Proyecto Fin de Carrera

DISEÑO Y FABRICACION DE DISPOSITIVO TÁCTIL PARA EL MANEJO Y CONTROL DE LUCES

AUTOR: MIGUEL PEREZ MAESTRO

ING. SUP. TELECOMUNICACIONES

TUTOR: ENRIQUE PREFASI

Leganés, Mayo de 2017. DPTO. TECNOLOGÍA ELECTRÓNICA

Agradecimientos

A todos mis compañeros de la universidad, con los que he compartido muchos momentos y me han ayudado enormemente a superar los pequeños baches que me han surgido durante la carrera. A los profesores que realmente han conseguido despertar en mí un gran interés por las materias que impartían. Especialmente a mi familia, que siempre ha creído en mí y lo han demostrado mostrando su apoyo incondicional.

Resumen del proyecto

El proyecto que se presenta en este documento, consiste en el diseño y fabricación de un dispositivo domótico, que aporta modularidad al usuario, permitiéndole configurar el dispositivo según las necesidades de su hogar, mediante un menú sencillo de configuración, en el cual el usuario establece el tipo y número de aparatos electrónicos que van a ser conectados al dispositivo.

Para llevar a cabo las fases de diseño y fabricación del prototipo que servirá como ejemplo durante la presentación de este proyecto, se han utilizado distintos programas y tecnologías. Estos programas y tecnologías serán introducidos a lo largo del documento.

El dispositivo presentado, estará fijado a la pared y estará conectado a los aparatos que va a controlar mediante el cableado existente en el edificio.

Una vez el dispositivo está configurado, el usuario podrá realizar diferentes funciones con cada uno de los aparatos eléctricos a los que está conectado. El usuario podrá realizar estas funciones mediante la interacción con una pantalla táctil capacitiva, en la que se selecciona el aparato eléctrico sobre el que se desea actuar, así como la función a desempeñar sobre ella.

Las funcionalidades que ofrece el dispositivo para los aparatos conectados, son Encender/Apagar, Subir/Bajar intensidad o programar el encendido/apagado. Los aparatos que pueden ser conectados son las luces, sobre las que se pueden realizar todas las acciones descritas, al igual que para los ventiladores, o persianas, que en este caso solo se podrá Subir/Bajar o programar la Subida/Bajada de la persiana.

Todas estas funciones estarán integradas en el dispositivo en el cual se basa este proyecto. Este dispositivo es un substituto ideal para las cajas de interruptores mecánicos con varios interruptores, aportando nuevas funcionalidades y un diseño innovador, además de la entrada en modo “Ahorro de energía” cuando el dispositivo no está siendo utilizado.

La aportación de este dispositivo respecto a lo existente actualmente en el mercado, será la modularidad que ofrece la fácil configuración de la que dispone, mediante la cual el usuario podrá establecer qué número y tipos de cada dispositivo van a ser conectados. Otra de las ventajas que el dispositivo aporta será el reducido precio del mismo, en comparación con los dispositivos actuales que realizan estas funciones.

Tabla de contenido

Agradecimientos 3

Resumen del proyecto 5

ÍNDICE DE FIGURAS: ..... **¡Error! Marcador no definido.**

INDICE DE TABLAS: 11

1. Motivación y objetivos 12

1.1 Motivación 12

1.2 Objetivos 13

2. Estado del arte 16

2.1 Tecnologías relacionadas existentes actualmente 17

2.2 Integración de los dispositivos involucrados 20

3. Parte Hardware del proyecto 22

3.1 Pantalla TFT-LCD con panel táctil capacitivo y micro de vídeo: 23

3.1.1 Micro de vídeo RA8875 de RAIO 25

3.1.2 Pantalla táctil capacitiva y chip controlador de pantalla táctil: 26

3.2 Fuente de alimentación 30

3.3 Driver de corriente 32

3.3.1 Calculo de los valores de las resistencias 33

3.3.2 Detector de paso por cero 35

3.4 Placas de circuito impreso (PCBs) y esquemático 38

3.4.1 Diseño electrónico y Altium Designer 38

3.4.2 Diseño de las PCBs 39

3.5 Microcontrolador 41

3.5.1 Relaciones entre el micro maestro y los periféricos 43

3.5.2 Kit de desarrollo STM32F4Discovery 46

4. Parte Software del proyecto 49

4.1 Funciones a desempeñar 50

4.2 Menús de la aplicación 51

4.2.1 Menú de configuración de fecha y hora 51

4.2.2 Menú principal o de luces 52

4.2.3 Menú de regulación de intensidad 53

4.2.4 Menú de selección de tipo de programación 54

4.2.5 Menú de programación de las salidas 55

4.2.6 Diagrama de flujo de los menús 57
4.3 Implementación Software de las funcionalidades a desempeñar 58
4.3.1 Regulación de intensidad 58
4.3.2 Detección de pulsaciones 59
4.3.3 Zonas sensibles de pulsación 60
4.3.4 Función ActionButton 65
4.3.5 Actuadores 66
4.3.6 Entrada en modo ahorro de energía 76
4.3.7 Interrupciones 77
5. Presupuesto 80
5.1 Resumen costes Hardware 84
6. Conclusiones 85
7. Futuras mejoras 88
7.1 ETHERNET 88
7.2 Bombillas de bajo consumo 90
8. ANEXOS 91
8.1 ANEXO I - Características de la pantalla: 91
8.2 ANEXO II – Características del chip GSL1680 94
8.3 ANEXO III – Esquemático completo y PCBs 95
8.4 ANEXO IV – Datasheet del optoacoplador 97
8.5 ANEXO V – Descripción de pines y funciones alternativas del micro 98
8.6 ANEXO VI – Estructura principal y sub-estructuras 107
8.7 ANEXO VII – Código de la aplicación 111
8.7.1 APP_Touch 111
8.7.2 Dimmer 134
8.7.3 Lights 140

ÍNDICE DE FIGURAS:

***Figura 1*: *Diagrama de Gantt estimado de las tareas requeridas* 15**

***Figura 2:** Forma de onda de la red eléctrica* 16

***Figura 3:** Circuito de interruptor simple* 17

***Figura 4:** Integración de los dispositivos* 20

***Figura 5:** Distribución de capas del prototipo* 22

***Figura 6:** Pantalla LCD ER-TFT-M050-3* 23

***Figura 7:** Enumeración de los accesorios de la pantalla LCD* 24

***Figura 8:** Malla de electrodos en pantallas táctiles* 27

***Figura 9:** Pines para la comunicación del chip GSL1680* 28

***Figura 10:** Esquema de la fuente de alimentación* 31

***Figura 11:** Circuito Interruptor digital* 32

***Figura 12:** Circuito Detector de paso por cero* 35

***Figura 13:** Formas de onda del detector de paso por cero*..... 36

***Figura 14**: Formas de onda de los Triacs de potencia en función de la intensidad*..... 37

***Figura 15:** Funcionalidades Altium Designer* 38

***Figura 16:** Imagen del chip utilizado en el proyecto* 41

***Figura 17:** Instrucciones de lectura/escritura en el micro de vídeo*.....43

***Figura 18:** Kit de desarrollo STM32F4Discovery* 48

***Figura 19:** Configuración jumpers CN3* 49

***Figura 20:** Imagen del menú de configuración de fecha y hora* 51

***Figura 21:** Imagen del menú principal con todas las luces apagadas*..... 52

***Figura 22:** Imagen del menú principal con dos luces encendidas* 52

***Figura 23:** Imagen del menú de regulación de intensidad* 53

***Figura 24:** Imagen del menú de selección de tipo de programado*..... 54

***Figura 25:** Imagen del menú de programación de luces* 55

***Figura 26:** Menú de programación de luces con programación de encendido activada* 55

***Figura 27:** Diagrama de flujo de los menús* 57

***Figura 28:** Definición de zona sensible de pulsación* 60

Figura 29: *Zonas sensibles de pulsación del menú de fecha y hora …… 61*

Figura 30: *Zonas sensibles de pulsación del menú principal …… 62*

Figura 31: *Zonas sensibles de pulsación del menú de regulación de intensidad …….. 62*

Figura 32: *Zonas sensibles de pulsación del menú de selección de tipo de programación … 63*

Figura 33: *Zonas sensibles de pulsación del menú programación …… 63*

Figura 34: *Primera zona sensible de pulsación del menú de fecha y hora …… 66*

Figura 35: *Primera zona sensible de pulsación del menú de fecha y hora …… 87*

Figura 36: *Esquemático conector RJ45 …… 88*

Figura 37: *Huella del conector RJ45 …… 89*

Figura 38: *Dimensiones de la pantalla LCD …… 91*

Figura 39: *Descripción de pines de la interfaz paralela de la pantalla LCD …… 92*

Figura 40: *Características eléctricas de la pantalla LCD …… 93*

Figura 41: *Esquemático del chip GSL1680 …… 94*

Figura 42: *Esquemático completo …… 95*

Figura 43: *Placa framework …… 96*

Figura 44: *Placa Driver de corriente …… 96*

Figura 45: *Características del optoacoplador VO3053 …… 97*

Figura 46: *Descripción de pines del micro STM32F407VGT6…… 98*

Figura 47: *Descripción de las funciones alternativas del puerto A del micro maestro…… 99*

Figura 48: *Descripción de las funciones alternativas del puerto B del micro maestro…… 100*

Figura 49: *Descripción de las funciones alternativas del puerto C del micro maestro…… 101*

Figura 50: *Descripción de las funciones alternativas del puerto D del micro maestro…… 102*

Figura 51: *Descripción de las funciones alternativas del puerto E del micro maestro…… 103*

INDICE DE TABLAS:

Tabla 1: *Voltajes y corrientes requeridos 30*

Tabla 2: *Pines utilizados para la comunicación con la pantalla táctil 44*

Tabla 3: *Pines utilizados por el driver de corriente 45*

Tabla 4*: Pines utilizados para la interfaz de programación del micro (SWD) 45*

Tabla 5: *Pines interfaz SWD 48*

Tabla 6: *Prioridades del proyecto Software 79*

Tabla 7: *Presupuesto pantalla TFT 80*

Tabla 8: *Presupuesto Fuente alimentación 81*

Tabla 9*: Presupuesto placas de circuito impreso 81*

Tabla 10: *Presupuesto conectores 82*

Tabla 11: *Presupuesto microprocesador y placa evaluación 82*

Tabla 12: *Presupuesto driver de corriente 83*

Tabla 13: *Presupuesto global 84*

Tabla 14: *Pines asociados a funciones Ethernet 89*

Tabla 15: *Pines asociados a funciones Ethernet 104*

Tabla 16: *Pines utilizados para la interfaz FSMC 105*

Tabla 17: *Pines de control para la interfaz FSMC 106*

1. Motivación y objetivos

1.1 Motivación

Este proyecto, pretende ser una inspiración para mí mismo, de forma que a partir de una idea, pueda llegar a producir un producto que satisfaga las necesidades de esa idea.

Desde que tengo uso de razón, he estado viendo los mismos interruptores mecánicos en las paredes. Dada la cantidad de avances tecnológicos que se han acontecido en los últimos años, me surgió la curiosidad de diseñar y producir un tipo diferente de interruptor, al cual añadir algunas funcionalidades nuevas para que el resultado no solo sea una mejora visual del mismo, sino también una mejora de sus funcionalidades.

Dado el coste de los interruptores mecánicos, unos 15€ aprox. por interruptor simple, comencé este proyecto con la intención de conseguir un producto, que sin alejarse mucho del precio del interruptor mecánico, consiguiera un nuevo interruptor, con un diseño mucho más innovador y con funcionalidades añadidas.

1.2 Objetivos

En este apartado se enumeran todos los objetivos que se pretenden conseguir durante la realización del proyecto. Estos objetivos serán analizados al final del proyecto y se expondrán las conclusiones de los mismos en el capítulo 6 (Conclusiones).

1. Llevar a cabo la fabricación de un prototipo que aporte modularidad al usuario, permitiéndole una configuración del dispositivo sencilla, en la que pueda escoger la cantidad de aparatos a la que va a ser conectado.

2. El dispositivo deberá llevar a cabo las funcionalidades para las cuales ha sido diseñado, permitiendo al usuario activar/desactivar, regular en intensidad y programar el encendido/apagado de todos los aparatos conectados al mismo.

3. Configurar el panel táctil correctamente, de manera que el dispositivo sea capaz de reconocer las pulsaciones de la pantalla de manera exacta, siendo capaz de identificar las coordenadas de la pulsación.

4. Establecer la comunicación entre el microcontrolador maestro y el micro de vídeo de la pantalla LCD, lo que permitirá la visualización en la pantalla de los menús de los cuales dispone la aplicación.

5. Conseguir la integración de todos los módulos independientes para que estos puedan comunicarse entre ellos llevando a cabo las funciones para las que han sido diseñados. Estos módulos son:

   - Un panel táctil

   - Una pantalla LCD con micro de vídeo

   - La electrónica necesaria

   - Un microcontrolador que gobierne el sistema

6. Implementar una fuente de alimentación que permita el funcionamiento de los componentes electrónicos de los que se compone el prototipo. Proporcionando a todos los componentes los voltajes y corrientes adecuados para su funcionamiento.

7. Encontrar el equilibrio entre los precios de los componentes utilizados, el rendimiento que ofrecen y las necesidades requeridas de los mismos para el proyecto.

8. Conseguir que el conjunto del dispositivo suponga un ahorro económico en comparación con los dispositivos completos con funcionalidades parecidas ya existentes en el mercado.

9. Encontrar un Software gratuito que sea adecuado para la programación del dispositivo, el cual aporte facilidades para el desarrollo de la aplicación y para el grabado en la memoria flash del micro maestro utilizado.

En la siguiente imagen se muestra el diagrama de Gantt del proyecto, donde se pueden observar los tiempos estimados que van a requerir cada una de las tareas requeridas:

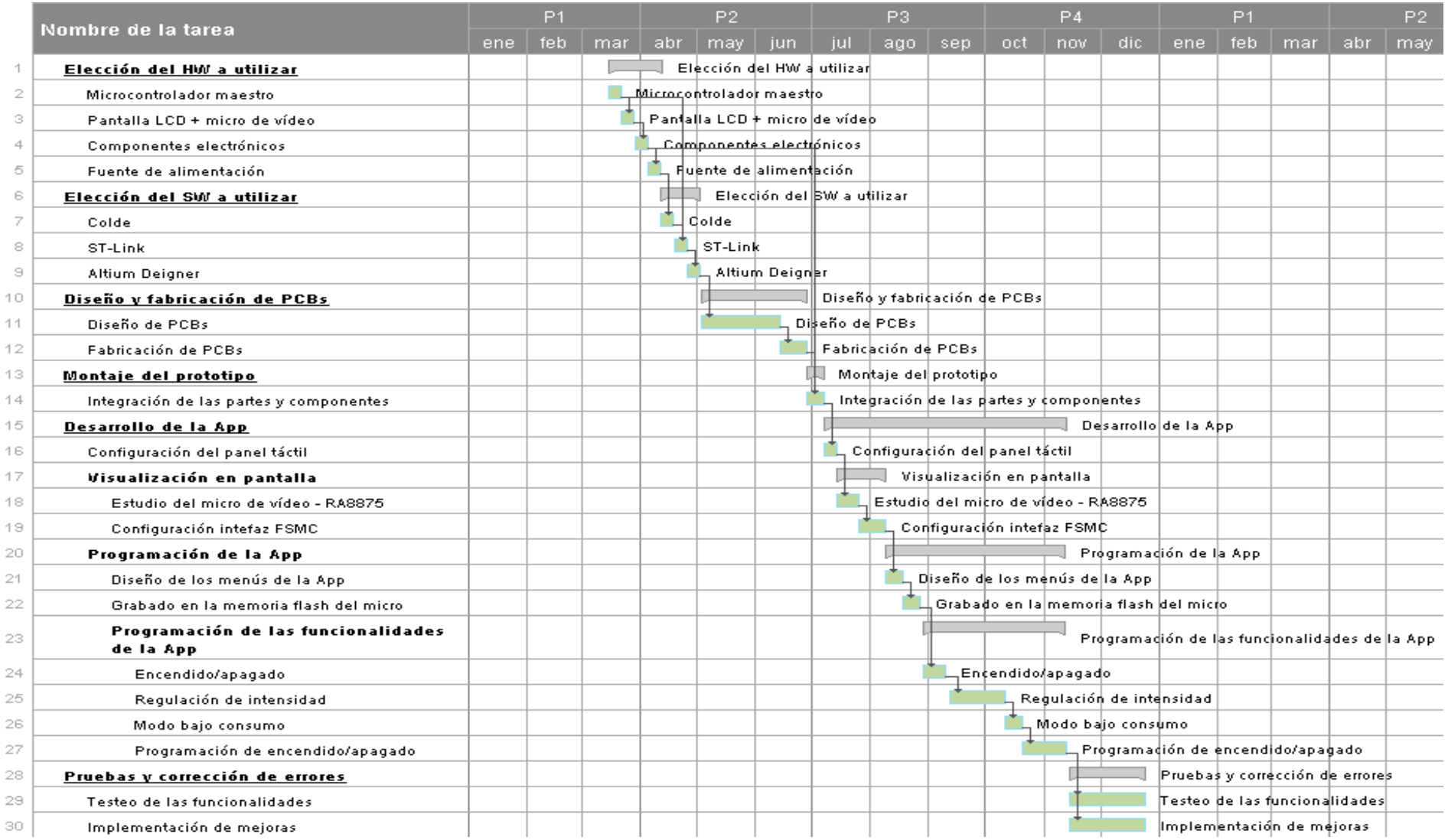


Figura 1: Diagrama de Gantt estimado de las tareas requeridas

2. Estado del arte

En este apartado se estudian y analizan los dispositivos actuales que existen en el mercado cuyo propósito es similar al del dispositivo diseñado en este proyecto. Antes de entrar en detalle sobre las tecnologías existentes actualmente, repasamos las características de la línea de red de las viviendas en España, así como el concepto de interruptor.

Características de la línea de red de las viviendas en España

En las viviendas españolas, disponemos de una línea eléctrica que nos permite conectar a la red eléctrica todo tipo de aparatos y electrodomésticos. Esta línea de corriente tiene la siguiente forma:

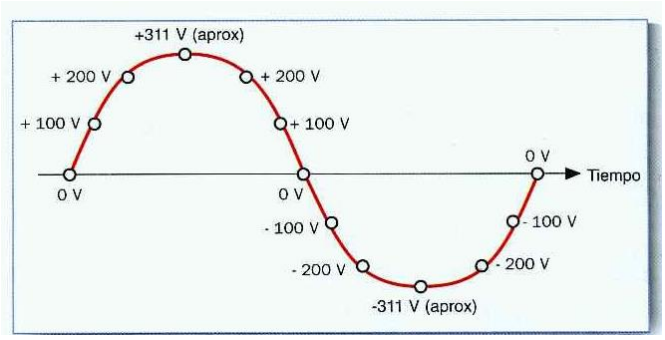


Figura 2: Forma de onda de la red eléctrica

En la figura anterior se puede observar que la línea de corriente, ofrece una corriente alterna de 230 x $\sqrt{2}$ Voltios eficaces, a una frecuencia de 50 Hz.

Las conexiones disponibles en las viviendas españolas son:

- **Fase**: La cual tiene 230 voltios de pico y es corriente alterna (sinusoidal a 50 Hz).
- **Neutro**: Esta línea cierra el circuito una vez ha pasado la corriente por la carga, no hay tensión pero sí circula corriente.
- **Tierra**: Se encarga de poner partes del circuito a un lugar sin potencial eléctrico.

Concepto de interruptor

El funcionamiento de un interruptor consiste en interrumpir o dejar pasar la corriente eléctrica mediante un contacto, al que normalmente está conectado una lámpara o cualquier otro tipo de aparato eléctrico. El siguiente esquema resume el funcionamiento de un interruptor simple:

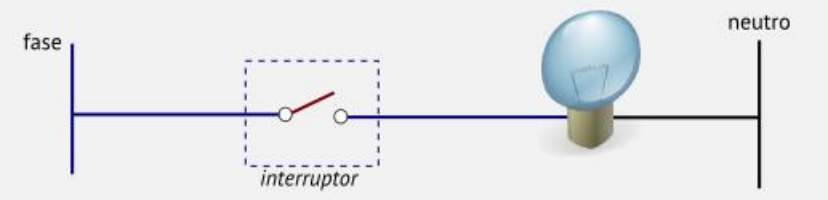


Figura 3: Circuito de interruptor simple

2.1 Tecnologías relacionadas existentes actualmente

Interruptores táctiles

En la actualidad, existen a la venta dispositivos como los interruptores táctiles, que no son más que interruptores mecánicos activados mediante la pulsación en un panel táctil. Esta pulsación es reconocida por el panel y se interpreta como un cambio en el contacto del interruptor, lo que permite/corta el paso de corriente a través del aparato que esté conectado a dicho interruptor.

Estos interruptores están en torno a los 15€ y la aportación frente a los interruptores habituales es muy limitada, ya que la única diferencia entre ellos es el panel táctil con el que se activan, en lugar del botón de los interruptores mecánicos habituales.

Interruptores programables

Los interruptores programables, son utilizados en aquellas ocasiones en las que se necesite cortar el paso de corriente a través del aparato eléctrico conectado a una hora determinada, o al cabo de un tiempo concreto.

La gran ventaja de estos interruptores frente a los interruptores mecánicos habituales, es que no es necesaria la presencia de una persona que active/desactive el interruptor, sino que el propio interruptor programable, como su nombre indica, permite programar la hora de activación/desactivación del aparato al que está conectado.

Este tipo de interruptores, ofrecen la ventaja anteriormente comentada, y aunque existen muchos formatos y modelos, el precio medio de un interruptor programable estándar está en torno a los 15 €.

Pantallas LCD

Las pantallas LCD son pantallas delgadas de cristal líquido formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz, habitualmente LEDs microscópicos.

Existen multitud de tipos de pantallas LCD, entre sus atributos se encuentra el tamaño o la resolución. Ambos atributos encarecen el precio de la pantalla a medida que crecen. Por ejemplo, el tamaño de las pantallas se mide en pulgadas, y a mayor número de pulgadas mayor es el precio de la pantalla. Este aspecto hay que combinarlo con la resolución de la pantalla, la cual se mide en píxeles cuadrados.

En el caso del proyecto estamos utilizando una pantalla LCD de 5 pulgadas y 800x480 píxeles de resolución.

Panel táctil

Un panel táctil es una fina lámina transparente basada en sensores, pudiendo ser estos sensores resistivos o capacitivos. Los sensores reconocen la pulsación en la pantalla y envían las coordenadas de la pulsación en forma de datos al dispositivo que se lo esté requiriendo, generalmente un microcontrolador.

Estos paneles táctiles suelen integrarse con pantallas LCD, de tal forma que tengan el mismo tamaño, y quede superpuesto el panel táctil transparente en la pantalla, de tal forma que el reconocimiento de la pulsación en el panel táctil, pueda ser interpretado como la pulsación en esa misma coordenada de la pantalla LCD.

Sistemas domóticos completos

Los sistemas domóticos completos son desde hace ya varios años utilizados frecuentemente en las viviendas. Estos sistemas permiten controlar los aparatos eléctricos a los que están conectados, que frecuentemente son las luces, persianas y ventiladores de los hogares.

Existen diferentes modelos de sistemas domóticos, cada uno de ellos permite el control de los aparatos conectados de distintas formas. Aunque todos los sistemas domóticos aportan facilidades para el control de los aparatos, dependiendo de la gama del sistema domótico, podremos realizar el control de los aparatos mediante paneles táctiles, mandos a distancia o incluso mediante aplicaciones web que podremos utilizar desde móviles u ordenadores.

El precio de este tipo de sistemas es muy variado, dependiendo de las funcionalidades que ofrezca así como de su diseño, pueden llegar a costar varios miles de euros incluyendo la instalación.

2.2 Integración de los dispositivos involucrados

Como veremos en apartados posteriores, el prototipo creado para la realización de este proyecto, es un dispositivo que permite el control de aparatos eléctricos (que serán luces en el caso de exposición del proyecto), tal y como realizan los sistemas domóticos completos. El usuario podrá realizar el control de dichos aparatos mediante la utilización de un panel táctil conectado a una pantalla LCD, donde todos los componentes están gobernados por un microcontrolador.

Para conseguir lo expuesto en el párrafo anterior, el dispositivo que se ha fabricado debe ser una integración de las tecnologías existentes actualmente que han sido expuestas en el apartado 2.1.

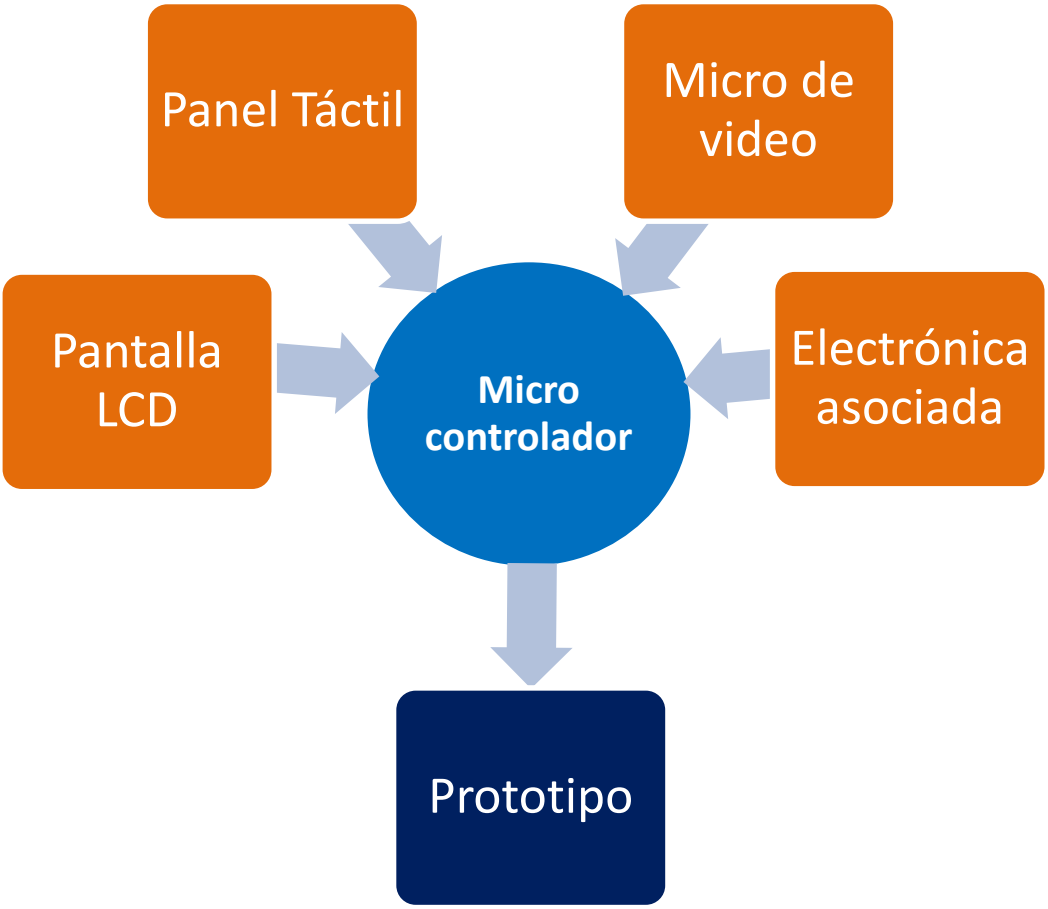


Figura 4: Integración de los dispositivos

Esta integración consistirá en hacer funcionar conjuntamente las tecnologías expuestas en el apartado anterior, de tal forma que el panel táctil esté superpuesto a una pantalla LCD del mismo tamaño, y ambos dispositivos sean gobernados por un microcontrolador que controle y de las órdenes necesarias a todas las partes del prototipo. Formando un sistema domótico completo que aporte modularidad al usuario a un precio competitivo.

La integración de estos dispositivos será una tarea clave durante la realización del proyecto, ya que se debe conseguir la intercomunicación entre ellos para lograr que el prototipo resultante realice todas las funciones para las que ha sido diseñado.

3. Parte Hardware del proyecto

Se denomina parte Hardware, a la parte del proyecto destinada al análisis, diseño e implementación de las partes físicas necesarias para el correcto funcionamiento del dispositivo que se pretende crear durante la realización de este proyecto. En este apartado entraremos en detalle de cada una de las partes Hardware involucradas en el proyecto.

El proyecto en sí, consiste en la fabricación de un prototipo a través del cual el usuario pueda realizar una serie de funciones, descritas en el apartado 4.1, mediante la utilización de una pantalla táctil. Para conseguir este objetivo, es necesario que el prototipo disponga de un panel táctil, una pantalla LCD, la electrónica necesaria para que el prototipo lleve a cabo las funciones para las cuales es diseñado, y un microcontrolador que gobierne el sistema actuando como maestro de todos los periféricos involucrados. Además, será necesario alimentar el sistema mediante una fuente de alimentación, que permita el funcionamiento de los componentes electrónicos de los que se compone el prototipo.

El esquema global del prototipo resultante se muestra en la imagen siguiente:

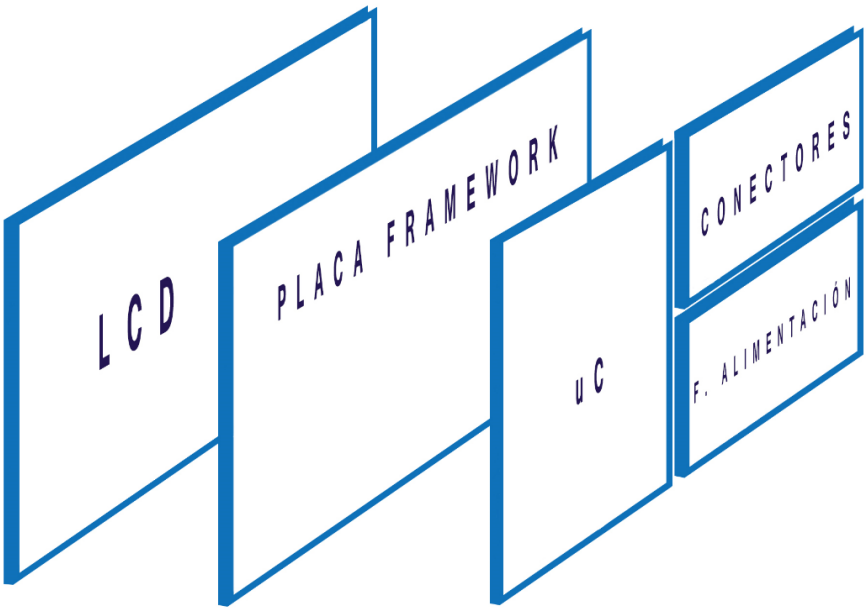


Figura 5: Distribución de capas del prototipo

En los siguientes sub-apartados, se van describiendo cada una de las partes físicas involucradas en el proyecto. Se utiliza la denominación “micro maestro”, haciendo referencia al microcontrolador STM32F407VGT6 que gobierna el prototipo.

3.1 Pantalla TFT-LCD con panel táctil capacitivo y micro de vídeo:

Para facilitar la integración de los componentes del prototipo y proporcionar una calidad de imagen óptima, se ha escogido una pantalla de cristal líquido (LCD) de transistores de película fina (TFT). La pantalla escogida, dispone de un panel táctil capacitivo integrado, así como de un micro de vídeo encargado de controlar la visualización en la pantalla. También dispone de accesorios para comunicación por interfaz paralela entre el micro maestro y el micro de vídeo integrado en la pantalla.

Esta pantalla, del fabricante EastRising, es el modelo ER-TFT-M050-3, y ha sido adquirida a través de BuyDisplay.com. En el apartado 5, presupuesto, viene indicado el valor de compra de esta pantalla así como de los demás componentes del proyecto.

A continuación se muestra la imagen de la pantalla proporcionada por el vendedor:

Figura 6: Pantalla LCD ER-TFT-M050-3

Aunque la pantalla utilizada para la fabricación del prototipo lleve integrado el panel táctil para el reconocimiento de pulsaciones, será necesaria la conexión del chip controlador del panel táctil al microcontrolador maestro para el procesamiento de los datos que envíe el chip controlador de panel táctil.

En la siguiente imagen, además del micro de vídeo RA8875 de RAIO, que viene incorporado en la pantalla, se muestran de forma enumerada los accesorios de los cuales se compone la pantalla desde su vista posterior:

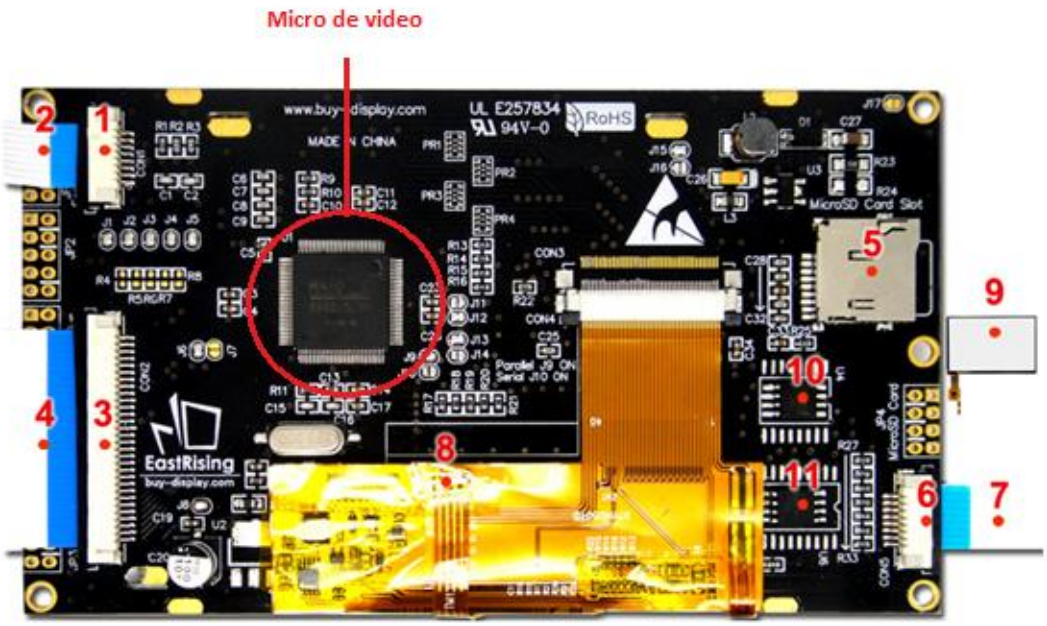


Figura 7: Enumeración de los accesorios de la pantalla LCD

A continuación se describen los accesorios enumerados:

1 y 2: Conexiones para la comunicación serie entre el microcontrolador maestro y el micro de vídeo de la pantalla. 1 se corresponde con un conector ZIF de 8 pines. 2 se corresponde con un conector FCC de 8 pines.

3 y 4: Conexiones para la comunicación en paralelo entre el microcontrolador maestro y el micro de vídeo de la pantalla. 3 se corresponde con un conector ZIF de 30 pines. 4 se corresponde con un conector FCC de 30 pines.

5: Slot para tarjeta MicroSD.

6 y 7: Conexiones para el manejo del slot MicroSD. 6 se corresponde con un conector ZIF de 8 pines. 7 se corresponde con un conector FCC de 8 pines.

8: Panel táctil resistivo de 5 pulgadas y 4 cables. (Por haberse escogido una pantalla capacitiva y no resistiva, esta opción no está montada en el proyecto, está la 9 en su lugar).

9: Panel táctil capacitivo mediante chip GSL1680 con 6 pines de salida.

10: Chip de memoria flash de 128Mb.

11: Chip de fuente para escribir textos en la pantalla.

Las características de la pantalla pueden observarse en el Anexo I, en la parte final de este documento.

3.1.1 Micro de vídeo RA8875 de RAIO

El micro de vídeo, es el chip que se encarga de controlar la visualización en la pantalla TFT-LCD. El modelo de micro utilizado, que viene integrado en la pantalla, es el RA8875 de RAIO. Este micro es utilizado como controlador de pantallas de hasta 800x480 píxeles, dispone de una memoria RAM embebida de 768Kb.

Dispone de una serie de registros que proporcionan las funcionalidades que este micro de vídeo ofrece. Algunas de las funcionalidades incluidas son, dibujar figuras geométricas en la pantalla o escribir texto de distintos tamaños y colores.

Para una mayor velocidad en la comunicación entre el micro maestro y el micro de vídeo, la comunicación se realiza en paralelo a través de la interfaz destinada para ello (3 y 4 en la descripción de los accesorios de la pantalla).

La descripción de los 30 pines de la interfaz paralela encargada de la comunicación entre ambos micros, se puede observar en el Anexo I de este documento.

En el proyecto, se ha utilizado la interfaz paralela, y ha sido configurada en modo 16 bits y 8080 bps. Los 16 bits de datos que se pueden mandar en paralelo entre el micro maestro y la pantalla, se encuentran en los pines 15 a 30 de esta interfaz.

El dispositivo funciona a un voltaje continuo de 3.3 voltios (V), y consume una corriente máxima de 450 miliamperios (mA), estos datos pueden observarse en la tabla de características eléctricas de la pantalla LCD, ubicada en el Anexo I de este documento.

3.1.2 Pantalla táctil capacitiva y chip controlador de pantalla táctil:

Las pantallas táctiles, son capaces de reconocer las pulsaciones del usuario en su superficie. Para poder realizar esta tarea, cada pantalla táctil, lleva un chip controlador de la pantalla, que es el encargado de gestionar las pulsaciones recibidas en la superficie, e interpretar dichas pulsaciones mediante el reconocimiento de las coordenadas de la pulsación en la pantalla.

Existen dos tipos de pantallas táctiles las resistivas y las capacitivas. Vamos a centrarnos en las pantallas capacitivas, ya que es el tipo de pantalla táctil que ha sido utilizada en este proyecto.

Las pantallas táctiles capacitivas están construidas por cierto número de electrodos sobre una película de cristal o substrato. Dichos electrodos suelen estar formados de óxido de indio y estaño (ITO). Ese material, se conforma en tiras extendidas en el eje X y en el eje Y como muestra la siguiente figura, la intersección de cada tira da lugar a un sensor capacitivo.

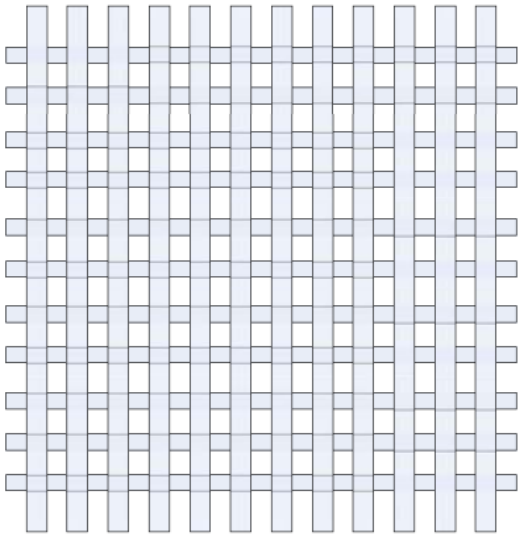


Figura 8: Malla de electrodos en pantallas táctiles

El sensor por tanto muestra un campo de electrones controlado con precisión tanto en el eje vertical como en el horizontal. El cuerpo humano también se puede considerar un dispositivo eléctrico, por lo que cuando el campo de capacitancia normal del sensor (su estado de referencia) es alterado por otro campo de capacitancia, como puede ser el dedo de una persona, los circuitos electrónicos situados en cada esquina de la pantalla miden la 'distorsión' resultante en la onda sinodal característica del campo de referencia y envía la información acerca de este evento al controlador para su procesamiento matemático.

La pantalla táctil que viene integrada con la pantalla TFT-LCD utilizada durante este proyecto, que ha sido presentada en el apartado 3.1, posee las siguientes características:

- Es capaz de detectar hasta 10 dedos de forma simultánea.

- Tiene una frecuencia máxima de pulsación de 200Hz.

- Provee una interfaz I2C compatible para la comunicación con el micro maestro, con una velocidad de transmisión de hasta 400KHz.

- La alimentación requerida para la pantalla táctil es de 3.3 voltios, y su consumo máximo de corriente es de 9 mA.

El chip controlador de la pantalla táctil, almacena las coordenadas de las pulsaciones recibidas en la pantalla. La comunicación entre el micro maestro y la pantalla táctil, es necesaria para que el micro maestro pueda recibir las coordenadas de los toques que se hayan producido en la pantalla, para posteriormente actuar en consecuencia según se explicará en el capítulo posterior (Parte Software del proyecto).

El chip controlador de pantalla táctil que viene integrado junto con la pantalla LCD, es el chip GSL1680 de Silead, y su descripción de pines puede observarse en el Anexo II.

Los pines de los que dispone el chip, son en su mayoría, los canales de escaneo del chip, hasta 16 canales en el eje X y 10 en el eje Y. El esquemático del circuito integrado que permite el funcionamiento del chip de control de pantalla táctil se encuentra en el Anexo II de este documento.

La comunicación entre el micro maestro y el chip controlador de pantalla táctil, se realiza mediante los siguientes pines de los que dispone el controlador:

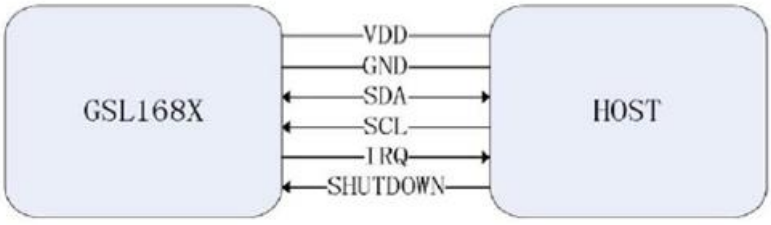


Figura 9: Pines para la comunicación del chip GSL1680

Como se aprecia en la figura, además de la alimentación, únicamente 4 pines son necesarios para la comunicación:

- SDA y SCL, son los pines de reloj y datos, correspondientes a la comunicación mediante el protocolo I2C, que se verá en más detalle en el apartado 3.5, destinado al microcontrolador maestro.

- IRQ es el pin de interrupción, utilizado por el chip controlador, para comunicarle al micro maestro, que se ha producido una nueva pulsación en la pantalla.

- El pin SHUTDOWN, es utilizado para habilitar/deshabilitar la pantalla táctil.

El chip, para ser utilizado por primera vez, requiere que su firmware sea cargado en determinadas posiciones de su memoria. Este firmware es proporcionado por el fabricante, así como el algoritmo a seguir para poder cargarlo de forma adecuada. Como ya se ha comentado, la comunicación entre el chip controlador de pantalla táctil y el micro maestro, se realiza utilizando el protocolo I2C. Este protocolo va a ser utilizado en ambos sentidos, ya que en primera instancia, será el micro maestro el que le envíe el firmware al chip controlador, y una vez configurado, será el chip controlador el que envíe al micro maestro las coordenadas de las pulsaciones recibidas en la pantalla táctil.

3.2 Fuente de alimentación

El funcionamiento principal del prototipo, es funcionar como interruptor digital de las luces que tenga conectadas. Dado que se requiere la utilización de la línea de 230 Voltios para alimentar los puntos de luz a los que está conectado el prototipo, y que toda la circuitería requerida funciona a un voltaje diferente, y en general, mucho menor de 230 Voltios, se requiere una fuente de alimentación independiente de la línea de 230 Voltios, que será la encargada de administrar la corriente necesaria a cada parte de los circuitos que componen el prototipo.

En la tabla siguiente, se muestran los voltajes y corrientes que necesita cada parte de la circuitería del prototipo para funcionar correctamente:

DISPOSITIVO	VOLTAJE REQUERIDO	CORRIENTE REQUERIDA
MICRO STM32F407	3.3 V	150 mA
CHIP GSL1680	3.3 V	< 60 mA
PANTALLA TFT-LCD	3.3 V	450 mA
ZERO CROSS DETECTOR	3.3 V	60 mA

Tabla 1: Voltajes y corrientes requeridos

Tal y como se muestra en la tabla, el único voltaje necesario en el prototipo, además de los 230 Voltios de la línea para alimentar las luces (y el circuito de optoacopladores), es de 3.3 Voltios en continua para el micro maestro (STM32F407VGT6), la pantalla TFT-LCD (ER-TFTM050-3), el chip controlador de pantalla táctil (GSL16080) y el detector de paso por cero. La corriente que debe suministrar la fuente de alimentación, debe ser de al menos 720 mA (la suma de las corrientes requeridas por los dispositivos (150 + 450+ 60 + 60 = 720 mA).

La solución que se ha llevado a cabo para obtener los voltajes y corrientes requeridos, pasa por un transformador AC-DC, que a partir de la línea de 230V, obtenga una salida de 3.3 Voltios y ofrezca una corriente de 750 mA. Para ello, se ha utilizado un

transformador Nokia AC-DC (Alterna – Continua) que transforma de 230V a 6.5 Voltio. Este transformador, normalmente es utilizado para la carga de teléfonos móviles, del cual se ha desprendido el mini-jack que conectaba al móvil, para poder utilizar los cables de salida, como entrada para un siguiente transformador DC – DC LM2596, el cual ha sido regulado mediante la utilización del potenciómetro del que dispone, para que obtenga los 3.3 Voltios requeridos.

En la siguiente imagen se muestra el esquema de la fuente de alimentación de 3.3 V y 750 mA:

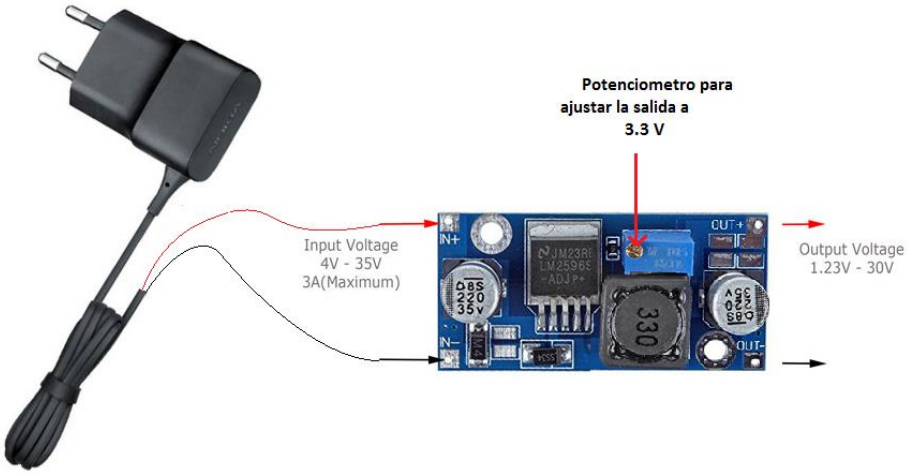


Figura 10: Esquema de la fuente de alimentación

3.3 Driver de corriente

El driver de corriente, es la parte encargada del manejo y la gestión de la línea de 230 V. El driver, entregará la corriente a las cargas (luces) cuando el micro maestro así lo indique mediante las señales de control.

Cada carga, dispone de un circuito como el que se muestra a continuación:

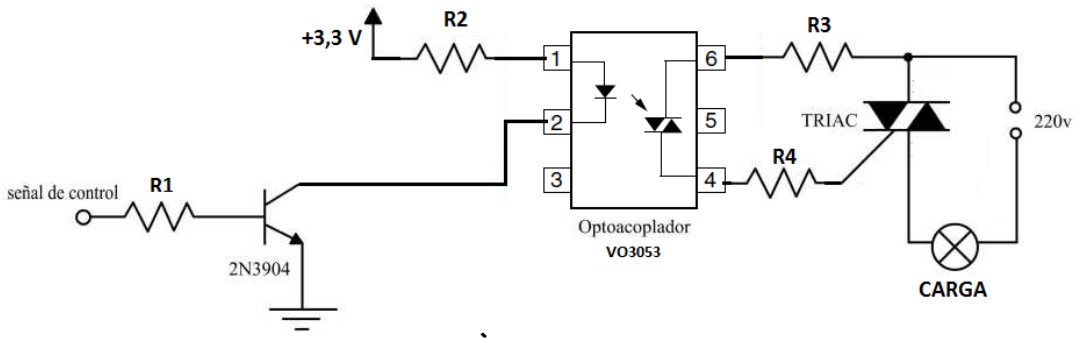


Figura 11: Circuito Interruptor digital

Dado que el prototipo permite el control de hasta cuatro salidas, el circuito mostrado en la figura anterior, se repetirá para cada una de las salidas, y el micro maestro dispondrá de 4 señales de control, una para cada salida.

Para el diseño del circuito, se ha tenido en cuenta que es necesario aislar el microcontrolador maestro de la señal de 230V, ya que en caso de derivación de la intensidad, podría dañar el microcontrolador. La manera que se ha llevado a cabo para aislar la onda de 230V del microcontrolador maestro, ha sido la utilización del optoacoplador VO3053, el cual a partir de un pequeño voltaje que habilita su funcionamiento, activa un TRIAC empleando un fotodiodo y un fotodetector, de esta forma ambas partes del circuito quedan aisladas hasta un valor de tensión en la salida

de 5300 Vrms, según las hojas de características del optoacoplador (adjuntas en el anexo IV).

Cuando el optoacoplador está en estado de funcionamiento, es decir, que permite el paso a través del TRIAC que lleva integrado en su parte de salida, se cierra el circuito formado por el TRIAC de potencia exterior y la carga, lo que hace que la bombilla luzca.

3.3.1 Calculo de los valores de las resistencias

El cálculo de los valores de las resistencias, ha sido realizado teniendo en cuenta los requerimientos de corriente de cada componente.

La circulación de corriente que debe recorrer el fotodiodo emisor del optoacoplador, es de 10 mA según sus hojas de características. Por tanto, para calcular el valor de R2, utilizamos la siguiente ecuación:

$$3.3\ v = R_2 \cdot\ I_{FD} + \ V_{FD} + \ V_{CE}$$

Despejando R2 y haciendo $I_{FD} = 10\ mA$, se obtiene:

$$R_2 = \frac{(3.3v - V_{FD} - V_{CE})}{10\ mA} = 190\ \Omega$$

El valor comercial escogido para esta resistencia ha sido $R_2 = 220\ \Omega$

De las hojas de características del optoacoplador, se recoge el dato V_{FD}, que es el valor de tensión que cae en el fotodiodo emisor. Este valor es $V_{FD} = 1.2\ v$. El valor de V_{CE},

es el voltaje colector-emisor del transistor, que en funcionamiento es igual a su valor en estado de saturación, es decir, $V_{CE} = 0.2\ v$.

Con una corriente de colector de 10 mA, y una ganancia de $\beta = h_{FE} = 100$, se obtiene una corriente de base, $I_b = 100\ \eta A$. Dado que la señal de control, cuando está activa, proporciona un voltaje de 3.3 v, podemos plantear la siguiente ecuación para el cálculo de R1:

$$V_{SC} = I_b \cdot R1 + V_{BE}$$

Siendo $V_{SC} = 3.3\ v$ y $V_{BE} = 0.7\ v$, se obtiene:

$$R_1 = \frac{(3.3v - 0.7v)}{100\ \eta A} = 26\ K\Omega$$

El valor comercial escogido para esta resistencia ha sido $R_1 = 30.1\ K\Omega$

La resistencia R3 no es obligatoria cuando la carga es resistiva, ya que la corriente está limitada por la corriente de disparo de puerta del TRIAC de potencia. Sin embargo, se ha incluido para evitar posibles daños en el TRIAC del optoacoplador. La función de la resistencia R4, es limitar la corriente que circula por la puerta (Gate) del TRIAC de potencia. El valor que se ha dado a ambas resistencias es:

$$R_3 = R_4 = 330\ \Omega$$

3.3.2 Detector de paso por cero

Otra parte del driver de corriente, que requiere otro circuito independiente, es el circuito encargado de la regulación de la intensidad de las salidas. Para ello, se utiliza un detector de paso por cero, el H11A11, que basándose en el siguiente circuito, activa el pin PE2 del microcontrolador en cada paso por cero de la onda de 50Hz y 230V.

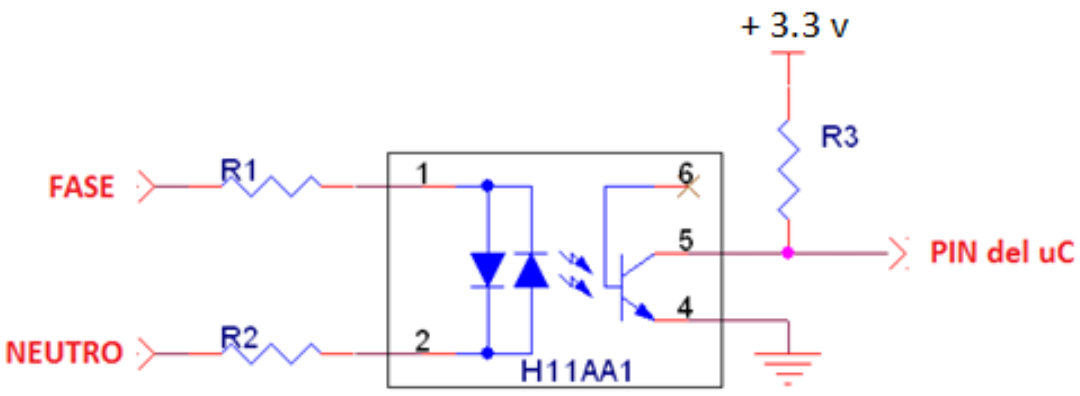
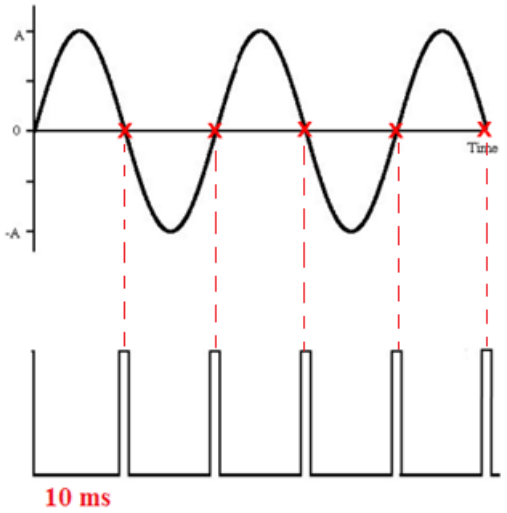


Figura 12: Circuito Detector de paso por cero

Cuando el voltaje de la onda alterna es inferior al voltaje consumido por el diodo, se interpreta que estamos pasando por un cero. El hecho de que existan dos diodos contrapuestos, sirve para poder identificar los pasos por ceros tanto cuando provenimos del semiciclo positivo como del negativo de la onda. La función de la resistencia de pull-up R3, es mantener la señal de control a cero voltios cuando el transistor del detector de paso por cero está en corte, lo que sucede cuando la onda no está pasando por cero voltios. Una vez que la onda pasa por cero voltios, el transistor entra en conducción, lo que provocará un cero lógico en el pin del microcontrolador conectado al detector de paso por cero.



Onda de entrada al detector de paso por cero

Salida del detector de paso por cero H11A11

Figura 13: Formas de onda del detector de paso por cero

El valor escogido para R3 ha sido:

$$R_3 = 10\ K\Omega$$

Dado que lo que comprueba el componente detector de paso por cero, el H11A11, es el paso de la tensión de la onda sinodal por cero voltios, y que la corriente de funcionamiento del fotodiodo emisor es de 10 mA, según sus hojas de características, se han escogido los siguientes valores de las resistencias R1 y R2:

$$R_1 = R_2 = 33\ K\Omega$$

Estos valores de resistencias, proporcionan el valor de corriente apropiado en los diodos:

$$V_{ef} = R_1 \cdot I_D$$

Siendo I_D la corriente que cirula por el diodo. Despejando I_D obtenemos:

$$I_D = \frac{V_{ef}}{R_1} = \frac{230 \cdot \sqrt{2}}{33\ K\Omega} = 9.85\ mA$$

Siendo V_{ef} el valor eficaz del voltaje, que resulta de multiplicar los 230V x $\sqrt{2}$.

Ya se han mostrado los valores de las resistencias R1 y R2, sin embargo, dado que en los bornes de estas resistencias habrá una gran caída de tensión cercano a los 230 V de la onda, se han escogido resistencias de 2 vatios, ya que la potencia de las resistencias comerciales más habituales, suelen ser de ¼ de vatio que resultan insuficientes.

La función del detector de paso por cero, es informar al microcontrolador maestro que la onda de 230V está pasando por los cero voltios. Por tanto, podemos disparar una interrupción en el micro maestro, cada vez que la salida del detector de paso por cero se active, lo que sucederá cada 10 ms (ver figura 25). El software del programa, mediante las señales de control, será el encargado de regular cuanto tiempo después del paso por cero se permitirá el paso de corriente a través del TRIAC (ver figura 23). Dado que existirá un paso por cero cada 10 ms, por ser una onda de 50 Hz, si activamos la señal de control 1 ms después del paso por cero, estaremos entregando a la carga el 90% de la corriente de la onda.

A continuación, se muestran diferentes formas de onda de los voltajes entregados a las cargas, según el momento en el que se permita el paso de corriente a través del TRIAC:

Intensidad al 90% Intensidad al 50% Intensidad al 10%

Figura 14: Formas de onda de los Triacs de potencia en función de la intensidad

3.4 Placas de circuito impreso (PCBs) y esquemático

Para la fabricación de las placas de circuito impreso, ha sido necesaria la utilización de un programa de diseño electrónico que nos permita diseñar las placas, a continuación se realiza una breve descripción del diseño electrónico así como del programa de diseño utilizado.

3.4.1 Diseño electrónico y Altium Designer

El *diseño electrónico*, consiste en la utilización de las distintas metodologías existentes, con el fin de desarrollar un circuito electrónico.

El diseño se realiza a distintos niveles. Por una parte tenemos la parte física, donde se diseña la estructura real de los componentes electrónicos que constituyen el circuito, sus dimensiones, materiales. Por encima podemos encontrar métodos de diseño de cada vez más alto nivel, hasta llegar a los llamados lenguajes de descripción de hardware. Éstos permiten introducir descripciones de los distintos bloques funcionales de un sistema para su simulación, verificación e incluso para la generación automática del circuito físico con la herramienta de síntesis apropiada.

ALTIUM DESIGNER

Este programa de diseño electrónico ha sido utilizado durante el proyecto para la creación de los circuitos impresos de los que se compone el prototipo. El programa tiene incorporadas librerías con los componentes habituales en electrónica (resistencias, capacitores, conectores, compuertas lógicas, etc.).

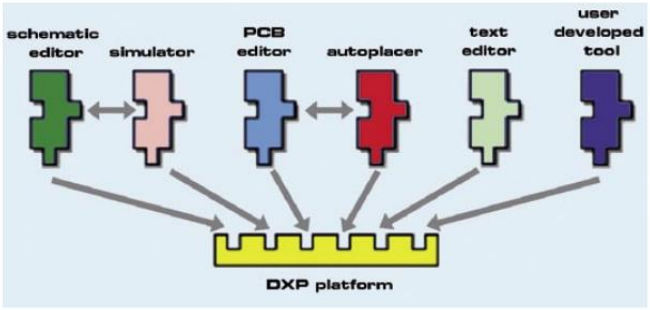


Figura 15: Funcionalidades Altium Designer

El primer paso a realizar, sería la creación del esquemático completo, donde se detallan las conexiones existentes entre las distintas interfaces del prototipo. Para ello, comenzamos a buscar los componentes que tiene nuestro dispositivo, tratando de poner en el centro del esquemático los componentes con más pines, que en nuestro caso es el microcontrolador. Los componentes hay que incluirlos una vez los hayamos localizado en las librerías de las que dispone el programa.

Una vez cargados los componentes en la hoja PCB, solo queda acomodar los componentes, para ello hacemos uso de la función “AutoRoute”, que nos ayudará enormemente a encontrar los caminos que deben seguir los paths que conectan cada uno de los pines del circuito con su destino. Será necesario modificar los grosores de los paths, para asegurarnos que cumplen con los tamaños requeridos según la corriente que vaya a atravesar el path.

3.4.2 Diseño de las PCBs

Durante la fase de diseño del prototipo, dado el elevado numero de conexiones existentes entre cada uno de los componentes del circuito, se vio la necesidad de fabricar dos PCBs para la integración de todos los componentes. El diseño de estas PCBs evita el cableado sustituyéndolo por paths impresos en las PCBs.

Para ello, se ha utilizado el programa de diseño electrónico Altium Designer, el cual ha sido introducido en el apartado anterior. Tal y como se comenta en ese apartado, antes de proceder al diseño de las PCBs, es necesario contruir el esquemático completo con los componentes que van a ser utilizados en las placas de circuito impreso. El cual queda como se muestra en el Anexo III.

Durante la fase de diseño, se decidió fabricar dos PCBs de doble cara. La primera de ellas, la placa Framework, contiene la mayoría de las partes del prototipo, entre ellas se encuentran:

- El microcontrolador maestro

- El puerto SWD necesario para la programación del micro maestro

- El puerto que conecta los pines del chip GSL1680 de control de pantalla táctil con el micro maestro,

- El puerto que conecta el micro maestro con el detector de paso por cero,

- Las señales de control que activan cada una de las salidas,

- Las tomas de tierra y alimentación que dotan de corriente a la placa

La placa Framework ha sido diseñada del mismo tamaño que la pantalla LCD, quedando fijada a la pantalla por la parte posterior de la misma.

La segunda placa diseñada, es la placa de conectores, en la que se encuentran los siguientes componentes:

- Detector de paso por cero

- Electrónica del driver de corriente (Resistencia, Optoacopladores y Triacs).

En esta segunda placa, también se encuentran las conexiones de la línea de 230 V con las luces conectadas.

En el anexo III , se puede observar la representación de ambas PCBs, donde las líneas o paths en rojo, hacen referencia a las líneas situadas en la cara frontal o “top layer”, mientras que las líneas en azul hacen referencia a los paths situados en la cara posterior o “bottom layer”.

3.5 Microcontrolador

Para la realización del proyecto se ha escogido el microcontrolador STM32F407VGT6 de STMicroelectronics, en su versión de encapsulado de 100 pines (STM32F4 LFQP100) la descripción de pines del chip se puede encontrar en el anexo V. La elección del micro se ha realizado teniendo en cuenta su precio y prestaciones, ya que se trata de un micro de 32 bits, lo suficientemente potente para realizar todas las posibles funcionalidades que se han pensado implementar en este proyecto y un coste aceptable. Otro criterio para la elección de este microcontrolador, es que dispone de un módulo de Ethernet y los pines necesarios para poder utilizar el protocolo si fuera necesario, ver apartado 7 (Futuras mejoras).

Para facilitar la utilización del micro, se ha utilizado en su formato EASY MX 1105MIKROE, el cual viene con el microcontrolador en el medio de la placa. Para facilitar el acceso a los pines del micro, el circuito EASY MX dispone de pines accesibles conectados a cada uno de los pines del microcontrolador, así como algunos pines preprogramados para utilizar los protocolos de comunicación más utilizados (I2C, SPI, USART, etc…). También dispone funcionalidades extras que faciliten la programación de las mismas (ETHERNET, USB_OTG).

Figura 16: Imagen del chip utilizado en el proyecto

El micro maestro, dispone de 5 puertos de 16 pines cada uno. Los pines de estos puertos pueden ser utilizados como pines de propósito general, o utilizando alguna de sus funciones alternativas. En las tablas que se encuentran en el Anexo V, se muestran las funciones alternativas de cada uno de los cinco puertos de los que dispone el microcontrolador. Cada una de las columnas, representa una función alternativa, que puede estar asociada a algunos de los pines de cada puerto.

Las funciones alternativas mostradas en el Anexo V, están a disposición del proyecto, aunque el hecho de utilizarlas, sacrificaría los pines escogidos para desempeñar la función alternativa asociada, no pudiendo utilizar el pin para otra finalidad. Esta limitación a la hora de escoger las funciones alternativas que debe realizar cada pin, supone la necesidad de una organización de todas las funciones a desarrollar, ya que se debe evitar la utilización de determinados pines para propósitos generales, si estos disponen de funciones alternativas que pueden ser útiles en el proyecto.

Por poner un ejemplo, si se utiliza el pin PE9 como pin de propósito general para realizar cualquier funcionalidad, ya no podrá utilizarse el canal 1 del Timer 1 en ese pin, sin embargo el microcontrolador, proporciona alternativas a las funciones más utilizadas, y en el ejemplo anterior, podríamos darle uso al canal 1 del Timer 1 utilizando el pin PA9 que también lleva la función alternativa de canal1 del Timer 1 asociada.

3.5.1 Relaciones entre el micro maestro y los periféricos

Control de pantalla LCD

El control de la visualización en la pantalla LCD, se lleva a cabo mediante la utilización del micro de vídeo RAIO8875, el cual va integrado en la pantalla TFT-LCD. Como ya se describió en el apartado 3 (parte Hardware del proyecto), el micro de vídeo es el encargado del control de la visualización en la pantalla. Además dispone de una memoria flash donde se almacenan los datos que deben ser visualizados. Para poder establecer la comunicación entre el microcontrolador maestro y el micro de vídeo, y programar la memoria flash del micro de vídeo con el fin de mostrar en pantalla el menú adecuado, se utiliza la interfaz FSMC (Flexible Static Memory Controller) del micro maestro, por tanto los pines que incluyen las funciones referentes a la interfaz FSMC, tendrán que utilizarse para este propósito, y no podrán ser utilizados para realizar otras funciones.

A parte de los pines de alimentación y tierra, serán necesarios los 16 pines de datos, los cuales se describen en el Anexo V.

Los pines RS y WR, son utilizados para indicar al micro de vídeo la función que se pretende realizar:

Cycle Type	RW#	RS	Description
Command Write	0	1	Register number write cycle
Status Read	1	1	Status read cycle
Data Write	0	0	Corresponding Register data/Memory data write cycle following the Command Write cycle.
Data Read	1	0	Corresponding Register data/Memory data read cycle following the Command Write cycle.

Figura 17: Instrucciones de lectura/escritura en el micro de vídeo

Control de panel táctil:

Para el control del panel táctil, y el consiguiente reconocimiento de pulsaciones en la pantalla, es utilizado el chip del panel táctil, el GSL1680 visto en el capítulo 4.1. Este chip envía los datos de las pulsaciones recogidos, utilizando el protocolo I2C. El protocolo I2C únicamente requiere de dos pines bidireccionales para establecer la comunicación, el pin SDA o de datos y el pin SCL o de reloj. La línea SDA contiene los datos, y la SCL es el reloj que marca los tiempos de la comunicación. El micro maestro, dispone de tres combinaciones de pines para el uso del protocolo I2C. Como se pueden ver en las tablas de funciones alternativas de los puertos, una de las combinaciones posibles es utilizar los pines PB6 y PB7 como líneas SCL y SDA, que es la configuración escogida para el proyecto.

El chip GSL1680, requiere de los pines de alimentación y tierra, y además de otros dos pines, uno utilizado para apagar el chip de la pantalla táctil, y otro para que el chip pueda indicar al micro maestro que la pantalla ha sido pulsada en determinado punto. Los pines elegidos que se van a asociar a estos dos pines han sido PC3 y PC2, ya que no disponen de funciones alternativas que vayan a ser utilizadas en el proyecto. Los pines encargados de la comunicación mediante el protocolo I2C (SAD y SCL) son:

PIN GSL1680	PIN STM32
TOUCH – INT	PC2
TOUCH – SHT	PC3
I2C – SCL	PB6
I2C – SDA	PB7

Tabla 2: Pines utilizados para la comunicación con la pantalla táctil

Control de los interruptores digitales

Para permitir el paso de corriente través de los TRIACs, es necesaria una señal de control que habilite cada una de las cuatro salidas. Para ello, se deben configurar cuatro pines del microcontrolador, para que actúen como las señales de control de cada uno de los cuatro interruptores digitales. Estos pines, serán configurados en el micro maestro como salidas, así cuando se requiera activar la conducción de uno de los TRIACs, se pondrá a nivel alto el pin correspondiente, lo que conllevará la activación del fotodiodo emisor del optoacoplador.

Los pines que hacen de señales de control para cada una de las luces son:

Señal de control	PIN STM32
Luz 1	PE2
Luz 2	PE0
Luz 3	PA10
Luz 4	PB9
Zero Cross Detector	PB8

Tabla 3: Pines utilizados por el driver de corriente

Programación del micro maestro

El ultimo requerimiento de pines, son los pines destinados a la programación del micro maestro mediante la placa de evaluación, utilizando la interfaz SWD. Los pines asociados son:

Pin SWD	PIN STM32
SWD – SWDIO	PA13
SWD – CLK	PA14

Tabla 4: Pines utilizados para la interfaz de programación del micro (SWD)

La descripción de los pines utilizados para cada una de las funciones alternativas que ofrece el microcontrolador

Una vez descritas todas las funciones alternativas de las que disponen los pines de los 5 puertos del microcontrolador, vamos a detallar que pines han sido utilizados en el proyecto y que funciones llevan asociadas. Para ello vamos enumerar cada una de las necesidades que han surgido para el diseño del proyecto, y a continuación se mostrará una tabla resumen de todos los pines utilizados y las funciones asociadas a ellos:

3.5.2 Kit de desarrollo STM32F4Discovery

La placa de desarrollo STM32F4Discovery está gobernada por el microcontrolador STM32F407VGT6 de arquitectura ARM Cortex-M4 con 1Mb de memoria flash y 192Kb de RAM. Incluye una serie de periféricos integrados como el acelerómetro LIS3DSH, micrófono digital MP45DT02, DAC de audio con controlador de altavoz integrado CS43L22, ocho LEDs, switch configurable, conector micro-USB AB OTG y la herramienta de desarrollo ST-LINK/V2 que permite programar el microcontrolador integrado o un microcontrolador de una placa externa mediante un cable conectado al interfaz SWD (*Serial Wire Debug*).

En la siguiente figura se puede ver la disposición física de estos elementos en el kit de desarrollo.

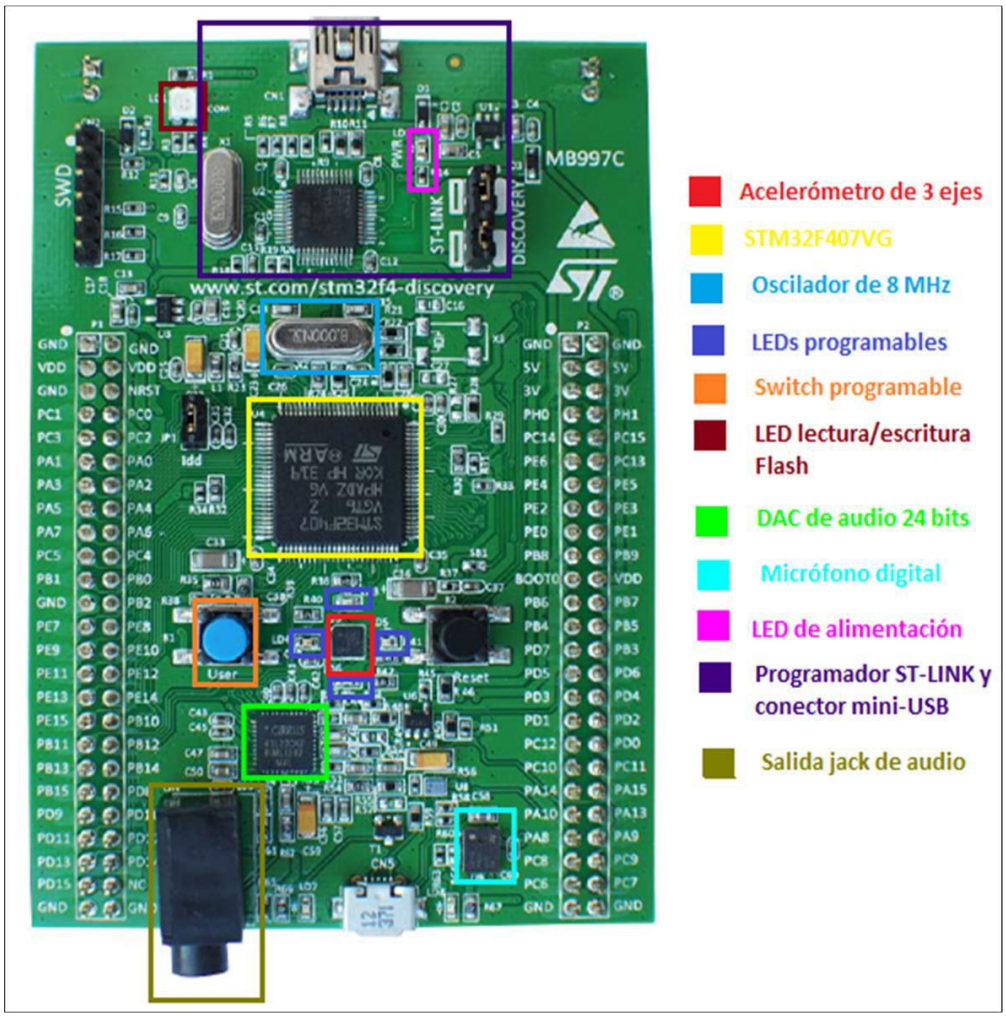


Figura 18: Kit de desarrollo STM32F4Discovery

Como ya se ha mencionado, el kit de desarrollo (o placa de evaluación), se utiliza para la programación de la memoria flash del micro objetivo. Para poder llevar a cabo esta tarea, es necesario configurar los jumpers de la placa de evaluación. La siguiente imagen muestra los jumpers de la placa de evaluación en su posición normal, que es la utilizada para programar el microcontrolador integrado en la placa:

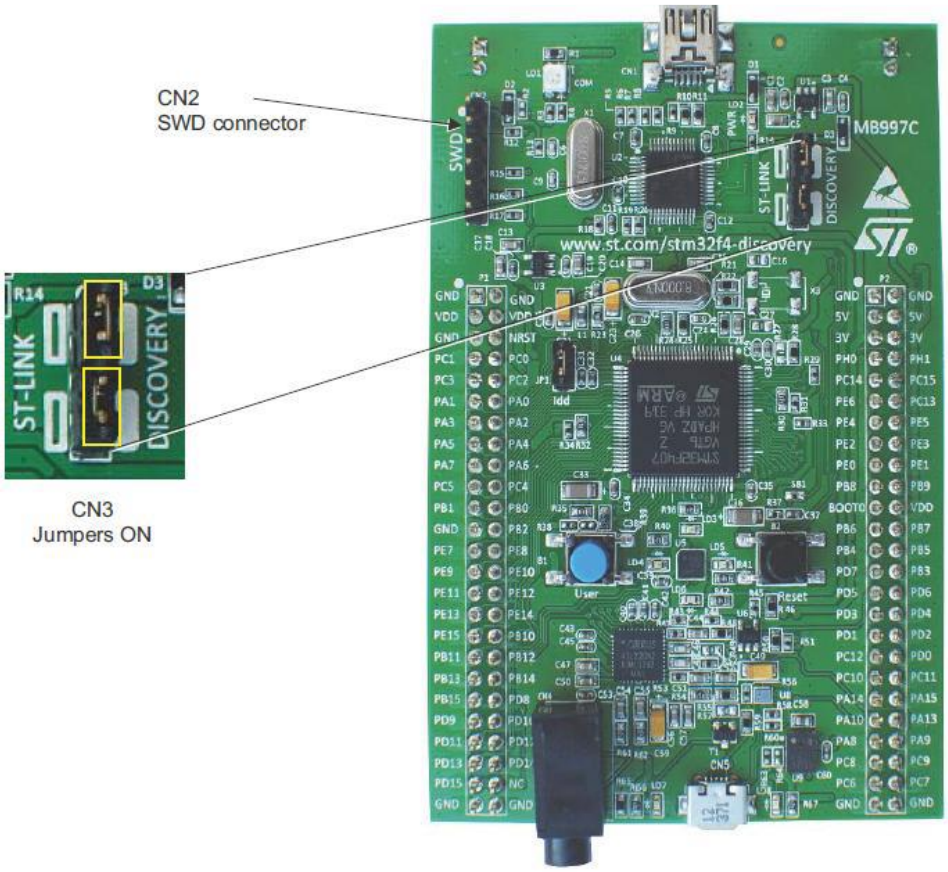


Figura 19: Configuración jumpers CN3 para uso del microcontrolador integrado

Sin embargo, lo que en este proyecto se necesita es programar el micro externo a la placa de evaluación, por lo que se deben retirar los jumpers mostrados en la figura 16, y conectar los pines del micro externo destinados a la programación de su memoria flash, con la interfaz SWD (*Serial Wire Debug*) de la placa de evaluación siguiendo la siguiente disposición de pines:

Pin	CN2	Asignación
1	VDD	VDD
2	SWCLK	Reloj SWD
3	GND	GND
4	SWDIO	input/output SWD
5	NRST	Reset del micro
6	SWO	Reservado

Tabla 5: Pines interfaz SWD

4. Parte Software del proyecto

Un entorno de desarrollo, es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software. Normalmente, un IDE consiste de un editor de código fuente, compilador, herramientas de construcción automáticas y un depurador.

Para la creación de la parte Software del proyecto, ha sido utilizado el entorno de desarrollo de Software **CoIDE**, de **CooCox** en el cual se ha desarrollado el proyecto implementándose en el lenguaje de programación C.

CoIDE es un entorno de desarrollo libre para microcontroladores con núcleo de la serie Cortex-M fabricados por ARM. Integra un compilador GCC, un entorno de desarrollo basado en Eclipse y un programador de memoria flash.

El entorno de desarrollo de CooCox, CoIDE, aporta librerías con funciones básicas implementadas, estas funciones proporcionan un entorno más intuitivo para el desarrollo de Software, ya que al utilizarlas, se evita tener que modificar cada uno de los registros de forma independiente y basta con nombrar a la función encargada de realizar las funciones pertinentes.

4.1 Funciones a desempeñar

Las funciones para las que el prototipo ha sido diseñado, así como los menús que van a constituir las pantallas de las que dispone la aplicación del prototipo, serán introducidas en este capítulo. En la idea inicial, el prototipo pretendía funcionar únicamente como un interruptor digital de hasta cuatro salidas, que sustituya a los interruptores mecánicos habituales. A esta funcionalidad principal del prototipo, se han añadido las siguientes funcionalidades:

- Regulación de la intensidad de cada salida de forma independiente.
- Programado de encendido y/o apagado de las salidas seleccionadas.
- Entrada en modo ahorro de energía.
- Configuración inicial de fecha y hora.

En el Implementación Software de las funcionalidades (4.3), se entrará en profundidad en cómo se han diseñado y desarrollado cada una de estas funcionalidades.

4.2 Menús de la aplicación

Para poder llevar a cabo las funcionalidades descritas anteriormente, el prototipo dispondrá de los siguientes menús, en cada uno de ellos podemos acceder a las diferentes funciones descritas.

El listado de los menús del prototipo se lista a continuación:

1. Menú de configuración de fecha y hora
2. Menú principal o de luces
3. Menú de regulación de intensidad
4. Menú de selección de tipo de programación
5. Menú de programación de luces

4.2.1 Menú de configuración de fecha y hora

Es el menú con el que arranca el prototipo cuando este es conectado a la red, en él se configura la fecha y hora del dispositivo. En el siguiente dibujo se observa el menú de configuración de fecha y hora:

Figura 20: Imagen del menú de configuración de fecha y hora

4.2.2 Menú principal o de luces

Tras configurar la fecha y la hora, y pulsar el botón “Save”, la pantalla realiza una breve introducción en la cual se da la bienvenida al usuario, tras la introducción, se entra en la pantalla principal, en la cual se muestra la fecha y hora configuradas en el menú anterior junto con los botones de cada una de las cuatro luces. La funcionalidad principal de este menú es el encendido/apagado de las luces.

El aspecto de la pantalla principal se muestra a continuación:

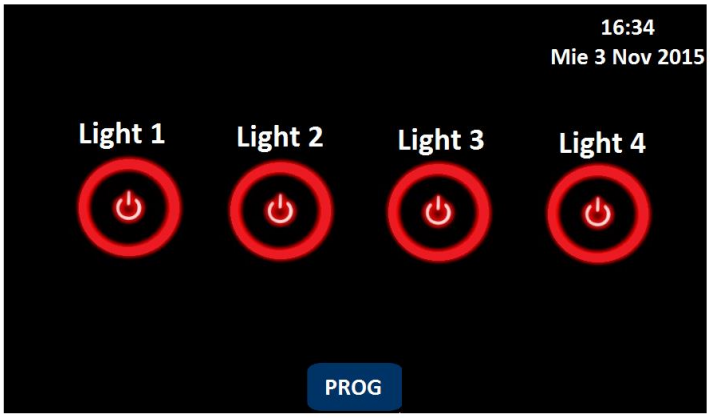


Figura 21: Imagen del menú principal con todas las luces apagadas

Y así es como se mostraría la pantalla principal con las salidas 1 y 3 encendidas:

Figura 22: Imagen del menú principal con dos luces encendidas

Como se puede apreciar en las imágenes anteriores, el botón inferior cambia cuando hay alguna de las luces encendidas. Esto se debe a que la función que realiza el botón, es diferente si no hay ninguna luz encendida a si la hay.

En el caso de no haber ninguna luz encendida, el botón tiene el nombre “PROG”, y si es pulsado lleva al menú de selección de tipo de programación. Cuando hay alguna luz encendida, el botón tiene el nombre “LUM”, y si es pulsado se entra en el menú de regulación de intensidad.

4.2.3 Menú de regulación de intensidad

Cuando alguna de las salidas está activa, y se pulsa el botón “LUM” pasamos al menú de selección de intensidad de la luz de las salidas, que es como se muestra en la siguiente imagen:

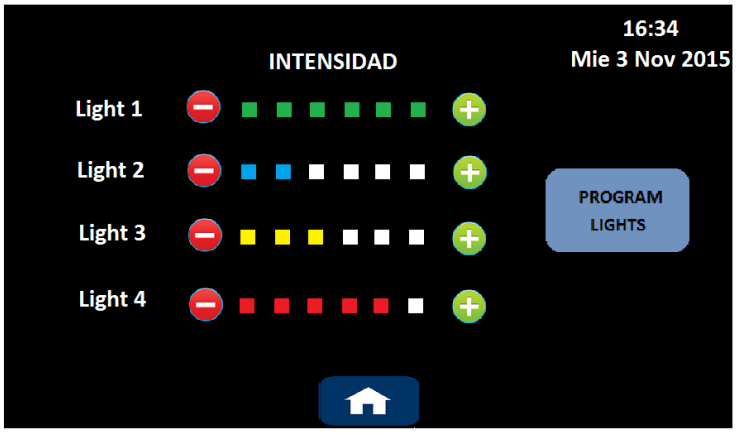


Figura 23: Imagen del menú de regulación de intensidad

En este menú, el usuario puede regular la intensidad de las 4 salidas, aunque la regulación sólo se hará efectiva cuando la salida que se esté regulando esté activa. Como se aprecia en la imagen anterior, desde este menú de regulación de intensidad, se puede ir al menú de programación de luces pulsando en el botón "PROGRAM LIGHTS", lo que nos llevaría al mismo menú que si pulsamos el botón inferior de la pantalla principal cuando no hay ninguna luz encendida.

4.2.4 Menú de selección de tipo de programación

Tanto si pulsamos el botón "Program Lights" en el menú de regulación de intensidad, como si pulsamos el botón PROG en el menú principal, accedemos al menú de programación de alarma. En este menú se podrá seleccionar el tipo de programación que se desea realizar, programación de encendido, o programación de pagado. Este menú es como se muestra en la siguiente imagen:

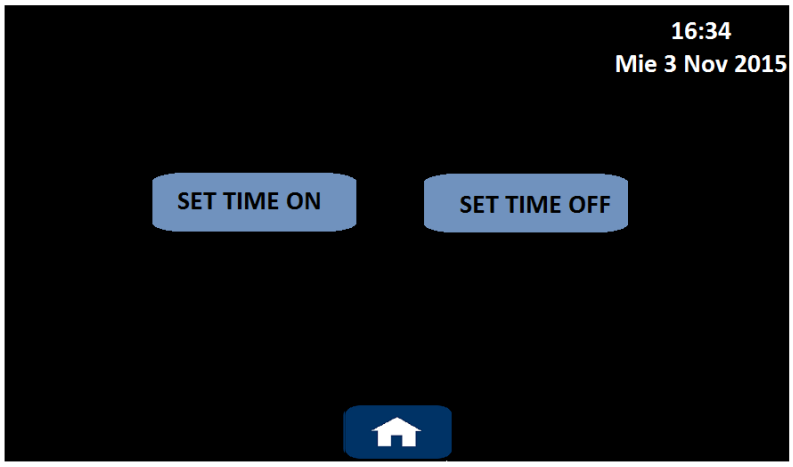


Figura 24: Imagen del menú de selección de tipo de programado

4.2.5 Menú de programación de las salidas

Una vez seleccionado el tipo de programación que se desea, se entra en el menú de programación, en el cual se pueden configurar la hora y las luces que se desea que se enciendan o apaguen. El menú de programación se muestra en la siguiente imagen:

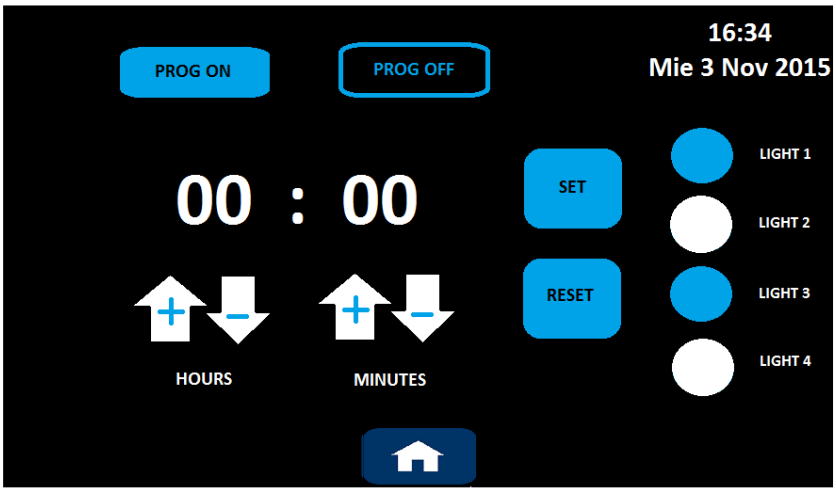


Figura 25: Imagen del menú de programación de luces

En la imagen anterior, está seleccionada la programación para las luces 1 y 3. Una vez seleccionada la hora y las luces para las cuales se quiere realizar la programación de encendido, se pulsa el botón “SET”, y la programación queda activada, mostrándose en pantalla como representa la siguiente imagen:

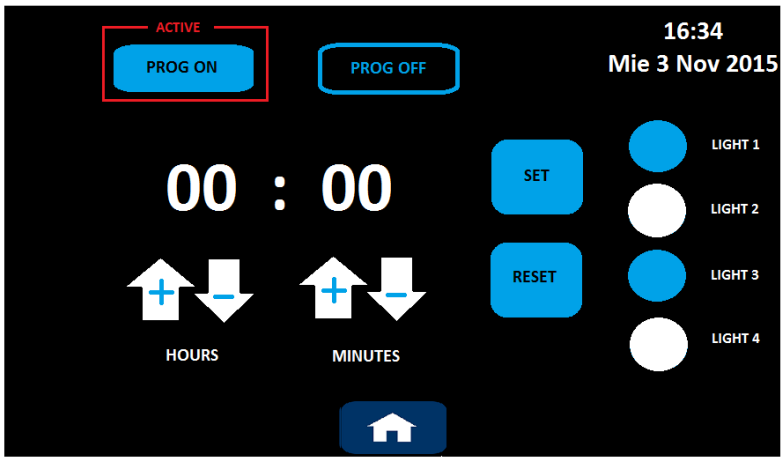


Figura 26: Menú de programación de luces con programación de encendido activada

El botón inferior cuando se está en cualquier menú que no sea el principal, el cual está representado por una casa blanca sobre un fondo azul, al ser pulsado, lleva al menú principal, mostrándose en el estado en el que se hayan dejado cada una de las luces.

4.2.6 Diagrama de flujo de los menús

Una vez que el dispositivo se conecta a la fuente de alimentación, este se enciende, y muestra el menú de configuración de fecha y hora para que el usuario pueda configurarlas. A continuación se muestra un breve video de bienvenida, el cual da lugar a la representación del menú principal, donde se muestran los cuatro botones que hacen de interruptor digital para cada una de las cuatro luces. A partir de aquí, es el usuario el que puede ir moviéndose por los menús para realizar las funciones que ofrece el dispositivo. El siguiente diagrama de flujo, representa los botones que deben ser pulsados para ir accediendo a cada uno de los menús del dispositivo:

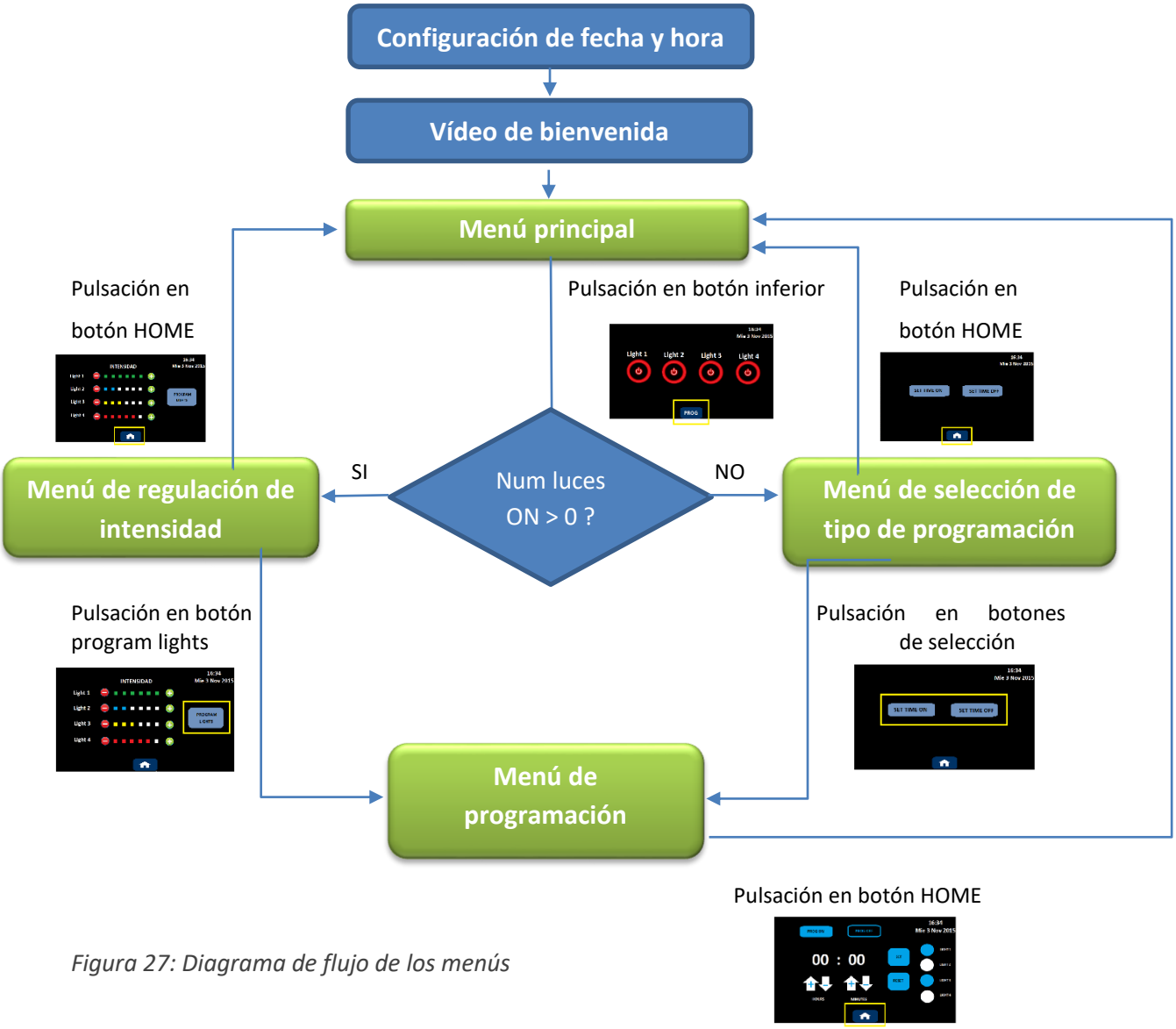


Figura 27: Diagrama de flujo de los menús

4.3 Implementación Software de las funcionalidades a desempeñar

4.3.1 Regulación de intensidad

El funcionamiento de la electrónica encargada de la regulación de intensidad de las salidas activas, ha sido descrito en el apartado 3.3. En este apartado se va a describir el software que ha sido necesario implementar para el control de la electrónica encargada de la regulación.

La regulación de intensidad de las salidas, se realiza partir de la detección de paso por cero, llevada a cabo por el detector de paso por cero H11A11 descrito en el apartado 3.3. Una vez que el microcontrolador ha recibido la interrupción que se produce en cada paso por cero, se habilita el Timer 2, el cual empieza a contar desde cero a un ritmo de 1 microsegundo. Dada la cadencia del timer, y que existe un paso por cero cada 10 milisegundos, el timer contará hasta 10.000 entre cada paso por cero. La regulación de intensidad se consigue realizando una espera ente el paso por cero, y la activación de la señal de control que habilitará el funcionamiento del optoacoplador, esta espera será directamente proporcional a la regulación realizada, por ejemplo, una espera de 5.000 microsegundos entre el paso por cero y la activación de la señal de control, supondrá una regulación de intensidad del 50% (5.000 / 10.000).

Cada salida, dispone de un valor de dimmer asociado, el cual puede variar entre cero y seis. Este valor de dimmer asociado a cada salida, será el que fije el tiempo esperar entre la detección de paso por cero y la activación de la señal de control de cada salida.

Para dotar de esta capacidad al prototipo, se ha desarrollado un software que inicializa el timer en cada detección de paso por cero, y genera una interrupción cada vez que el valor del timer ha llegado al valor de espera fijado por el valor del dimmer de cada salida. Esta interrupción será la encargada de activar la señal de control que habilita el optoacoplador asociado a la salida que se está regulando.

4.3.2 Detección de pulsaciones

Cada vez que el usuario toca la pantalla, el chip GSL1680 reconoce que ha recibido una pulsación, y activa el pin IRQ del chip (ver sección 3.1.2). Como se vio en la sección referente al chip GSL1680, el microcontrolador maestro está conectado al chip GSL1680, uniendo el canal IRQ del chip, con el pin PC2 del microcontrolador. Se ha configurado dicho pin en el micro como una entrada, y declarado como una línea de interrupción externa (EXTI_Line), para que se genere una interrupción, cada vez que el pin IRQ del chip GSL1680 se active. De esta manera, el microcontrolador entra en la rutina de atención a la interrupción cada vez que el usuario toca la pantalla.

Entre otras acciones, la rutina de atención a la interrupción, cuando se produce una pulsación en la pantalla, llama a la función Read_coords_MultiFingers, que es la encargada de almacenar la información de las pulsaciones en la pantalla. La definición de esta función puede visualizarse en el Anexo VII.

La información de las posiciones de las pulsaciones en la pantalla, se almacena en la variable _touch_event de la estructura principal _screen, teniendo que cuando sólo hay un dedo tocando la pantalla, el único elemento _coords que contiene información es coords [0], mientras que si hay dos dedos en la pantalla, serán coords [0] y coords[1] los elementos que dispongan información útil y así sucesivamente. Si no hay dedos tocando la pantalla, los cinco elementos _coords de la estructura _touch_event, estarán vacíos (X=0, Y=0).

4.3.3 Zonas sensibles de pulsación

El funcionamiento del dispositivo, requiere la identificación de las pulsaciones realizadas por parte del usuario en la pantalla. Como ya se ha descrito en el apartado anterior, la información referente a la posición de las pulsaciones, se guarda en la variable _touch_event de la estructura principal _screen.

El procedimiento para identificar si las pulsaciones en la pantalla conllevan una acción a realizar por parte del dispositivo, consiste en establecer las zonas de la pantalla que llaman a alguna tarea. Se denomina **zona sensible de pulsación**, a aquellas zonas dentro de un menú, que al ser pulsadas, requieren una acción por parte del dispositivo. Hay que definir cada zona sensible de pulsación en cada uno de los menús que existen en la aplicación.

Las zonas sensibles, son definidas por las posiciones de los botones que conllevan la acción, así pues, cada zona sensible forma una región en la pantalla, estas regiones están constituidas por cuatro puntos, formando un rectángulo:

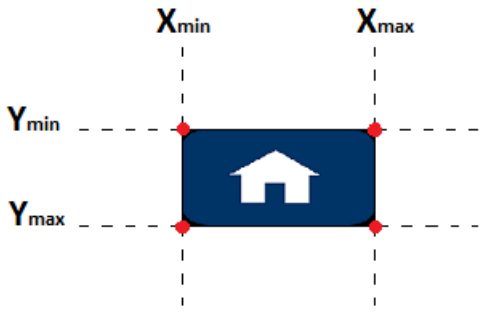


Figura 28: Definición de zona sensible de pulsación

Por ejemplo, observando el dibujo anterior, la zona sensible para el botón Home, que como se explicó en el capítulo 5.2, al ser pulsado, lleva al menú principal o de luces,

queda definido por los cuatro puntos (Xmin, Ymin), (Xmax, Ymin), (Xmin, Ymax), y (Xmax, Ymax). Por tanto, una vez detectada una pulsación en la pantalla, al consultar las coordenadas de esta pulsación, si los valores de X e Y se encuentran dentro de la región definida por los 4 puntos, y nos encontramos en uno de los menús que posean el botón Home, podremos asumir que el botón ha sido pulsado y actuar en consecuencia.

A continuación, se resaltan las zonas sensibles de pulsación para cada uno de los menús marcadas con rectángulos de color rojo:

Menú de configuración de fecha y hora:

Figura 29: Zonas sensibles de pulsación del menú de fecha y hora

Menú principal o de luces:

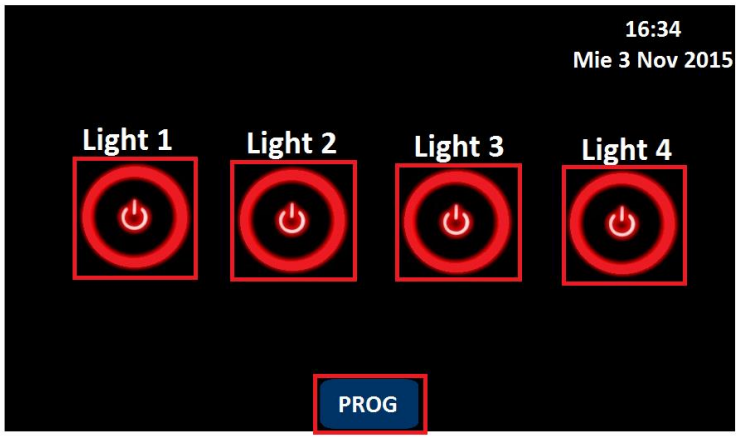


Figura 30: Zonas sensibles de pulsación del menú principal

Menú de regulación de intensidad:

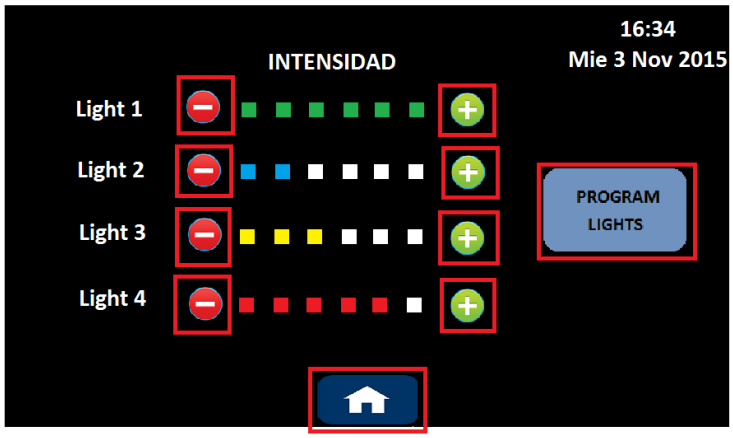


Figura 31: Zonas sensibles de pulsación del menú de regulación de intensidad

Menú de selección de tipo de programación:

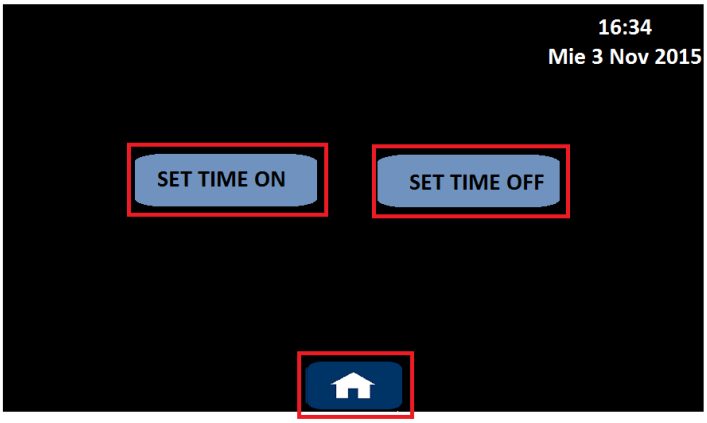


Figura 32: Zonas sensibles de pulsación del menú de selección de tipo de programación

Menú de programación de encendido/apagado:

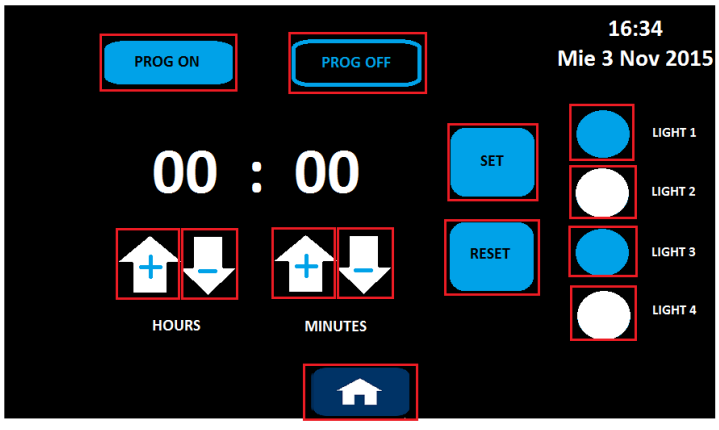


Figura 33: Zonas sensibles de pulsación del menú programación

Para facilitar las llamadas a las posiciones de todas las zonas sensibles existentes en el proyecto, y para mejorar la organización de la aplicación software, se ha creado el archivo “ButtonsPosition.h” donde se describen las posiciones que marcan las regiones sensibles de cada botón disponible en cada uno de los menús.

El archivo “buttonsPosition.h” se muestra en el Anexo VII.

4.3.4 Función ActionButton

Una vez se ha reconocido la pulsación en la pantalla, y almacenado los valores de las coordenadas de la pulsación en la estructura principal mediante la función Read_coords_MultiFinger, se llama a la función ActionButton, para que compruebe, si las coordenadas pulsadas, en el menú en el que se encuentra el usuario, corresponden con alguna zona sensible de pulsación, y en caso afirmativo, llame al actuador correspondiente.

La función ActionButton, se encuentra en el archivo Touch.c, el cual se puede ver en el anexo VII. Como variable auxiliar a la función ActionButton, se ha definido la variable touchReleased, que vale cero cuando la pulsación aún se mantiene en la pantalla, y vale uno cuando la pulsación ha sido retirada de la pantalla. Esta variable es utilizada por la función ActionButton, para comprobar si la pulsación está siendo sobre un botón de incremento/decremento (De fecha, de tiempo o de tiempo de programación), ya que de ser así, gestiona el incremento/decremento a una velocidad adecuada. Mientras que si el botón pulsado ha sido cualquier otro botón que no implique un incremento o decremento, el uso de esta variable, impide que se vuelva a realizar la acción correspondiente al botón indefinidamente hasta que el usuario retire el dedo de la pantalla. Después de chequear el valor de la variable auxiliar "touchReleased", la función ActionButton consulta la estructura principal screen, para conocer en que menú se encuentra el usuario, y a continuación, examina si las coordenadas de la pulsación, se corresponde con alguna zona sensible de pulsación, llamando al actuador correspondiente.

4.3.5 Actuadores

Teniendo en cuenta el funcionamiento del método ActionButton, expuesto en el apartado anterior, cada vez que existe una coincidencia entre las coordenadas de la pulsación, el menú actual, y una zona sensible de pulsación del menú actual, se ejecuta el código del actuador correspondiente al botón pulsado.

Existe un actuador por cada una de las zonas sensibles de pulsación existente en el dispositivo. En el apartado 4.3.3, hemos visto de manera gráfica las zonas sensibles de pulsación existentes. En este apartado, vamos a ver los actuadores correspondientes a cada una de las zonas sensibles de actuación de cada menú.

4.3.5.1 Actuadores del menú de configuración de fecha y hora

El primer actuador que vamos a examinar, es el correspondiente a la primera zona sensible (representada en la siguiente imagen por un rectángulo rojo) del menú de configuración de fecha y hora:

Figura 34: Primera zona sensible de pulsación del menú de fecha y hora

Cuando el usuario se encuentra en el menú de configuración de fecha y hora, y toca el botón marcado con un cuadrado rojo en la imagen anterior, que es cuando el usuario

pretende incrementar el día en el menú de configuración de fecha y hora, la función ActionButton comprueba si el menú actual es el de configuración de fecha y hora, y en caso afirmativo, comprueba si las coordenadas de la pulsación recibida, las cuales están almacenadas en la estructura touch_event, se encuentran dentro de la zona de pulsación del botón mencionado. En caso de haberse producido la pulsación en ese botón, la función ActionButon llama al actuador **Touch_DayUP**, que se encargará de incrementar en una unidad el día mostrado en pantalla, realizando las comprobaciones pertinentes, las cuales se explican a continuación.

Este actuador, recibe como parámetro la dirección de memoria de la estructura principal, para poder acceder a las variables de la misma para su lectura/escritura. El otro parámetro que recibe, **"monthYearChanged"** es un indicador, que indica si el actuador esta llamado por otro actuador, en cuyo caso solo debe recorrer parte del código del actuador **"Touch_DayUp",** o si por el contrario ha sido llamado por la función ActionButton.

La variable **"monthYearChanged"** valdrá uno cuando se llame al actuador desde otro actuador, y cero cuando la llamada sea realizada desde la función ActionButton. Esta variable ha sido introducida para el control de los días que tiene cada mes y cada año, ya que si por ejemplo, estamos situados en el día 30 del mes 1, y cambiamos al mes 2, el actuador correspondiente al botón de incrementar el mes, llamará al actuador **"Touch_DayUp"** para que compruebe si el día actual (30 en el ejemplo), es válido para el nuevo mes.

En el caso descrito, el programa cambiará el día automáticamente al día 1 del mes 2. Las comprobaciones que realiza este actuador, también controlan el número de días de febrero por año bisiesto. Por último, el actuador llama a la función LCD_PrintDate, para que actualice el nuevo valor del día en la pantalla. El código de todos los actuadores se muestra en el archivo Touch.c, el cual se puede visualizar en el Anexo VII.

El siguiente actuador del menú de configuración de fecha y hora, es **“Touch_DayDown”,** que recibe los mismos parámetros que **“Touch_DayUp”**, y realiza las mismas comprobaciones, pero en vez de incrementar el día, lo decrementa y almacena el nuevo valor en la variable screenDate de la estructura principal.

Los actuadores **“Touch_MonthUp”** y **“Touch_MonthDown”**, únicamente reciben como parámetro la dirección de memoria donde se encuentra la estructura principal, ya que el número de meses siempre es 12, independientemente del año en el que nos encontremos, por lo que no necesitan realizar comprobaciones. Su función es incrementar, y decrementar, respectivamente, el mes mostrado en pantalla, y almacenar el nuevo valor en la variable screenDate de la estructura principal.

Los actuadores **“Touch_YearUp”** y **“Touch_YearDown”**, también reciben únicamente como parámetro la dirección de memoria donde se encuentra la estructura principal. Su función es incrementar, y decrementar, respectivamente, el año mostrado en pantalla y almacenar el nuevo valor en la variable screenDate de la estructura principal.

Los actuadores **“touch_TimeHUp”,** y **“touch_TimeHDown”**, reciben como parámetro la dirección de memoria donde se encuentra la estructura principal, controlan que la hora sea un valor entre 0 y 23, y su función es incrementar y decrementar, respectivamente, la hora mostrada en pantalla y almacenar el nuevo valor en la variable screenTime de la estructura principal.

Los actuadores **“touch_TimeMUp”,** y **“touch_TimeMDown”**, reciben como parámetro la dirección de memoria donde se encuentra la estructura principal, controlan que el minuto seleccionado, sea un valor entre 0 y 59, y su función es incrementar y

decrementar, respectivamente, el minuto mostrado en pantalla y almacenar el nuevo valor en la variable screenTime de la estructura principal.

El último actuador del menú de configuración de fecha y hora, es el que hace referencia a la zona sensible de pulsación correspondiente al botón **"Save"**. Este actuador se llama **"touch_Save"**, recibe como parámetro la dirección de memoria donde se encuentra la estructura principal, y su función es establecer la fecha del sistema, guardar los valores de fecha y hora mostrados por pantalla en las variables screenTime y screenDate de la estructura principal y además hacer efectivo el cambio de menú de configuración de fecha y hora al menú principal o de luces.

4.3.5.2 Actuadores del menú principal o de luces

En el menú principal existen dos actuadores, el primero de ellos es el actuador **"touch_Light_Button"**, que es el encargado de manejar las pulsaciones en los cuatro botones de encendido/apagado de las luces. La función ActionButton es la encargada de decirle a este actuador que luz es la que ha sido pulsada, pasándole como parámetro un número que identifique la luz sobre la que el usuario ha realizado la pulsación. El actuador que activa/desactiva las luces, **"touch_Light_Button"**, recibe como parámetro la dirección de memoria donde se encuentra la estructura principal, así como el identificador de la luz que ha sido pulsada.

El segundo actuador de este menú, es el encargado de gestionar la pulsación en el botón inferior del menú, y recibe el nombre **"touch_Settings_Button"**. Como ya se comentó en el apartado 4.2, menús de la aplicación, el botón inferior de la pantalla principal, es inicialmente el botón **"PROG"**, el cual al ser pulsado, lleva al usuario al menú de selección de tipo de programación. Cuando alguna luz está activa y el usuario

está en el menú principal, el botón inferior pasa a ser el botón **“LUM”**, que lleva al usuario al menú de regulación de intensidad de las luces.

Para este actuador, la función ActionButton no pasa como parámetro un indicador para que el actuador sepa si el botón pulsado ha sido **“PROG”** o **“LUM”**, sino que el propio actuador, recibe como parámetro la dirección de memoria de la estructura principal, y consulta en dicha estructura la variable **“numLightsON”**. Si la variable es cero, el actuador interpreta que el botón pulsado ha sido el botón **“PROG”**, mientras que si es mayor que cero, el botón pulsado habrá sido **“LUM”**.

4.3.5.3 Actuadores del menú de regulación de intensidad

En este menú, los actuadores existentes pueden clasificarse en tres tipos:

Actuadores referentes a los botones de incremento/decremento de intensidad para cada luz:

Para cada luz, existe un actuador encargado de gestionar la pulsación en el botón de incremento de la intensidad, y otro para el decremento. Los actuadores encargados del decremento de la intensidad, son **"touch_Minus_Dimmer1", "touch_Minus_Dimmer2", "touch_Minus_Dimmer3"** y **"touch_Minus_Dimmer4",** y su función es decrementar la intensidad de cada una de las luces respectivamente.

Los actuadores que gestionan el incremento de la luminosidad de cada luz son, **"touch_ Plus _Dimmer1", "touch_Plus_Dimmer2", "touch_ Plus _Dimmer3"y "touch_ Plus _Dimmer4"** respectivamente.

Todos estos actuadores, reciben como parámetro la dirección de memoria de la estructura principal, y modifican, para cada luz, el valor de la variable Dimmer asociada a cada luz. Cada pulsación implica un incremento/decremento de intensidad de la luz que haya sido pulsada, siendo 7 el número de niveles de intensidad que acepta cada luz. Todas las luces comienzan con su nivel máximo de intensidad cuando el dispositivo se conecta por primera vez. La manera en la que se realiza la regulación de intensidad de cada luz, en función del valor de su variable Dimmer, fue descrita en el capítulo 3.3 Driver de corriente.

Actuador referente al botón "Home":

El actuador correspondiente a la zona sensible de pulsación del botón **"Home", "touch_Home_Button".** Recibe por parámetro la dirección de memoria de la estructura principal, y su función es cambiar el menú de la estructura, al menú de luces. También realiza la visualización del menú principal en el dispositivo.

Actuador correspondiente al botón de programación de luces:

El último actuador que vamos a describir del menú de regulación de intensidad, es el que hace referencia al botón de programación de encendido/apagado, **“touch_ProgramLights”,** y al igual que el actuador del botón Home, su función es cambiar el menú de la estructura principal así como de encargarse de la visualización del nuevo menú, en este caso el menú de selección de tipo de programación.

4.3.5.4 Actuadores del menú de selección de tipo de programación

El menú de selección de tipo de programación, es utilizado por el usuario, para seleccionar si quiere entrar en el menú de programación de encendido o de apagado de las luces. También existe otro actuador en este menú, cuya función es la de volver al menú principal mediante la pulsación del botón “Home”.

Los actuadores de este menu, son **“touch_SetON”, “touch_SetOFF”, y “touch_Home_Button”**. El actuador **“touch_Home_Button”**, es el mismo descrito en el apartado anterior, el cual estaba destinado a describir los actuadores del menú de regulación de intensidad, y la función que desempeña es cambiar el valor del menú en la estructura principal, así como la visualización del menú principal en el dispositivo.

Los actuadores que se corresponden con los botones de elección de tipo de programación, se encargan de modificar las variables **menú** y **alarmSel** de la estructura principal. Ambos actuadores cambiarán el menú almacenado en la estructura principal, al menú de programación de encendido/apagado, dependiendo del botón que pulse el usuario. El actuador “**touch_SetON**”, cambiará la variable **alarmSel** a cero, mientras que “**touch_SetOFF**” la cambiará a uno, de esta forma, la estructura **screen**, ya dispondrá de la información que indica que tipo de programación se está realizando.

4.3.5.5 Actuadores del menú de programación de encendido/apagado

Las posibilidades que ofrece este menú, son utilizadas para la programación de encendido o apagado de las luces que sean seleccionadas. Existen distintos tipos de actuadores en este menú;

Por un lado están los actuadores referentes al tipo de programación que se desea realizar, aunque siempre que se haya llegado hasta este menú, en la estructura principal ya se habrá cargado la información del tipo de programación que se haya seleccionado, y esta información estará en la variable alarmSel de la estructura principal, puede ser modificada pulsando sobre uno de los dos actuadores siguientes:

touch_SetTimeOFF, que de ser pulsado, cambiará la variable alarmSel a 1, lo que provocará que el tipo de programación que se va a realizar, será programación de apagado de las luces.

touch_SetTimeON, que de ser pulsado, cambiará la variable alarmSel a 0, lo que provocará que el tipo de programación que se va a realizar, será programación de encendido de las luces.

Por otro lado, tenemos los actuadores referentes a la hora en la que se pretende fijar la programación de encendido o apagado. Estos actuadores nos permitirán modificar la hora y el minuto en el que queremos que actúe la programación. **touch_HourUP** incrementará la hora en una unidad, **touch_HourDOWN** decrementará la hora en una unidad, **touch_MinuteUP** incrementará los minutos en una unidad, y **touch_MinuteDown**, decrementará los minutos en una unidad.

Otro tipo de actuadores de este menú son los que hacen referencia a las luces que son seleccionadas para realizar la programación. Estos actuadores son **touch_ALRM_Light1, touch_ALRM_Light2, touch_ALRM_Light3 y touch_ALRM_Light4.** Los cuales modifican la variable ALRM_lightsSet, la cual contiene la información de las luces sobre las que tiene que actuar la programación.

Existen otros dos actuadores que son los encargados de activar/desactivar la programación realizada. Una vez seleccionado el tipo de programación, la hora y minuto, y las luces sobre las que debe actuar la programación, se debe pulsar sobre el botón **SET**, lo cual llama al actuador **touch_SetProgram,** el cual configura los parámetros y activa la alarma del micro maestro, para que actúe en el minuto y hora seleccionado. El actuador **touch_ResetProgram** realiza el proceso inverso, cuando una alarma está activa, al pulsar el botón **RESET**, el actuador **touch_ResetProgram** es llamado, y se encarga de desactivar la alarma correspondiente.

En este menú también existe el actuador **"touch_Home_Button"**, que es el mismo descrito en los dos apartados anteriores.

4.3.6 Entrada en modo ahorro de energía

Esta funcionalidad, fue la última en incluirse en el proyecto. Una vez estaba el proyecto funcionando, la experiencia percibida tras las pruebas con el dispositivo, sugería introducir esta nueva funcionalidad, que aporta un ahorro en el consumo de energía, ya que apaga el dispositivo tras no recibir ninguna pulsación en un periodo de 60 segundos. Además de la mejora del ahorro de energía que supone la inclusión de esta nueva funcionalidad, la principal motivación que llevó a incluir esta mejora, fue tratar de proporcionar un ambiente relajado para el usuario, ya que la luz que desprende el dispositivo, puede llegar a ser demasiado luminosa en ambientes oscuros.

La manera en la que se realiza el control del tiempo que ha pasado desde la última pulsación, es la utilización de uno de los Timers de los que dispone el microcontrolador maestro. Este Timer se resetea cada vez que el usuario realiza una pulsación en la pantalla. Este Timer ha sido configurado para que salte una interrupción cuando hayan pasado 60 segundos desde la última pulsación.

La rutina de atención a la interrupción de este Timer, es la encargada de apagar el dispositivo tras haber guardado el estado de todas las variables del mismo.

La forma en la que el sistema se reestablece tras haber entrado en modo ahorro de energía, se realiza en la rutina de atención a la interrupción de detección de pulsaciones, cuando se recibe una pulsación, se comprueba si el sistema está en modo ahorro de energía, en caso afirmativo se reestablece el sistema y se muestra en la pantalla el menú principal, en caso de no estar en modo ahorro de energía se proceden a extraer las coordenadas de la pulsación recibida.

4.3.7 Interrupciones

En el proyecto, existen diferentes fuentes de interrupción, que deben ser priorizadas para asegurar un correcto funcionamiento del dispositivo.

Las fuentes de interrupción existentes en el proyecto son:

- Pulsación en la pantalla
- Detección de paso por cero
- Activación de las señales de control
- Desbordamiento del timer de ahorro de energía
- Desbordamiento del timer de systick
- Actualización de fecha y hora en la pantalla
- Alarma

Las interrupciones más críticas del proyecto, son aquellas que deben ser fijadas con mayor prioridad, ya que requieren que sus rutinas de atención a la interrupción sean atendidas en el momento en el que son generadas.

Este es el caso de las **interrupciones encargadas de la regulación de la intensidad**, ya que al existir un paso por cero cada 10 milisegundos, y unos valores de dimmer que requiere realizar esperas de entre 500 y 7.500 milisegundos, la desviación de estas esperas supondrá un malfuncionamiento en la regulación de intensidad (ver capítulo 4.3.1), ya que una espera de 1 milisegundo más, supone un decremento de intensidad del 10%. Por este motivo, las interrupciones con mayor prioridad son tanto las de activación de las señales de control, como la de detección de paso por cero.

Por otro lado, la **interrupción** que menos prioridad requiere, es la **encargada del systick**. Esta interrupción se utiliza para controlar los delays que se quieran introducir en el código, es decir, las esperas durante las cuales una función se debe quedar ociosa. Por tanto, si alguna otra interrupción, irrumpe cuando un delay se está

ejecutando, esta nueva interrupción debe apropiarse del procesador, y pasar a ejecutarse inmediatamente.

La **interrupción** que hace referencia a la **alarma**, se activa cuando existe una concordancia entre la hora actual del sistema y la hora de la alarma activa. Cuando esto ocurre, se debe atender a la rutina de atención a la interrupción de la alarma, lo que siempre va a conllevar un encendido o apagado de alguna/as de las luces. Dado que lo que se selecciona en la alarma es la hora y el minuto, no será necesaria una alta prioridad para esta interrupción, ya que no es relevante el hecho de que las luces se encienda/apaguen en el segundo exacto en el que se igualan los minutos de la hora del sistema y de la alarma, pudiendo realizarse el encendido/apagado de la luz en el segundo siguiente a haber coincidido.

La **interrupción de pulsación en la pantalla**, se genera cada vez que una pulsación ha sido realizada en el panel táctil. Esta interrupción debe ser atendida lo antes posible para evitar posibles pulsaciones realizadas de manera rápida, pero no debe interferir con las interrupciones destinadas a la regulación de intensidad, ya que provocaría un malfuncionamiento en la regulación.

En el entorno de software utilizado en el proyecto, COIDE, la prioridad se fija de manera inversa, es decir, una prioridad cero, indica la mayor prioridad. Existe también en este entorno de desarrollo, una subprioridad, la cual es utilizada para discriminar que interrupción debe ser atendida si hay más de una interrupción activa con el mismo nivel de prioridad.

Los niveles de prioridad son configurables, disponiendo de 4 bits para asignar las prioridades, el entorno de desarrollo se puede configurar para usar esos 4 bits como niveles de prioridad, o como niveles de subprioridad, en el proyecto se ha configurado para utilizar dos bits para la prioridad y otros dos bits para la subprioridad, por tanto la menor prioridad será de 3, igualmente que la menor subprioridad.

A continuación se muestra la tabla con las prioridades asignadas a cada una de las interrupciones existentes en el proyecto:

INTERRUPCION	PRIORIDAD	SUBPRIORIDAD
PULSACIÓN EN LA PANTALLA	1	0
DETECCIÓN DE PASO POR CERO	0	0
ESPERA DE REGULACIÓN DE INTENSIDAD CONCLUIDA	0	1
DESBORDAMIENTO DEL TIMER DE AHORRO DE ENERGÍA	1	1
DESBORDAMIENTO DEL TIMER DE SYSTICK	3	3
ACTUALIZACIÓN DE FECHA Y HORA EN LA PANTALLA	3	0
ALARMA	2	2

Tabla 6: Prioridades del proyecto Software

5. Presupuesto

En este apartado se va a mostrar de manera detallada el presupuesto desglosado del proyecto, especificando los diferentes gastos que han sido necesarios para su realización. Dado que el entorno de desarrollo de Software es un entorno gratuito, los costes asociados al proyecto son únicamente de la parte Hardware del proyecto.

Tal y como se ha visto en el apartado 3, parte Hardware del proyecto, la parte de Hardware se puede dividir en cada uno de los sub-apartados que esta parte incluye, esto es:

- Pantalla TFT-LCD
- Fuente de alimentación
- Placas de circuito impreso
- Microprocesador y placa de evaluación.
- Driver de corriente

A continuación, se van a mostrar los precios de cada una de las partes, indicando en cada caso el desglose de componentes de cada una de las partes.

Pantalla TFT - LCD

La pantalla utilizada para el proyecto, lleva integrado el micro de vídeo y el chip controlador de pantalla táctil, además del coste de la pantalla, ha sido necesario adquirir un conector FCC para el chip GSL1680

DESCRIPCION	FABRICANTE	PRECIO unitario
PANTALLA	EASTRISING	49,85 €
FFC DIP ADAPTER 6C PIN	E-SALE2010	0,74 €
		50,59 €

Tabla 7: Presupuesto pantalla TFT

Fuente de alimentación

Para la fuente de alimentación, únicamente ha sido necesario adquirir los transformadores que convierten el voltaje alterno de 230 V a los 3.3 V requeridos en el sistema.

DESCRIPCION	FABRICANTE	PRECIO unitario
TRANSFORMADOR DC-DC LM2596	EASTRISING	1,33 €
TRANSFORMADOR AC -DC	NOKIA	1,50 €
		2,83 €

Tabla 8: Presupuesto Fuente alimentación

Placas de circuito impreso

Tras el diseño de las placas, la fabricación para el montaje del prototipo se ha realizado mediante la empresa PCBWAY, la cual ha dado los siguientes precios según los diseños enviados:

DESCRIPCION	FABRICANTE	PRECIO unitario
FRAMEWORK PCB	PCB WAY	2,60 €
DRIVER PCB	PCB WAY	5,20 €
		7,80 €

Tabla 9: Presupuesto placas de circuito impreso

En este apartado, también vamos a incluir todos los costes de los conectores utilizados, ya que han sido montados sobre las mismas placas:

MODELO	DESCRIPCION	FABRICANTE	Precio
SSW-113-01-L-D	HEMBRA 2X13	SAMTEC	0,42 €
SSW-115-01-F-D	HEMBRA 2X15	SAMTEC	0,56 €
TSW-106-07-L-S	MACHO 1X6	SAMTEC	0,75 €
TSW-102-07-l-s	MACHO 1X2	SAMTEC	0,84 €
SSW-103-01-l-d	HEMBRA 2X3	SAMTEC	0,48 €
SSW-104-01-F-D	HEMBRA 2X4	SAMTEC	0,65 €
FHP-02-01-T-S	HEMBRA 1X2	SAMTEC	0,35 €
SNT-100-bk-g	JUMPER	SAMTEC	0,20 €
			4,25 €

Tabla 10: Presupuesto conectores

Por tanto, el coste de las placas de circuito impresos, incluyendo los costes de los conectores, asciende a **12.05€.**

Microprocesador y placa de evaluación

La adquisición del microcontrolador y de la placa de evaluación para la programación del mismo, se realizó a través de la empresa Farnell, y estos son los costes asociados:

MODELO	DESCRIPCION	FABRICANTE	PRECIO unitario
STM32F407 DISCOVERY	PLACA EVALUACION	STM	13,75 €
EASYMX PRO V7	MICROCONTROLADOR	MIKROE	20,35 €
			34,10 €

Tabla 11: Presupuesto microprocesador y placa evaluación

Driver de corriente

En esta parte se incluyen los componentes electrónicos que forman tanto los interruptores digitales como el detector de paso por cero para la regulación de intensidad:

PARTE	TIPO	MODELO	DESCRIPCION	FABRICANTE	CANTIDAD	PRECIO unit.	PRECIO Total
Interruptor digital	RESISTENCIA	ERJ-8ENF2200V	220 Ohm	PANASONIC	1	0,03 €	0,03 €
Interruptor digital	RESISTENCIA	TNPW120630K1BEEN	30,1 KOhm	VISHAY	1	0,77 €	0,77 €
Interruptor digital	RESISTENCIA	ERJ-8ENF3300V	330 Ohm	PANASONIC	2	0,03 €	0,07 €
Interruptor digital	COMPONENTE	VO3053	MOC 3053	VISHAY	1	0,88 €	0,88 €
Interruptor digital	COMPONENTE	ACST610-8G	TRIAC 6A 800V	STM	1	1,59 €	1,59 €
Interruptor digital	COMPONENTE	MMBT3904	TRT NPN VCE 40v	FAIRCHILD	1	0,12 €	0,12 €
Detector - ZCD	RESISTENCIA	CRCW120610K0FKEA	10 KOhm	VISHAY	1	0,05 €	0,05 €
Detector - ZCD	RESISTENCIA	SMF233KJT	33KOhm 2W 300V	TE CONNECTIVITY	2	0,20 €	0,40 €
Detector - ZCD	COMPONENTE	H11AA1-X007T	H11AA1 5300 AC	VISHAY	1	1,32 €	1,32 €
							5,23 €

Tabla 12: Presupuesto driver de corriente

5.1 Resumen costes Hardware

La siguiente tabla, muestra de forma conjunta, los gastos totales de la fabricación del prototipo:

DESCRIPCION	COSTE
Pantalla TFT -LCD	50,59 €
Fuente de alimentación	2,83 €
Microprocesador y placa evaluación	34,10 €
Placas de circuito impreso	12.05 €
Driver de corriente	5,23 €
	104.80 €

Tabla 13: Presupuesto global

6. Conclusiones

Este capítulo contiene las reflexiones que han derivado de la realización del proyecto, así como de la fabricación del prototipo. Basándonos en los objetivos expuestos al principio de este documento, llegamos a unas conclusiones las cuales se exponen a lo largo de este capítulo.

1. El diseño de los menús del prototipo tal como se ha realizado, consigue aportar al usuario la modularidad deseada, permitiendo que el dispositivo resultante sea configurable por cualquier usuario, independientemente de la configuración que el usuario requiera.

2. Ha sido posible implementar todas las funcionalidades que se requerían para el dispositivo. Además se ha implementado la funcionalidad extra de entrada en modo e ahorro de energía, la cual fue expuesta en el apartado 4.3.6.

3. La configuración del panel táctil, llevó más tiempo del esperado tal y como se puede apreciar en el diagrama de Gantt expuesto en estas conclusiones, sin embargo finalmente se consiguió establecer la comunicación entre el microcontrolador maestro y el chip de control de pantalla táctil GSL1680, permitiendo extraer las coordenadas exactas de las pulsaciones en la pantalla.

4. Se ha conseguido establecer la comunicación entre el microcontrolador maestro y micro de vídeo encargado de la visualización en la pantalla LCD, mediante la configuración de la interfaz FSMC del microcontrolador maestro. Este objetivo también ha llevado algo más de tiempo de lo que se esperaba, tal y como se aprecia en el diagrama de Gantt que se expone en este capítulo.

5. La integración de todos los módulos involucrados, ha sido posible gracias a la fabricación de las PCBs diseñadas durante este proyecto, las cuales han servido como nexo entre los módulos, tanto a nivel Hardware como a nivel Software.

6. Se ha conseguido alimentar todo el sistema a partir de una fuente de alimentación enchufable a la red y construida con diferentes componentes que ofrezcan los voltajes y corrientes requeridos.

7. El estudio de los componentes electrónicos a incluir en el dispositivo, ha llevado un tiempo mayor al esperado como se puede ver en el siguiente diagrama de Gantt. El motivo del incremento en el tiempo de búsqueda de componentes, ha sido el elevado número de fabricantes y precios que estos ofrecen.

8. Los dispositivos con funcionalidades parecidas existentes actualmente en el mercado, parten desde un precio de 300€ llegando hasta valores muy superiores, por lo que este dispositivo tendría un precio menor en comparación con lo que se puede encontrar actualmente. Sin embargo dados los requerimientos del prototipo, la elección de una pantalla LCD más económica que la utilizada, reduciría el coste total del dispositivo, sin reducir la calidad del mismo de forma significativa, ya que aunque la pantalla escogida lleva un panel táctil integrado que hay que conectar con el microcontrolador de forma independiente a la pantalla, con el fin de reducir costes, sería más adecuado integrar una pantalla LCD con un panel táctil independiente a la pantalla, en lugar de utilizar una pantalla que lleve el panel táctil incluido. Esta forma de proceder abarataría el precio final de la pantalla obteniendo el mismo resultado.

9. Existen entornos de desarrollo de software de código abierto a libre disposición de los usuarios que disponen de librerías y repositorios con funcionalidades de los microcontroladores, que permiten la programación de dispositivos como el mostrado en este proyecto sin tener que realizar inversión económica en programas de desarrollo de Software.

En la siguiente imagen se muestra el diagrama de Gantt del proyecto, donde se pueden observar las modificaciones realizadas de las tareas requeridas respecto a la estimación inicial:

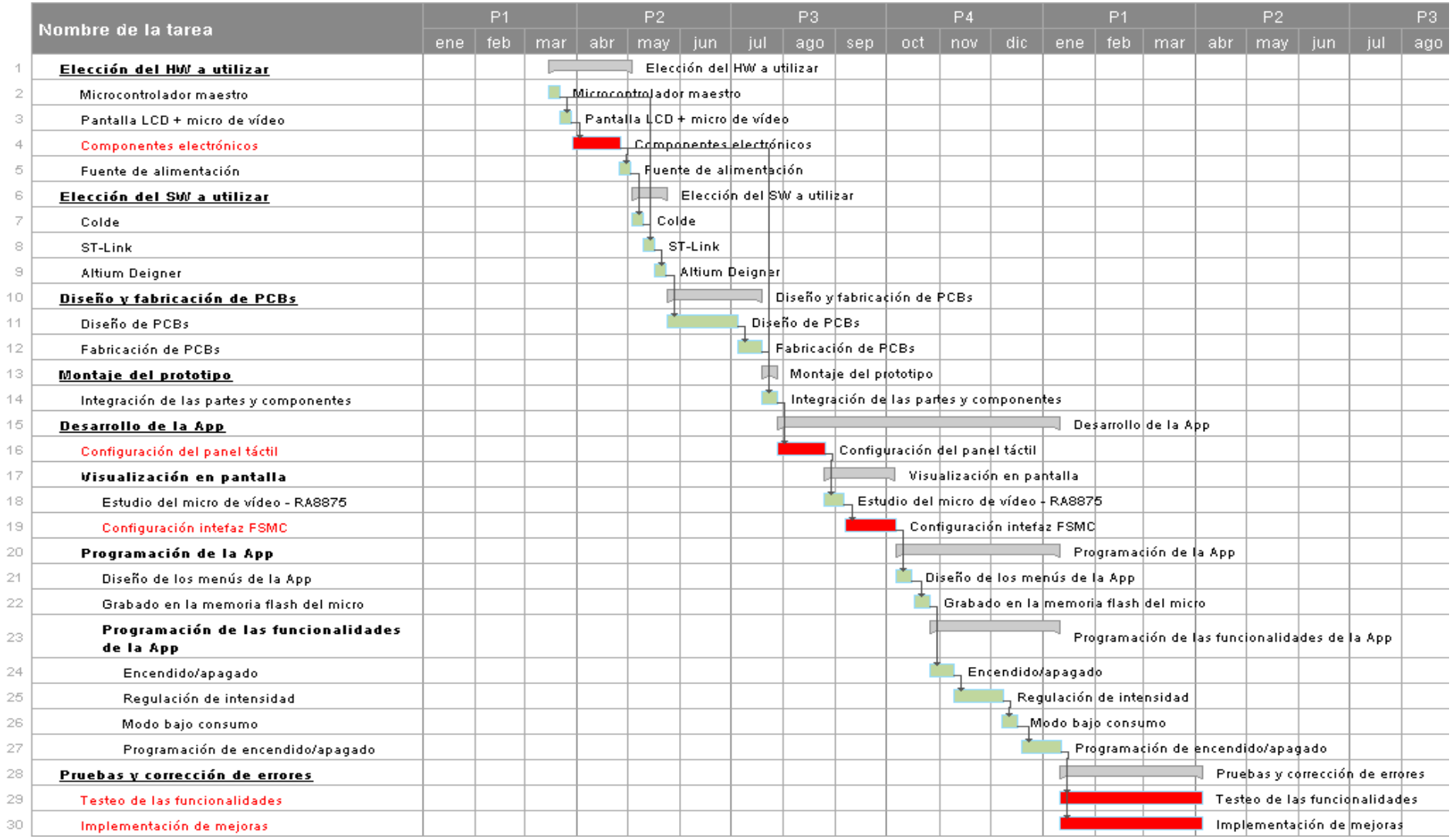


Figura 35: Diagrama de Gantt final

7. Futuras mejoras

En este capítulo se van a comentar distintas mejoras que podrían ser incluidas en el prototipo en un futuro próximo. Estas mejoras supondrían un incremento de las funcionalidades que puede llevar a cabo el dispositivo.

7.1 ETHERNET

Una posible mejora a introducir en el dispositivo, sería el desarrollo de una aplicación web y/o móvil para el control del dispositivo de forma remota.

Para aportar esta funcionalidad, sería necesario llevar hasta el dispositivo, un cable ETHERNET y conectarlo al dispositivo mediante un conector RJ45. Dicho conector debería integrarse en la placa Framework, junto con su circuitería asociada. El siguiente esquema muestra la circuitería asociada del conector, necesaria para para la adaptación del conector RJ45 con el micro maestro:

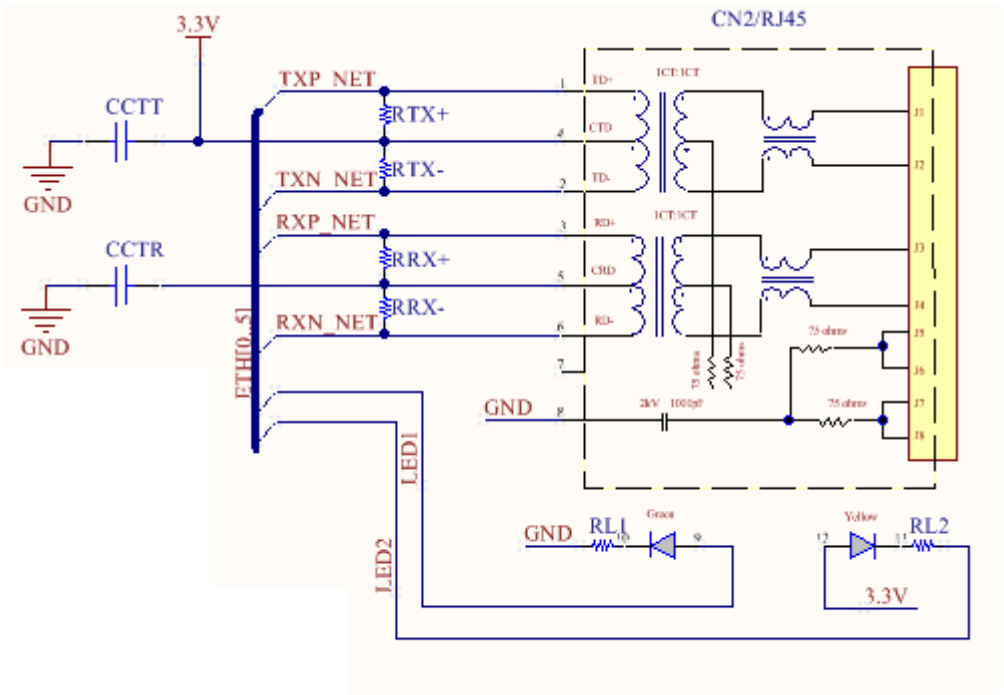


Figura 36: Esquemático conector RJ45

El microcontrolador escogido, dispone de un módulo de ETHERNET, por lo que bastaría con conectar los pines del micro maestro destinados a tal efecto, con los correspondientes pines del conector RJ45. La huella que habría que incluir en la placa framework para poder soldar a ella el conector RJ45, es como la que se muestra a continuación:

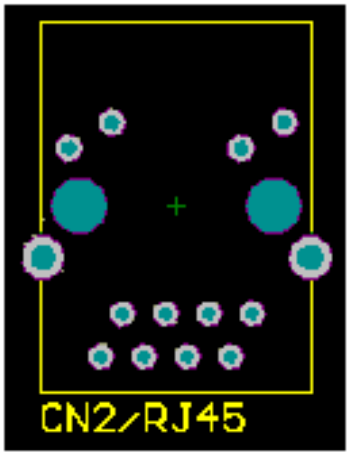


Figura 37 Huella del conector RJ45

Los pines del micro maestro destinados al protocolo ETHERNET, que son los que habría que conectar al conector RJ45, se muestran en la siguiente tabla junto con sus funciones asociadas:

PIN Micro	Función asociada
PB10	**ETH_MII_RXER**
PB11	**ETH_MII_TXEN**
PB12	**ETH_MII_TXD0**
PB13	**ETH_MII_TXD1**
PC4	**ETH_MII_RXD0**
PC5	**ETH_MII_RXD1**

Tabla 14: Pines asociados a funciones Ethernet

Como se puede comprobar en el listado de pines utilizados en el microcontrolador maestro, estos pines se han dejado libres para posibilitar la futura inclusión del módulo de ETHERNET.

Una vez conectado el cable al dispositivo, habría que implementar un protocolo de comunicación basado en TCP/IP, y desarrollar la aplicación móvil y/o web que permita el manejo del dispositivo de forma remota.

7.2 Bombillas de bajo consumo

La utilización de bombillas de bajo consumo, no soporta la función de regulador de intensidad, debido a que estas bombillas llevan un transformador interno que modifica completamente el funcionamiento.

Dado que cada fabricante de bombillas, incluye un transformador diferente para cada bombilla, no es posible encontrar un regulador de intensidad que funcione de manera universal para todas las bombillas de bajo consumo.

Debido a estas restricciones, la utilización del regulador de intensidad en bombillas de bajo consumo, normalmente no reducirá la intensidad de la luz que emite. En las ocasiones en las que el transformador incluido por el fabricante tenga unas características determinadas, sí se conseguirá regular la intensidad de la luz, en otras ocasiones solo se conseguirá un parpadeo en la luz cuando se eligen valores bajos de luminosidad. Por último, habrá transformadores que no permitan ningún tipo de regulación en la intensidad de la luz.

La posible solución a este problema, pasará por la universalización de los estándares de diseño de los transformadores que llevan integrados las bombillas de bajo consumo.

8. ANEXOS

8.1 ANEXO I - Características de la pantalla:

La pantalla con la que se va a trabajar en el proyecto, tiene una resolución de 800x480 píxeles y las dimensiones (en mm) que se especifican a continuación:

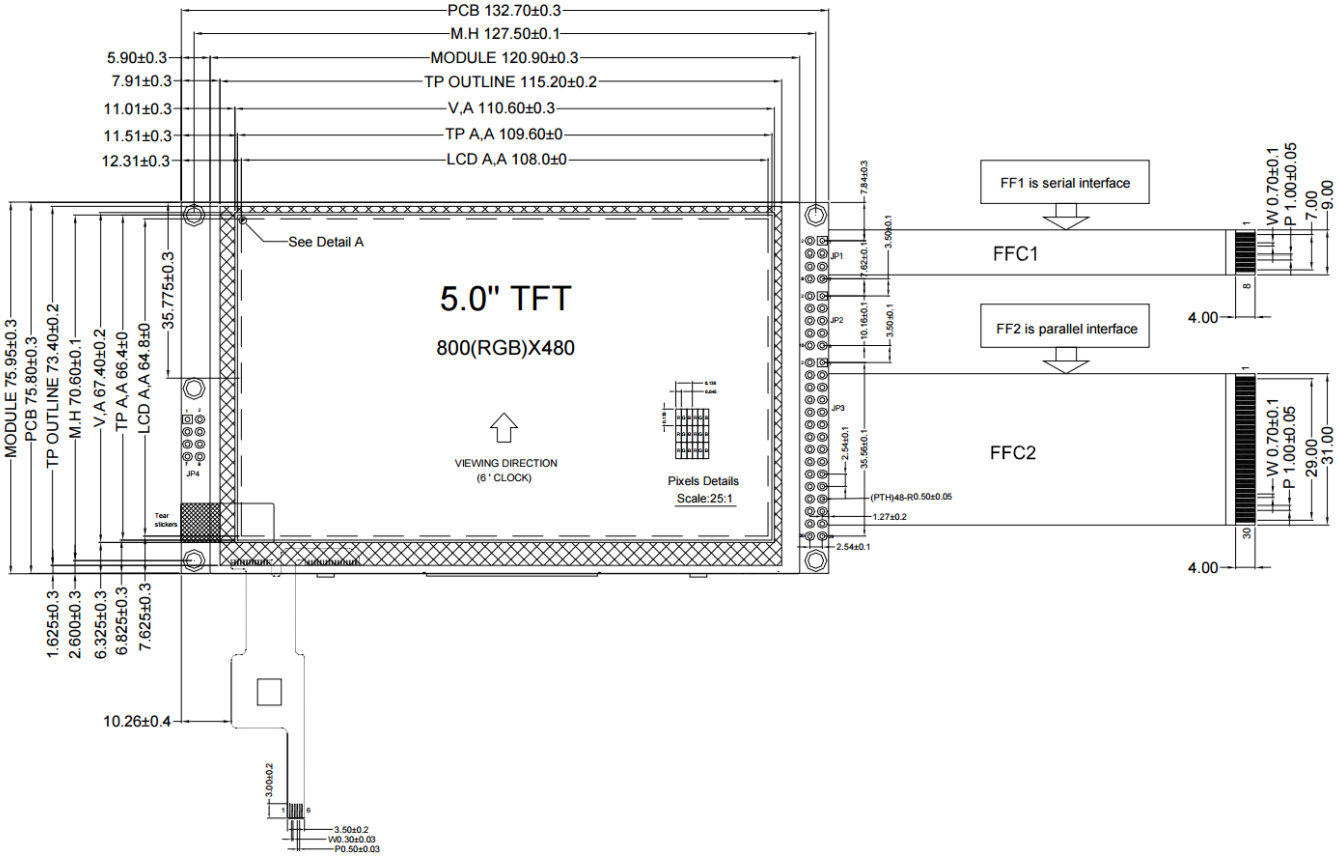


Figura 38: Dimensiones de la pantalla LCD

La siguiente figura, muestra la descripción de pines de la interfaz paralela de la pantalla LCD, que es la interfaz encargada de la comunicación entre el micro maestro y el micro de vídeo integrado en la pantalla.

Pin No	Symbol	Descriptions
1,2	VSS	Ground
3,4	VDD	Power Supply
5	E_/RD	Enable/Read Enable When MCU interface (I/F) is 8080 series, this pin is used as RD# signal (Data Read) , active low. When MCU I/F is 6800 series, this pin is used as EN signal (Enable), active high.
6	R/W_/WR	Write/Read-Write When MCU I/F is 8080 series, this pin is used as WR# signal (data write) , active low. When MCU I/F is 6800 series, this pin is used as RW# signal (data read/write control) . Active high for read and active low for write.
7	/CS	Chip Select Input Low active chip select pin.
8	RS	Command / Data Select Input The pin is used to select command/data cycle. RS = 0, data Read/Write cycle is selected. RS = 1, status read/command write cycle is selected. In 8080 interface, usually it connects to “A0” address pin.
9	WAIT	Wait Signal Output This is a WAIT# output to indicate the RA8875 is in busy state. The RA8875 can’t access MCU cycle when WAIT# pin is active. It is active low and could be used for MCU to poll busy status by connecting it to I/O port.
10	INT	Interrupt Signal Output The interrupt output for MCU to indicate the status of RA8875.
11	/RESET(NC)	Master synchronizes reset, Active Low. RC Reset circuit on board.
12	C86(NC)	MCU Interface Select 0: 8080 interface is selected 1: 6800 interface is selected Internally connected to the low level.
13	VSS	Ground
14	BL_CONTROL	BackLight control signal input.When using the internal PWM signal this pin floating.
15-30	DB0-DB15	Data Bus.These are data bus for data transfer between MCU and RA8875. When setting register number and register data, DB[7:0] is used. When writing data to display RAM, DB[15:0] is used according to data bus mode setting. DB[15:8] will be input and should be pull-low or pullhigh when 8-bit data bus mode is used.

Figura 39: Descripción de pines de la interfaz paralela de la pantalla LCD

La siguiente figura ha sido obtenida del datasheet de la pantalla LCD, y muestra las características eléctricas de la misma:

4.7 Electrical Characteristics

ITEM	SYMBOL	MIN.	TYP.	MAX.	UNIT
Power supply voltage	VDD	3.0	3.3	3.6	V
Input voltage 'H' level	VIH	0.8VDDIO	-	VDDIO	V
Input voltage 'L' level	VIL	GND	-	0.2VDDIO	V
Output voltage 'H' level	VOH	VDDIO-0.4	-	VDDIO	V
Output voltage 'L' level	VCL	GND	-	GND+0.4	V
Schmitt-Trigger Input (*1)					
Input voltage 'H' level	VIH	0.7VDDIO	-	VDDIO	V
Input voltage 'L' level	VIL	GND	-	0.3VDDIO	V
Input Leakage Current 1（*2）	VIH	--	+	+2	uA
Input Leakage Current 2（*2）	VIL	--		-2	uA
Model Current	IDD	--	--	450	mA
Sleep Mode（*2）	ISLP	--	320	--	uA

Note1: Signals RD#, WR#, CS#, RS, RST# are inputs of Schmitt-trigger.

Note2: Oscillator Clock = 20MHz, System Clock = 20~60MHz,VSYNC = 45~65Hz, T_A=25 ℃

Figura 40: Características eléctricas de la pantalla LCD

8.2 ANEXO II – Características del chip GSL1680

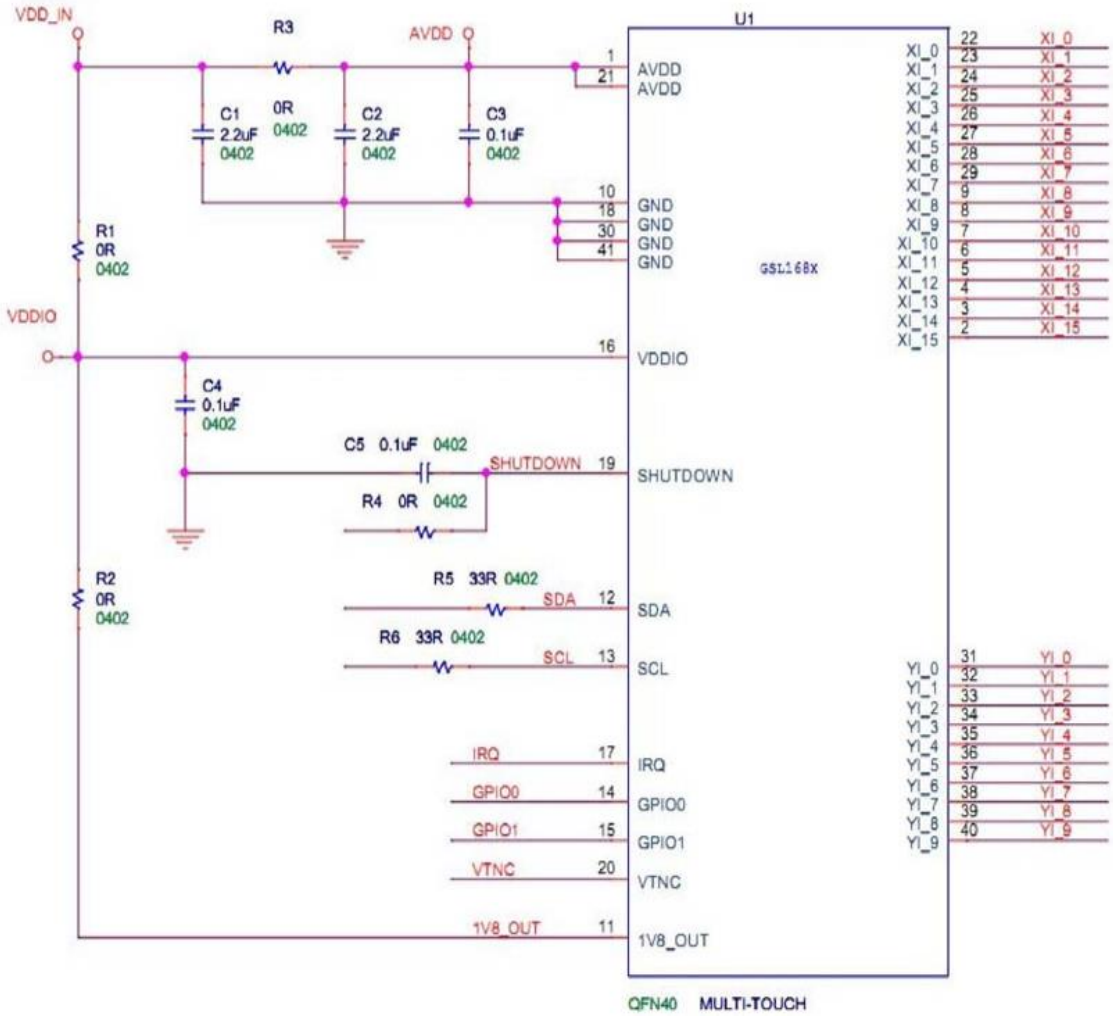


Figura 41: Circuito interno del chip GSL1680

8.3 ANEXO III – Esquemático completo y PCBs

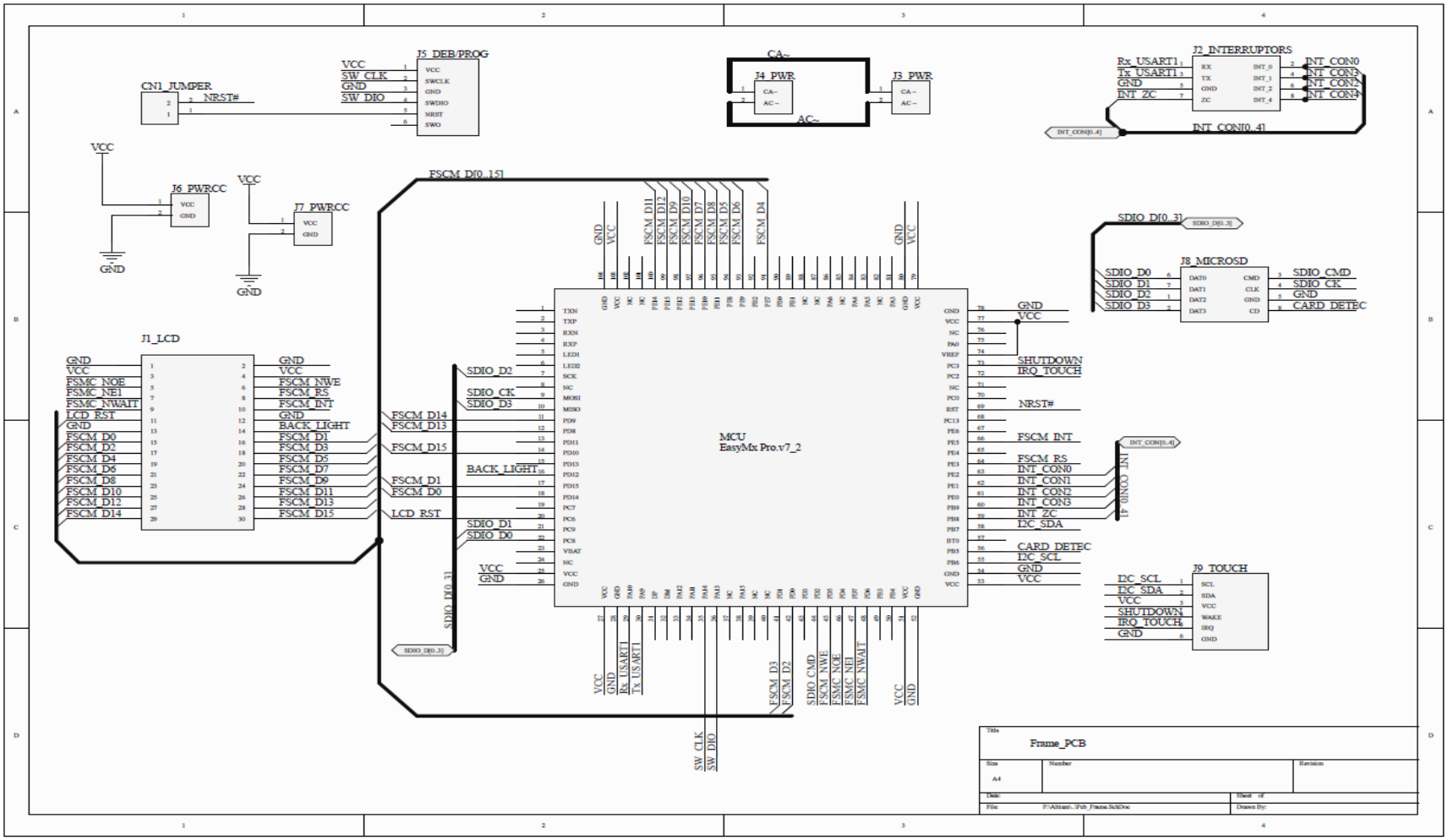


Figura 42: Esquemático completo

PCB de la placa Framework:

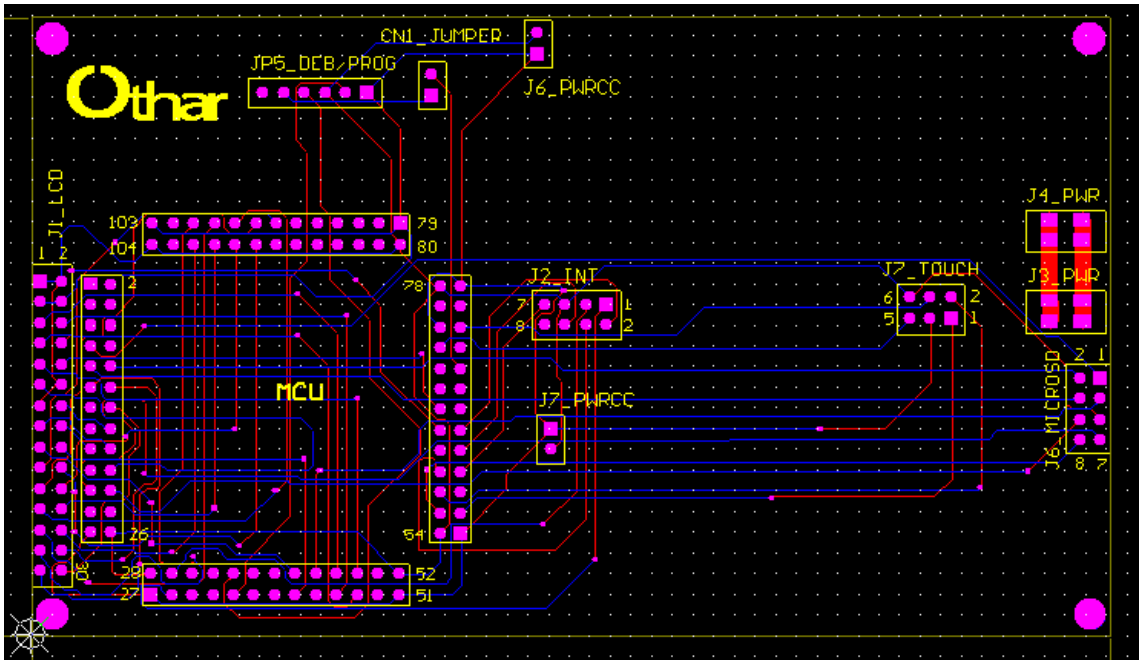


Figura 43: Placa framework

PCB del driver de corriente:

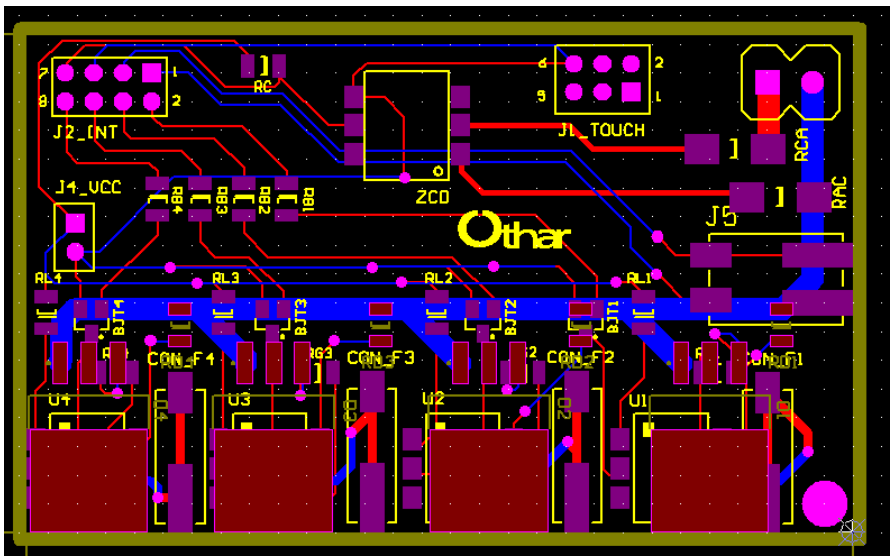


Figura 44: Placa Driver de corriente

8.4 ANEXO IV – Datasheet del optoacoplador

ABSOLUTE MAXIMUM RATINGS (T_{amb} = 25 °C, unless otherwise specified)				
PARAMETER	**TEST CONDITION**	**SYMBOL**	**VALUE**	**UNIT**
INPUT				
Reverse voltage		V_R	6	V
Forward current - continuous		I_F	60	mA
Power dissipation		P_{diss}	100	mW
OUTPUT				
Off state output terminal voltage		V_{DRM}	600	V
Peak non-repetitive surge current	PW = 100 ms, 120 pps	I_{TSM}	1	A
Power dissipation		P_{diss}	200	mW
On-state RMS current		$I_{T(RMS)}$	100	mA
COUPLER				
Isolation test voltage	t = 1 s	V_{ISO}	5300	V_{RMS}
Total power dissipation		P_{tot}	300	mW
Operating temperature		T_{amb}	- 55 to + 100	°C
Storage temperature		T_{stg}	- 55 to + 150	°C
Soldering temperature [1]	10 s	T_{sld}	260	°C

Figura 45: Características del optoacoplador VO3053

8.5 ANEXO V – Descripción de pines y funciones alternativas del micro

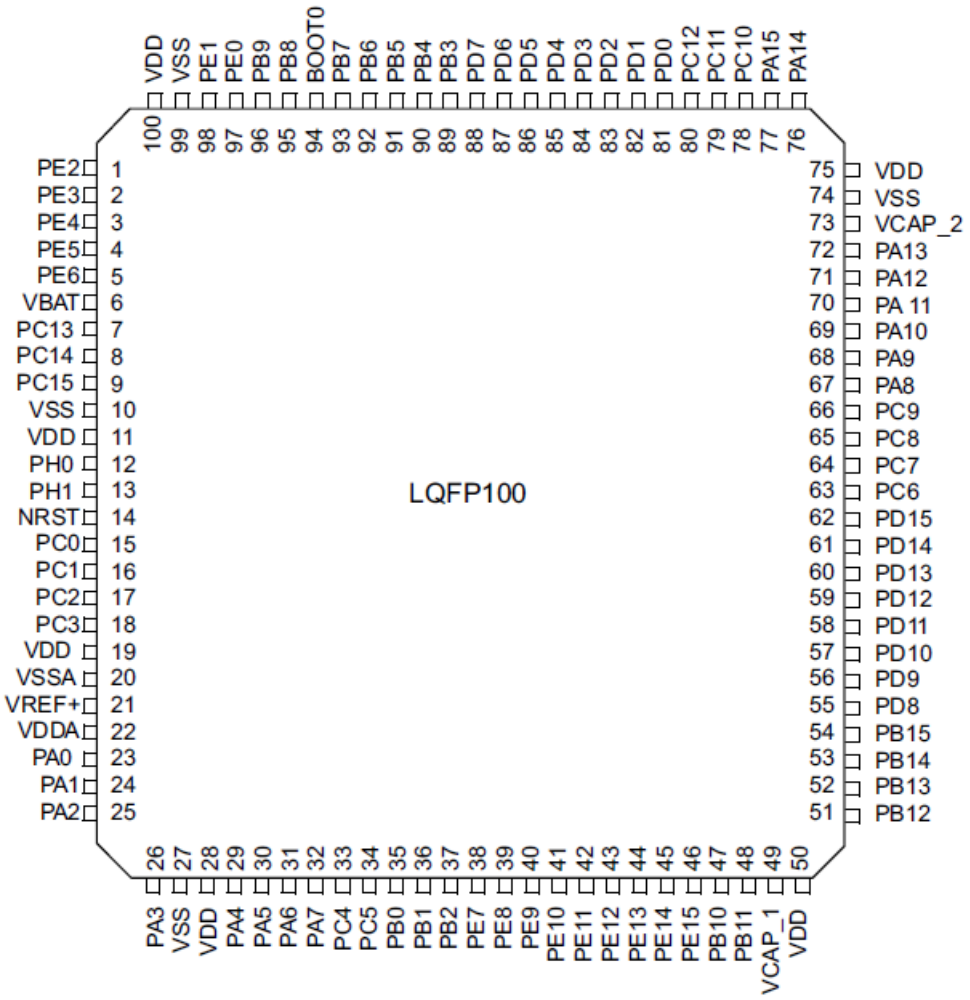


Figura 46: Descripción de pines del micro STM32F407VGT6

Puerto A:

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11	I2C1/2/3	SPI1/SPI2/I2S2/I2S2ext	SPI3/I2Sext/I2S3	USART1/2/3/I2S3ext	UART4/5/USART6	CAN1/CAN2/TIM12/13/14	OTG_FS/OTG_HS	ETH	FSMC/SDIO/OTG_FS	DCMI		
Port A	PA0		TIM2_CH1_ETR	TIM 5_CH1	TIM8_ETR				USART2_CTS	UART4_TX			ETH_MII_CRS				EVENTOUT
	PA1		TIM2_CH2	TIM5_CH2					USART2_RTS	UART4_RX			ETH_MII_RX_CLK ETH_RMII__REF_CLK				EVENTOUT
	PA2		TIM2_CH3	TIM5_CH3	TIM9_CH1				USART2_TX				ETH_MDIO				EVENTOUT
	PA3		TIM2_CH4	TIM5_CH4	TIM9_CH2				USART2_RX			OTG_HS_ULPI_D0	ETH _MII_COL				EVENTOUT
	PA4						SPI1_NSS	SPI3_NSS I2S3_WS	USART2_CK					OTG_HS_SOF	DCMI_HSYNC		EVENTOUT
	PA5		TIM2_CH1_ETR		TIM8_CH1N		SPI1_SCK					OTG_HS_ULPI_CK					EVENTOUT
	PA6		TIM1_BKIN	TIM3_CH1	TIM8_BKIN		SPI1_MISO				TIM13_CH1				DCMI_PIXCK		EVENTOUT
	PA7		TIM1_CH1N	TIM3_CH2	TIM8_CH1N		SPI1_MOSI				TIM14_CH1		ETH_MII _RX_DV ETH_RMII _CRS_DV				EVENTOUT
	PA8	MCO1	TIM1_CH1			I2C3_SCL			USART1_CK			OTG_FS_SOF					EVENTOUT
	PA9		TIM1_CH2			I2C3_SMBA			USART1_TX						DCMI_D0		EVENTOUT
	PA10		TIM1_CH3						USART1_RX			OTG_FS_ID			DCMI_D1		EVENTOUT
	PA11		TIM1_CH4						USART1_CTS		CAN1_RX	OTG_FS_DM					EVENTOUT
	PA12		TIM1_ETR						USART1_RTS		CAN1_TX	OTG_FS_DP					EVENTOUT
	PA13	JTMS-SWDIO															EVENTOUT
	PA14	JTCK-SWCLK															EVENTOUT
	PA15	JTDI	TIM 2_CH1 TIM 2_ETR				SPI1_NSS	SPI3_NSS/I2S3_WS									EVENTOUT

Figura 47: Descripción de las funciones alternativas del puerto A del micro maestro

Puerto B:

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11	I2C1/2/3	SPI1/SPI2/I2S2/I2S2ext	SPI3/I2Sext/I2S3	USART1/2/3/I2S3ext	UART4/5/USART6	CAN1/CAN2/TIM12/13/14	OTG_FS/OTG_HS	ETH	FSMC/SDIO/OTG_FS	DCMI		
Port B	PB0		TIM1_CH2N	TIM3_CH3	TIM8_CH2N							OTG_HS_ULPI_D1	ETH _MII_RXD2				EVENTOUT
	PB1		TIM1_CH3N	TIM3_CH4	TIM8_CH3N							OTG_HS_ULPI_D2	ETH _MII_RXD3				EVENTOUT
	PB2																EVENTOUT
	PB3	JTDO/TRACESWO	TIM2_CH2				SPI1_SCK	SPI3_SCK I2S3_CK									EVENTOUT
	PB4	NJTRST		TIM3_CH1			SPI1_MISO	SPI3_MISO	I2S3ext_SD								EVENTOUT
	PB5			TIM3_CH2		I2C1_SMBA	SPI1_MOSI	SPI3_MOSI I2S3_SD			CAN2_RX	OTG_HS_ULPI_D7	ETH _PPS_OUT		DCMI_D10		EVENTOUT
	PB6			TIM4_CH1		I2C1_SCL			USART1_TX		CAN2_TX				DCMI_D5		EVENTOUT
	PB7			TIM4_CH2		I2C1_SDA			USART1_RX					FSMC_NL	DCMI_VSYNC		EVENTOUT
	PB8			TIM4_CH3	TIM10_CH1	I2C1_SCL					CAN1_RX		ETH _MII_TXD3	SDIO_D4	DCMI_D6		EVENTOUT
	PB9			TIM4_CH4	TIM11_CH1	I2C1_SDA	SPI2_NSS I2S2_WS				CAN1_TX			SDIO_D5	DCMI_D7		EVENTOUT
	PB10		TIM2_CH3			I2C2_SCL	SPI2_SCK I2S2_CK		USART3_TX			OTG_HS_ULPI_D3	ETH_ MII_RX_ER				EVENTOUT
	PB11		TIM2_CH4			I2C2_SDA			USART3_RX			OTG_HS_ULPI_D4	ETH _MII_TX_EN ETH _RMII_TX_EN				EVENTOUT
	PB12		TIM1_BKIN			I2C2_SMBA	SPI2_NSS I2S2_WS		USART3_CK		CAN2_RX	OTG_HS_ULPI_D5	ETH _MII_TXD0 ETH _RMII_TXD0	OTG_HS_ID			EVENTOUT
	PB13		TIM1_CH1N				SPI2_SCK I2S2_CK		USART3_CTS		CAN2_TX	OTG_HS_ULPI_D6	ETH _MII_TXD1 ETH _RMII_TXD1				EVENTOUT
	PB14		TIM1_CH2N		TIM8_CH2N		SPI2_MISO	I2S2ext_SD	USART3_RTS		TIM12_CH1			OTG_HS_DM			EVENTOUT
	PB15	RTC_REFIN	TIM1_CH3N		TIM8_CH3N		SPI2_MOSI I2S2_SD				TIM12_CH2			OTG_HS_DP			EVENTOUT

Figura 48: Descripción de las funciones alternativas del puerto B del micro maestro

Puerto C:

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11	I2C1/2/3	SPI1/SPI2/ I2S2/I2S2ext	SPI3/I2Sext/ I2S3	USART1/2/3/ I2S3ext	UART4/5/ USART6	CAN1/ CAN2/ TIM12/13/14	OTG_FS/ OTG_HS	ETH	FSMC/SDIO/ OTG_FS	DCMI		
Port C	PC0											OTG_HS_ULPI_ STP					EVENTOUT
	PC1												ETH_MDC				EVENTOUT
	PC2						SPI2_MISO	I2S2ext_SD				OTG_HS_ULPI_ DIR	ETH _MII_TXD2				EVENTOUT
	PC3						SPI2_MOSI I2S2_SD					OTG_HS_ULPI_ NXT	ETH _MII_TX_CLK				EVENTOUT
	PC4												ETH_MII_RXD0 ETH_RMII_RXD0				EVENTOUT
	PC5												ETH _MII_RXD1 ETH _RMII_RXD1				EVENTOUT
	PC6			TIM3_CH1	TIM8_CH1		I2S2_MCK			USART6_TX				SDIO_D6	DCMI_D0		EVENTOUT
	PC7			TIM3_CH2	TIM8_CH2			I2S3_MCK		USART6_RX				SDIO_D7	DCMI_D1		EVENTOUT
	PC8			TIM3_CH3	TIM8_CH3					USART6_CK				SDIO_D0	DCMI_D2		EVENTOUT
	PC9	MCO2		TIM3_CH4	TIM8_CH4	I2C3_SDA	I2S_CKIN							SDIO_D1	DCMI_D3		EVENTOUT
	PC10							SPI3_SCK/ I2S3_CK	USART3_TX/	UART4_TX				SDIO_D2	DCMI_D8		EVENTOUT
	PC11						I2S3ext_SD	SPI3_MISO/	USART3_RX	UART4_RX				SDIO_D3	DCMI_D4		EVENTOUT
	PC12							SPI3_MOSI I2S3_SD	USART3_CK	UART5_TX				SDIO_CK	DCMI_D9		EVENTOUT
	PC13																EVENTOUT
	PC14																EVENTOUT
	PC15																EVENTOUT

Figura 49: Descripción de las funciones alternativas del puerto C del micro maestro

Puerto D:

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11	I2C1/2/3	SPI1/SPI2/I2S2/I2S2ext	SPI3/I2Sext/I2S3	USART1/2/3/I2S3ext	UART4/5/USART6	CAN1/CAN2/TIM12/13/14	OTG_FS/OTG_HS	ETH	FSMC/SDIO/OTG_FS	DCMI		
Port D	PD0										CAN1_RX			FSMC_D2			EVENTOUT
	PD1										CAN1_TX			FSMC_D3			EVENTOUT
	PD2			TIM3_ETR						UART5_RX				SDIO_CMD	DCMI_D11		EVENTOUT
	PD3								USART2_CTS					FSMC_CLK			EVENTOUT
	PD4								USART2_RTS					FSMC_NOE			EVENTOUT
	PD5								USART2_TX					FSMC_NWE			EVENTOUT
	PD6								USART2_RX					FSMC_NWAIT			EVENTOUT
	PD7								USART2_CK					FSMC_NE1/FSMC_NCE2			EVENTOUT
	PD8								USART3_TX					FSMC_D13			EVENTOUT
	PD9								USART3_RX					FSMC_D14			EVENTOUT
	PD10								USART3_CK					FSMC_D15			EVENTOUT
	PD11								USART3_CTS					FSMC_A16			EVENTOUT
	PD12			TIM4_CH1					USART3_RTS					FSMC_A17			EVENTOUT
	PD13			TIM4_CH2										FSMC_A18			EVENTOUT
	PD14			TIM4_CH3										FSMC_D0			EVENTOUT
	PD15			TIM4_CH4										FSMC_D1			EVENTOUT

Figura 50: Descripción de las funciones alternativas del puerto D del micro maestro

Puerto E:

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11	I2C1/2/3	SPI1/SPI2/ I2S2/I2S2ext	SPI3/I2Sext/ I2S3	USART1/2/3/ I2S3ext	UART4/5/ USART6	CAN1/ CAN2/ TIM12/13/14	OTG_FS/ OTG_HS	ETH	FSMC/SDIO/ OTG_FS	DCMI		
Port E	PE0			TIM4_ETR										FSMC_NBL0	DCMI_D2		EVENTOUT
	PE1													FSMC_NBL1	DCMI_D3		EVENTOUT
	PE2	TRACECLK											ETH _MII_TXD3	FSMC_A23			EVENTOUT
	PE3	TRACED0												FSMC_A19			EVENTOUT
	PE4	TRACED1												FSMC_A20	DCMI_D4		EVENTOUT
	PE5	TRACED2			TIM9_CH1									FSMC_A21	DCMI_D6		EVENTOUT
	PE6	TRACED3			TIM9_CH2									FSMC_A22	DCMI_D7		EVENTOUT
	PE7		TIM1_ETR											FSMC_D4			EVENTOUT
	PE8		TIM1_CH1N											FSMC_D5			EVENTOUT
	PE9		TIM1_CH1											FSMC_D6			EVENTOUT
	PE10		TIM1_CH2N											FSMC_D7			EVENTOUT
	PE11		TIM1_CH2											FSMC_D8			EVENTOUT
	PE12		TIM1_CH3N											FSMC_D9			EVENTOUT
	PE13		TIM1_CH3											FSMC_D10			EVENTOUT
	PE14		TIM1_CH4											FSMC_D11			EVENTOUT
	PE15		TIM1_BKIN											FSMC_D12			EVENTOUT

Figura 51: Descripción de las funciones alternativas del puerto E del micro maestro

Descripción de pines utilizados del microcontrolador:

En la siguiente tabla se muestran todos los pines del microcontrolador maestro, y la interfaz a las que están asociados.

	Pin 0	Pin 1	Pin 2	Pin 3	Pin 4	Pin 5	Pin 6	Pin 7	Pin 8	Pin 9	Pin 10	Pin 11	Pin 12	Pin 13	Pin 14	Pin 15
PUERTO A											**S.C**			**SWD**	**SWD**	
PUERTO B							**GSL**	**GSL**	**ZCD**	**S.C**						
PUERTO C			**GSL**	**GSL**			**FSMC**									
PUERTO D	**FSMC**	**FSMC**		**FSMC**	**FSMC**	**FSMC**			**FSMC**	**FSMC**	**FSMC**		**FSMC**		**FSMC**	**FSMC**
PUERTO E	**S.C**		**S.C**						**FSMC**	**FSMC**	**FSMC**	**FSMC**	**FSMC**	**FSMC**	**FSMC**	**FSMC**

Tabla 15: Relación de pines utilizados e interfaces asociadas

Donde las siglas mostradas en la tabla anterior, representan las siguientes interfaces:

S.C – Señal de control

ZCD – Zero cross detector

GSL – Chip GSL1680, (control de pantalla táctil)

SWD – Interfaz de programación del micro (Serial Wire Debug)

FSMC – Interfaz de comunicación entre micro de vídeo y micro maestro

Descripción de pines para la interfaz FSMC del micro:

PIN FSMC	PIN STM32
FSMC – DO	PD14
FSMC – D1	PD15
FSMC – D2	PD0
FSMC – D3	PD1
FSMC – D4	PE7
FSMC – D5	PE8
FSMC – D6	PE9
FSMC – D7	PE10
FSMC – D8	PE11
FSMC – D9	PE12
FSMC – D1O	PE13
FSMC – D11	PE14
FSMC – D12	PE15
FSMC – D13	PD8
FSMC – D14	PD9
FSMC – D15	PD10

Tabla 16: Pines de datos de la interfaz FSMC

Como se puede observar en la tabla de funciones alternativas del puerto D, el reloj para el funcionamiento de la interfaz FSMC, está asociado al pin PD3. Mientras que el pin asociado a la función de reset de la interfaz FSMC, es el pin PC6.

El resto de pines para la definición de la interfaz FSMC son:

PIN FSMC	PIN STM32
FSMC – WR	PD5
FSMC – RS	PD4
FSMC – BL	PD12
FSMC – Reset	PC6
FSMC – CLK	PD3

Tabla 17: Pines de control para la interfaz FSMC

El pin BL, sirve para controlar la luminosidad de la pantalla. Este pin está funcionando en modo PWM del Timer 4.

8.6 ANEXO VI – Estructura principal y sub-estructuras

A continuación se muestra la definición de la estructura principal "**_screen**":

```
typedef struct {
      _time dateTime;
      uint8_t timeSet;
      uint8_t numLightsON;
      LightsSelected lightsSet;
      MenuTypes menu;
      AlarmSel alarmSel;
      _alarm alarms[2];
      _touch_event touch_event[5];
      LIGHT_t LIGHTS[4];
} _screen ;
```

Los tipos de las variables de la estructura principal se definen a continuación:

Definición del tipo **LightsSelected:**

```
typedef enum {
      No_LightsSel =      0b00000000,
      Light_1 =           0b11000000,
      Light_2 =           0b00110000,
      Light_3 =           0b00001100,
      Light_4 =           0b00000011,
      Lights_1_2 =        0b11110000,
      Lights_1_3 =        0b11001100,
      Lights_1_4 =        0b11000011,
      Lights_2_3 =        0b00111100,
      Lights_2_4 =        0b00110011,
      Lights_3_4 =        0b00001111,
      Lights_1_2_3 =      0b11111100,
      Lights_1_2_4 =      0b11110011,
      Lights_1_3_4 =      0b11001100,
      Lights_2_3_4 =      0b00111111,
```

```
    allLights_sel =     0b11111111,
} LightsSelected;
```

Definición del tipo **MenuTypes:**

```
typedef enum {
    BUTTONS = 0X00, DIMMERS = 0X01, SEL_PROGRAM = 0x02,
PROGRAM = 0x03, CONF_TIME = 0x04,
} MenuTypes;
```

Definición del tipo **AlarmSel:**

```
typedef enum {
    NO_AlarmSel = 0X00, SET_ON = 0X01, SET_OFF = 0x02,
} AlarmSel;
```

La estructura **_alarm** se define a continuación:

```
typedef struct{
    char *name;
    Status status;
    LightsSelected ALRM_lightsSet;
    RTC_TimeTypeDef alarmTime;
} _alarm;
```

Definición de la estructura **_time**:

```
typedef struct  {
    RTC_DateTypeDef screenDate;
    RTC_TimeTypeDef screenTime;
}_time;
```

Definición de la estructura **RTC_DateTypeDef**:

```
typedef struct
{
  uint8_t RTC_WeekDay; // Specifies the RTC Date WeekDay.
From 1 to 7
  uint8_t RTC_Month;   // Specifies the RTC Date Month.
From 1 to 12
  uint8_t RTC_Date;    // Specifies the RTC Date. From 1 to
31
  uint8_t RTC_Year;    // Specifies the RTC Date Year. From
0 to 99
}RTC_DateTypeDef;
```

Definición de la estructura **RTC_TimeTypeDef**:

```
typedef struct
{
  uint8_t RTC_Hours;    // Specifies the RTC Time Hour.
From 1 to 12
  uint8_t RTC_Minutes;  // Specifies the RTC Time Minute.
From 0 to 59
  uint8_t RTC_Seconds;  // Specifies the RTC Time Second.
From 0 to 59
  uint8_t RTC_H12;      // Specifies the RTC AM/PM Time.
}RTC_TimeTypeDef;
```

La estructura **LIGHT_t** se define a continuación:

```
typedef struct {
     char *name;
     Status status;
```

```
    uint16_t dimmer;
    Status trigger
} LIGHT_t;
```

La estructura **_touch_event** se define a continuación:

```
typedef struct  {
    uint8_t n_fingers;
_coord coords;
}_touch_event;
```

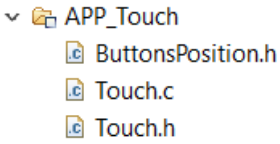
La estructura **_coords** se define a continuación:

```
typedef struct {
    uint32_t x, y;
    uint8_t finger;
}_coord
```

8.7 ANEXO VII – Código de la aplicación

Debido a lo extenso del código, únicamente se incluye el código más relevante de la aplicación, es decir, el concerniente al desarrollo de las funcionalidades requeridas para el dispositivo.

8.7.1 APP_Touch



8.7.1.1 ButtonsPosition.h

```
//******      Define buttons position for BUTTONS screen     *********
//Date buttons
#ifndef YminDateUp
#define YminDateUp 150
#define YmaxDateUp 225
#define YminDateDown 280
#define YmaxDateDown 355
#define XminDayUp 50
#define XmaxDayUp 125
#define XminDayDown 50
#define XmaxDayDown 125
#define XminMonthUp 170
#define XmaxMonthUp 245
#define XminMonthDown 170
#define XmaxMonthDown 245
#define XminYearUp 290
#define XmaxYearUp 365
#define XminYearDown 290
#define XmaxYearDown 365
//Time buttons
#define XminTimeHUp 540
#define XmaxTimeHUp 615
#define XminTimeHDown 540
#define XmaxTimeHDown 615
#define XminTimeMUp 660
#define XmaxTimeMUp 735
#define XminTimeMDown 660
#define XmaxTimeMDown 735
//Save buttons
#define YminSave 395
#define YmaxSave 480
#define XminSave 280
#define XmaxSave 460
```

```
//******      Define buttons position for BUTTONS screen     *********
//Light buttons
#define YminLights 150
#define YmaxLights 300
#define XminLight1 40
#define XmaxLight1 190
#define XminLight2 230
#define XmaxLight2 380
#define XminLight3 420
#define XmaxLight3 570
#define XminLight4 610
#define XmaxLight4 760
//Settings button
#define YminSettings 420
#define YmaxSettings 480
#define XminSettings 340
#define XmaxSettings 460

//******      Define buttons position for DIMMERS screen     *********
//Dimmer buttons
#define XminDimmers 250
#define XmaxDimmers 520
#define YminDim1 120
#define YmaxDim1 160
#define YminDim2 190
#define YmaxDim2 230
#define YminDim3 260
#define YmaxDim3 300
#define YminDim4 330
#define YmaxDim4 370
//ProgLights button
#define YminProgLights 200
#define YmaxProgLights 280
#define XminProgLights 620
#define XmaxProgLights 760

//******      Define buttons position for SEL_PROGRAM screen
*********
//Selection Program buttons
#define YminSetTime 160
#define YmaxSetTime 280
#define XminSetTimeON 150
#define XmaxSetTimeON 350
#define XminSetTimeOFF 450
#define XmaxSetTimeOFF 650

//******      Define buttons position for PROGRAM screen     *********
//Selection Lights buttons
#define XminSelLights 630
#define XmaxSelLights 700
#define YminSelLight1 110
#define YmaxSelLight1 180
#define YminSelLight2 200
#define YmaxSelLight2 270
#define YminSelLight3 290
#define YmaxSelLight3 360
#define YminSelLight4 380
#define YmaxSelLight4 450
// up/down time buttons
#define YminTimeButtons 280
```

```
#define YmaxTimeButtons 350
#define XminHourUP 100
#define XmaxHourUP 160
#define XminHourDOWN 170
#define XmaxHourDOWN 230
#define XminMinuteUP 280
#define XmaxMinuteUP 340
#define XminMinuteDOWN 350
#define XmaxMinuteDOWN 410
//SET/RESTORE buttons
#define XminSETREST 480
#define XmaxSETREST 580
#define YminSET 170
#define YmaxSET 270
#define YminREST 300
#define YmaxREST 400
//SetON/SetOFF selection buttons
#define YminAlarmType 50
#define YmaxAlarmType 130
#define XminAlarmON 70
#define XmaxAlarmON 220
#define XminAlarmOFF 320
#define XmaxAlarmOFF 470

//******      Define buttons position appearing in many screens
*********
//Home buttons
#define YminHome 420
#define YmaxHome 480
#define XminHome 340
#define XmaxHome 460

#endif
```

8.7.1.2 Touch.c

```
#include "stm32f4xx_conf.h"
#include "APP_Touch/Touch.h"
#include "ButtonsPosition.h"

#ifndef GSLX680_I2C_ADDR
#define GSLX680_I2C_ADDR            0x40
#endif

#ifndef GSL_DATA_REG_ADDR
#define GSL_DATA_REG_ADDR           0x80
#endif

#ifndef GSL_STATUS_REG_ADDR
#define GSL_STATUS_REG_ADDR         0xe0
#endif

#ifndef GSL_PAGE_REG_ADDR
#define GSL_PAGE_REG_ADDR           0xf0
#endif

extern _screen screen;

LightsSelected AUX_alarmLights = No_LightsSel;
uint8_t AUX_Hour = 0;
uint8_t AUX_Minute = 0;
uint8_t touch_data[8] = { 0 };
uint8_t touchReleased = 0;
uint8_t auxDay, auxMonth, auxHour, auxMinute = 0;
uint16_t auxYear = 0;

char buf[50];

/**************     Action Functions
*****************************/

void Read_Coords(_screen *screen) {

      i2c_Read(GSL_DATA_REG_ADDR, touch_data, 8);

      (*screen).touch_event.n_fingers = touch_data[0];
      (*screen).touch_event.coords.x = ((((uint32_t) touch_data[5]) <<
8)
                  | (uint32_t) touch_data[4]) & 0x00000FFF; // 12 bits
of X coord
      (*screen).touch_event.coords.y = ((((uint32_t) touch_data[7]) <<
8)
                  | (uint32_t) touch_data[6]) & 0x00000FFF; // 12 bits
of Y coord
      (*screen).touch_event.coords.finger = (uint32_t) touch_data[7]
>> 4; // finger that did the touch
      if (touch_data[0] == 0) {
            touchReleased = 1;
      }
}
/*
 void Read_Coords_MultiFingers() {

 uint8_t touch_data[24] = { 0 };
 int n = i2c_Read(GSL_DATA_REG_ADDR, touch_data, 24);
 uint8_t i;
 screen.touch_event.n_fingers = touch_data[0];
```

```
 for (i = 0; i < screen.touch_event.n_fingers; i++) {
 screen.touch_event.coords[i].x = ((((uint32_t) touch_data[(i * 4) +
5])
 << 8) | (uint32_t) touch_data[(i * 4) + 4]) & 0x00000FFF; // 12 bits
of X coord
 screen.touch_event.coords[i].y = ((((uint32_t) touch_data[(i * 4) +
7])
 << 8) | (uint32_t) touch_data[(i * 4) + 6]) & 0x00000FFF;
 screen.touch_event.coords[i].finger = (uint32_t) touch_data[(i * 4) +
7]
 >> 4; // finger that did the touch
 }

 }*/

/*************   Button-Touched Functions
*****************************/
// **************       MENU CONF_TIME   ******************
void touch_DayUp(_screen *screen, uint8_t monthYearChanged) {

      if (monthYearChanged == 0) {
            (*screen).dateTime.screenDate.RTC_Date++;
      }

      if ((*screen).dateTime.screenDate.RTC_Month == 1
                  || (*screen).dateTime.screenDate.RTC_Month == 3
                  || (*screen).dateTime.screenDate.RTC_Month == 5
                  || (*screen).dateTime.screenDate.RTC_Month == 7
                  || (*screen).dateTime.screenDate.RTC_Month == 8
                  || (*screen).dateTime.screenDate.RTC_Month == 10
                  || (*screen).dateTime.screenDate.RTC_Month == 12) {
            if ((*screen).dateTime.screenDate.RTC_Date > 31) {
                  (*screen).dateTime.screenDate.RTC_Date = 1;
            }
      } else if ((*screen).dateTime.screenDate.RTC_Month == 4
                  || (*screen).dateTime.screenDate.RTC_Month == 6
                  || (*screen).dateTime.screenDate.RTC_Month == 9
                  || (*screen).dateTime.screenDate.RTC_Month == 11) {
            if ((*screen).dateTime.screenDate.RTC_Date > 30) {
                  (*screen).dateTime.screenDate.RTC_Date = 1;
            }
      } else if ((*screen).dateTime.screenDate.RTC_Month == 2
                  && (*screen).dateTime.screenDate.RTC_Year % 4 != 0) {
            if ((*screen).dateTime.screenDate.RTC_Date > 28) {
                  (*screen).dateTime.screenDate.RTC_Date = 1;
            }
      } else if ((*screen).dateTime.screenDate.RTC_Month == 2
                  && (*screen).dateTime.screenDate.RTC_Year % 4 == 0) {
            if ((*screen).dateTime.screenDate.RTC_Date > 29) {
                  (*screen).dateTime.screenDate.RTC_Date = 1;
            }
      }
      LCD_PrintDate((*screen).dateTime.screenDate.RTC_Date,
                  (*screen).dateTime.screenDate.RTC_Month,
                  (*screen).dateTime.screenDate.RTC_Year);
}
void touch_DayDown(_screen *screen, uint8_t monthYearChanged) {

      if (monthYearChanged == 0) {
            (*screen).dateTime.screenDate.RTC_Date--;
      }
```

```
	if ((*screen).dateTime.screenDate.RTC_Month == 1
			|| (*screen).dateTime.screenDate.RTC_Month == 3
			|| (*screen).dateTime.screenDate.RTC_Month == 5
			|| (*screen).dateTime.screenDate.RTC_Month == 7
			|| (*screen).dateTime.screenDate.RTC_Month == 8
			|| (*screen).dateTime.screenDate.RTC_Month == 10
			|| (*screen).dateTime.screenDate.RTC_Month == 12) {
		if ((*screen).dateTime.screenDate.RTC_Date < 1) {
			(*screen).dateTime.screenDate.RTC_Date = 31;
		}
	} else if ((*screen).dateTime.screenDate.RTC_Month == 4
			|| (*screen).dateTime.screenDate.RTC_Month == 6
			|| (*screen).dateTime.screenDate.RTC_Month == 9
			|| (*screen).dateTime.screenDate.RTC_Month == 11) {
		if ((*screen).dateTime.screenDate.RTC_Date < 1) {
			(*screen).dateTime.screenDate.RTC_Date = 30;
		}
	} else if ((*screen).dateTime.screenDate.RTC_Month == 2
			&& (*screen).dateTime.screenDate.RTC_Year % 4 != 0) {
		if ((*screen).dateTime.screenDate.RTC_Date < 1) {
			(*screen).dateTime.screenDate.RTC_Date = 28;
		}
	} else if ((*screen).dateTime.screenDate.RTC_Month == 2
			&& (*screen).dateTime.screenDate.RTC_Year % 4 == 0) {
		if ((*screen).dateTime.screenDate.RTC_Date < 1) {
			(*screen).dateTime.screenDate.RTC_Date = 29;
		}
	}
	LCD_PrintDate((*screen).dateTime.screenDate.RTC_Date,
			(*screen).dateTime.screenDate.RTC_Month,
			(*screen).dateTime.screenDate.RTC_Year);
}
void touch_MonthUp(_screen *screen) {
	(*screen).dateTime.screenDate.RTC_Month++;

	if ((*screen).dateTime.screenDate.RTC_Month > 12) {
		(*screen).dateTime.screenDate.RTC_Month = 1;
	}

	touch_DayDown(screen, 1);
	touch_DayUp(screen, 1);

	LCD_PrintDate((*screen).dateTime.screenDate.RTC_Date,
			(*screen).dateTime.screenDate.RTC_Month,
			(*screen).dateTime.screenDate.RTC_Year);
}
void touch_MonthDown(_screen *screen) {
	(*screen).dateTime.screenDate.RTC_Month--;

	if ((*screen).dateTime.screenDate.RTC_Month < 1) {
		(*screen).dateTime.screenDate.RTC_Month = 12;
	}

	touch_DayDown(screen, 1);
	touch_DayUp(screen, 1);

	LCD_PrintDate((*screen).dateTime.screenDate.RTC_Date,
			(*screen).dateTime.screenDate.RTC_Month,
			(*screen).dateTime.screenDate.RTC_Year);
}
```

```
void touch_YearUp(_screen *screen) {
	(*screen).dateTime.screenDate.RTC_Year++;

	if ((*screen).dateTime.screenDate.RTC_Year > 99) {
		(*screen).dateTime.screenDate.RTC_Year = 0;
	}

	touch_DayDown(screen, 1);
	touch_DayUp(screen, 1);

	LCD_PrintDate((*screen).dateTime.screenDate.RTC_Date,
			(*screen).dateTime.screenDate.RTC_Month,
			(*screen).dateTime.screenDate.RTC_Year);
}
void touch_YearDown(_screen *screen) {
	(*screen).dateTime.screenDate.RTC_Year--;

	if ((*screen).dateTime.screenDate.RTC_Year > 100) {
		(*screen).dateTime.screenDate.RTC_Year = 99;
	}

	touch_DayDown(screen, 1);
	touch_DayUp(screen, 1);

	LCD_PrintDate((*screen).dateTime.screenDate.RTC_Date,
			(*screen).dateTime.screenDate.RTC_Month,
			(*screen).dateTime.screenDate.RTC_Year);
}
void touch_TimeHUp(_screen *screen) {

	(*screen).dateTime.screenTime.RTC_Hours++;

	if ((*screen).dateTime.screenTime.RTC_Hours > 23) {
		(*screen).dateTime.screenTime.RTC_Hours = 0;
	}
	LCD_PrintTime((*screen).dateTime.screenTime.RTC_Hours,
			(*screen).dateTime.screenTime.RTC_Minutes);

}
void touch_TimeHDown(_screen *screen) {
	(*screen).dateTime.screenTime.RTC_Hours--;

	if ((*screen).dateTime.screenTime.RTC_Hours > 100) {
		(*screen).dateTime.screenTime.RTC_Hours = 23;
	}
	LCD_PrintTime((*screen).dateTime.screenTime.RTC_Hours,
			(*screen).dateTime.screenTime.RTC_Minutes);
}
void touch_TimeMUp(_screen *screen) {
	(*screen).dateTime.screenTime.RTC_Minutes++;

	if ((*screen).dateTime.screenTime.RTC_Minutes > 59) {
		(*screen).dateTime.screenTime.RTC_Minutes = 0;
	}
	LCD_PrintTime((*screen).dateTime.screenTime.RTC_Hours,
			(*screen).dateTime.screenTime.RTC_Minutes);
}
void touch_TimeMDown(_screen *screen) {
	(*screen).dateTime.screenTime.RTC_Minutes--;

	if ((*screen).dateTime.screenTime.RTC_Minutes > 100) {
```

```
            (*screen).dateTime.screenTime.RTC_Minutes = 59;
      }
      LCD_PrintTime((*screen).dateTime.screenTime.RTC_Hours,
                  (*screen).dateTime.screenTime.RTC_Minutes);
}
void touch_Save(_screen *screen) {

      SetTime(screen);
}

// **************       MENU BOTONES      ******************
void touch_Light_Button(_screen *screen, uint8_t *numLight, uint8_t
alarm) {

      if (alarm == 0) {
            LCD_ToogleLightImage(screen, numLight);
      }
      Toogle_Light(screen, numLight);
      if (alarm == 0) {
            settingsButton(screen);
      }
      CheckLights_ZCD(screen);
}

void touch_Settings_Button(_screen *screen) {

      if ((*screen).numLightsON > 0) {
            (*screen).menu = DIMMERS;
            LCD_SetDimmerImage(); //imprime la imagen de los dimmers
      } else {
            (*screen).menu = SEL_PROGRAM;
            LCD_SetSelProgramImage(); //imprime la imagen de seleccion
de alarmas
      }
}

// **************       MENU DIMMERS      ******************
void touch_MinusDimmer1(_screen *screen) {
      (*screen).LIGHTS[0].dimmer += 1;
      if ((*screen).LIGHTS[0].dimmer > 6)
            (*screen).LIGHTS[0].dimmer = 6;
      LCD_ChangeDim1Image((*screen).LIGHTS[0].dimmer);
}
void touch_MinusDimmer2(_screen *screen) {

      (*screen).LIGHTS[1].dimmer += 1;
      if ((*screen).LIGHTS[1].dimmer > 6)
            (*screen).LIGHTS[1].dimmer = 6;
      LCD_ChangeDim2Image((*screen).LIGHTS[1].dimmer);
}
void touch_MinusDimmer3(_screen *screen) {

      (*screen).LIGHTS[2].dimmer += 1;
      if ((*screen).LIGHTS[2].dimmer > 6)
            (*screen).LIGHTS[2].dimmer = 6;
      LCD_ChangeDim3Image((*screen).LIGHTS[2].dimmer);
}
void touch_MinusDimmer4(_screen *screen) {

      (*screen).LIGHTS[3].dimmer += 1;
      if ((*screen).LIGHTS[3].dimmer > 6)
```

```
            (*screen).LIGHTS[3].dimmer = 6;
      LCD_ChangeDim4Image((*screen).LIGHTS[3].dimmer);
}
void touch_PlusDimmer1(_screen *screen) {
      (*screen).LIGHTS[0].dimmer -= 1;
      if ((*screen).LIGHTS[0].dimmer > 250)
            (*screen).LIGHTS[0].dimmer = 0;
      LCD_ChangeDim1Image((*screen).LIGHTS[0].dimmer);
}
void touch_PlusDimmer2(_screen *screen) {

      (*screen).LIGHTS[1].dimmer -= 1;
      if ((*screen).LIGHTS[1].dimmer > 250)
            (*screen).LIGHTS[1].dimmer = 0;
      LCD_ChangeDim2Image((*screen).LIGHTS[1].dimmer);
}
void touch_PlusDimmer3(_screen *screen) {

      (*screen).LIGHTS[2].dimmer -= 1;
      if ((*screen).LIGHTS[2].dimmer > 250)
            (*screen).LIGHTS[2].dimmer = 0;
      LCD_ChangeDim3Image((*screen).LIGHTS[2].dimmer);
}
void touch_PlusDimmer4(_screen *screen) {

      (*screen).LIGHTS[3].dimmer -= 1;
      if ((*screen).LIGHTS[3].dimmer > 250)
            (*screen).LIGHTS[3].dimmer = 0;
      LCD_ChangeDim4Image((*screen).LIGHTS[3].dimmer);
}

void touch_ProgramLights(_screen *screen) {
      (*screen).menu = SEL_PROGRAM;
      LCD_SetSelProgramImage();
}

// **************       MENU SEL_PROGRAM     ******************

void touch_SetON(_screen *screen) {
      (*screen).alarmSel = SET_ON;
      LCD_SetSelTimeONImage(); //Imprime el boton SetTimeOn encendido
      LCD_SetAlarmViewImage(); // Imagen de la vista de alarma sin los
botones de seleccion de alarma
      touch_SetTimeON(screen);
      (*screen).menu = PROGRAM;
}
void touch_SetOFF(_screen *screen) {
      (*screen).alarmSel = SET_OFF;
      LCD_SetSelTimeOFFImage(); //Imprime el boton SetTimeOff
encendido
      LCD_SetAlarmViewImage(); // Imagen de la vista de alarma sin los
botones de seleccion de alarma
      touch_SetTimeOFF(screen);
      (*screen).menu = PROGRAM;
}

// **************       MENU PROGRAM     ******************
void touch_SetTimeON(_screen *screen) { // incluir estado de la alrma
******** **** *** ***

      (*screen).alarmSel = SET_ON;
```

```
	LCD_SetTimeONImage(); //Imprime el boton SetOn encendido
	LCD_ResetTimeOFFImage(); //Imprime el boton SetOff apagado
	AUX_alarmLights = (*screen).alarms[ALRM_ON].ALRM_lightsSet;
	AUX_Hour = (*screen).alarms[ALRM_ON].alarmTime.RTC_Hours;
	AUX_Minute = (*screen).alarms[ALRM_ON].alarmTime.RTC_Minutes;
	LCD_PrintAlarmLights(screen, &AUX_alarmLights);
	LCD_ShowAlarmState(screen);
	LCD_PrintAlarmTime((*screen).alarms[ALRM_ON].alarmTime.RTC_Hours
,
			(*screen).alarms[ALRM_ON].alarmTime.RTC_Minutes);

}

void touch_SetTimeOFF(_screen *screen) {

	(*screen).alarmSel = SET_OFF;
	LCD_SetTimeOFFImage();
	LCD_ResetTimeONImage();
	AUX_alarmLights = (*screen).alarms[ALRM_OFF].ALRM_lightsSet;
	AUX_Hour = (*screen).alarms[ALRM_OFF].alarmTime.RTC_Hours;
	AUX_Minute = (*screen).alarms[ALRM_OFF].alarmTime.RTC_Minutes;
	LCD_PrintAlarmLights(screen, &AUX_alarmLights);
	LCD_ShowAlarmState(screen);
	LCD_PrintAlarmTime((*screen).alarms[ALRM_OFF].alarmTime.RTC_Hour
s,
			(*screen).alarms[ALRM_OFF].alarmTime.RTC_Minutes);

}

void touch_HourUP(_screen *screen) {
	if ((*screen).alarmSel == SET_ON) {
		AUX_Hour++;
		if (AUX_Hour > 23) {
			AUX_Hour = 0;
		}
		LCD_PrintAlarmTime(AUX_Hour, AUX_Minute);
	}
	if ((*screen).alarmSel == SET_OFF) {
		AUX_Hour++;
		if (AUX_Hour > 23) {
			AUX_Hour = 0;
		}
		LCD_PrintAlarmTime(AUX_Hour, AUX_Minute);
	}
}
void touch_HourDOWN(_screen *screen) {

	if ((*screen).alarmSel == SET_ON) {
		AUX_Hour--;
		if (AUX_Hour > 24) {
			AUX_Hour = 23;
		}
		LCD_PrintAlarmTime(AUX_Hour, AUX_Minute);
	}
	if ((*screen).alarmSel == SET_OFF) {
		AUX_Hour--;
		if (AUX_Hour > 24) {
			AUX_Hour = 23;
		}
		LCD_PrintAlarmTime(AUX_Hour, AUX_Minute);
	}
```

```
}
void touch_MinuteUP(_screen *screen) {
	if ((*screen).alarmSel == SET_ON) {
		AUX_Minute++;
		if (AUX_Minute > 59) {
			AUX_Minute = 0;
		}
		LCD_PrintAlarmTime(AUX_Hour, AUX_Minute);
	}
	if ((*screen).alarmSel == SET_OFF) {
		AUX_Minute++;
		if (AUX_Minute > 59) {
			AUX_Minute = 0;
		}
		LCD_PrintAlarmTime(AUX_Hour, AUX_Minute);
	}
}
void touch_MinuteDOWN(_screen *screen) {
	if ((*screen).alarmSel == SET_ON) {
		AUX_Minute--;
		if (AUX_Minute > 59) {
			AUX_Minute = 59;
		}
		LCD_PrintAlarmTime(AUX_Hour, AUX_Minute);
	}
	if ((*screen).alarmSel == SET_OFF) {
		AUX_Minute--;
		if (AUX_Minute > 59) {
			AUX_Minute = 59;
		}
		LCD_PrintAlarmTime(AUX_Hour, AUX_Minute);
	}
}

void touch_SetProgram(_screen *screen) {

	if (AUX_alarmLights != No_LightsSel) {
		if ((*screen).alarmSel == SET_ON) {
			(*screen).alarms[ALRM_ON].status = ACTIVE;
			(*screen).alarms[ALRM_ON].ALRM_lightsSet =
AUX_alarmLights;
			(*screen).alarms[ALRM_ON].alarmTime.RTC_Hours =
AUX_Hour;
			(*screen).alarms[ALRM_ON].alarmTime.RTC_Minutes =
AUX_Minute;
			Disable_Alarm(screen);
			LCD_ShowAlarmState(screen);
			Set_Alarm(screen);

		} else if ((*screen).alarmSel == SET_OFF) {
			(*screen).alarms[ALRM_OFF].status = ACTIVE;
			(*screen).alarms[ALRM_OFF].ALRM_lightsSet =
AUX_alarmLights;
			(*screen).alarms[ALRM_OFF].alarmTime.RTC_Hours =
AUX_Hour;
			(*screen).alarms[ALRM_OFF].alarmTime.RTC_Minutes =
AUX_Minute;
			Disable_Alarm(screen);
			LCD_ShowAlarmState(screen);
			Set_Alarm(screen);
		}
```

```
      } else {
            int i = 0;
            for (i = 0; i < 2; i++) {
                  Draw_circle_fill(
                              XmaxSelLights - (XmaxSelLights -
XminSelLights) / 2,
                              YmaxSelLight1 - (YmaxSelLight1 -
YminSelLight1) / 2,
                               (YmaxSelLight1 - YminSelLight1) / 2 - 1,
Blue2);
                  Chk_Busy();
                  Draw_circle_fill(
                              XmaxSelLights - (XmaxSelLights -
XminSelLights) / 2,
                              YmaxSelLight2 - (YmaxSelLight2 -
YminSelLight2) / 2,
                               (YmaxSelLight2 - YminSelLight2) / 2 - 1,
Blue2);
                  Chk_Busy();
                  Draw_circle_fill(
                              XmaxSelLights - (XmaxSelLights -
XminSelLights) / 2,
                              YmaxSelLight3 - (YmaxSelLight3 -
YminSelLight3) / 2,
                               (YmaxSelLight3 - YminSelLight3) / 2 - 1,
Blue2);
                  Chk_Busy();
                  Draw_circle_fill(
                              XmaxSelLights - (XmaxSelLights -
XminSelLights) / 2,
                              YmaxSelLight4 - (YmaxSelLight4 -
YminSelLight4) / 2,
                               (YmaxSelLight4 - YminSelLight4) / 2 - 1,
Blue2);
                  Chk_Busy();

                  Draw_circle_fill(
                              XmaxSelLights - (XmaxSelLights -
XminSelLights) / 2,
                              YmaxSelLight1 - (YmaxSelLight1 -
YminSelLight1) / 2,
                               (YmaxSelLight1 - YminSelLight1) / 2 - 1,
Black);
                  Chk_Busy();
                  Draw_circle_fill(
                              XmaxSelLights - (XmaxSelLights -
XminSelLights) / 2,
                              YmaxSelLight2 - (YmaxSelLight2 -
YminSelLight2) / 2,
                               (YmaxSelLight2 - YminSelLight2) / 2 - 1,
Black);
                  Chk_Busy();
                  Draw_circle_fill(
                              XmaxSelLights - (XmaxSelLights -
XminSelLights) / 2,
                              YmaxSelLight3 - (YmaxSelLight3 -
YminSelLight3) / 2,
                               (YmaxSelLight3 - YminSelLight3) / 2 - 1,
Black);
                  Chk_Busy();
                  Draw_circle_fill(
```

```
                                    XmaxSelLights - (XmaxSelLights -
XminSelLights) / 2,
                                    YmaxSelLight4 - (YmaxSelLight4 -
YminSelLight4) / 2,
                                    (YmaxSelLight4 - YminSelLight4) / 2 - 1,
Black);
                  Chk_Busy();
            }
      }
}

void touch_RestoreProgram(_screen *screen) {

      if ((*screen).alarmSel == SET_ON) {
            (*screen).alarms[ALRM_ON].status = INACTIVE;
            (*screen).alarms[ALRM_ON].alarmTime.RTC_Hours = 0;
            (*screen).alarms[ALRM_ON].alarmTime.RTC_Minutes = 0;
            AUX_Hour = 0;
            AUX_Minute = 0;
            (*screen).alarms[ALRM_ON].ALRM_lightsSet = No_LightsSel;
            AUX_alarmLights = No_LightsSel;

      LCD_PrintAlarmTime((*screen).alarms[ALRM_ON].alarmTime.RTC_Hours
,

      (*screen).alarms[ALRM_ON].alarmTime.RTC_Minutes);
            LCD_PrintAlarmLights(screen, &AUX_alarmLights);
            LCD_ShowAlarmState(screen);
            Disable_Alarm(screen);
      } else if ((*screen).alarmSel == SET_OFF) {
            (*screen).alarms[ALRM_OFF].status = INACTIVE;
            (*screen).alarms[ALRM_OFF].alarmTime.RTC_Hours = 0;
            (*screen).alarms[ALRM_OFF].alarmTime.RTC_Minutes = 0;
            //toogleRESETBUTTON();
            (*screen).alarms[ALRM_OFF].ALRM_lightsSet = No_LightsSel;
            AUX_alarmLights = No_LightsSel;

      LCD_PrintAlarmTime((*screen).alarms[ALRM_OFF].alarmTime.RTC_Hour
s,

      (*screen).alarms[ALRM_OFF].alarmTime.RTC_Minutes);
            LCD_PrintAlarmLights(screen, &AUX_alarmLights);
            LCD_ShowAlarmState(screen);
            Disable_Alarm(screen);
      }
}

void touch_ALRM_Light1(_screen *screen) {
      if ((*screen).alarmSel == SET_ON) {

            if ((AUX_alarmLights & Light_1) != 0) {
                  AUX_alarmLights &= Lights_2_3_4;
            } else {
                  AUX_alarmLights |= Light_1;
            }
            LCD_PrintAlarmLights(screen, &AUX_alarmLights);
      } else if ((*screen).alarmSel == SET_OFF) {
            if ((AUX_alarmLights & Light_1) != 0) {
                  AUX_alarmLights &= Lights_2_3_4;
            } else {
                  AUX_alarmLights |= Light_1;
```

```
            }
            LCD_PrintAlarmLights(screen, &AUX_alarmLights);
      }
}
void touch_ALRM_Light2(_screen *screen) {
      if ((*screen).alarmSel == SET_ON) {
            if ((AUX_alarmLights & Light_2) != 0) {
                  AUX_alarmLights &= Lights_1_3_4;
            } else {
                  AUX_alarmLights |= Light_2;
            }
            LCD_PrintAlarmLights(screen, &AUX_alarmLights);
      } else if ((*screen).alarmSel == SET_OFF) {
            if ((AUX_alarmLights & Light_2) != 0) {
                  AUX_alarmLights &= Lights_1_3_4;
            } else {
                  AUX_alarmLights |= Light_2;
            }
            LCD_PrintAlarmLights(screen, &AUX_alarmLights);
      }
}
void touch_ALRM_Light3(_screen *screen) {
      if ((*screen).alarmSel == SET_ON) {
            if ((AUX_alarmLights & Light_3) != 0) {
                  AUX_alarmLights &= Lights_1_2_4;
            } else {
                  AUX_alarmLights |= Light_3;
            }
            LCD_PrintAlarmLights(screen, &AUX_alarmLights);
      } else if ((*screen).alarmSel == SET_OFF) {
            if ((AUX_alarmLights & Light_3) != 0) {
                  AUX_alarmLights &= Lights_1_2_4;
            } else {
                  AUX_alarmLights |= Light_3;
            }
            LCD_PrintAlarmLights(screen, &AUX_alarmLights);
      }
}
void touch_ALRM_Light4(_screen *screen) {
      if ((*screen).alarmSel == SET_ON) {
            if ((AUX_alarmLights & Light_4) != 0) {
                  AUX_alarmLights &= Lights_1_2_3;
            } else {
                  AUX_alarmLights |= Light_4;
            }
            LCD_PrintAlarmLights(screen, &AUX_alarmLights);
      } else if ((*screen).alarmSel == SET_OFF) {
            if ((AUX_alarmLights & Light_4) != 0) {
                  AUX_alarmLights &= Lights_1_2_3;
            } else {
                  AUX_alarmLights |= Light_4;
            }
            LCD_PrintAlarmLights(screen, &AUX_alarmLights);
      }
}

// ******  Appears in various MENUs:
void touch_HOMEButton(_screen *screen) {

      (*screen).menu = BUTTONS;
      (*screen).alarmSel = NO_AlarmSel;
```

```
      LCD_SetButtonsImage(screen); //imprime el menu principal de
luces

}

/*********************************************************************
**************
 *                                                ActionButton
                                                */
/*********************************************************************
**************
 * 1. Verifies if received coords aim a button in a concrete screen
 * 2. Execute the proper function.
 *
*********************************************************************
***********/
void ActionButton(_screen *screen) {

      uint8_t nLight;

      if (touchReleased == 1) {
            //Touch done in CONF_TIME screen
            if ((*screen).menu == CONF_TIME) {
                  //If touch is done in any plus button
                  if (((*screen).touch_event.coords.y > YminDateUp)
                              && ((*screen).touch_event.coords.y <
YmaxDateUp)) {
                        //If touch is done in Day up button
                        if (((*screen).touch_event.coords.x >
XminDayUp)
                                    && ((*screen).touch_event.coords.x
< XmaxDayUp)) {
                              touch_DayUp(screen, 0);
                              touchReleased = 0;
                        }
                        //If touch is done in month up button
                        if (((*screen).touch_event.coords.x >
XminMonthUp)
                                    && ((*screen).touch_event.coords.x
< XmaxMonthUp)) {
                              touch_MonthUp(screen);
                              touchReleased = 0;
                        }
                        //If touch is done in year up button
                        if (((*screen).touch_event.coords.x >
XminYearUp)
                                    && ((*screen).touch_event.coords.x
< XmaxYearUp)) {
                              touch_YearUp(screen);
                              touchReleased = 0;
                        }
                        //If touch is done in Hour up button
                        if (((*screen).touch_event.coords.x >
XminTimeHUp)
                                    && ((*screen).touch_event.coords.x
< XmaxTimeHUp)) {
                              touch_TimeHUp(screen);
                              touchReleased = 0;
                        }

                        //If touch is done in Minute up button
```

```
                        if (((*screen).touch_event.coords.x >
XminTimeMUp)
                                    && ((*screen).touch_event.coords.x
< XmaxTimeMUp)) {
                              touch_TimeMUp(screen);
                              touchReleased = 0;
                        }
                        //If touch is done in any minus button
                  } else if (((*screen).touch_event.coords.y >
YminDateDown)
                              && ((*screen).touch_event.coords.y <
YmaxDateDown)) {
                        //If touch is done in Day down button
                        if (((*screen).touch_event.coords.x >
XminDayDown)
                                    && ((*screen).touch_event.coords.x
< XmaxDayDown)) {
                              touch_DayDown(screen, 0);
                              touchReleased = 0;
                        }
                        //If touch is done in month down button
                        if (((*screen).touch_event.coords.x >
XminMonthDown)
                                    && ((*screen).touch_event.coords.x
< XmaxMonthDown)) {
                              touch_MonthDown(screen);
                              touchReleased = 0;
                        }
                        //If touch is done in year down button
                        if (((*screen).touch_event.coords.x >
XminYearDown)
                                    && ((*screen).touch_event.coords.x
< XmaxYearDown)) {
                              touch_YearDown(screen);
                              touchReleased = 0;
                        }
                        //If touch is done in Hour down button
                        if (((*screen).touch_event.coords.x >
XminTimeHDown)
                                    && ((*screen).touch_event.coords.x
< XmaxTimeHDown)) {
                              touch_TimeHDown(screen);
                              touchReleased = 0;
                        }
                        //If touch is done in Minute down button
                        if (((*screen).touch_event.coords.x >
XminTimeMDown)
                                    && ((*screen).touch_event.coords.x
< XmaxTimeMDown)) {
                              touch_TimeMDown(screen);
                              touchReleased = 0;
                        }

                        //If touch is done Save button
                  } else if (((*screen).touch_event.coords.y >
YminSave)
                              && ((*screen).touch_event.coords.y <
YmaxSave)) {
                        if (((*screen).touch_event.coords.x > XminSave)
                                    && ((*screen).touch_event.coords.x
< XmaxSave)) {
```

```
                        touch_Save(screen);
                        touchReleased = 0;
                    }
                }
            }
            //Touch done in BUTTONS screen
            if ((*screen).menu == BUTTONS) {

                //If touch is done in any Light button
                if (((*screen).touch_event.coords.y > YminLights)
                        && ((*screen).touch_event.coords.y <
YmaxLights)) {
                    //If touch is done in Light1 button
                    if (((*screen).touch_event.coords.x >
XminLight1)
                            && ((*screen).touch_event.coords.x
< XmaxLight1)) {
                        nLight = 1;
                        touch_Light_Button(screen, &nLight, 0);
                        touchReleased = 0;
                    }
                    //If touch is done in Light2 button
                    if (((*screen).touch_event.coords.x >
XminLight2)
                            && ((*screen).touch_event.coords.x
< XmaxLight2)) {
                        nLight = 2;
                        touch_Light_Button(screen, &nLight, 0);
                        touchReleased = 0;
                    }
                    //If touch is done in Light3 button
                    if (((*screen).touch_event.coords.x >
XminLight3)
                            && ((*screen).touch_event.coords.x
< XmaxLight3)) {
                        nLight = 3;
                        touch_Light_Button(screen, &nLight, 0);
                        touchReleased = 0;
                    }
                    //If touch is done in Light4 button
                    if (((*screen).touch_event.coords.x >
XminLight4)
                            && ((*screen).touch_event.coords.x
< XmaxLight4)) {
                        nLight = 4;
                        touch_Light_Button(screen, &nLight, 0);
                        touchReleased = 0;
                    }
                }

                //If touch is done in settings button
                if (((*screen).touch_event.coords.y > YminSettings)
                        && ((*screen).touch_event.coords.y <
YmaxSettings)) {
                    if (((*screen).touch_event.coords.x >
XminSettings)
                            && ((*screen).touch_event.coords.x
< XmaxSettings)) {
                        touch_Settings_Button(screen);
                        touchReleased = 0;
                    }
```

```
                }
            } //END screen BUTTONS

            //Touch done in DIMMERS screen
            else if ((*screen).menu == DIMMERS) {
                //If touch is done in any - button
                if (((*screen).touch_event.coords.x > XminDimmers -
100)
                        && ((*screen).touch_event.coords.x <
XminDimmers)) {
                    //If touch is done in dimmer1
                    if (((*screen).touch_event.coords.y > YminDim1)
                            && ((*screen).touch_event.coords.y
< YmaxDim1)) {
                        touch_MinusDimmer1(screen);
                        touchReleased = 0;
                    }
                    //If touch is done in dimmer2
                    if (((*screen).touch_event.coords.y > YminDim2)
                            && ((*screen).touch_event.coords.y
< YmaxDim2)) {
                        touch_MinusDimmer2(screen);
                        touchReleased = 0;
                    }
                    //If touch is done in dimmer3
                    if (((*screen).touch_event.coords.y > YminDim3)
                            && ((*screen).touch_event.coords.y
< YmaxDim3)) {
                        touch_MinusDimmer3(screen);
                        touchReleased = 0;
                    }
                    //If touch is done in dimmer4
                    if (((*screen).touch_event.coords.y > YminDim4)
                            && ((*screen).touch_event.coords.y
< YmaxDim4)) {
                        touch_MinusDimmer4(screen);
                        touchReleased = 0;
                    }

                }
                //If touch is done in any + button
                if (((*screen).touch_event.coords.x > XmaxDimmers)
                        && ((*screen).touch_event.coords.x <
XmaxDimmers + 100)) {
                    //If touch is done in dimmer1
                    if (((*screen).touch_event.coords.y > YminDim1)
                            && ((*screen).touch_event.coords.y
< YmaxDim1)) {
                        touch_PlusDimmer1(screen);
                        touchReleased = 0;
                    }
                    //If touch is done in dimmer2
                    if (((*screen).touch_event.coords.y > YminDim2)
                            && ((*screen).touch_event.coords.y
< YmaxDim2)) {
                        touch_PlusDimmer2(screen);
                        touchReleased = 0;
                    }
                    //If touch is done in dimmer3
                    if (((*screen).touch_event.coords.y > YminDim3)
```

```
                                        && ((*screen).touch_event.coords.y
< YmaxDim3)) {
                                  touch_PlusDimmer3(screen);
                                  touchReleased = 0;
                            }
                            //If touch is done in dimmer4
                            if (((*screen).touch_event.coords.y > YminDim4)
                                        && ((*screen).touch_event.coords.y
< YmaxDim4)) {
                                  touch_PlusDimmer4(screen);
                                  touchReleased = 0;
                            }
                      }
                      //If touch is done in ProgLights button
                      if (((*screen).touch_event.coords.x > XminProgLights)
                                  && ((*screen).touch_event.coords.x <
XmaxProgLights)) {
                            if (((*screen).touch_event.coords.y >
YminProgLights)
                                        && ((*screen).touch_event.coords.y
< YmaxProgLights)) {
                                  touch_ProgramLights(screen);
                                  touchReleased = 0;
                            }
                      }
                      //If touch is done in HOME button
                      if (((*screen).touch_event.coords.x > XminHome)
                                  && ((*screen).touch_event.coords.x <
XmaxHome)) {
                            if (((*screen).touch_event.coords.y > YminHome)
                                        && ((*screen).touch_event.coords.y
< YmaxHome)) {
                                  touch_HOMEButton(screen);
                                  touchReleased = 0;
                            }
                      }
                } //END screen DIMMERS

                //Touch done in SEL_PROGRAM screen
                else if ((*screen).menu == SEL_PROGRAM) {

                      //If touch is done in any SetTime button
                      if (((*screen).touch_event.coords.y > YminSetTime)
                                  && ((*screen).touch_event.coords.y <
YmaxSetTime)) {
                            //If touch is done in SetTimeON button
                            if (((*screen).touch_event.coords.x >
XminSetTimeON)
                                        && ((*screen).touch_event.coords.x
< XmaxSetTimeON)) {
                                  touch_SetON(screen);
                                  touchReleased = 0;
                            }
                            //If touch is done in SetTimeOFF button
                            if (((*screen).touch_event.coords.x >
XminSetTimeOFF)
                                        && ((*screen).touch_event.coords.x
< XmaxSetTimeOFF)) {
                                  touch_SetOFF(screen);
                                  touchReleased = 0;
                            }
```

```
                }
                //If touch is done in HOME button
                if (((*screen).touch_event.coords.x > XminHome)
                            && ((*screen).touch_event.coords.x <
XmaxHome)) {
                      if (((*screen).touch_event.coords.y > YminHome)
                                  && ((*screen).touch_event.coords.y
< YmaxHome)) {
                            touch_HOMEButton(screen);
                            touchReleased = 0;
                      }
                }
          } //END screen SEL_PROGRAM

          //Touch done in PROGRAM screen
          else if ((*screen).menu == PROGRAM) {

                //If touch is done in any selection light button
                if (((*screen).touch_event.coords.x > XminSelLights)
                            && ((*screen).touch_event.coords.x <
XmaxSelLights)) {

                      //If touch is done in selection light1 button
                      if (((*screen).touch_event.coords.y >
YminSelLight1)
                                  && ((*screen).touch_event.coords.y
< YmaxSelLight1)) {
                            touch_ALRM_Light1(screen);
                            touchReleased = 0;
                      }
                      //If touch is done in selection light2 button
                      if (((*screen).touch_event.coords.y >
YminSelLight2)
                                  && ((*screen).touch_event.coords.y
< YmaxSelLight2)) {
                            touch_ALRM_Light2(screen);
                            touchReleased = 0;
                      }
                      //If touch is done in selection light3 button
                      if (((*screen).touch_event.coords.y >
YminSelLight3)
                                  && ((*screen).touch_event.coords.y
< YmaxSelLight3)) {
                            touch_ALRM_Light3(screen);
                            touchReleased = 0;
                      }
                      //If touch is done in selection light4 button
                      if (((*screen).touch_event.coords.y >
YminSelLight4)
                                  && ((*screen).touch_event.coords.y
< YmaxSelLight4)) {
                            touch_ALRM_Light4(screen);
                            touchReleased = 0;
                      }
                }
                //if touch is done in any time button
                if (((*screen).touch_event.coords.y >
YminTimeButtons)
                            && ((*screen).touch_event.coords.y <
YmaxTimeButtons)) {
```

```
                        //If touch is done in set hour UP button
                        if (((*screen).touch_event.coords.x >
XminHourUP)
                                    && ((*screen).touch_event.coords.x
< XmaxHourUP)) {
                              touch_HourUP(screen);
                              touchReleased = 0;
                        }
                        //If touch is done in set hour DOWN button
                        if (((*screen).touch_event.coords.x >
XminHourDOWN)
                                    && ((*screen).touch_event.coords.x
< XmaxHourDOWN)) {
                              touch_HourDOWN(screen);
                              touchReleased = 0;
                        }
                        //If touch is done in set hour UP button
                        if (((*screen).touch_event.coords.x >
XminMinuteUP)
                                    && ((*screen).touch_event.coords.x
< XmaxMinuteUP)) {
                              touch_MinuteUP(screen);
                              touchReleased = 0;
                        }
                        //If touch is done in set hour DOWN button
                        if (((*screen).touch_event.coords.x >
XminMinuteDOWN)
                                    && ((*screen).touch_event.coords.x
< XmaxMinuteDOWN)) {
                              touch_MinuteDOWN(screen);
                              touchReleased = 0;
                        }
                  }

                  //If touch is done in any SET/RESETORE button
                  if (((*screen).touch_event.coords.x > XminSETREST)
                              && ((*screen).touch_event.coords.x <
XmaxSETREST)) {

                        //If touch is done in SET button
                        if (((*screen).touch_event.coords.y > YminSET)
                                    && ((*screen).touch_event.coords.y
< YmaxSET)) {
                              touch_SetProgram(screen);
                              touchReleased = 0;
                        }
                        //If touch is done in RESTORE button
                        if (((*screen).touch_event.coords.y > YminREST)
                                    && ((*screen).touch_event.coords.y
< YmaxREST)) {
                              touch_RestoreProgram(screen);
                              touchReleased = 0;
                        }
                  }

                  //If touch is done in any SetON/setOFF selection
button
                  if (((*screen).touch_event.coords.y > YminAlarmType)
                              && ((*screen).touch_event.coords.y <
YmaxAlarmType)) {
```

```
                        //If touch is done in SetON button
                        if (((*screen).touch_event.coords.x >
XminAlarmON)
                                    && ((*screen).touch_event.coords.x
< XmaxAlarmON)) {
                              touch_SetTimeON(screen);
                              touchReleased = 0;

                        }
                        //If touch is done in SetOFF button
                        if (((*screen).touch_event.coords.x >
XminAlarmOFF)
                                    && ((*screen).touch_event.coords.x
< XmaxAlarmOFF)) {
                              touch_SetTimeOFF(screen);
                              touchReleased = 0;
                        }

                  }
                  //If touch is done in HOME button
                  if (((*screen).touch_event.coords.x > XminHome)
                              && ((*screen).touch_event.coords.x <
XmaxHome)) {
                        if (((*screen).touch_event.coords.y > YminHome)
                                    && ((*screen).touch_event.coords.y
< YmaxHome)) {
                              touch_HOMEButton(screen);
                              touchReleased = 0;
                        }
                  }
            } //END screen PROGRAM
      } else {
            if ((*screen).menu == PROGRAM) {
                  //if touch is done in any time button
                  if (((*screen).touch_event.coords.y >
YminTimeButtons)
                              && ((*screen).touch_event.coords.y <
YmaxTimeButtons)) {

                        //If touch is done in set hour UP button
                        if (((*screen).touch_event.coords.x >
XminHourUP)
                                    && ((*screen).touch_event.coords.x
< XmaxHourUP)) {
                              touch_HourUP(screen);
                              Delay_ms(100);
                        }
                        //If touch is done in set hour DOWN button
                        if (((*screen).touch_event.coords.x >
XminHourDOWN)
                                    && ((*screen).touch_event.coords.x
< XmaxHourDOWN)) {
                              touch_HourDOWN(screen);
                              Delay_ms(100);
                        }
                        //If touch is done in set hour UP button
                        if (((*screen).touch_event.coords.x >
XminMinuteUP)
                                    && ((*screen).touch_event.coords.x
< XmaxMinuteUP)) {
                              touch_MinuteUP(screen);
```

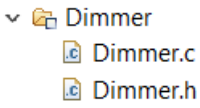
```
                              Delay_ms(100);
                        }
                        //If touch is done in set hour DOWN button
                        if (((*screen).touch_event.coords.x >
XminMinuteDOWN)
                                    && ((*screen).touch_event.coords.x
< XmaxMinuteDOWN)) {
                              touch_MinuteDOWN(screen);
                              Delay_ms(100);
                        }
                  }
            } else if ((*screen).menu == CONF_TIME) {
                  //If touch is done in any plus button
                  if (((*screen).touch_event.coords.y > YminDateUp)
                              && ((*screen).touch_event.coords.y <
YmaxDateUp)) {
                        //If touch is done in Day up button
                        if (((*screen).touch_event.coords.x >
XminDayUp)
                                    && ((*screen).touch_event.coords.x
< XmaxDayUp)) {
                              touch_DayUp(screen, 0);
                              Delay_ms(100);
                        }
                        //If touch is done in month up button
                        if (((*screen).touch_event.coords.x >
XminMonthUp)
                                    && ((*screen).touch_event.coords.x
< XmaxMonthUp)) {
                              touch_MonthUp(screen);
                              Delay_ms(100);
                        }
                        //If touch is done in year up button
                        if (((*screen).touch_event.coords.x >
XminYearUp)
                                    && ((*screen).touch_event.coords.x
< XmaxYearUp)) {
                              touch_YearUp(screen);
                              Delay_ms(100);
                        }
                        //If touch is done in Hour up button
                        if (((*screen).touch_event.coords.x >
XminTimeHUp)
                                    && ((*screen).touch_event.coords.x
< XmaxTimeHUp)) {
                              touch_TimeHUp(screen);
                              Delay_ms(100);
                        }

                        //If touch is done in Minute up button
                        if (((*screen).touch_event.coords.x >
XminTimeMUp)
                                    && ((*screen).touch_event.coords.x
< XmaxTimeMUp)) {
                              touch_TimeMUp(screen);
                              Delay_ms(100);
                        }
                  }
            }
      }
```

```
}
```

8.7.1.3 Touch.h

```
#include "stm32f4xx_conf.h"
#include "Includes/defines.h"

/*void Read_Coords(_screen *screen);*/
void Read_Coords_MultiFingers();

// **************       MENU BOTONES      ******************
void touch_Light_Button(_screen *screen, uint8_t *numLight);
void touch_Settings_Button();
// **************       MENU DIMMERS      ******************
void touch_Dimmer1();
void touch_Dimmer2();
void touch_Dimmer3();
void touch_Dimmer4();
void touch_ProgramLights();
// **************       MENU SEL_PROGRAM      ******************
void touch_SetON();
void touch_SetOFF();
// **************       MENU PROGRAM      ******************
void touch_SetTimeON();
void touch_SetTimeOFF();
void touch_HourUP(_screen *screen);
void touch_HourDOWN(_screen *screen);
void touch_MinuteUP(_screen *screen);
void touch_MinuteDOWN(_screen *screen);
void touch_SetProgram();
void touch_RestoreProgram();
void touch_ALRM_Light1();
void touch_ALRM_Light2();
void touch_ALRM_Light3();
void touch_ALRM_Light4();
void touch_HOMEButton();

void ActionButton(_screen *screen);
```

8.7.2 Dimmer



8.7.2.1 Dimmer.c

```
#include "Dimmer.h"

uint32_t valTim2 = 0;
extern _screen screen;
uint8_t ZCD_ENABLED = 0;
extern uint32_t cont;
extern uint32_t cont2;
extern uint8_t dimmer1Flag;
extern uint8_t dimmer2Flag;
extern uint8_t dimmer3Flag;
extern uint8_t dimmer4Flag;
char text[50];

/* Configure pin PD2 to be interrupt for zero crossing detection*/
void Init_Dimmer(void) {
	GPIO_LIGHTS_Init();
	Configure_ZCInterrupt();
	Config_Interrupt();
	TIM_Config();
}

void Configure_ZCInterrupt() {
	GPIO_InitTypeDef GPIO_InitStruct;
	/* Enable clock for GPIO */
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
	GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8;
	GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
	GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(GPIOD, &GPIO_InitStruct);

	EXTI_InitTypeDef EXTI_InitStruct;
	// Enable clock for SYSCFG
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource8);
	EXTI_InitStruct.EXTI_Line = EXTI_Line8;
	EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
	EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
	EXTI_InitStruct.EXTI_LineCmd = ENABLE;
	EXTI_Init(&EXTI_InitStruct);

	NVIC_InitTypeDef NVIC_InitStruct;
	// Add IRQ vector to NVIC
	NVIC_InitStruct.NVIC_IRQChannel = EXTI9_5_IRQn;
	NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x00;
	NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;
	NVIC_InitStruct.NVIC_IRQChannelCmd = DISABLE;
	NVIC_Init(&NVIC_InitStruct);
}
```

```
void Config_Interrupt() {
	GPIO_InitTypeDef GPIO_InitStruct;
	/* Enable clock for GPIO */
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
	GPIO_InitStruct.GPIO_Pin = GPIO_Pin_12;
	GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
	GPIO_InitStruct.GPIO_OType = GPIO_OType_OD;
	GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
	GPIO_Init(GPIOA, &GPIO_InitStruct);

	EXTI_InitTypeDef EXTI_InitStruct;
	// Enable clock for SYSCFG
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource12);
	EXTI_InitStruct.EXTI_Line = EXTI_Line12;
	EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
	EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
	EXTI_InitStruct.EXTI_LineCmd = ENABLE;
	EXTI_Init(&EXTI_InitStruct);

	// Enable clock for SYSCFG
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource11);
	EXTI_InitStruct.EXTI_Line = EXTI_Line11;
	EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
	EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
	EXTI_InitStruct.EXTI_LineCmd = ENABLE;
	EXTI_Init(&EXTI_InitStruct);

	// Enable clock for SYSCFG
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource15);
	EXTI_InitStruct.EXTI_Line = EXTI_Line15;
	EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
	EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
	EXTI_InitStruct.EXTI_LineCmd = ENABLE;
	EXTI_Init(&EXTI_InitStruct);

	// Enable clock for SYSCFG
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOD, EXTI_PinSource13);
	EXTI_InitStruct.EXTI_Line = EXTI_Line13;
	EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
	EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
	EXTI_InitStruct.EXTI_LineCmd = ENABLE;
	EXTI_Init(&EXTI_InitStruct);

	NVIC_InitTypeDef NVIC_InitStruct;
	// Add IRQ vector to NVIC
	NVIC_InitStruct.NVIC_IRQChannel = EXTI15_10_IRQn;
	NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x00;
	NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x01;
	NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStruct);
}

void Enable_ZCInterrupt() {

	if (ZCD_ENABLED == 0) {
```

```
            NVIC_InitTypeDef NVIC_InitStruct;
            NVIC_InitStruct.NVIC_IRQChannel = EXTI9_5_IRQn;
            NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x00;
            NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;
            NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
            NVIC_Init(&NVIC_InitStruct);
            ZCD_ENABLED = 1;
      }
}

void Disable_ZCInterrupt() {
      if (ZCD_ENABLED == 1) {
            NVIC_InitTypeDef NVIC_InitStruct;
            NVIC_InitStruct.NVIC_IRQChannel = EXTI9_5_IRQn;
            NVIC_InitStruct.NVIC_IRQChannelCmd = DISABLE;
            NVIC_Init(&NVIC_InitStruct);
            ZCD_ENABLED = 0;
      }
}

void TIM_Config(void) {

      TIM_TimeBaseInitTypeDef timerInitStructure2;
      // TIM2 Configuration
      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
      timerInitStructure2.TIM_Prescaler = 0x0053;
//timer_tick_frequency = 84000000 / 83+1 = 1 MHz --> 1 us
      timerInitStructure2.TIM_CounterMode = TIM_CounterMode_Up;
      timerInitStructure2.TIM_Period = 0x2134; // 10 < Dimer < 9000
      TIM_TimeBaseInit(TIM2, &timerInitStructure2);
      TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
      TIM_Cmd(TIM2, DISABLE);

      NVIC_InitTypeDef NVIC_InitStruct;
      NVIC_InitStruct.NVIC_IRQChannel = TIM2_IRQn;
      NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x02;
      NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x02;
      NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
      NVIC_Init(&NVIC_InitStruct);

}

void EXTI15_10_IRQHandler() {

      uint8_t num = 0;

      if (EXTI_GetITStatus(EXTI_Line12) != RESET) {
            if (screen.LIGHTS[0].status == ACTIVE) {
                  if (dimmer1Flag == 1) {
                        num = 1;
                        Set_Light(&screen, &num);
                  }
            }

            EXTI_ClearITPendingBit(EXTI_Line12);
      }

      if (EXTI_GetITStatus(EXTI_Line11) != RESET) {
            if (screen.LIGHTS[1].status == ACTIVE) {
                  if (dimmer2Flag == 1) {
                        num = 2;
```

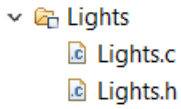
```
                        Set_Light(&screen, &num);
                  }
            }
            EXTI_ClearITPendingBit(EXTI_Line11);
      }

      if (EXTI_GetITStatus(EXTI_Line13) != RESET) {
            if (screen.LIGHTS[2].status == ACTIVE) {
                  if (dimmer3Flag == 1) {
                        num = 3;
                        Set_Light(&screen, &num);
                  }
            }
            EXTI_ClearITPendingBit(EXTI_Line13);
      }

      if (EXTI_GetITStatus(EXTI_Line15) != RESET) {
            if (screen.LIGHTS[3].status == ACTIVE) {
                  if (dimmer4Flag == 1) {
                        num = 4;
                        Set_Light(&screen, &num);
                  }
            }

            EXTI_ClearITPendingBit(EXTI_Line15);
      }

}

void EXTI9_5_IRQHandler() {
      // Make sure that interrupt flag is set
      if (EXTI_GetITStatus(EXTI_Line8) != RESET) {

            //if (screen.LIGHTS[0].status == ACTIVE) {
            //TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
            TIM_SetCounter(TIM2, 0);
            TIM_Cmd(TIM2, ENABLE);
            TIM_SetCounter(TIM2, 0);
            dimmer1Flag = 1;
            dimmer2Flag = 1;
            dimmer3Flag = 1;
            dimmer4Flag = 1;
            //}

            EXTI_ClearITPendingBit(EXTI_Line8);
      }

}

//Timer 2 handler
void TIM2_IRQHandler() {

      uint8_t nLight = 0;
      //Timer 2 update event
      if (TIM_GetITStatus(TIM2, TIM_IT_Update) == SET) {

            if (screen.LIGHTS[0].status == ACTIVE) {
                  nLight = 1;
                  Reset_Light(&screen, &nLight);
            }
            if (screen.LIGHTS[1].status == ACTIVE) {
```

```
                nLight = 2;
                Reset_Light(&screen, &nLight);
            }
            if (screen.LIGHTS[2].status == ACTIVE) {
                nLight = 3;
                Reset_Light(&screen, &nLight);
            }
            if (screen.LIGHTS[3].status == ACTIVE) {
                nLight = 4;
                Reset_Light(&screen, &nLight);
            }

            //DISABLE TIMER2
            TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
            TIM_SetCounter(TIM2, 0);
            TIM_Cmd(TIM2, DISABLE);

        }

}
```

8.7.2.2 Dimmer.h

```
#include "stm32f4xx_conf.h"
#include "Includes/defines.h"

void Init_Dimmer(void);
void Config_Interrupt(void);
void Configure_ZCInterrupt(void);
void TIM_Config(void);
void TIM3_IRQHandler();
void Disable_ZCInterrupt();
void Enable_ZCInterrupt();
void EXTI9_5_IRQHandler();
void EXTI15_10_IRQHandler();
```

8.7.3 Lights



8.7.3.1 Lights.c

```
#include "Lights/Lights.h"

/**********************************************************************
**********
 * Function Name  : initializeGPIO_LIGHTS
 * Description    : Initialize GPIO LIGHTS PINS
 * Input          : None
 * Output         : PE2 -> LIGHT1
 *                          PE0 -> lIGHT2
 *                          PA10 -> LIGHT3
 *                          PB9 -> LIGHT4
 * Return         : None
 * Attention        : None

**********************************************************************
*********/
void GPIO_LIGHTS_Init(void) {

	GPIO_InitTypeDef GPIO_InitStructure;
	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIOE, &GPIO_InitStructure);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIOE, &GPIO_InitStructure);

	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIOA, &GPIO_InitStructure);

	RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
	GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
	GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
	GPIO_Init(GPIOB, &GPIO_InitStructure);
```

```
}

/*********************************************************************
**********
 * Function Name  : Init_Lights
 * Description    : Initialize all LIGHT structures


*********************************************************************
*********/
void Init_Lights(_screen *screen) {

      // Initial values of each Light Structure
      (*screen).LIGHTS[0].name = "Light 1";
      (*screen).LIGHTS[0].dimmer = 0;
      (*screen).LIGHTS[0].status = INACTIVE;
      (*screen).LIGHTS[0].trigger = INACTIVE;

      (*screen).LIGHTS[1].name = "Light 2";
      (*screen).LIGHTS[1].dimmer = 0;
      (*screen).LIGHTS[1].status = INACTIVE;
      (*screen).LIGHTS[1].trigger = INACTIVE;

      (*screen).LIGHTS[2].name = "Light 3";
      (*screen).LIGHTS[2].dimmer = 0;
      (*screen).LIGHTS[2].status = INACTIVE;
      (*screen).LIGHTS[2].trigger = INACTIVE;

      (*screen).LIGHTS[3].name = "Light 4";
      (*screen).LIGHTS[3].dimmer = 0;
      (*screen).LIGHTS[3].status = INACTIVE;
      (*screen).LIGHTS[3].trigger = INACTIVE;

}

/*********************************************************************
**********
 * Function Name  : CheckLights_ZCD
 * Description    : Enables ZCDInterrupt if needed

*********************************************************************
*********/
void CheckLights_ZCD(_screen *screen) {
      if ((*screen).numLightsON == 1) {
            Enable_ZCInterrupt();
      } else if ((*screen).numLightsON == 0) {
            Disable_ZCInterrupt();
      }
}

/*********************************************************************
**********
 * Function Name  : Toggle_Light
 * Description    : Changes the specified light status

*********************************************************************
*********/
void Toogle_Light(_screen *screen, uint8_t *numLight) {

      if ((*screen).LIGHTS[(*numLight) - 1].status == ACTIVE) {
            (*screen).LIGHTS[(*numLight) - 1].status = INACTIVE;
```

```
			(*screen).numLightsON--;
			if ((*numLight) == 1) {
				(*screen).lightsSet &= 0b00111111;
			} else if ((*numLight) == 2) {
				(*screen).lightsSet &= 0b11001111;
			} else if ((*numLight) == 3) {
				(*screen).lightsSet &= 0b11110011;
			} else if ((*numLight) == 4) {
				(*screen).lightsSet &= 0b11111100;
			}
			Reset_Light(screen, numLight);
		} else {
			(*screen).LIGHTS[(*numLight) - 1].status = ACTIVE;
			(*screen).numLightsON++;
			if ((*numLight) == 1) {
				(*screen).lightsSet |= 0b11000000;
			} else if ((*numLight) == 2) {
				(*screen).lightsSet |= 0b00110000;
			} else if ((*numLight) == 3) {
				(*screen).lightsSet |= 0b00001100;
			} else if ((*numLight) == 4) {
				(*screen).lightsSet |= 0b00000011;
			}
		}
}

/*****************************************************************
**********
 * Function Name  : SetLight
 * Description    : Set trigger lightx

*****************************************************************
*********/
void Set_Light(_screen *screen, uint8_t *nLight) {
		switch (*nLight) {
		case 1:
			GPIO_SetBits(GPIOE, GPIO_Pin_2);
			(*screen).LIGHTS[0].trigger = ACTIVE;
			break;
		case 2:
			GPIO_SetBits(GPIOE, GPIO_Pin_0);
			(*screen).LIGHTS[1].trigger = ACTIVE;
			break;
		case 3:
			GPIO_SetBits(GPIOA, GPIO_Pin_10);
			(*screen).LIGHTS[2].trigger = ACTIVE;
			break;
		case 4:
			GPIO_SetBits(GPIOB, GPIO_Pin_9);
			(*screen).LIGHTS[3].trigger = ACTIVE;
			break;
		default:
			break;
		}
}
/*****************************************************************
**********
 * Function Name  : ResetLight
 * Description    : Reset trigger lightx
```

```
*********************************************************************
*********/
void Reset_Light(_screen *screen, uint8_t *nLight) {
	switch (*nLight) {
	case 1:
		GPIO_ResetBits(GPIOE, GPIO_Pin_2);
		(*screen).LIGHTS[0].trigger = INACTIVE;
		break;
	case 2:
		GPIO_ResetBits(GPIOE, GPIO_Pin_0);
		(*screen).LIGHTS[1].trigger = INACTIVE;
		break;
	case 3:
		GPIO_ResetBits(GPIOA, GPIO_Pin_10);
		(*screen).LIGHTS[2].trigger = INACTIVE;
		break;
	case 4:
		GPIO_ResetBits(GPIOB, GPIO_Pin_9);
		(*screen).LIGHTS[3].trigger = INACTIVE;
		break;
	default:
		break;
	}
}
```

8.7.3.2 Lights.h

```
#include "stm32f4xx.h"
#include "Includes/defines.h"


void GPIO_LIGHTS_Init(void);
void Init_Lights(_screen *screen);
void CheckLights_ZCD(_screen *screen);
void Toogle_Light(_screen *screen, uint8_t *numLight);
void Set_Light(_screen *screen, uint8_t *nLight);
void Reset_Light(_screen *screen, uint8_t *nLight);
```