diseño actual del sistema: configurar el circuito integrado ZXLD1362 para que genere una corriente de salida de 500 mA. lo cual supuso un aumento del número de *drivers* empleados (de cinco a diez), aunque el número de tiras de LED se mantenía igual. De esta forma, se asegura la integridad del circuito integrado, trabajando en una zona bastante cómoda para él.

Una vez determinada la configuración del integrado, y comprobar que así soporta la disipación térmica, se procedió a probar la regulación de la salida lumínica mediante una señal PWM.

Para realizar estas pruebas, se generó una señal cuadrada de 3.3 Vpp y 100 Hz con un generador de funciones. Al introducir la señal PWM en la patilla ADJ a través del transistor bipolar, se pudo observar que al principio se producía la regulación de luz de la forma que se esperaba, pero al cabo de unos minutos en funcionamiento en *dimming*, los LEDs empezaban a parpadear y, finalmente, el circuito integrado terminaba explotando.

Ante este resultado, se plantearon varias alternativas que afectaban a la selección de componentes empleados. En primer lugar, se pensó que el diodo Schottky empleado no era lo suficientemente rápido, por lo que se sustituyó por otro con un tiempo de recuperación menor. Tras hacer este cambio, seguían obteniéndose los mismos resultados destructivos.

En este punto existía una gran incertidumbre con lo que estaba pasando, ya que no se apreciaba nada extraño en el diseño del *driver* que pudiera producir dichos resultados.

Finalmente, observando el modo de funcionamiento del circuito de control montado sobre la patilla ADJ del ZXLD1362, se pensó que al estar haciendo conmutar el transistor bipolar, se podría generar algún tipo de ruido o señal molesta que afectara negativamente a la electrónica interna del integrado. Por ello, se probó a colocar un condensador a modo de filtro entre el pin ADJ y masa, en paralelo al transistor bipolar.

Con sólo 100 pF de capacidad, se consiguió anular dicho efecto, haciendo funcionar el *driver* durante un periodo indefinido de tiempo sin que el integrado terminara por destruirse.

Con estos cambios, se validó el diseño y la fiabilidad del *driver*, comentado en más detalle en el apartado 3 de la memoria.

4.3.- PRUEBAS DEL SISTEMA COMPLETO

Una vez probado el funcionamiento de la etapa de potencia de nuestro sistema, se procederá a validar la implementación del código desarrollado para el microcontrolador, comprobando que las operaciones de recepción y transmisión se llevan a cabo sin pérdidas o alteración de la información en el bus, y que el procesado de comandos se realiza de forma correcta, observando su efecto sobre la luminaria.

Las pruebas se han realizado empleando un solo *driver* puesto que el comportamiento de los otros nueve será exactamente idéntico, ahorrándose componentes y materiales antes de dar por válido el funcionamiento del sistema completo. También serán necesarios la fuente de alimentación de 3.3 V, el *transceiver* y el circuito de test de presencia de lámpara.

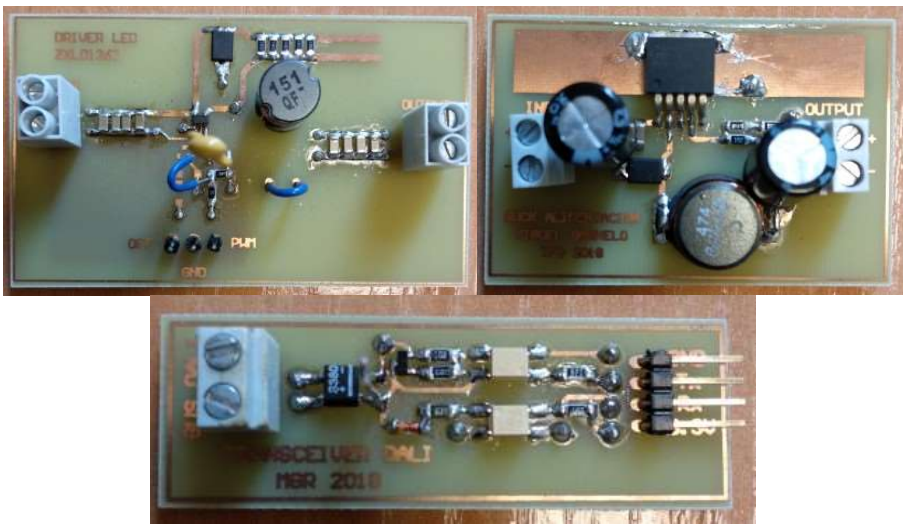


Figura 4.2.- Ejemplos del driver LED (arriba izda), la fuente de alimentación de 3.3 V (arriba dcha) y del transceiver DALI (abajo) diseñados y empleados en el proceso experimental.

Para realizar estas pruebas se ha empleado como fuente de alimentación del bus DALI el modelo DALI_Gateway TW del fabricante JUNG.

Como maestro se ha utilizado el dispositivo DALI-USB de Tridonic, que sirve como módulo de interfaz entre un ordenador y un bus DALI a través de un puerto USB. Contar con un elemento certificado como dispositivo DALI nos permitirá comprobar la validez del

sistema diseñado, ya que solamente envía y recibe información que cumpla con las especificaciones de la norma.

El propio Tridonic facilita un software para el envío de comandos a través del bus: el programa masterCONFIGURATOR, el cual es de descarga gratuita y permite realizar operaciones de direccionamiento, programación y configuración de dispositivos DALI.

Figura 4.3.- Fuente de alimentación del bus DALI (izda.) y DALI-USB (dcha).

Una vez reunidos todos los elementos necesarios, se procede a realizar el montaje del conjunto para poder comenzar con la fase de testeo. Dicho montaje se puede observar en la Figura 4.4:

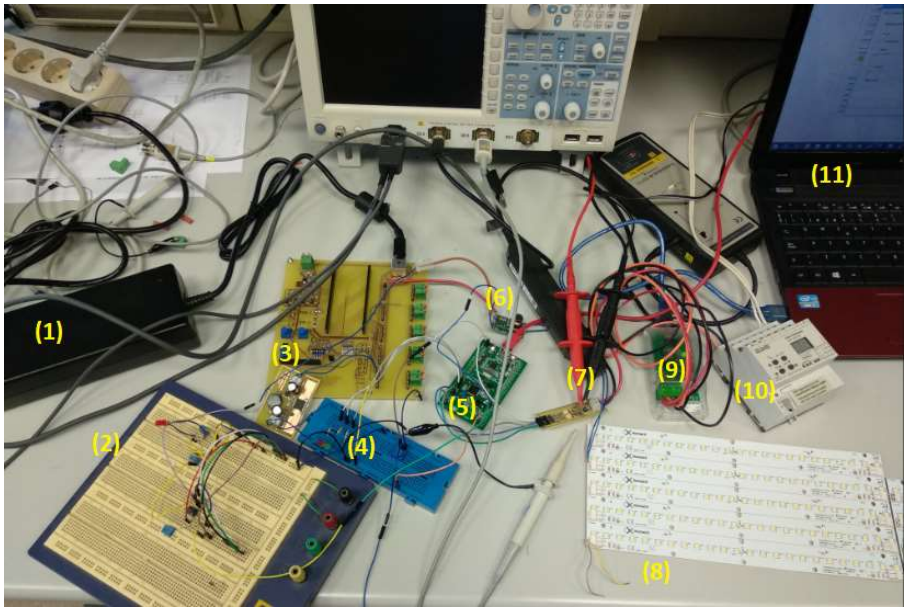


Figura 4.4.- Montaje de la fase experimental del sistema.

En donde tenemos:

- **(1)** Fuente de alimentación de 48 V.
- **(2)** Circuito que comprueba si hay lámpara conectada (montado sobre *protoboard*).
- **(3)** Fuente de alimentación de 3.3 V.
- **(4)** *Protoboard* con LEDs indicadores (Rx, Tx y fallo).
- **(5)** Placa STMF0Discovery sobre cuyo microcontrolador se carga el código a probar.
- **(6)** *Driver* LED.
- **(7)** *Transceiver* DALI.
- **(8)** Tiras de LED.
- **(9)** DALI-USB.
- **(10)** Fuente de alimentación del bus DALI.
- **(11)** Ordenador que contiene el software masterCONFIGURATOR.

Una vez realizado el montaje, hubo que solventar algunos fallos en el código, relativos al proceso de recepción/transmisión y de la generación de la señal PWM para la regulación de la intensidad lumínica de la luminaria.

Una vez solucionados, se probó de nuevo la implementación en el prototipo, determinando que este cumple con todas las especificaciones del protocolo DALI, ya que, tras procesar las diferentes peticiones enviadas por el maestro, el sistema responde tal y como indica la normativa para cada comando. Algunos de estos resultados se muestran a continuación.

4.3.1.- Comprobación lógica inversa en el transceiver

Se adjunta una imagen (Figura 4.5) que muestra como, enviando una trama cualquiera, el nivel de los datos que llegan al pin Rx del microcontrolador (salida del *transceiver*) es complementario al nivel de los datos a la entrada, correspondiente a la conexión con el bus DALI. También ocurre lo mismo con el nivel de los datos transmitidos por el pin Tx.

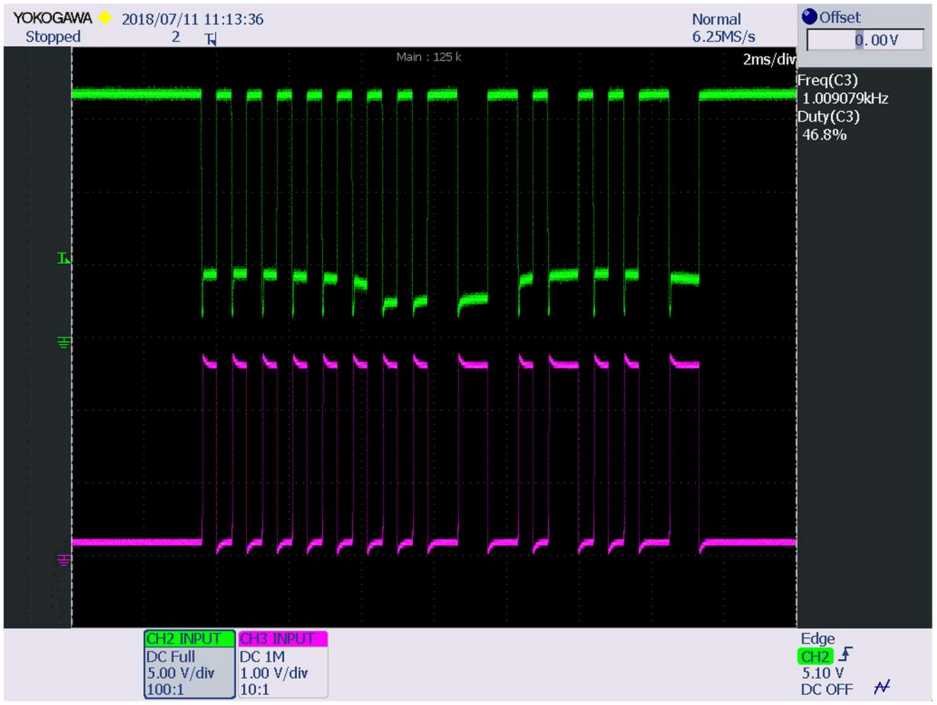


Figura 4.5.- Trama en el bus DALI (verde) vs. trama que llega al pin Rx del microcontrolador (morado).

4.3.2.- *Dimming de la luminaria.*

El objetivo de esta prueba es comprobar que el sistema diseñado atiende a las peticiones del maestro de modificar el nivel de luminosidad de la luminaria (*dimming*).

En la Figura 4.6 se puede observar la interfaz de usuario del masterCONFIGURATOR con la configuración del comando DAPC (Direct Arc Power Control), el cual requiere al microcontrolador que regule la salida de luz en un valor entre 0 y 254 (0 – 100%). En este caso, se envía el valor 229 (0xE5 en hexadecimal), que corresponde con un porcentaje del 51%.

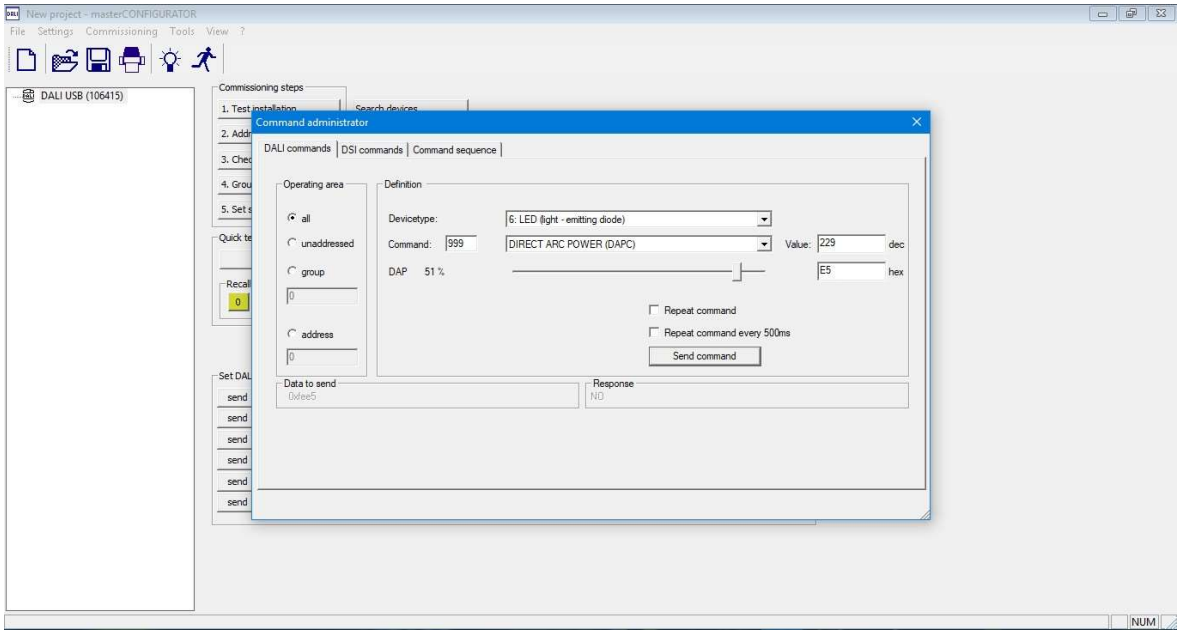


Figura 4.6.- Captura la interfaz gráfica del masterCONFIGURATOR al enviar el comando DAPC para regular la salida de luz en un 51%.

Una vez el microcontrolador recibe este comando, deberá procesarlo generando una señal PWM con el ciclo de trabajo indicado. Esta señal llegará al *driver* LED, el cual se encargará de regular la corriente que circula por las tiras de LED para brillen al nivel indicado por el maestro.

Se adjunta una captura del osciloscopio en el cual se muestran las señales que intervienen en este proceso (Figura 4.7). En ella se puede apreciar que, una vez termina la recepción de la trama, el microcontrolador comienza a generar la señal PWM, observando en la parte superior derecha de la imagen el valor de la frecuencia configurada (200 Hz) y el *duty* necesario para atender la demanda del maestro (49% = 100% – 51%, recordando que el pin de ajuste del ZXLD1362 es de lógica inversa). Una vez generada dicha señal, se aprecia la variación de la corriente por los LED.

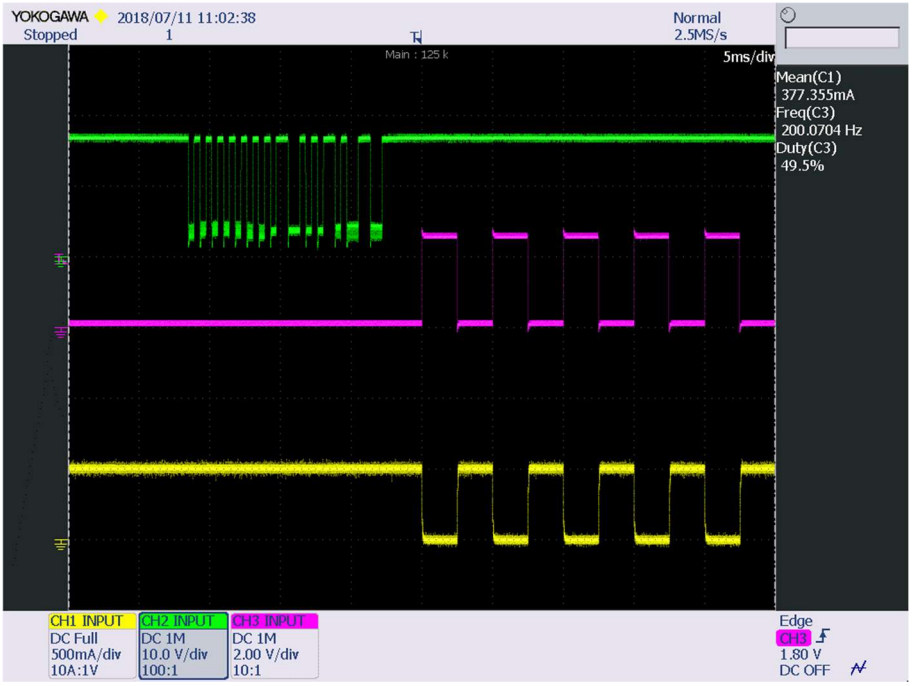


Figura 4.7.- Trama enviada por el maestro (verde), señal PWM generada por el maestro (morado) y corriente circulando por las tiras de LED (amarillo).

En la Figura 4.8, se muestra un *zoom* de la trama enviada por el maestro. En ella se puede comprobar como la codificación Manchester de la trama se corresponde con el identificador hexadecimal del comando DAPC (0xFE) y el nivel deseado a fijar (0xE5).

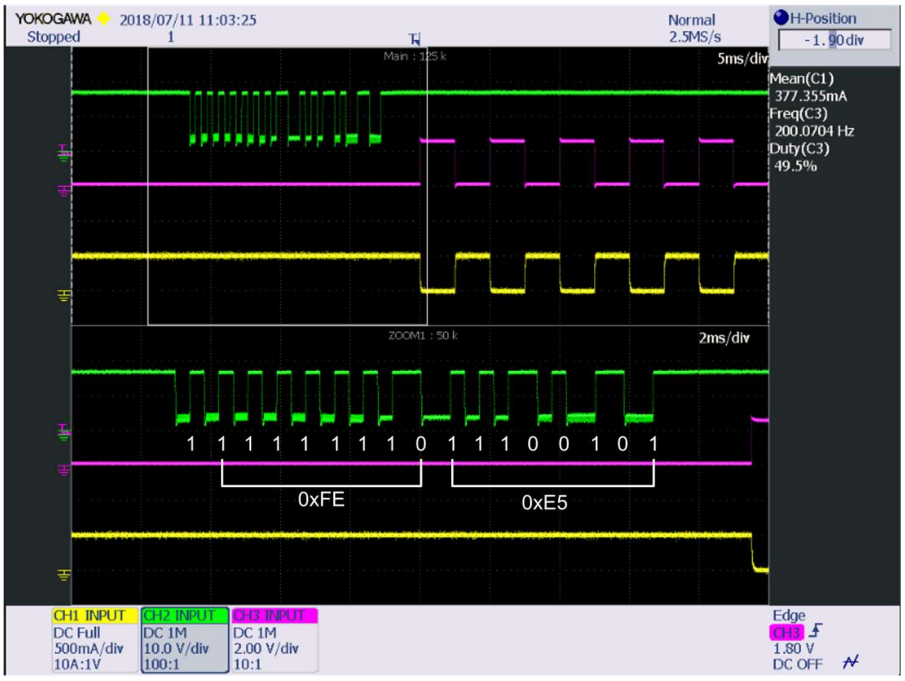


Figura 4.8.- Zoom trama enviada por el maestro.

4.3.3.- Envío de la respuesta a una consulta realizada por el maestro

Una vez fijado el nivel del apartado anterior, se consulta al esclavo que nivel de luz tiene la luminaria actualmente.

Esto se puede realizar enviando el comando QUERY_ACTUAL_LEVEL. En la propia interfaz gráfica del masterCONFIGURATOR se puede comprobar la respuesta enviada por el sistema (Figura 4.9).

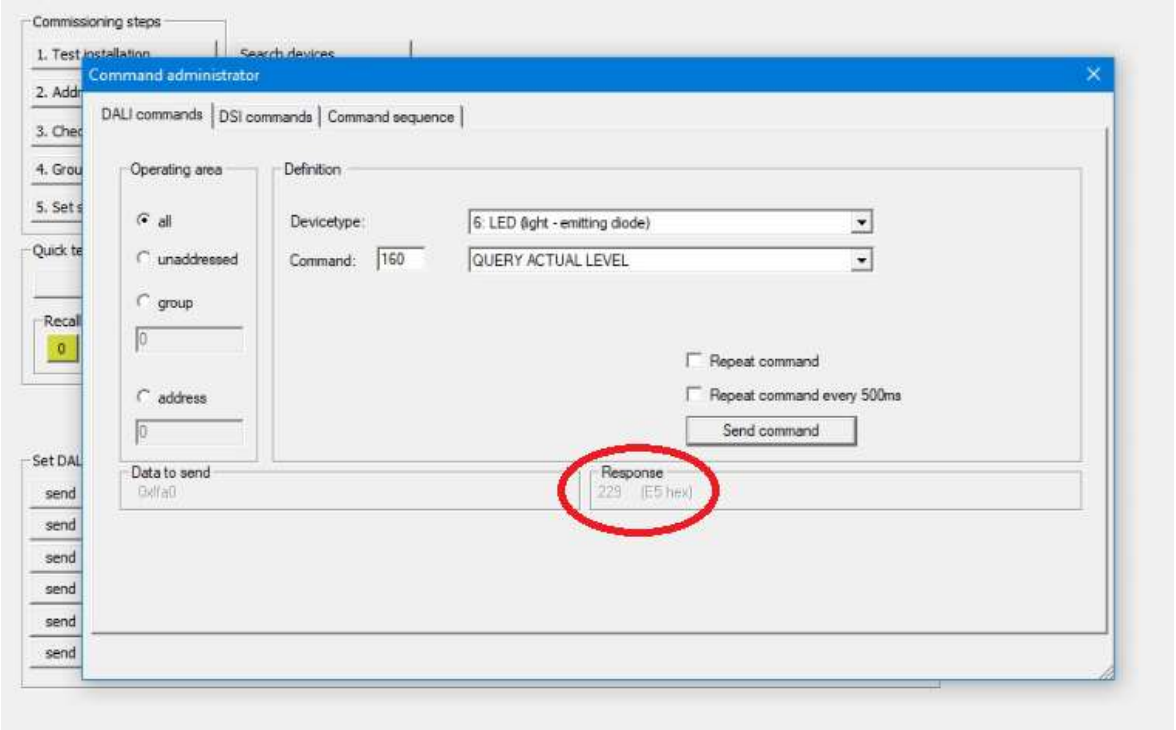


Figura 4.9.- Captura de la interfaz de usuario del masterCONFIGURATOR con el valor respondido por el esclavo.

El esclavo responde que tiene un nivel actual de 229 (0xE5). Al coincidir este valor con el fijado por el maestro con el comando anterior, se demuestra que el proceso de transmisión funciona de forma satisfactoria.

En la Figura 4.10 se muestra una captura del osciloscopio donde se representan las tramas correspondientes a la consulta realizada por el maestro y a la respuesta enviada por el esclavo. Se hace un zoom sobre las mismas para ver las tramas con más detalle. Esta imagen corrobora el envío correcto de la respuesta

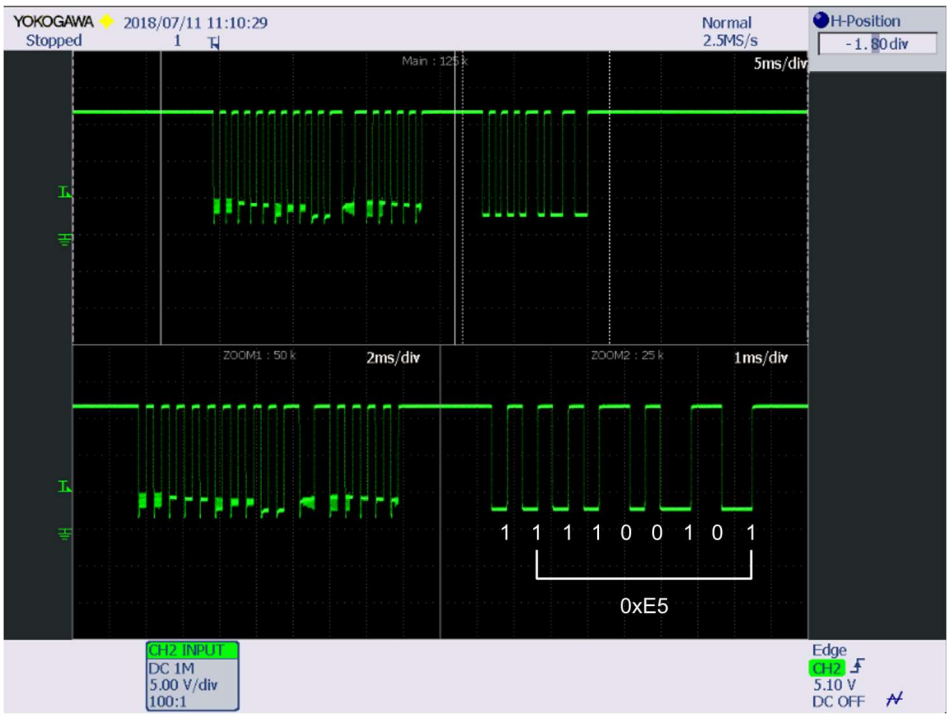


Figura 4.10.- Consulta de nivel actual de la luminaria realizada por el maestro (izquierda) y respuesta con dicho valor por parte del esclavo (derecha).

4.3.4.- Pruebas con otro dispositivo esclavo

Una vez se ha comprobado el correcto funcionamiento del sistema diseñado de forma individual, es interesante ver cómo se comporta integrado en una red DALI en la que hay más elementos conectados al bus.

Para observar este fenómeno, se ha empleado el *driver* LED comercial OPTOTRONIC INTELIGENT OTi DALI 25/220 700 LT2 de OSRAM (Figura 4.11), el cual está certificado como dispositivo que cumple con la normativa DALI.

Conectando al mismo bus el *driver* diseñado para este proyecto y el *driver* de OSRAM, se comprueba que ambos dispositivos responden a las órdenes enviadas a través del dispositivo DALI-USB usado como maestro, demostrando así que el sistema que se ha desarrollado no interfiere con dispositivos comerciales y que es perfectamente integrable en una instalación controlada por protocolo DALI.

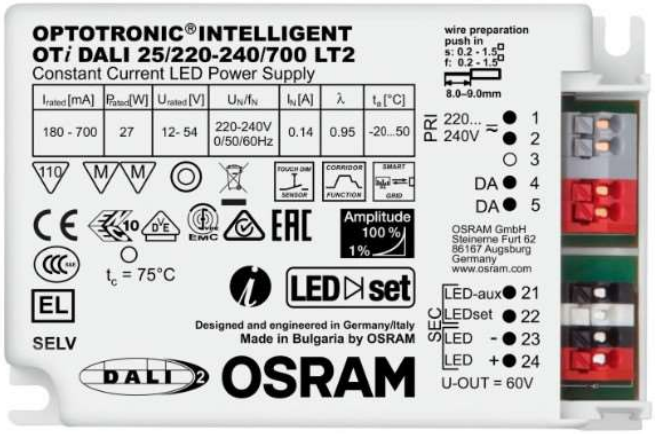


Figura 4.11.- Driver OPTOTRONIC INTELIGENT OTi DALI 25/220 700 LT2.

Una vez realizada la fase experimental y comprobado el correcto funcionamiento del sistema en conjunto, se realizó una placa que integra los diez *drivers* LED (Figuras 4.20 y 4.21). De este modo, se consigue cumplir con el objetivo inicial de diseñar un *driver* de LED de 200 W de potencia controlado mediante protocolo DALI.

A continuación, se muestra una imagen (Figura 4.12) de cómo sería la placa de circuito impreso del producto final (solamente se ha soldado un *driver* para probar su funcionamiento).

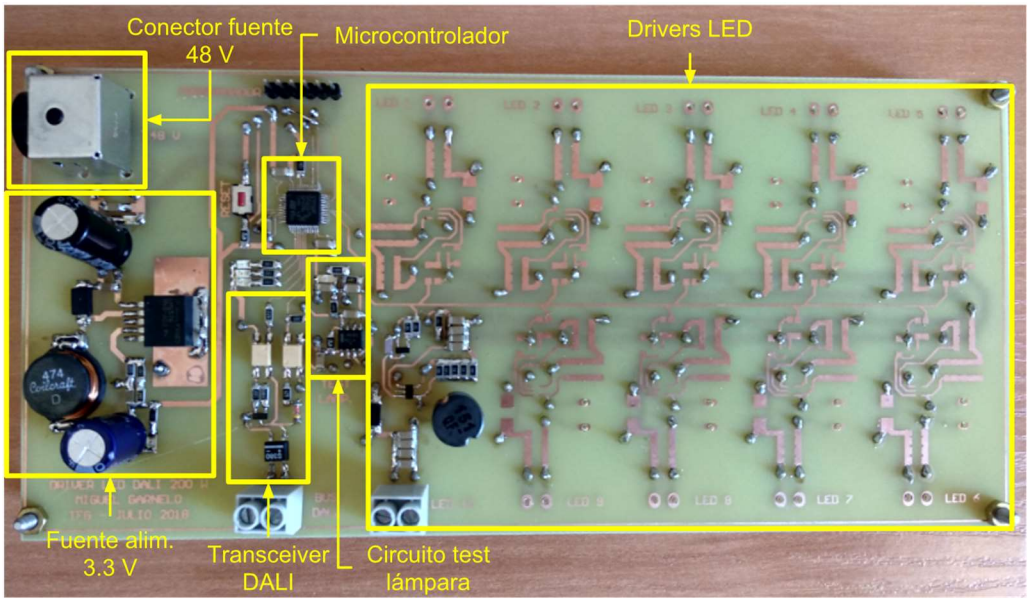


Figura 4.12.- Placa de circuito impreso del sistema completo.

5.- Conclusiones y trabajos futuros

5.1.- CONCLUSIONES

Diseñar un *driver* LED de 200 W no es una tarea sencilla por sí sola. Si a eso se añade que deba ser controlado por un protocolo normalizado y específico de iluminación, el reto que ello supone es todavía mayor.

A la vista de los resultados tan favorables que se han obtenido en el procedimiento experimental del prototipo, se puede concluir que los objetivos propuestos al comienzo del proyecto se han alcanzado de forma satisfactoria, desarrollando un sistema de iluminación que funciona correctamente y, además, es capaz de integrarse en una red DALI en sintonía con otros dispositivos.

Por otra parte, el desarrollo del proceso ha supuesto una mejora personal y profesional, no sólo por los conocimientos teóricos adquiridos durante el desarrollo del proyecto, sino también por la capacidad de tomar decisiones a la hora de acometer un diseño y de lidiar con los numerosos problemas surgidos, siendo capaz de solucionarlos consiguiendo grandes resultados.

5.2.- TRABAJOS FUTUROS

Aunque el sistema diseñado funciona correctamente, no deja de ser un prototipo y, si algún día se pudiera convertir en un producto comercial, se podrían hacer algunas mejoras.

En primer lugar, se podría estudiar el emplear otro circuito integrado para los *drivers* de las tiras LED, que sea capaz de trabajar con una corriente de mayor valor sin fallar o destruirse. De esta manera, se podría disminuir el número empleado de estos *drivers*, traduciéndose, a su vez, en una reducción de la placa de circuito impreso diseñada.

Otra mejora que se podría realizar sería el diseño de un convertidor electrónico AC-DC (inversor) para generar la tensión de alimentación de 48 V desde la red eléctrica. Esto se debe a que la fuente de alimentación usada para el proyecto es demasiado pesada y voluminosa, lo cual podría suponer un problema si la luminaria se llegase a empotrar en un techo, por ejemplo.

6.- Bibliografía y referencias

[1] Red Eléctrica, “Preliminary report 2017.”

[2] D. Gacio Vaquero, “PhD Thesis - OFF-LINE SUPPLY OF SOLID-STATE LAMPS. LAMP MODELLING, APPLICATION OF THE INTEGRATED BUCK-FLYBACK CONVERTER, AND PROPOSAL OF A NEW OPTIMISED DIMMING SCHEME,” 2013.

[3] P. J. Quintana Barcia, “Phd Thesis - POWER ELECTRONIC SUPPLIES FOR PUBLIC LIGHTING SYSTEMS WITH DISTRIBUTED GENERATION CAPABILITY: SOLUTION PROPOSALS FOR POWER AND CONTROL STAGES, CHARACTERIZATION AND MINIMIZATION OF THE IMPACT IN GRID QUALITY,” 2015.

[4] B. Stewart, “Incandescence —,” no. August, pp. 1201–1204, 1934.

[5] M. F. Da Silva *et al.*, “Analysis and design of a single-stage high-power-factor dimmable electronic ballast for electrodeless fluorescent lamp,” *IEEE Trans. Ind. Electron.*, vol. 60, no. 8, pp. 3081–3091, 2013.

[6] C. Brañas, F. J. Azcondo, and S. Bracho, “Evaluation of an electronic ballast circuit for HID lamps with passive power factor correction,” *Lamp*, pp. 371–376, 2002.

[7] E. L. C. et Al., “A Novel Low Cost Two-Stage Electronic Ballast for 250W High Pressure Mercury Vapor Lamps Based on Current-Mode-Controlled Buck-Boost Inverter,” 2001.

[8] M. Rico-Secades *et al.*, “Complete low-cost two-stage electronic ballast for 70-W high-pressure sodium vapor lamp based on current-mode-controlled buck-boost inverter,” *IEEE Trans. Ind. Appl.*, vol. 41, no. 3, pp. 728–734, 2005.

[9] M. A. D. Costa, J. M. Alonso, J. Cardesín, and J. Ribas, “Analysis , Design and Experimentation of a Closed- Loop Metal Halide Lamp Electronic Ballast,” vol. 00, no. c, pp. 1384–1390, 2006.

[10] M. A. Dalla Costa, A. L. Kirsten, J. M. Alonso, J. García, and D. Gacio, "Analysis, design, and experimentation of a closed-loop metal halide lamp electronic ballast," *IEEE Trans. Ind. Appl.*, vol. 48, no. 1, pp. 28–36, 2012.

[11] E. F. Schubert, *Light-Emitting Diodes*. 2006.

[12] Nichia Corporation, "NFSL757DT-V1 (R9050)," vol. 1, no. 131226.

[13] K. Górecki, "Modelling power LEDs in the COB case with thermal phenomena taken into account," no. September, pp. 1–6, 2017.

[14] Nichia Corporation, "Light Emitting Diode(LED)." [Online]. Available: http://www.nichia.co.jp/en/product/led.html. [Accessed: 12-Jul-2018].

[15] K. H. Loo, W. K. Lun, S. C. Tan, Y. M. Lai, and C. K. Tse, "On driving techniques for LEDs: Toward a generalized methodology," *IEEE Trans. Power Electron.*, vol. 24, no. 12, pp. 2967–2976, 2009.

[16] A. Gulati, "Modulation Techniques for LED Dimming," no. 001, pp. 1–10, 2008.

[17] A. Lagunov, L. Morozova, D. Fedin, N. Podorojnyak, and V. Terehin, "Polychromatic LED device for measuring CFFF," *Proc. 2015 IEEE 8th Int. Conf. Intell. Data Acquis. Adv. Comput. Syst. Technol. Appl. IDAACS 2015*, vol. 2, no. September, pp. 682–687, 2015.

[18] Cypress, "PrISM™ Technology for LED Dimming – AN47372," no. 001, pp. 1–6, 2011.

[19] A. Wilkins, J. Veitch, and B. Lehman, "LED lighting flicker and potential health concerns: IEEE standard PAR1789 update," *2010 IEEE Energy Convers. Congr. Expo. ECCE 2010 - Proc.*, pp. 171–178, 2010.

[20] K. H. Loo, Y. M. Lai, S. C. Tan, and C. K. Tse, "On the color stability of phosphor-converted white LEDs under DC, PWM, and bilevel drive," *IEEE Trans. Power Electron.*, vol. 27, no. 2, pp. 974–984, 2012.

[21] J. Garcia, M. A. Dalla-Costa, J. Cardesin, J. M. Alonso, and M. Rico-Secades, "Dimming of high-brightness LEDs by means of luminous flux thermal estimation," *IEEE Trans. Power Electron.*, vol. 24, no. 4, pp. 1107–1114, 2009.

[22] P. M. Pérez, "Creación de un Master DALI. Control de una/varias luminarias basado en un microcontrolador Cortex M4," 2016.

[23] S. Varghese, "A study of Communication Protocols and Wireless Networking Systems for Lighting Control Application," vol. 5, pp. 22–25, 2015.

[24] "KNX Association - KNX Association [Official website]." [Online]. Available: https://www2.knx.org/knx-es/index.php.

[25] IEC, "Digital addressable lighting interface - Part 101: General requirements - System components (IEC 62386-101:2014)," 2015.

[26] IEC, "Digital addressable lighting interface - Part 102: General requirements - Control gear (IEC 62386-102:2014)," 2015.

[27] IEC, "Digital addressable lighting interface - Part 103: General requirements - Control devices (IEC 62386-103:2014)," 2015.

[28] Zhaga Consortium, "Zhaga Interface Specification Book 1 Edition 1.5 February 2014," no. February, 2014.

[29] Zhaga Consortium, "Zhaga Interface Specification Book 7 Edition 1.2 March 2014," no. March, 2014.

[30] M. Well, "GS220 Series," pp. 5–6, 2015.

[31] Diodes Incorporated, "ZXLD1362," no. October, pp. 1–10, 2014.

[32] Panasonic, "High Power Inductors Types: 06D, 08D, 10D, 16B, 18B, 10E, 12E, 15E, 18E," pp. 34–41.

[33] Diodes Incorporated, "B220/A - B260/A Shottky Barrier Rectifier," no. July, pp. 1–5,

2015.

[34] N. X. P. Semiconductors, “BC847 series,” no. September, pp. 1–17, 2014.

[35] Texas Instruments, “LM2576xx Series SIMPLE SWITCHER ® 3-A Step-Down Voltage Regulator,” no. 1, 2016.

[36] Coilcraft, “SMT Power Inductors – DO5010H Series,” pp. 7–8, 2009.

[37] Diodes Incorporated, “PDU620,” pp. 4–7, 2003.

[38] Texas Instruments, “LMx93-N , LM2903-N Low-Power , Low-Offset Voltage , Dual Comparators,” no. 1, pp. 1–20, 2014.

[39] Microchip, “DALI AN1465,” no. Figure 1, pp. 1–14, 2012.

[40] P. Visconti, A. Lay-Ekuakille, P. Primiceri, and G. Cavalera, “Wireless smart system for monitoring and driving of household electrical facilities remotely controlled by Internet,” *EEEIC 2016 - Int. Conf. Environ. Electr. Eng.*, pp. 0–5, 2016.

[41] D. Semiconductor, “S380 Bridge Rectifier,” pp. 1–2, 2003.

[42] Toshiba, “TLP181,” pp. 1–9, 2017.

[43] STMicroelectronics, “STM32F051x4 STM32F051x6 STM32F051x8,” no. July, 2012.

[44] STMicroelectronics, “Description of STM32F0xx HAL and Low-layer drivers - User Manual,” no. November, 2016.

[45] Motorola, “Digitally Addressable Lighting Interface (DALI) Unit Using the MC68HC908KX8,” 2002.

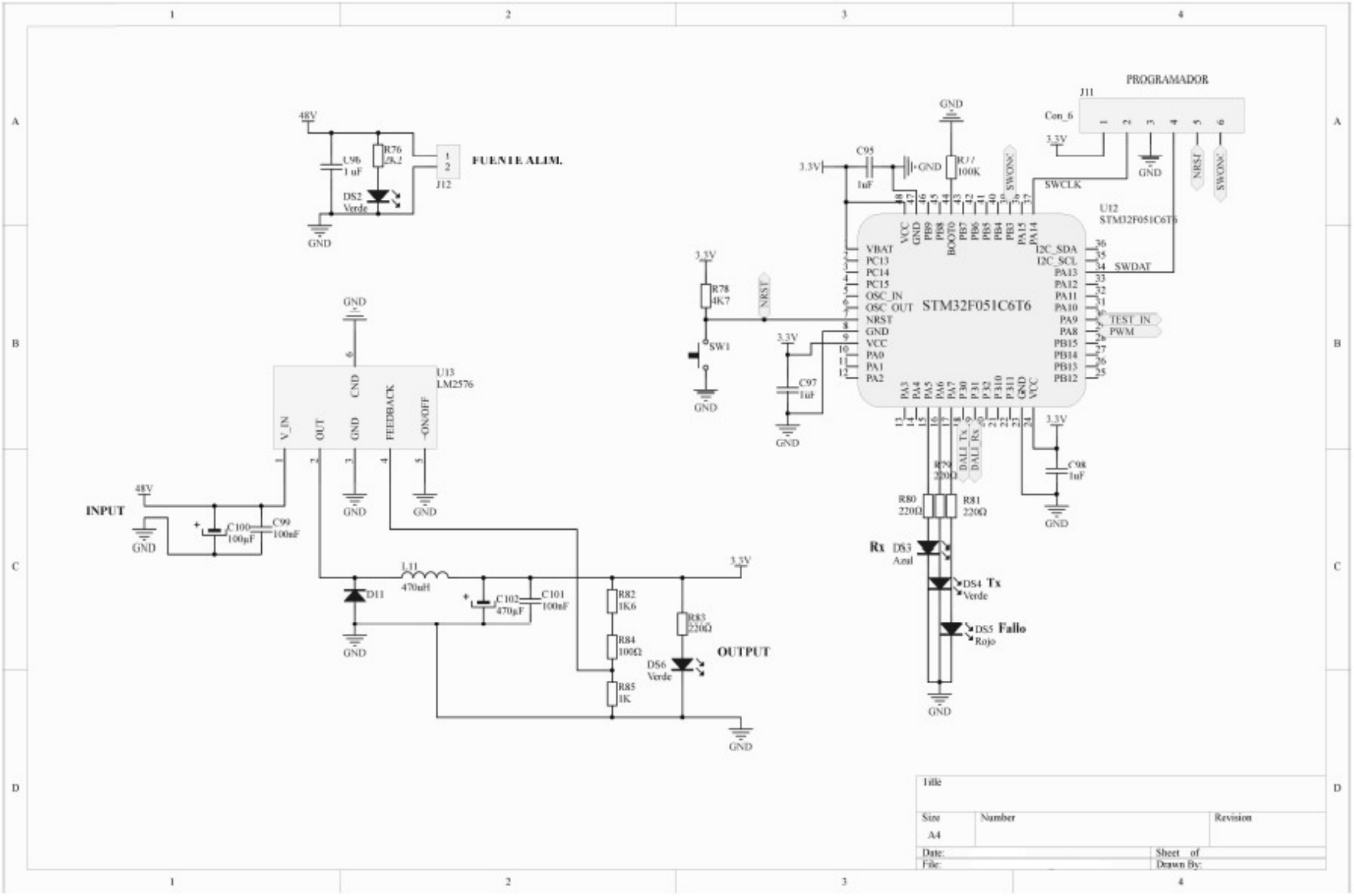
[46] STMicroelectronics, “STM32F0x1/STM32F0x2/STM32F0x8 advanced ARM®-based 32-bit MCUs - Reference Manual,” *Yade Doc. 2nd ed*, no. January 2017, 2015.

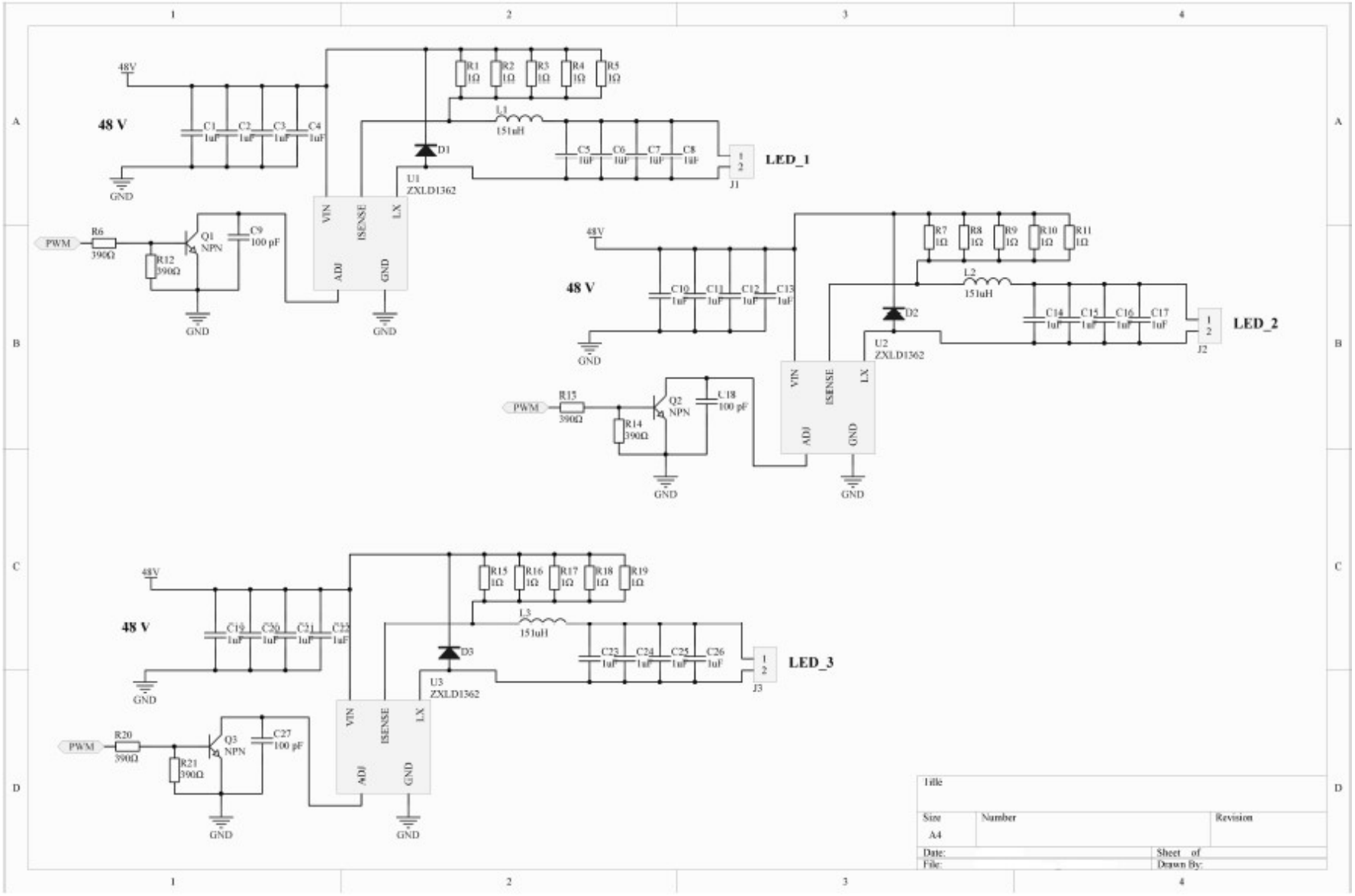
[47] “STM32CubeMX - STM32Cube initialization code generator - STMicroelectronics.” [Online]. Available: https://www.st.com/en/development-

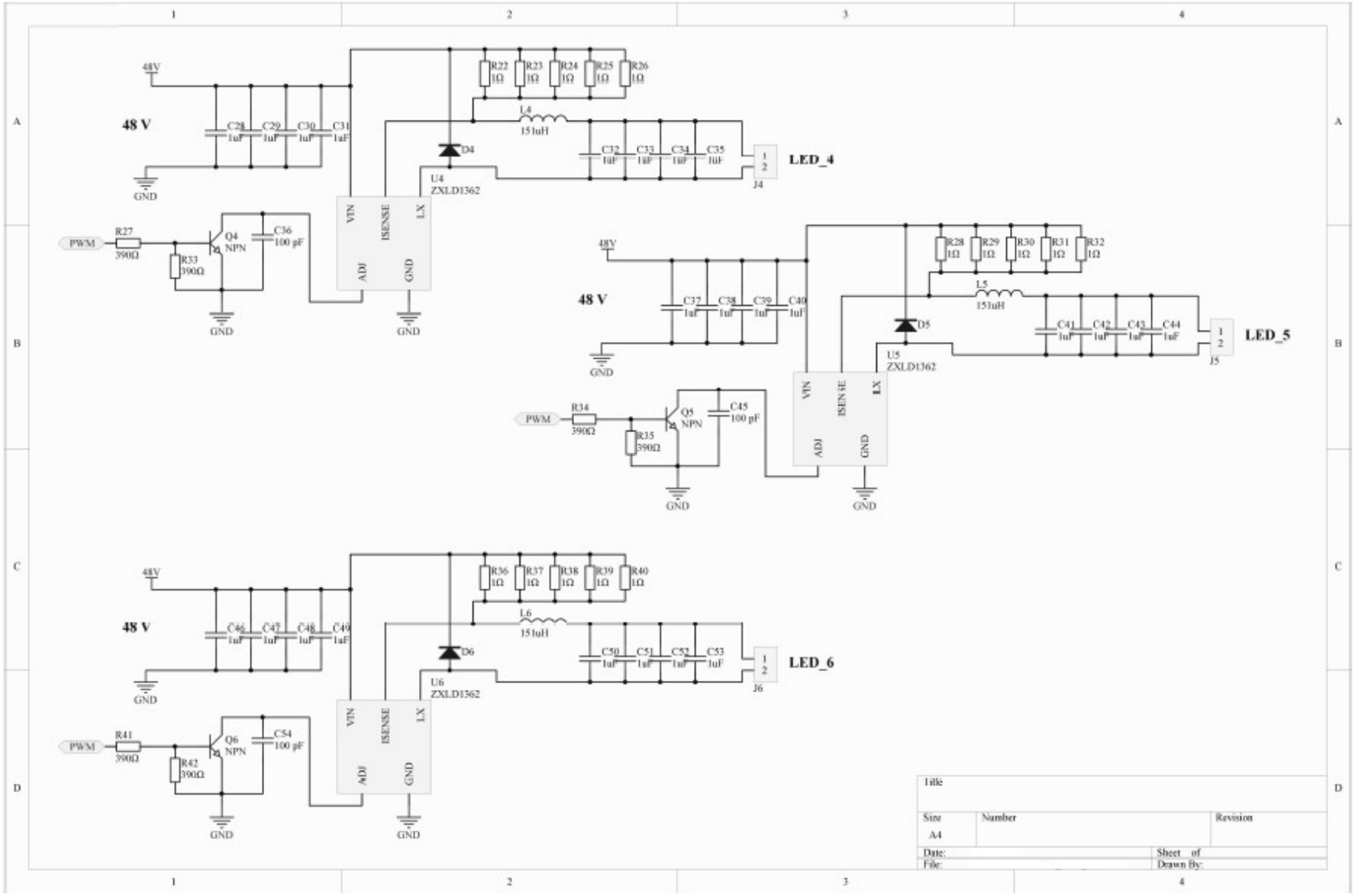
tools/stm32cubemx.html.

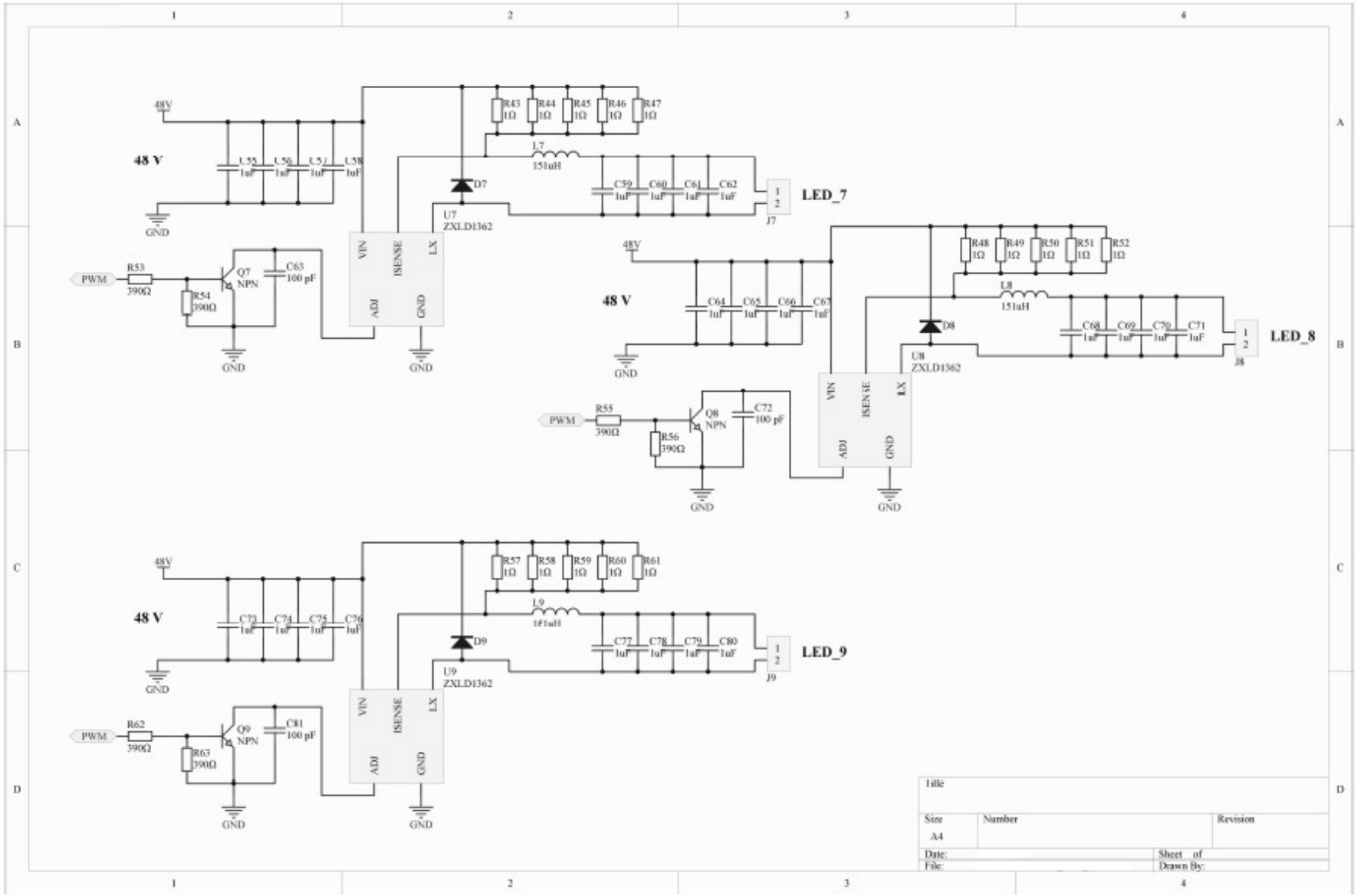
[48] “Keil Embedded Development Tools for Arm, Cortex-M, Cortex-R4, 8051, C166, and 251 processor families.” [Online]. Available: https://www.keil.com/.

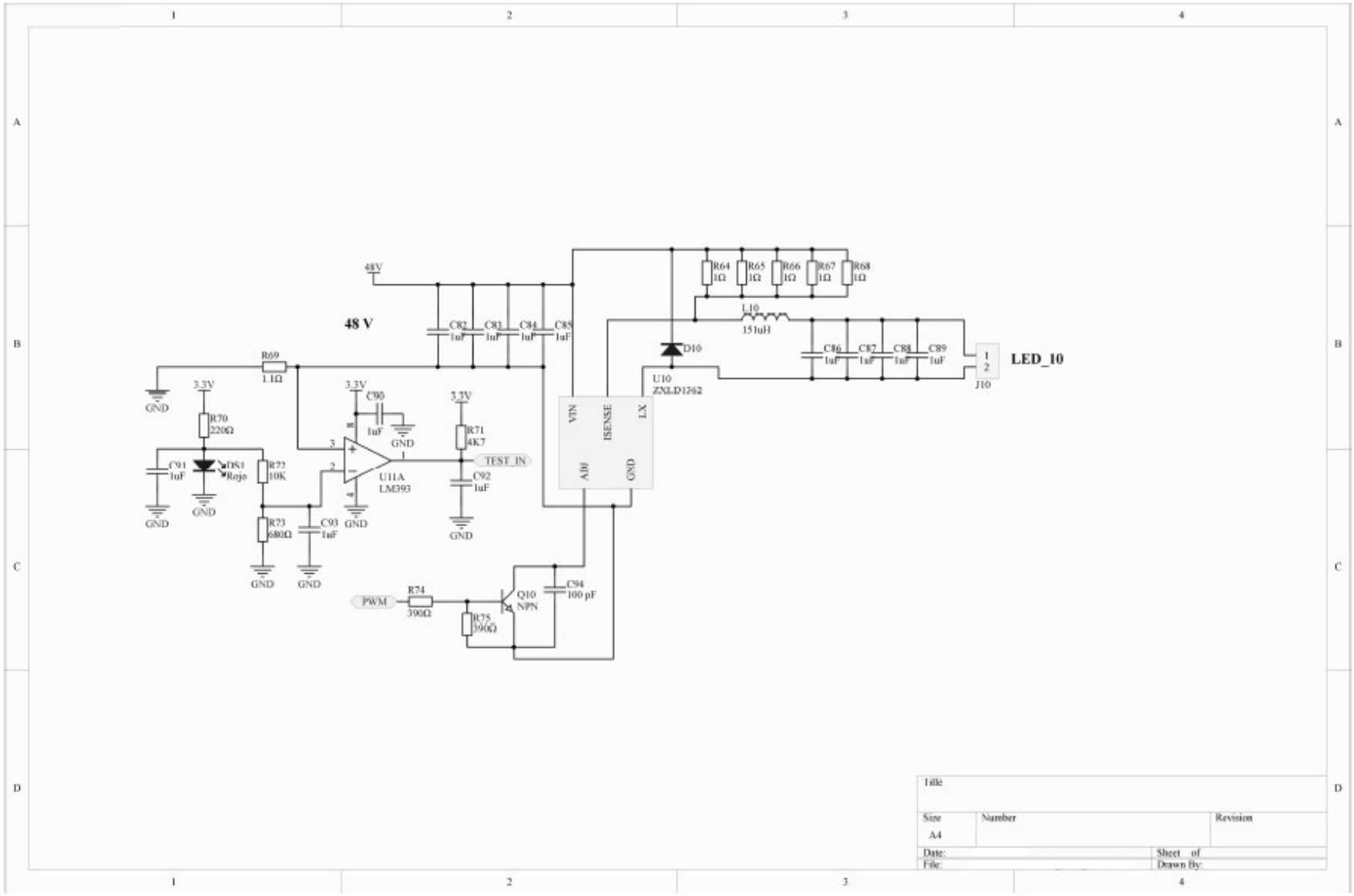
Anexo I: Esquemáticos

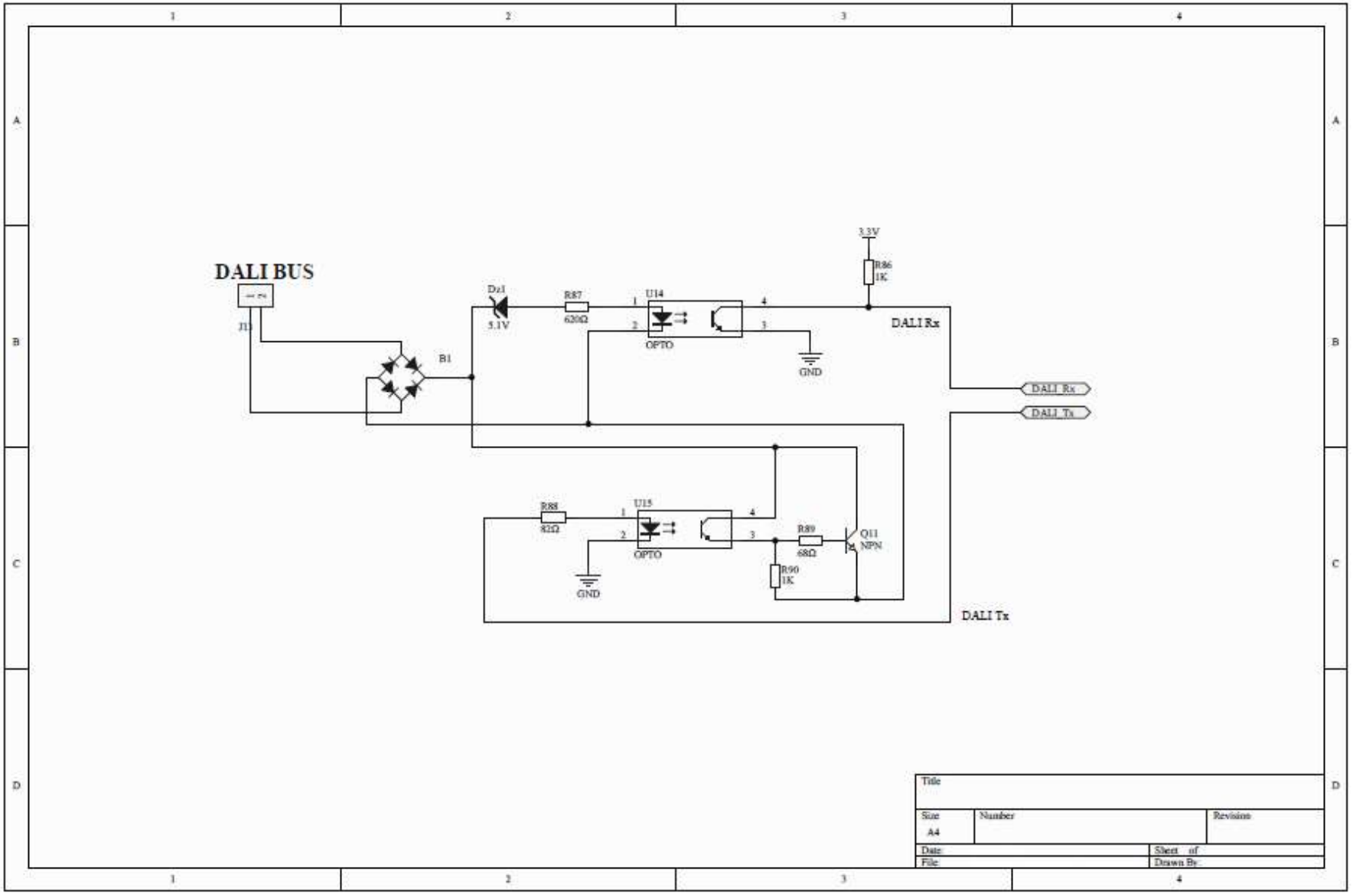

PROGRAMADOR
J11
Con_6
FUENTE ALIM.
J12
48V
R76
DS2
Verde
U96
1 uF
GND
U12
STM32F051C6T6
STM32F051C6T6
VBAT
PC13
PC14
PC15
OSC_IN
OSC_OUT
NRST
VCC
PA0
PA1
PA2
I2C_SDA
I2C_SCL
SWDAT
TEST_IN
PWM
SWCLK
3.3V
C95
1uF
R77
100K
R78
4K7
SW1
C97
1uF
C98
1uF
R79
220Ω
R80
220Ω
R81
220Ω
DALI_Tx
DALI_Rx
Rx
DS3
Azul
DS4 Tx
Verde
DS5 Fallo
Rojo
U13
LM2576
V_IN
OUT
GND
FEEDBACK
ON/OFF
INPUT
C100
100µF
C99
100nF
L11
470uH
D11
C102
470µF
C101
100nF
R82
1K6
R84
100Ω
R85
1K
R83
220Ω
DS6
Verde
OUTPUT
Title
Size
A4
Number
Revision
Date:
File:
Sheet of
Drawn By:


48V
48 V
C1 1uF
C2 1uF
C3 1uF
C4 1uF
GND
R1 1Ω
R2 1Ω
R3 1Ω
R4 1Ω
R5 1Ω
L1 151uH
D1
C5 10uF
C6 10uF
C7 10uF
C8 10uF
J1
LED_1
U1 ZXLD1362
VIN
ISENSE
LX
ADJ
GND
PWM
R6 390Ω
Q1 NPN
C9 100 pF
R12 390Ω
R7 1Ω
R8 1Ω
R9 1Ω
R10 1Ω
R11 1Ω
L2 151uH
D2
C10 1uF
C11 1uF
C12 1uF
C13 1uF
C14 1uF
C15 1uF
C16 1uF
C17 1uF
J2
LED_2
U2 ZXLD1362
R13 390Ω
Q2 NPN
C18 100 pF
R14 390Ω
R15 1Ω
R16 1Ω
R17 1Ω
R18 1Ω
R19 1Ω
L3 151uH
D3
C19 1uF
C20 1uF
C21 1uF
C22 1uF
C23 1uF
C24 1uF
C25 1uF
C26 1uF
J3
LED_3
U3 ZXLD1362
R20 390Ω
Q3 NPN
C27 100 pF
R21 390Ω
Title
Size
A4
Number
Revision
Date:
File:
Sheet of
Drawn By:


48V
48 V
C28 1uF
C29 1uF
C30 1uF
C31 1uF
GND
R22 1Ω
R23 1Ω
R24 1Ω
R25 1Ω
R26 1Ω
L4 15uH
D4
C32 10uF
C33 10uF
C34 10uF
C35 10uF
J4
LED_4
U4 ZXLD1362
VIN
ISENSE
LX
ADJ
GND
PWM
R27 390Ω
Q4 NPN
C36 100 pF
R33 390Ω
R28 1Ω
R29 1Ω
R30 1Ω
R31 1Ω
R32 1Ω
L5 15uH
C37 1uF
C38 1uF
C39 1uF
C40 1uF
D5
C41 1uF
C42 1uF
C43 1uF
C44 1uF
J5
LED_5
U5 ZXLD1362
R34 390Ω
Q5 NPN
C45 100 pF
R35 390Ω
R36 1Ω
R37 1Ω
R38 1Ω
R39 1Ω
R40 1Ω
L6 15uH
C46 1uF
C47 1uF
C48 1uF
C49 1uF
D6
C50 1uF
C51 1uF
C52 1uF
C53 1uF
J6
LED_6
U6 ZXLD1362
R41 390Ω
Q6 NPN
C54 100 pF
R42 390Ω
Title
Size
A4
Number
Revision
Date:
File:
Sheet of
Drawn By:


48V
48 V
GND
R43 1Ω
R44 1Ω
R45 1Ω
R46 1Ω
R47 1Ω
L7 151uH
D7
U7 ZXLD1362
VIN
ISENSE
LX
ADJ
GND
C59 1uF
C60 1uF
C61 1uF
C62 1uF
J7
LED_7
PWM
R53 390Ω
R54 390Ω
Q7 NPN
C63 100 pF
R48 1Ω
R49 1Ω
R50 1Ω
R51 1Ω
R52 1Ω
L8 151uH
D8
U8 ZXLD1362
C68 1uF
C69 1uF
C70 1uF
C71 1uF
J8
LED_8
R55 390Ω
R56 390Ω
Q8 NPN
C72 100 pF
R57 1Ω
R58 1Ω
R59 1Ω
R60 1Ω
R61 1Ω
L9 151uH
D9
U9 ZXLD1362
C77 1uF
C78 1uF
C79 1uF
C80 1uF
J9
LED_9
R62 390Ω
R63 390Ω
Q9 NPN
C81 100 pF
Title
Size
A4
Number
Revision
Date:
File:
Sheet of
Drawn By:


48V
48 V
C82 1uF
C83 1uF
C84 1uF
C85 1uF
R64 1Ω
R65 1Ω
R66 1Ω
R67 1Ω
R68 1Ω
L10 151uH
D10
U10 ZXLD1362
VIN
ISENSE
LX
A/0
GND
C86 1uF
C87 1uF
C88 1uF
C89 1uF
J10
LED_10
R69 1.1Ω
3.3V
R70 220Ω
C91 1uF
DS1 Rojo
R72 10K
C90 1uF
R71 4K7
TEST_IN
U11A LM393
C92 1uF
R73 680Ω
C93 1uF
PWM
R74 390Ω
R75 390Ω
Q10 NPN
C94 100 pF
Title
Size A4
Number
Revision
Date:
File:
Sheet of
Drawn By:


DALI BUS
J11
B1
Dz1
5.1V
R87
620Ω
U14
OPTO
GND
3.3V
R86
1K
DALI Rx
DALI_Rx
DALI_Tx
R88
82Ω
U15
OPTO
GND
R89
68Ω
Q11
NPN
R90
1K
DALI Tx
Title
Size
A4
Number
Revision
Date:
File:
Sheet of
Drawn By:

Anexo II: Código fuente

slave_dali_comandos.h

```
//Autor: Miguel Garnelo Rodríguez
//Grado en Ingeniería Electrónica Industrial y Automática
//Trabajo Fin de Grado – Julio 2018
//Escuela Politécnica de Ingeniería de Gijón
//Universidad de Oviedo

//Lista de comandos DALI definidos en el estándar IEC62386-102

//..... COMANDOS DE NIVEL .....
#define OFF                         (uint8_t) 0x00
#define UP                          (uint8_t) 0x01
#define DOWN                        (uint8_t) 0x02
#define STEP_UP                     (uint8_t) 0x03
#define STEP_DOWN                   (uint8_t) 0x04
#define RECALL_MAX_LEVEL            (uint8_t) 0x05
#define RECALL_MIN_LEVEL            (uint8_t) 0x06
#define STEP_DOWN_AND_OFF           (uint8_t) 0x07
#define ON_AND_STEP_UP              (uint8_t) 0x08
#define ENABLE_DAPC_SEQUENCE        (uint8_t) 0x09
#define GO_TO_LAST_ACTIVE_LEVEL     (uint8_t) 0x0A
#define GO_TO_SCENE                 (uint8_t) 0x10

//..... COMANDOS DE CONFIGURACION .....
#define RESET                                (uint8_t) 0x20
#define STORE_ACTUAL_LEVEL_IN_DTR0           (uint8_t) 0x21
#define SAVE_PERSISTENT_VARIABLES            (uint8_t) 0x22
#define SET_OPERATING_MODE_DTR0              (uint8_t) 0x23
#define RESET_MEMORY_BANK_DTR0               (uint8_t) 0x24
#define IDENTIFY_DEVICE                      (uint8_t) 0x25

#define SET_MAX_LEVEL_DTR0                   (uint8_t) 0x2A
#define SET_MIN_LEVEL_DTR0                   (uint8_t) 0x2B
#define SET_SYSTEM_FAILURE_LEVEL_DTR0        (uint8_t) 0x2C
#define SET_POWER_ON_LEVEL_DTR0              (uint8_t) 0x2D
#define SET_FADE_TIME_DTR0                   (uint8_t) 0x2E
#define SET_FADE_RATE_DTR0                   (uint8_t) 0x2F
#define SET_EXTENDED_FADE_TIME_DTR0          (uint8_t) 0x30
#define SET_SCENE_DTR0                       (uint8_t) 0x40
#define REMOVE_FROM_SCENE                    (uint8_t) 0x50
#define ADD_TO_GROUP                         (uint8_t) 0x60
#define REMOVE_FROM_GROUP                    (uint8_t) 0x70
#define SET_SHORT_ADDRESS_DTR0               (uint8_t) 0x80
#define ENABLE_WRITE_MEMORY                  (uint8_t) 0x81

//..... COMANDOS DE CONSULTA (QUERY) .....
#define QUERY_STATUS                         (uint8_t) 0x90
#define QUERY_CONTROL_GEAR_PRESENT           (uint8_t) 0x91
#define QUERY_LAMP_FAILURE                   (uint8_t) 0x92
```

```
#define QUERY_LAMP_POWER_ON                 (uint8_t) 0x93
#define QUERY_LIMIT_ERROR                   (uint8_t) 0x94
#define QUERY_RESET_STATE                   (uint8_t) 0x95
#define QUERY_MISSING_SHORT_ADDRESS         (uint8_t) 0x96
#define QUERY_VERSION_NUMBER                (uint8_t) 0x97
#define QUERY_CONTENT_DTR0                  (uint8_t) 0x98
#define QUERY_DEVICE_TYPE                   (uint8_t) 0x99
#define QUERY_PHYSICAL_MINIMUM              (uint8_t) 0x9A
#define QUERY_POWER_FAILURE                 (uint8_t) 0x9B
#define QUERY_CONTENT_DTR1                  (uint8_t) 0x9C
#define QUERY_CONTENT_DTR2                  (uint8_t) 0x9D
#define QUERY_OPERATING_MODE                (uint8_t) 0x9E
#define QUERY_LIGHT_SOURCE_TYPE             (uint8_t) 0x9F
#define QUERY_ACTUAL_LEVEL                  (uint8_t) 0xA0
#define QUERY_MAX_LEVEL                     (uint8_t) 0xA1
#define QUERY_MIN_LEVEL                     (uint8_t) 0xA2
#define QUERY_POWER_ON_LEVEL                (uint8_t) 0xA3
#define QUERY_SYSTEM_FAILURE_LEVEL          (uint8_t) 0xA4
#define QUERY_FADE_TIME_RATE                (uint8_t) 0xA5
#define QUERY_MANUFACTURER_SPECIFIC_MODE    (uint8_t) 0xA6
#define QUERY_NEXT_DEVICE_TYPE              (uint8_t) 0xA7
#define QUERY_EXTENDED_FADE_TIME            (uint8_t) 0xA8
#define QUERY_CONTROL_GEAR_FAILURE          (uint8_t) 0xAA

#define QUERY_SCENE_LEVEL                   (uint8_t) 0xB0
#define QUERY_GROUPS_0_7                    (uint8_t) 0xC0
#define QUERY_GROUPS_8_15                   (uint8_t) 0xC1
#define QUERY_RANDOM_ADDRESS_H              (uint8_t) 0xC2
#define QUERY_RANDOM_ADDRESS_M              (uint8_t) 0xC3
#define QUERY_RANDOM_ADDRESS_L              (uint8_t) 0xC4
#define QUERY_MEMORY_LOCATION               (uint8_t) 0xC5
#define QUERY_EXTENDED_VERSION_NUMBER     (uint8_t) 0xFF

//..... COMANDOS ESPECIALES .....
//NOTA: Los siguientes valores hexadecimales se refieren al byte
//de DIRECCION, no al byte de OPCODE como los anteriores
#define TERMINATE                                 (uint8_t) 0xA1
#define DTR_0                           (uint8_t) 0xA3
#define INITIALISE                      (uint8_t) 0xA5
#define RANDOMISE                                 (uint8_t) 0xA7
#define COMPARE                                   (uint8_t) 0xA9
#define WITHDRAW                                  (uint8_t) 0xAB
#define PING                            (uint8_t) 0xAD
#define SEARCHADDRH                     (uint8_t) 0xB1
#define SEARCHADDRM                     (uint8_t) 0xB3
#define SEARCHADDRL                     (uint8_t) 0xB5
#define PROGRAM_SHORT_ADDRESS           (uint8_t) 0xB7
#define VERIFY_SHORT_ADDRESS            (uint8_t) 0xB9
#define QUERY_SHORT_ADDRESS             (uint8_t) 0xBB
#define PHYSICAL_SELECTION              (uint8_t) 0xBD
#define ENABLE_DEVICE_TYPE              (uint8_t) 0xC1
#define DTR1                            (uint8_t) 0xC3
#define DTR2                            (uint8_t) 0xC5
```

```
#define WRITE_MEMORY_LOCATION          (uint8_t) 0xC7
#define WRITE_MEMORY_LOCATION_NO_REPLY      (uint8_t) 0xC9
```

slave_dali_comun.h

```
//Autor: Miguel Garnelo Rodríguez
//Grado en Ingeniería Electrónica Industrial y Automática
//Trabajo Fin de Grado – Julio 2018
//Escuela Politécnica de Ingeniería de Gijón
//Universidad de Oviedo

//Este archivo contiene definiciones y variables utilizadas de forma compartida
//por los diferentes ficheros de código que conforman el proyecto

#define TRUE  0x01
#define FALSE 0x00
#define YES  0xFF
#define MASK 0xFF
#define NUEVO_DATO 0x01                 //Flag para indicar que un nuevo comando se ha
                                        //recibido  por el puerto
#define ENVIAR_RESPUESTA  0x02          //Flag para indicar que se va a enviar una respuesta
                                        //al maestro
#define ERROR_BUS  0x04                 //Flag para indicar que se ha producido un error en el bus

//Variables globales
extern unsigned char direccion;     //Almacena la dirección enviada por el maestro DALI
extern unsigned char comando;       //Almacena el comando enviado por el maestro DALI
extern unsigned char respuesta;    //Almacena la respuesta que el esclavo DALI envía al //maestro
extern unsigned int flag;           //Flag para indicar todos los eventos que se producen
```

slave_dali_protocol.h

```
//Autor: Miguel Garnelo Rodríguez
//Grado en Ingeniería Electrónica Industrial y Automática
//Trabajo Fin de Grado - Julio2018
//Escuela Politécnica de Ingeniería de Gijón
//Universidad de Oviedo

//Este archivo contiene los prototipos de las funciones necesarias para realizar la parte de envío y recepción
//de tramas del protocolo DALI (Digital Addressable Lighting Interface) por parte de un esclavo.

//Inicialización de los elementos que intervienen en el proceso de comunicación del protocolo
//(configuración pines Rx y Tx, timer para la base de tiempos, así como interrupciones necesarias)
void dali_Inic(void);

//Preparación del esclavo DALI para el envío de una respuesta al maestro a través del bus
void dali_EnvioRespuesta(void);
```

slave_dali_lampara.h

```
//Autor: Miguel Garnelo Rodríguez
//Grado en Ingeniería Electrónica Industrial y Automática
//Trabajo Fin de Grado – Julio 2018
//Escuela Politécnica de Ingeniería de Gijón
//Universidad de Oviedo

//Este archivo contiene los prototipos de las funciones necesarias para realizar el procesamiento de la
//diferente información recibida a través del bus DALI, así como la interfaz con la lámpara LED para la
//ejecución de las diferentes acciones que sea necesario realizar (cambiar nivel de luz, consultas del estado
//de la lámpara, etc.)

//Inicialización de los elementos que intervienen en el proceso de tratamiento de la informacion recibida
//(configuración diferentes pines (PWM, test de fallos, etc.), timer para la base de tiempos, así como
//interrupciones necesarias)
void lampara_Inic(void);

//Función para el procesamiento de los comandos recibidos
void lampara_ProcesaComando(void);

//Función para situar la lámpara en nivel de fallo
void lampara_NivelFallo(void);
```

slave_dali_protocol.c

```
//Autor: Miguel Garnelo Rodríguez
//Grado en Ingeniería Electrónica Industrial y Automática
//Trabajo Fin de Grado – Julio 2018
//Escuela Politécnica de Ingeniería de Gijón
//Universidad de Oviedo

//Este archivo contiene la implementación de las funciones necesarias para manejar
//todas las tareas de recepción y envío de tramas relacionadas con el bus DALI.

//Inclusión de ficheros
#include "stm32f0xx.h"
#include "stm32f0xx_hal.h"
#include "stm32f0xx_it.h"
#include "slave_dali_comun.h"
#include "slave_dali_protocol.h"

//Prototipos de funciones HAL creadas por el programa STM32CubeMX al realizar
//la selección de los pines de los periféricos mediante su interfaz gráfica
extern TIM_HandleTypeDef htim2;
void SystemClock_Config(void);
static void MX_GPIO_Init_dali(void);
```

```
//Definición de variables locales
extern unsigned char recep_on;    //Variable para indicar si la recepción ha comenzado
unsigned char envio_on;           //Variable para indicar si el envío ha comenzado
unsigned char direccion_aux;          //Variable para almacenar el valor de la dirección que //se está
                                      recibiendo
unsigned char comando_aux;            //Variable para almacenar el valor del comando que //se está
                                      recibiendo
unsigned char posicion;           //Posición en la trama a recibir o a enviar
unsigned char valor;                  //Contiene el nivel lógico recibido en el pin Rx o a //enviar por
                                      Tx
unsigned int cont_nbajo;              //Contador de interrupciones del timer durante las //cuales el
                                      nivel del bus ha permanecido bajo

//Funciones de configuración de los periféricos que intervienen en el control de las tareas //relacionadas
con la funcionalidad de la lámpara (generadas por el STM32CubeMX)

//Configuración SysTick: base de tiempos para las comunicaciones a través del bus DALI
//Se configura para generar una interrupción cada 104,17 us (t_halfbit/4)
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  //Inicialización de los relojes de los buses CPU, AHB y APB
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL12;
  RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
|RCC_CLOCKTYPE_PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
//Configuración del Systick
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
  //Configuramos el SysTick para que genere una interrupción
//cada 5000 ticks, que con 48 MHz, corresponde a los 104,17 us
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/9600);
  //Configuración de la interrupción del SysTick
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}//Fin SystemClock_Config
```

```
//Configuración de los pines GPIO que intervienen en el
//proceso de comunicación: Rx, Tx y LEDs de señalización
static void MX_GPIO_Init_dali(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  //Habilitación de los relojes asociados a los puertos GPIOA y GPIOB
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  //Configuración de los niveles de los pines de salida
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  //Configuración pines GPIO: PA5 PA6 PA7 (LEDs indicación)
  GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  //Configuración pin GPIO: PB0 (Tx)
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);


  //Configuración pin GPIO: PB1 (Rx)
  GPIO_InitStruct.Pin = GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}//Fin MX_GPIO_Init_dali


//Función de inicialización del módulo de comunicación vía bus DALI
void dali_Inic(void)
{
   recep_on = FALSE;         //Iniciamos las variables de comienzo de recepción //transmision a
   envio_on = FALSE;   //0 a la espera de que suceda el evento correspondiente que inicie
                              //uno de los procesos

   //Llamada a funciones de configuración de pines (Tx, Rx y leds de indicación) y
   //timer (SysTick) que intervienen en la comunicación, así como de las interrupciones
   //necesarias para manejar los eventos asociados
   SystemClock_Config();
   MX_GPIO_Init_dali();
   //Habilitamos la interrupción externa en Rx (generada al recibir un flanco de bajada ->
   //bit Start)
   HAL_NVIC_SetPriority(EXTI0_1_IRQn, 3, 0);
   HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
}//Fin dali_Inic(void)

//Función para el envío de una respuesta por parte del esclavo
```

```
void dali_EnvioRespuesta(void)
{
   while (recep_on || envio_on);   //Esperamos a que el bus DALI esté en estado IDLE
                                   //(reposo) para no interferir
   //Desactivar interrupciones del pin Rx (para que no interfiera una nueva recepción de
   //trama enviada por el maestro) y del timer que gestiona las diferentes operaciones de
   //la lámpara
    HAL_TIM_Base_Stop_IT(&htim2);
    HAL_NVIC_DisableIRQ(EXTI0_1_IRQn);
    HAL_NVIC_ClearPendingIRQ(EXTI0_1_IRQn);
    //Preparación de las variables que intervienen en el envío
    valor = 1;                  //Mantenemos el bus a 1 al comienzo del envío para cumplir el
                                //tiempo de establecimiento entre la recepción de una trama del
                                //maestro y la correspondiente respuesta del esclavo
     posicion = 0;
     envio_on = TRUE;           //Variable para activar la parte de envío en la interrupción de la
                                //base de tiempos (SysTick)
} //Fin dali_EnvioRespuesta(void)

//Interrupción en el pin Rx que detecta el comienzo de la recepción: transición de 1
//(bus DALI idle) a 0 (primera parte del bit de START (codificación Manchester))
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
   if ((GPIO_Pin == GPIO_PIN_1) && (recep_on == FALSE))
   {
      while (recep_on || envio_on);          //Esperamos a que el bus DALI esté en estado
                                             //IDLE (reposo) para no interferir

      //Desactivar interrupciones del pin Rx (para que no interfiera una nueva recepción
      //de trama enviada por el maestro) y del timer que gestiona las diferentes
      //operaciones de la lámpara
      HAL_TIM_Base_Stop_IT(&htim2);
      HAL_NVIC_DisableIRQ(EXTI0_1_IRQn);
      HAL_NVIC_ClearPendingIRQ(EXTI0_1_IRQn);

      //Preparación de las variables que intervienen en la recepción
      direccion_aux = 0x00;          //Inicialización de las variables que almacenan el valor de la dirección
      comando_aux = 0x00;            // y el comando enviados por el maestro DALI durante la recepción
      posicion = 0;
      valor = 0;
      recep_on = TRUE;    //Variable para activar la parte de recepción en la interrupción de la base
                          //de tiempos (SysTick
   }
}//Fin HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)


//Interrupción del módulo SysTick (base de tiempos para las tareas correspondientes a la comunicación
//maestro-esclavo a través del bus DALI) que se produce cada 104,17 us (t_halfbit/4) para un mejor
//procesamiento de la trama
void SysTick_Handler (void)
{
   HAL_SYSTICK_IRQHandler();
   HAL_IncTick();
```

```
unsigned int valor_aux;          //Variable auxiliar

if (recep_on == TRUE)            //Recepción de una trama DALI enviada por el maestro
{
      switch (posicion)
      {
          case 5:          //Bit Start
          {
             valor = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1);        //Leemos el valor recibido en el pin Rx
             HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);//Encendemos el led indicando que
                                                                //hemos recibido un nuevo bit

             if (valor == 1)         //Lógica inversa en el transceiver. Estaríamos recibiendo un 0 en la
             {                       //segunda mitad del bit de start, cuando debería estar a 1 -> ERROR!
                                     //Fin de la recepción
                HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);  //Apagamos el led de Rx

                recep_on = FALSE;

                //Activar interrupciones del pin Rx y del timer que gestiona las diferentes operaciones de
                //la lámpara
                HAL_TIM_Base_Start_IT(&htim2);
                HAL_NVIC_ClearPendingIRQ(EXTI0_1_IRQn);
                HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
             }
             break;
          }

          case 13:       //Bit 7 de la dirección
          case 21:       //Bit 6 de la dirección
          case 29:       //Bit 5 de la dirección
          case 37:       //Bit 4 de la dirección
          case 45:       //Bit 3 de la dirección
          case 53:       //Bit 2 de la dirección
          case 61:       //Bit 1 de la dirección
          case 69:       //Bit 0 de la dirección
          {
              valor = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1);        //Leemos el valor recibido en el pin Rx
              HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);  //Encendemos el led indicando
                                                                 //que hemos recibido un nuevo
              if (valor == 0)        //Lógica inversa en el transceiver, en realidad el bus está a 1
              {
                 valor = 1;
              }
              else
              {
                 valor = 0;
              }

              direccion_aux = (direccion_aux << 1)|valor;
              break;
          }
```

```
case 77:    //Bit 7 del comando
case 85:    //Bit 6 del comando
case 93:    //Bit 5 del comando
case 101:   //Bit 4 del comando
case 109:   //Bit 3 del comando
case 117:   //Bit 2 del comando
case 125:   //Bit 1 del comando
case 133:   //Bit 0 del comando
{
  valor = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1);       //Leemos el valor recibido en el pin Rx
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET); //Encendemos el led indicando
                                                    //que hemos recibido un nuevo bit
  if (valor == 0)       //Lógica inversa en el transceiver, en realidad el bus está a 1
  {
    valor = 1;
  }
  else
  {
    valor = 0;
  }

  comando_aux = (comando_aux << 1)|valor;
  break;
}

case 143:   //Fin primer bit de stop
{
  valor = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1);       //Leemos el valor recibido en el pin Rx
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET); //Encendemos el led indicando
                                                    //que hemos recibido un nuevo bit

  if (valor == 1)       //Lógica inversa en el transceiver. Estaríamos recibiendo un 0 en la
  {                     //segunda mitad del bit de stop, cuando debería estar a 1 -> ERROR!
                        //Fin de la recepción
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);      //Apagamos el led de Rx

    recep_on = FALSE;

    //Activar interrupciones del pin Rx y del timer que gestiona las diferentes operaciones de
    //la lámpara
    HAL_TIM_Base_Start_IT(&htim2);
    HAL_NVIC_ClearPendingIRQ(EXTI0_1_IRQn);
    HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
  }
  break;
}
```

```
            case 151:   //Fin segundo bit de stop. Recepción completada
            {
               direccion = direccion_aux;      //Si la recepción se realiza correctamente, actualizamos el
               comando = comando_aux;      //valor anterior de la dirección y del comando por los nuevos

                HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);          //Apagamos el led de Rx


               //Activar interrupciones del pin Rx y del timer que gestiona las diferentes operaciones de
               //la lámpara
                HAL_TIM_Base_Start_IT(&htim2);
                HAL_NVIC_ClearPendingIRQ(EXTI0_1_IRQn);
                HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);

                flag |= NUEVO_DATO;      //Indicamos que se dispone de un nuevo comando a procesar

                break;
            }

            default:
            {
               HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);          //Apagamos el led de Rx

                break;
            }
        }//Fin switch (posicion)

        posicion++;          //Aumentamos el valor de posicion para la siguiente interrupción del SysTick
    }//Fin if (recep_on == TRUE)

    else if (envio_on == TRUE)
    {
        //Se esperan 24 interrupciones del SysTick para asegurar que se cumple el tiempo de
        //establecimiento entre la recepción de una trama enviada por el maestro y el envío de la
        //respuesta por parte del esclavo. Segun la norma: ts(min)= 2,4 ms -> 24x104,16 us = 2,5 ms

        if ((posicion >= 23) && (posicion < 27))        //Primera mitad del bit de start
        {
            valor = 0;
        }

        else if ((posicion >= 27) && (posicion < 31))  //Segunda mitad del bit de start
        {
            valor = 1;
         }

        else if ((posicion >= 31) && (posicion < 95))   //Bits que conforman la respuesta
        {
             //Extraemos de la respuesta el nivel del bit a enviar
             valor_aux = (respuesta >> ((94 - posicion)/8)) & 0x01;
```

```
        //Comprobamos si es la primera o la segunda mitad del bit
        if (((posicion + 1) & 0x04) == 0)          //Primera mitad del bit
        {
            if (valor_aux == 0)             //Estamos enviando un 0. En cod Manchester se representa
            {                               //como un flanco de bajada (primera mitad del bit a 1)
                valor = 1;
            }
            else                            //Estamos enviando un 1. En cod Manchester se representa
            {                               //como un flanco de subida (primera mitad del bit a 0)
                valor = 0;
            }
        }
        else                                        //Segunda mitad del bit
        {
            if (valor_aux == 0)             //Estamos enviando un 0. En cod Manchester se representa
            {                               //como un flanco de bajada (segunda mitad del bit a 0)
                valor = 0;
            }
            else                            //Estamos enviando un 1. En cod Manchester se representa
            {                               //como un flanco de subida (segunda mitad del bit a 1)
                valor = 1;
            }
        }
}//Fin else if ((posicion >= 31) && (posicion < 95))

else if ((posicion >= 95) && (posicion < 112))          //Comienzo condición de stop
{
    valor = 1;      //Mantenemos el bus en estado IDLE
}

else if (posicion >= 112)    //Fin del envío. Ha transcurrido el tiempo equivalente a dos bits de stop
{
    envio_on = FALSE;

    //Activar interrupciones del pin Rx y del timer que gestiona las diferentes operaciones de la
    //lámpara
    HAL_TIM_Base_Start_IT(&htim2);
    HAL_NVIC_ClearPendingIRQ(EXTI0_1_IRQn);
    HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
}

posicion++;         //Aumentamos el valor de posicion para la siguiente interrupción del SysTick

if (valor == 0)     //Cambiamos el valor, debido a que nuestro transceiver es de lógica inversa al
{                   //bus DALI
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_0,GPIO_PIN_SET);           //Ponemos en el pin Tx el
                                                                //valor correspondiente
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6,GPIO_PIN_RESET);         //Apagamos el led cuando se
                                                                //envia un 0
}
```

```
            else
            {
                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_0,GPIO_PIN_RESET);        //Ponemos en el pin Tx el
                                                                           //valor correspondiente
                HAL_GPIO_WritePin(GPIOA,GPIO_PIN_6,GPIO_PIN_SET);          //Encendemos el led cuando
                                                                           //se envia un 1
            }
        }//Fin if (envio_on == TRUE)

        else if ((HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1)) == 0)            //Leemos el valor del bus en el pin Rx
        {
            cont_nbajo = 0;   //Sino estamos recibiendo ni enviando y el bus permanece idle (a 1), todo correcto.
            HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_RESET);    //Apagamos el led indicador de fallo
        }                                                              //en el bus
        else       //Si el bus está a 0, almacenamos en cont_nbajo el nº de
        {          //interrupciones de la base de tiempos que permanece a 0.
            cont_nbajo++;
            if (cont_nbajo > 4800)        //Si el bus llega a estar 500 ms o más a 0: ERROR! Debemos poner la
            {                             //lámpara en failureLevel
                cont_nbajo = 0;
                flag |= ERROR_BUS;
             }
        }
} //Fin interrupcion SysTick
```

slave_dali_lampara.c

```
//Autor: Miguel Garnelo Rodríguez
//Grado en Ingeniería Electrónica Industrial y Automática
//Trabajo Fin de Grado - Julio 2018
//Escuela Politécnica de Ingeniería de Gijón
//Universidad de Oviedo

//Inclusión de ficheros
#include "stm32f0xx.h"
#include "stm32f0xx_hal.h"
#include "stm32f0xx_it.h"
#include "slave_dali_comandos.h"
#include "slave_dali_comun.h"
#include "slave_dali_lampara.h"
```

```
//Variables HAL
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;

//Prototipos de funciones HAL creadas por el programa STM32CubeMX al realizar
//la selección de los pines de los periféricos mediante su interfaz gráfica
static void MX_GPIO_Init_lamp(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);
void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);

//Definiciones
#define VERSION 0x00
#define PHM 0x01          //Physical Minimun Level (mínimo nivel de luminosidad al que la lámpara puede
                          //operar)
//Estados de la lámpara manejados por su base de tiempos
#define IDLE    0x00      //Mientras no se recibe ninguna instrucción nueva. No hay cambio
#define INIC    0x01      //Inicialización. Durante los primeros 600 ms (según norma) o hasta que
                          //se recibe la primera trama
#define FADE    0x02      //Estado en el que la lámpara se encuentra en transición del nivel actual
                          //al nuevo nivel
//Definición de variables locales
unsigned char actualLevel;          //Nivel de luminosidad actual de la lámpara
unsigned char powerOnLevel;         //Nivel de luminosidad de la lámpara al encendido
unsigned char systemFailureLevel; //Nivel de luminosidad de la lámpara cuando se produce un fallo en el
                                  //bus DALI
unsigned char minLevel;             //Mínimo nivel (hasta el momento) que se ha establecido en la lámpara
unsigned char maxLevel;             //Máximo nivel (hasta el momento) que se ha establecido en la lámpara
unsigned char fadeRate;             //Permite elegir la tasa (steps/s) que se usará en un proceso de fade.
                                    //Valor del 1 al 15.
unsigned char fadeTime;           //Permite elegir el tiempo que durará un proceso de fade. Valor de 0 a 15.
unsigned char shortAddress;
unsigned char randomAddress_h;          //Parte alta de randomAddress
unsigned char randomAddress_m;          //Parte media de randomAddress
unsigned char randomAddress_l;          //Parte baja de randomAddress
```

```
unsigned char searchAddress_h;          //Parte alta de searchAddress
unsigned char searchAddress_m;          //Parte media de searchAddress
unsigned char searchAddress_l;          //Parte baja de searchAddress
unsigned char group_0_7;                //Grupos de luminarias del 0 al 7
unsigned char group_8_15;               //Grupos de luminarias del 8 al 15
unsigned char sceneNumber[15];          //Número de escena (16 posibles). Definen niveles
                                        //luminosidad preestablecidos.
unsigned char powerCycleSeen;           //Flag a 1 hasta que se reciban ciertos comandos (relacionado
                                        //con el power on)
unsigned char limitError;               //Indica si el último nivel recibido está fuera de los límites
                                        //permitidos
unsigned char lampFailure;              //Verdadera si hay un fallo de conexión con la lámpara
unsigned char dtr0;                     //Registro DTR0
unsigned char estado;                   //Estado actual en el que se encuentra la lámpara
unsigned int tiempo_estado;          //Tiempo correspondiente que dura cada estado (en interrupciones)

unsigned char fade_actualLevel;    //Nivel de luz al comienzo del fade
unsigned char fade_targetLevel;    //Nivel de luz de la lámpara al finalizar el fade
unsigned int tiempo_fade;          //Tiempo de duración del fade (contador de interrupciones)

unsigned char tiempo_repeticion; //Tiempo disponible para repetir los comandos que han de enviarse dos
                                 //veces
unsigned char direccion_rep;       //Dirección que debe enviarse dos veces (send-twice en la norma)
unsigned char comando_rep;         //Comando que debe enviarse dos veces (send-twice en la norma)

unsigned int tiempo_dir;  //Tiempo durante el cual no pueden recibirse comandos de direccionamiento

//Valor exponencial para el dimming resultante de aplicar la fórmula 10^(((nuevo nivel-1)/(253/3))-1)
const uint16_t dim_nivel_log[253] = {
48, 49, 50, 52, 53, 55, 56, 58, 59, 61, 63, 64, 66, 68, 70, 72,
74, 76, 78, 80, 82, 85, 87, 89, 92, 94, 97, 100, 103, 105, 108, 111,
114, 118, 121, 124, 128, 131, 135, 139, 143, 147, 151, 155, 159, 163,
168, 173, 177, 182, 187, 193, 198, 204, 209, 215, 221, 227, 233, 240,
247, 253, 260, 268, 275, 283, 290, 299, 307, 315, 324, 333, 342, 352,
362, 372, 382, 392, 403, 414, 426, 438, 450, 462, 475, 488, 502, 516,
530, 545, 560, 575, 591, 608, 624, 642, 660, 678, 697, 716, 736, 756,
```

```
777, 799, 821, 843, 867, 891, 915, 941, 967, 994, 1021, 1049, 1078,
1108, 1139, 1171, 1203, 1236, 1271, 1306, 1342, 1379, 1417, 1456, 1497,
1538, 1581, 1625, 1670, 1716, 1763, 1812, 1862, 1914, 1967, 2021, 2077,
2135, 2194, 2255, 2317, 2381, 2447, 2515, 2585, 2656, 2730, 2805, 2883,
2963, 3045, 3129, 3216, 3305, 3396, 3490, 3587, 3686, 3788, 3893, 4001,
4111, 4225, 4342, 4462, 4586, 4713, 4843, 4977, 5115, 5257, 5402, 5552,
5706, 5864, 6026, 6193, 6364, 6540, 6721, 6907, 7099, 7295, 7497, 7705,
7918, 8137, 8362, 8594, 8832, 9076, 9327, 9585, 9851, 10123, 10404,
10692, 10988, 11292, 11604, 11926, 12256, 12595, 12944, 13302, 13670, 14048,
14437, 14837, 15248, 15670, 16103, 16549,17007, 17478, 17962, 18459, 18970,
19495, 20035, 20589, 21159, 21745, 22347, 22965, 23601, 24254, 24926, 25616,
26325, 27053, 27802, 28572, 29363, 30175, 31011, 31869, 32751, 33658, 34589,
35547, 36531, 37542, 38581, 39649, 40746, 41874, 43033, 44225, 45449, 46707};

//Número de interrupciones necesarias para cumplir los "fade time" especificados por la norma. P.e. para
//fadeTime = 1, el tiempo de fade es de 0,7 s (valor nominal según la norma); teniendo un timer que
//produce una interrupción cada 20 ms, el tiempo estipulado se cumplirá cada 0,7/0.020 = 35
//interrupciones, y así con todos los tiempos
static const unsigned int fade_ticks[15] = {35, 50, 71, 100, 141, 200, 283, 400,
 566, 800, 1131, 1600, 2263, 3200, 4525};

//Número de steps necearios para cumplir las "fade rate" estipuladas por la norma, en un tiempo fijo de
//200 ms (también indicado en la norma). P.e. para fadeRate=1, tenemos una tasa de 358 steps/s (valor
//nominal según la norma), por ello, 358*0.200=72, y así sucesivamente con todos los valores especificados
static const unsigned char fade_step[15] = {72, 51, 36, 25, 18, 13, 9, 6, 4, 3, 2,
 2, 1, 1, 1 };

//Funciones de configuración de los periféricos que intervienen en el control de las tareas relacionadas con
//la funcionalidad de la lámpara (generadas por el STM32CubeMX)

//Configuración GPIO pin PA9: pin sobre el que se leerá el valor del circuito
//encargado de realizar el test que comprueba si la lámpara está conectada
 static void MX_GPIO_Init_lamp(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;
```

```
    //Habilitación del reloj asociado al puerto GPIOA
    __HAL_RCC_GPIOA_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_9;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLDOWN;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
} //Fin MX_GPIO_Init

//Configuración TIM1: encargado de generar la señal PWM encargada de realizar el dimming sobre los LED
//que constituyen la lámpara. Dicha señal está configurada para una frecuencia de 200 Hz.
static void MX_TIM1_Init(void)
{
   TIM_ClockConfigTypeDef sClockSourceConfig;
   TIM_MasterConfigTypeDef sMasterConfig;
   TIM_OC_InitTypeDef sConfigOC;
   TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig;

    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 4;
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = 47999;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.RepetitionCounter = 0;
    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
     }

    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }
```

```
  if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
  sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }
  HAL_TIM_MspPostInit(&htim1);
} //Fin MX_TIM1_Init
```

```
//Configuración TIM2: encargado de establecer la base de tiempos para el procesamiento de comandos y
//ejecución de las acciones correspondientes desencadenadas por dicho manejo de la información recibida
static void MX_TIM2_Init(void)
{
    TIM_ClockConfigTypeDef sClockSourceConfig;
    TIM_MasterConfigTypeDef sMasterConfig;

     htim2.Instance = TIM2;
     htim2.Init.Prescaler = 19;
     htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
     htim2.Init.Period = 47999;
     htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
     htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
     if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
     {
         _Error_Handler(__FILE__, __LINE__);
     }

     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
     if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
     {
         _Error_Handler(__FILE__, __LINE__);
     }

     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
    if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
    {
        _Error_Handler(__FILE__, __LINE__);
    }
} //Fin MX_TIM2_Init
```

```
//Funciones internas necesarias en el proceso

//Función para fijar el nivel lumínico de la lámpara
void set_Nivel(unsigned char targetLevel)
{
    unsigned char nivel_aux;
    uint16_t duty_1;
    uint16_t duty_2;

    limitError = FALSE;
    nivel_aux = targetLevel;

    if ((nivel_aux >= 1) && (nivel_aux <= 254))        //Si el nivel recibido es válido, comprobamos si está
    {                                                  //dentro de los límites. Sino, lo descartamos y lo
                                                       // fijamos en los límites.
        if (nivel_aux < minLevel)
        {
            nivel_aux = minLevel;
            limitError = TRUE;
        }
        else if (nivel_aux > maxLevel)
        {
             nivel_aux = maxLevel;
             limitError = TRUE;
         }
    }

    if (nivel_aux == 0)        //Lámpara apagada
    {
        duty_1 = 0;
    }
    else if ((nivel_aux >= 0x01) && (nivel_aux <= 0xFD))    //Dimming a un determinado porcentaje de duty
    {
        duty_1 = dim_nivel_log[nivel_aux - 1];          //Extraemos de la tabla de valores el porcentaje
    }
```

```
    else if (nivel_aux == 0xFE)        //Lámpara a máxima intensidad
    {
        duty_1 = 48000;                //Equivalente a 100%
    }

    //Generar salida PWM correspondiente
    duty_2= (48000 - duty_1);          //Logica inversa en la patilla de control del dimming en el driver
    htim1.Instance->CCR1 = duty_2;     //Cargamos el valor del duty calculado en el registro correspondiente

    //Almacenamos el nivel actual donde corresponde
    actualLevel = nivel_aux;
} //Fin set_Nivel

//Función para comprobar si hay un fallo de conexión con la lámpara
void test_FalloLamp(void)
{
    unsigned char nivel_aux2;
    unsigned char lectura;             //Variable para detectar si hay fallo o no

    nivel_aux2 = actualLevel;          //Almacenamos el valor actual de la intensidad lumínica
    set_Nivel(254);                    //Situamos la lámpara al máximo de intensidad para el test
    HAL_Delay(50);                     //Espera para que se estabilice el nivel
    lectura = HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_9);          //Pin sobre el que montamos el circuito de test
                                                           //de deteccion de lampara
    HAL_Delay(10);
    set_Nivel(nivel_aux2);             //Reestablecemos el nivel de luz que tenía la lámpara antes del test

    if (lectura == 1)
    {
        lampFailure = FALSE;           //Lámpara funcionando
    }
    else
    {
        lampFailure = TRUE;            //Fallo en la lámpara
    }
} //Fin test_FalloLamp
```

```
//Función que realiza dimming en la lámpara empleando el fade time
void dimming_FadeTime(unsigned char targetLevel)
{
    if (targetLevel != MASK)
    {
        if (fadeTime == 0)                //No hay fade
        {
            set_Nivel(targetLevel);
            estado = IDLE;
        }
        else          //Preparamos las variables que intervienen en el fade
        {
            fade_actualLevel = actualLevel;
            fade_targetLevel = targetLevel;
            tiempo_fade = fade_ticks[fadeTime - 1];
            tiempo_estado = tiempo_fade;
            estado = FADE;
        }
    }
} //Fin dimming_FadeTime

//Función para procesar los comandos especiales del protocolo DALI (según indica la norma IEC62386-102)
void procesa_ComandoEspecial(void)
{
    unsigned char withdrawn;
    unsigned char selection;
    unsigned long randomAddress;
    unsigned long searchAddress;

    //Comandos especiales
    if ((direccion >= 0xA7) && (direccion <= 0xC9) && (tiempo_dir == 0))
    {
        return;                  //Comandos no válidos hasta que se reciba INITIALISE
     }
```

```
switch (direccion)
{
    case TERMINATE:
    {
        tiempo_dir = 0;
        break;
    }
    case DTR_0:
    {
        dtr0 = comando;
        break;
    }
    case INITIALISE:
    {
        if ((comando == 0x00) || (((comando & 0x7E) >> 1) == shortAddress))
        {
            tiempo_dir = 45000;      //15 minutos disponibles para recibir direcciones (apartado 9.14.2 IEC
                                     //62386.102)
            withdrawn = FALSE;
            selection = FALSE;
        }
        break;
    }
    case RANDOMISE:
    {
        randomAddress_h = htim2.Instance->CNT;  //Leemos el valor del cont del timer en ese instante
        randomAddress_m = htim2.Instance->CNT; //Leemos el valor del cont del timer en ese instante
        randomAddress_l = htim2.Instance->CNT;    //Leemos el valor del cont del timer en ese instante
        break;
    }
    case COMPARE:
    {
        randomAddress = randomAddress_h << 16 | randomAddress_m << 8 | randomAddress_l;
        searchAddress = searchAddress_h << 16 | searchAddress_m << 8 | searchAddress_l;
```

```
    if ((randomAddress != searchAddress) || (withdrawn == FALSE))
    {
        if (randomAddress <= searchAddress)
        {
            respuesta = YES;
           flag |= ENVIAR_RESPUESTA;
        }
    }
   break;
}
case WITHDRAW:
{
    withdrawn = TRUE;
    break;
 }
 case SEARCHADDRH:
 {
    searchAddress_h = comando;
    break;
 }
 case SEARCHADDRM:
 {
    searchAddress_m = comando;
    break;
  }
 case SEARCHADDRL:
 {
    searchAddress_l = comando;
    break;
  }
```

```
case PROGRAM_SHORT_ADDRESS:
{
    if (selection == TRUE)
    {
        test_FalloLamp();
        if (lampFailure == TRUE)
        {
            if (comando == MASK)
            {
                shortAddress = MASK;
            }
            else
            {
                if (((dtr0 & 0x7E) >> 1) <= 63)
                {
                    shortAddress = ((comando & 0x7E) >> 1);
                }
            }
        }
    }
    else
    {
        randomAddress = randomAddress_h << 16 | randomAddress_m << 8 | randomAddress_l;
        searchAddress = searchAddress_h << 16 | searchAddress_m << 8 | searchAddress_l;
        if (randomAddress == searchAddress)
        {
            if (comando == MASK)
            {
                shortAddress = MASK;
            }
            else
            {
                if (((dtr0 & 0x7E) >> 1) <= 63)
                {
                    shortAddress = ((comando & 0x7E) >> 1);
                }
```

```
            }
          }
        }
    break;
}
case VERIFY_SHORT_ADDRESS:
{
   if (shortAddress == ((comando & 0x7E) >> 1))
   {
       respuesta = YES;
       flag |= ENVIAR_RESPUESTA;
    }
    break;
}
case QUERY_SHORT_ADDRESS:
{
    if (selection == TRUE)
    {
        test_FalloLamp();
        if (lampFailure == TRUE)
        {
            respuesta = shortAddress;
            flag |= ENVIAR_RESPUESTA;
        }
    }
    else
    {
        randomAddress = randomAddress_h << 16 | randomAddress_m << 8 | randomAddress_l;
        searchAddress = searchAddress_h << 16 | searchAddress_m << 8 | searchAddress_l;
        if (randomAddress == searchAddress)
        {
            respuesta = shortAddress;
            flag |= ENVIAR_RESPUESTA;
        }
    }
    break;
```

```
        }
        case PHYSICAL_SELECTION:
        {
            selection = TRUE;
            break;
        }
        default:
        {
            break;
        }
    } //Fin switch (direccion)
} //Fin procesa_ComandoEspecial

//Función para procesar el resto de comandos del protocolo DALI según indica la normativa IEC62386-102
void procesa_ComandoNormal(void)
{
    switch (comando)
    {
        //Comandos de nivel
        case OFF:
        {
            powerCycleSeen = FALSE;
            set_Nivel(0);
            estado = IDLE;
            break;
        }
        case UP:
        {
            if ((actualLevel != 0) && (actualLevel != maxLevel))
            {
                fade_actualLevel = actualLevel;

                if (fade_step[fadeRate- 1] > (maxLevel - actualLevel))
                {
                    fade_targetLevel = maxLevel;
```

```
            tiempo_fade = (unsigned int) ((maxLevel - actualLevel)*10/fade_step[fadeRate - 1]);  //ms
        }                                                                    hasta alcanzar MAX_LEVEL
        else
        {
            fade_targetLevel = (actualLevel + fade_step[fadeRate - 1]);
            tiempo_fade = 10;              //Número de interrupciones necesarias para temporizar 200 ms
        }
        tiempo_estado = tiempo_fade;
        estado = FADE;
    }
    break;
}
case DOWN:
{
    if ((actualLevel != 0) && (actualLevel != minLevel))
    {
        fade_actualLevel = actualLevel;

        if (fade_step[fadeRate - 1] > (actualLevel - minLevel))
        {
            fade_targetLevel = minLevel;
            tiempo_fade = (unsigned int) ((actualLevel - minLevel)*10/fade_step[fadeRate - 1]);  //ms
        }                                                                    hasta alcanzar MIN_LEVEL
        else
        {
            fade_targetLevel = (actualLevel - fade_step[fadeRate - 1]);
            tiempo_fade = 10;              //Número de interrupciones necesarias para temporizar 200 ms
        }
        tiempo_estado = tiempo_fade;
        estado = FADE;
    }
    break;
}
```

```
case STEP_UP:
{
    if ((actualLevel != 0) && (actualLevel != maxLevel))
    {
        set_Nivel(actualLevel + 1);
        estado = IDLE;              //La transición debe de ser inmediata
     }
    break;
}
case STEP_DOWN:
{
    if ((actualLevel != 0) && (actualLevel != minLevel))
    {
        set_Nivel(actualLevel - 1);
        estado = IDLE;              //La transición debe de ser inmediata
    }
    break;
}
case RECALL_MAX_LEVEL:
{
    powerCycleSeen = FALSE;
    set_Nivel(maxLevel);
    estado = IDLE;                  //La transición debe de ser inmediata
    break;
 }
 case RECALL_MIN_LEVEL:
{
    powerCycleSeen = FALSE;
    set_Nivel(minLevel);
    estado = IDLE;                  //La transición debe de ser inmediata
    break;
}
```

```
case STEP_DOWN_AND_OFF:
{
    powerCycleSeen = FALSE;
    if (actualLevel != 0)
    {
        if (actualLevel == minLevel)
        {
            set_Nivel(0);
        }
        else
        {
            set_Nivel(actualLevel - 1);
        }
        estado = IDLE;              //La transición debe de ser inmediata
     }
    break;
}
case ON_AND_STEP_UP:
{
    powerCycleSeen = FALSE;
    if (actualLevel != maxLevel)
    {
        if (actualLevel == 0)
        {
            set_Nivel(minLevel);
        }
        else
        {
            set_Nivel(actualLevel + 1);
        }
        estado = IDLE;              //La transición debe de ser inmediata
    }
    break;
}
```

```
//Comandos de configuración
case RESET:
{
    //Valores de reset de las variables (especificados en la norma)
    powerOnLevel = 254;
    systemFailureLevel = 0254;
    minLevel = PHM;
    maxLevel = 254;
    fadeRate = 7;
    fadeTime = 0;
    randomAddress_h = 0xFF;
    randomAddress_m = 0xFF;
    randomAddress_l = 0xFF;
    searchAddress_h = 0xFF;
    searchAddress_m = 0xFF;
    searchAddress_l = 0xFF;
    powerCycleSeen = FALSE;
    limitError = FALSE;
    lampFailure = FALSE;
    group_0_7 = 0x00;
    group_8_15 = 0x00;
    for (int i=0; i<16; i++)
    {
        sceneNumber[i] = MASK;
    }
    set_Nivel(254);
    estado = IDLE;
    break;
}
case STORE_ACTUAL_LEVEL_IN_DTR0:
{
    dtr0 = actualLevel;
    break;
}
```

```
case SET_MAX_LEVEL_DTR0:
{
    if ((dtr0 >= minLevel) && (dtr0 <= 254))
    {
        maxLevel = dtr0;
    }
    break;
}
case SET_MIN_LEVEL_DTR0:
{
    if ((dtr0 >= PHM) && (dtr0 <= maxLevel))
    {
        minLevel = dtr0;
    }
    break;
}
case SET_SYSTEM_FAILURE_LEVEL_DTR0:
{
    systemFailureLevel = dtr0;
    break;
}
case SET_POWER_ON_LEVEL_DTR0:
{
    if ((dtr0 >= 1) && (dtr0 <= 254))
    {
        powerOnLevel = dtr0;
    }
    break;
}
case SET_FADE_TIME_DTR0:
{
    fadeTime = (dtr0&0x0F);
    break;
}
```

```
case SET_FADE_RATE_DTR0:
{
    if (dtr0 & 0x0F)
    {
        fadeRate = (dtr0&0x0F);
    }
    break;
}
case SET_SHORT_ADDRESS_DTR0:
{
    if (dtr0 == MASK)
    {
        shortAddress = MASK;
    }
    else
    {
        if (((dtr0 & 0x7E) >> 1) <= 63)
        {
            shortAddress = ((dtr0 & 0x7E) >> 1);
        }
    }
    break;
}
//Comandos de consulta
case QUERY_STATUS:
{
    respuesta = 0x00;
    test_FalloLamp();
    if (lampFailure == TRUE)
    {
        respuesta |= 0x02;
    }
     if (actualLevel)
     {
        respuesta |= 0x04;
     }
```

```
    if (limitError == TRUE)
    {
        respuesta |= 0x08;
    }
    if (estado == FADE)
    {
        respuesta |= 0x10;
    }
    if (estado == INIC)
    {
        respuesta |= 0x20;
    }
    if (shortAddress == MASK)
    {
        respuesta |= 0x40;
     }
     if (powerCycleSeen == TRUE)
     {
        respuesta |= 0x80;
     }
     flag |= ENVIAR_RESPUESTA;
     break;
}
case QUERY_CONTROL_GEAR_PRESENT:
{
    respuesta = YES;
    flag |= ENVIAR_RESPUESTA;
    break;
}
case QUERY_LAMP_FAILURE:
{
    test_FalloLamp();
    if (lampFailure != FALSE)
    {
        respuesta = YES;
        flag |= ENVIAR_RESPUESTA;
```

```
    }
    break;
}
case QUERY_LAMP_POWER_ON:
{
        if (actualLevel)
        {
            respuesta = YES;
            flag |= ENVIAR_RESPUESTA;
        }
        break;
}
case QUERY_LIMIT_ERROR:
{
    if (limitError == TRUE)
    {
        respuesta = YES;
        flag |= ENVIAR_RESPUESTA;
    }
   break;
}
case QUERY_RESET_STATE:
{
    if (estado == INIC)
    {
        respuesta = YES;
        flag |= ENVIAR_RESPUESTA;
    }
    break;
}
case QUERY_MISSING_SHORT_ADDRESS:
{
    if (shortAddress == MASK)
    {
        respuesta = YES;
        flag |= ENVIAR_RESPUESTA;
```

```
    }
    break;
}
case QUERY_VERSION_NUMBER:
{
    respuesta = VERSION;
    flag |= ENVIAR_RESPUESTA;
    break;
}
case QUERY_CONTENT_DTR0:
{
    respuesta = dtr0;
    flag |= ENVIAR_RESPUESTA;
    break;
}
case QUERY_DEVICE_TYPE:
{
    respuesta = 6;
    flag |= ENVIAR_RESPUESTA;
    break;
}
case QUERY_PHYSICAL_MINIMUM:
{
    respuesta = PHM;
    flag |= ENVIAR_RESPUESTA;
    break;
}
case QUERY_POWER_FAILURE:
{
    if (powerCycleSeen == TRUE)
    {
        respuesta = YES;
        flag |= ENVIAR_RESPUESTA;
    }
    break;
}
```

```
case QUERY_ACTUAL_LEVEL:
{
   respuesta = actualLevel;
   flag |= ENVIAR_RESPUESTA;
   break;
}
case QUERY_MAX_LEVEL:
{
   respuesta = maxLevel;
   flag |= ENVIAR_RESPUESTA;
   break;
}
case QUERY_MIN_LEVEL:
{
   respuesta = minLevel;
   flag |= ENVIAR_RESPUESTA;
   break;
}
case QUERY_POWER_ON_LEVEL:
{
   respuesta = powerOnLevel;
   flag |= ENVIAR_RESPUESTA;
   break;
}
case QUERY_SYSTEM_FAILURE_LEVEL:
{
   respuesta = systemFailureLevel;
   flag |= ENVIAR_RESPUESTA;
   break;
}
case QUERY_FADE_TIME_RATE:
{
   respuesta = (fadeTime << 4)|fadeRate;
   flag |= ENVIAR_RESPUESTA;
   break;
}
```

```
        case QUERY_GROUPS_0_7:
        {
            respuesta = group_0_7;
            flag |= ENVIAR_RESPUESTA;
            break;
        }
        case QUERY_GROUPS_8_15:
        {
            respuesta = group_8_15;
            flag |= ENVIAR_RESPUESTA;
            break;
        }
        case QUERY_RANDOM_ADDRESS_H:
        {
            respuesta = randomAddress_h;
            flag |= ENVIAR_RESPUESTA;
            break;
        }
        case QUERY_RANDOM_ADDRESS_M:
        {
            respuesta = randomAddress_m;
            flag |= ENVIAR_RESPUESTA;
            break;
        }
        case QUERY_RANDOM_ADDRESS_L:
        {
            respuesta =randomAddress_l;
            flag |= ENVIAR_RESPUESTA;
            break;
        }
        default:
        {
            break;
        }
    } //Fin switch (comando)
```

```
switch (comando & 0xF0)
{
    case GO_TO_SCENE:
    {
        powerCycleSeen = FALSE;
        dimming_FadeTime(sceneNumber[comando&0x0F]);
        break;
    }
    case SET_SCENE_DTR0:
    {
        sceneNumber[comando&0x0F] = dtr0;
        break;
    }
    case REMOVE_FROM_SCENE:
     {
        sceneNumber[comando&0x0F] = MASK;
        break;
    }
    case ADD_TO_GROUP:
    {
        if ((comando & 0x0F) <= 7)
        {
            group_0_7 = group_0_7|(1<<(comando&0x0F));
        }
        else
        {
            group_8_15 = group_8_15|(1<<((comando&0x0F)-8));
        }
        break;
    }
    case REMOVE_FROM_GROUP:
    {
        if ((comando & 0x0F) <= 7)
        {
            group_0_7 = group_0_7|(~(1<<(comando&0x0F)));
        }
```

```
            else
            {
                group_8_15 = group_8_15|(~(1<<((comando&0x0F)-8)));
            }
            break;
        }
        case QUERY_SCENE_LEVEL:
        {
            respuesta = sceneNumber[comando&0x0F];
            flag |= ENVIAR_RESPUESTA;
            break;
        }
        default:
        {
            break;
        }
    } //Fin switch (comando & 0xF0)
} //Fin procesa_ComandoNormal (void)

//Funciones usadas externamente

//Función de inicialización del módulo lámpara
void lampara_Inic(void)
{
    //Inicializar las variables en sus valores de POWER ON por defecto según indica la norma
    powerCycleSeen = TRUE;
    limitError = FALSE;
    lampFailure = FALSE;

    dtr0 = 0;
    actualLevel = 0;

    searchAddress_h = 0xFF;
    searchAddress_m = 0xFF;
    searchAddress_l = 0xFF;
```

```
    shortAddress = MASK;

    powerOnLevel = 254;
    systemFailureLevel = 254;
    minLevel = PHM;
    maxLevel = 254;
    fadeRate = 7;
    fadeTime = 0;
    randomAddress_h = 0xFF;
    randomAddress_m = 0xFF;
    randomAddress_l = 0xFF;
    group_0_7 = 0x00;
    group_8_15 = 0x00;
    for (int i=0; i<16; i++)
    {
        sceneNumber[i] = MASK;
    }

    tiempo_repeticion = 0;
    tiempo_dir = 0;
    tiempo_estado = 30;    //Para temporizar los 600 ms de inicialización de la lámpara como dice la norma
    estado = INIC;

    //Llamada a las funciones de configuración de los pines de los perifericos que intervienen en la operacion
    //de la lampara (pin GPIO para detectar fallo en la lámpara, timer para base de tiempos y timer para
    //generar la señal PWM que se empleará para hacer el dimming) y sus interrupciones correspondientes
    MX_GPIO_Init_lamp();
    MX_TIM1_Init();
    MX_TIM2_Init();

    //Habilitamos el canal 1 del TIMER 1 en modo PWM
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);

    //Habilitamos la interrupción de la base de tiempos (TIM2)
    HAL_TIM_Base_Start_IT(&htim2);
} //Fin lampara_Inic
```

```
//Función para el procesamiento de las tramas recibidas, considerando tanto los comandos especiales
//como los normales
void lampara_ProcesaComando(void)
{
    unsigned int grupo;

    //Comprobar si el mensaje tiene como destino este esclavo
    //Comando especial o mensaje de broadcast
    if (((direccion & 0xE1)!=0xA1)&&((direccion&0xE1)!=0xC1)&&((direccion & 0xFE)!=0xFE))
    {
        if ((direccion & 0xE0) == 0x80)                    //Dirección de grupo
        {
            grupo = (group_8_15 << 8) | group_0_7;
            if ((grupo >> ((direccion & 0x1E) >> 1) & 0x01) == 0)
            {
                return;          //No pertenece a este grupo
            }
        }
        else if ((direccion & 0x80) == 0x00)               //Short address
       {
            if (((direccion & 0x7E) >> 1) != shortAddress)
            {
                return;                    //Short address incorrecta
            }
        }
    }

    //Comprobado que el mensaje va para este esclavo.
    //Ahora comprobar si se interrumpe la repetición del dato.
    if (tiempo_repeticion != 0)
    {
        if ((direccion != direccion_rep) || (comando != comando_rep))
        {
            tiempo_repeticion = 0;              //Se limpia el contador de repetición y se ignora el mensaje
```

```
        return;
    }
}

if ((direccion & 0x01) == 0)        //Direct Arc Power Control (DAPC). El byte de opcode
{                                   //proporciona el nivel lumínico de salida deseado.
    powerCycleSeen = FALSE;
    dimming_FadeTime(comando);      //En DAPC, la transición al nuevo nivel se realiza usando
                                    //el fade time correspondiente
    return;
}

//Comprobar si el comando a manejar deber repetirse (send-twice)
else if (((direccion & 0xE0) == 0xA0) || ((direccion & 0xE0) == 0xC0))          //Comandos especiales
{
    if ((direccion == INITIALISE) || (direccion == RANDOMISE))        //Unicos comandos especiales que se
    {                                                                 //envian dos veces
        if (tiempo_repeticion == 0)
        {
            direccion_rep = direccion;
            comando_rep = comando;
            tiempo_repeticion = 5;          //Numero de interrupciones para temporizar 100 ms (tiempo de
                                            //establecimiento entre el envío del comando por primera vez y
            return;                         //el comienzo de la repetición del mismo comando).
        }
    }
}

if ((comando >= 0x20) && (comando <= 0x80))     //Comandos normales (solo los comandos en esta
{                                               //franja de opcodes son los que repiten su envio)
    if (tiempo_repeticion == 0)
    {
        direccion_rep = direccion;
        comando_rep = comando;
        tiempo_repeticion = 5;          //Numero de interrupciones para temporizar 100 ms (tiempo de
                                        //establecimiento entre el envío del comando por primera vez y
```

```
            return;                                 // el comienzo de la repetición del mismo comando).
        }
    }

    tiempo_repeticion = 0;          //El resto de comandos no son de send-twice

    if(((direccion & 0xE0) == 0xA0) || ((direccion & 0xE0) == 0xC0))          //Comandos especiales
    {
        procesa_ComandoEspecial();
    }
    else            //Comandos normales
    {
        procesa_ComandoNormal();
    }
} //Fin lampara_ProcesaComando

//Función para situar la lámpara en nivel de fallo
void lampara_NivelFallo(void)
{
    estado = IDLE;
    set_Nivel(systemFailureLevel);
}

//Interrupción del timer que establece la base de tiempos para el manejo
//de las tareas relacionadas con la lámpara. Se produce cada 20 ms.
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Instance == TIM2)
    {
        unsigned char nuevo_valor;
        unsigned char diferencial;

        //Manejo de las temporizaciones que no influyen en los estados de la lámpara
        if (tiempo_repeticion != 0)
        {
            tiempo_repeticion--;
```

```
}
else if (tiempo_dir != 0)
{
    tiempo_dir--;
}

//Manejo de los estados de la lámpara
switch (estado)
{
    case IDLE:                  //Estado de reposo. No hay cambios.
    {
        break;
    }
    case INIC:                  //Estado de inicialización.
    {
        tiempo_estado--;

        if (tiempo_estado == 0)         //Fin de la inicialización. Cambio de estado a reposo.
        {
            estado = IDLE;
            set_Nivel(powerOnLevel);    //Fijamos el nivel de encendido. Por defecto, máximo de
        }                               //luminosidad.
        break;
    }
    case FADE:                  //Estado de transición del nivel actual al nuevo nivel indicado.
    {
        tiempo_estado--;

        if (tiempo_estado == 0)         //Fin del fade. Cambio de estado a reposo.
        {
            estado = IDLE;
        }
        //Calculamos el nuevo nivel a fijar
        if (fade_actualLevel > fade_targetLevel)        //Fade descendente
        {
            if (fade_targetLevel == 0)
```

```
    {
        if (tiempo_estado == 0)
        {
            nuevo_valor = 0;
        }
        else                    //Usar MIN_LEVEL en lugar de 0
        {
            //Transición lineal del nivel actual al objetivo
            diferencial = (unsigned char) ((((unsigned long)fade_actualLevel - (unsigned long)
                        minLevel)*((unsigned long)tiempo_fade – (unsignedlong)tiempo_estado))/
                        (unsigned long)tiempo_fade);
            nuevo_valor = (unsigned char)fade_actualLevel - diferencial;
        }
    }
    else                        //Usar el targetLevel
    {
        //Transición lineal del nivel actual al objetivo
        diferencial = (unsigned char) ((((unsigned long)fade_actualLevel - (unsigned long)
                    fade_targetLevel)*((unsigned long)tiempo_fade - (unsigned
                    long)tiempo_estado))/( unsigned long)tiempo_fade);
        nuevo_valor = (unsigned char)fade_actualLevel - diferencial;
    }
}
else                //Fade ascendente
{
    //Transición lineal del nivel actual al objetivo
    diferencial = (unsigned char) ((((unsigned long)fade_targetLevel - (unsigned long)
                fade_actualLevel)*((unsigned long)tiempo_fade - (unsigned
                long)tiempo_estado))/(unsigned long)tiempo_fade);
   nuevo_valor = (unsigned char)fade_actualLevel + diferencial;
}

if (nuevo_valor != actualLevel)
{
    set_Nivel(nuevo_valor);
}
```

```
            break;
        }
        default:
        {
            break;
        }
    } //Fin switch(estado)
  } //Fin if (htim->Instance == TIM2)
} //Fin de la interrupción
```

main.c

```
//Autor: Miguel Garnelo Rodríguez
//Grado en Ingeniería Electrónica Industrial y Automática
//Trabajo Fin de Grado - Julio 2018
//Escuela Politécnica de Ingeniería de Gijón
//Universidad de Oviedo

//Includes
#include "main.h"
#include "stm32f0xx_hal.h"
#include "stm32f0xx.h"
#include "stm32f0xx_it.h"
#include "slave_dali_comandos.h"
#include "slave_dali_comun.h"
#include "slave_dali_protocol.h"
#include "slave_dali_lampara.h"

//Variables globales
extern TIM_HandleTypeDef htim2;
unsigned char direccion;        //Almacena la dirección enviada por el maestro DALI
unsigned char comando;          //Almacena el comando enviado por el maestro DALI
unsigned char respuesta;        //Almacena la respuesta que el esclavo DALI envía al maestro
unsigned int flag;              //Flag para indicar todos los eventos que se producen
unsigned char recep_on;         //Variable para indicar si la recepción ha comenzado
```

```
int main(void)
{
   //Inicialización del driver HAL, variables globlales, comunicaciones DALI y actividad de la lámpara
   HAL_Init();

   direccion = 0x00;
   comando = 0x00;
   respuesta = 0x00;
   flag = 0x00;

   lampara_Inic();
   dali_Inic();

   while (1)        //Bucle de ejecución continua. Supervisa y controla todos los procesos
   {
      if (flag & NUEVO_DATO)       //Recepción de una nueva trama DALI enviada por un dispositivo
      {                            //maestro a través del bus. Nuevo comando a procesar.
         flag &= ~NUEVO_DATO;      //Borrado del flag
         lampara_ProcesaComando();       //Procesamiento de la nueva información recibida
         recep_on = FALSE;               //Indicamos que la recepción ha terminado
      }
      else if (flag & ENVIAR_RESPUESTA)    //Envío de respuesta correspondiente (si tiene que haberla) tras
      {                                    //procesar una nueva trama
         flag &= ~ENVIAR_RESPUESTA;        //Borrado del flag
         dali_EnvioRespuesta();            //Envío de la respuesta al maestro a través del bus DALI
      }
      else if (flag & ERROR_BUS)           //Manejo de error en el sistema
      {
         flag &= ~ERROR_BUS;               //Borrado del flag
         HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_SET);      //Encendemos el led indicador de
                                                                //fallo en el bus
         lampara_NivelFallo();       //Lampara situada en el nivel de fallo fijado
      }
   } //Fin while (1)
} //Fin main
```

```
/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void _Error_Handler(char * file, int line)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */
    while(1)
    {
    }
    /* USER CODE END Error_Handler_Debug */
}
#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 CODE BEGIN 6 */
    /* 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) */
    /* USER CODE END 6 */
 }
#endif
/**
  * @}
  */
/**
  * @}
*/
```