

CONTROL DE SEMÁFOROS DE TRENES A ESCALA CON ARDUINO

v 1.0 - Darío Calvo Sánchez, 2021

Más información en: <https://elgajoelegante.com>

Programa de control de señales LED basado en Arduino y placas PCA9685 de señal PWM, con las siguientes características:

- Control de hasta 12 señales con un máximo de 48 focos. En la práctica puede manejar casi cualquier tipo de señal:
 - En señales de ánodo común, el número máximo de focos por señal es de 4, de los cuales 3 podrán estar encendidos simultáneamente.
 - En señales de cátodo común, el número máximo de focos por señal es de 5, los cuales podrán encenderse todos simultáneamente en caso necesario.
- Compatible con sistemas analógicos y digitales.
- Se puede conectar una mezcla de señales de ánodo común y cátodo común, indistintamente.
- Se pueden conectar señales con diferente número de focos o configuraciones a la misma o a diferentes placas PCA9685. Incluso una señal puede tener algunos focos conectados a una de las placas y el resto a otra placa.
- Todos los focos de una señal de ánodo común brillarán con la misma intensidad, no importando cuántos de ellos estén encendidos simultáneamente.
- Atenuación (o encendido progresivo) ajustable e independiente del número de señales o focos conectados, permitiendo una representación realista de cualquier señal luminosa: no habrá una atenuación más lenta cuantas más señales conectemos.
- Velocidad de parpadeo ajustable.
- Todos los focos de cualquier aspecto de una misma señal se encenderán o apagarán simultáneamente, sin retardo entre uno y otro.
- Sin límite práctico en el número de placas PCA9685 que se pueden conectar (hasta su máximo de 62) a un solo Arduino.
- Preparado para las señales más típicas de RENFE/ADIF, pero configurable para cualquier sistema de señalización.
- Se incluyen archivos SVG compatibles con Rocrail.

CONTENIDO

1	Introducción.....	3
2	Requisitos.....	3
3	Instalación.....	4
3.1	Esquema general	4
3.1.1	Sistema digital	4
3.1.2	Sistema analógico	5
3.1.2.1	Modo manual	6
3.1.2.2	Modo automático.....	7
3.2	Conexión del Arduino y las placas PCA9685.....	7
3.3	Conexión de las señales.....	9
3.3.1	Señales de cátodo común	9
3.3.2	Señales de ánodo común	10
3.4	Alimentación eléctrica.....	11
3.5	Configuración del programa.....	11
3.5.1	Parámetros comunes a las dos versiones	11
3.5.2	Parámetros propios de la versión digital	14
3.5.3	Parámetros propios de la versión analógica	14
4	Funcionamiento e instrucciones de uso	15
4.1	Versión digital.....	16
4.2	Versión analógica	20
5	Solución de problemas más comunes.....	21
	Apéndice 1: Decodificador DCC con Arduino.....	22
	Apéndice 2: Circuito adaptador a ánodo común	25
	Apéndice 3: Cambio del sistema de señalización	27

1 INTRODUCCIÓN

Este sistema es un controlador de señales LED para modelismo ferroviario que únicamente se encarga de que cada señal luzca con el aspecto comandado en cada momento. Requiere por lo tanto un modo de indicar dicho comando (ver la sección 3.1).

En Internet hay cientos de algoritmos para controlar señales con Arduino, pero la mayoría sólo consideran los focos rojo y verde, o no permiten mezclar con facilidad señales de distintos tipos. Este sistema ofrece una mayor flexibilidad, permitiendo modelar técnicamente cualquier aspecto de una señal luminosa ferroviaria, y hacerlo tanto en instalaciones analógicas como digitales. También es totalmente configurable en términos de atenuación y parpadeo.

La principal diferencia de este sistema radica en el uso de las placas PCA9685, que están pensadas principalmente para controlar servos. Sin embargo, al ser controladores PWM también pueden manejar LEDs. Es cierto que eso se puede hacer también con los pines normales del Arduino, pero usando estas placas se consiguen fácilmente algunas de las ventajas mencionadas (como la posibilidad de mezclar señales de ánodo común y cátodo común de forma sencilla) a la vez que permiten conectar un mayor número de focos y señales sin tener que recurrir a un Arduino Mega (y en cualquier caso pudiendo ahorrar un buen número de pines en la placa de Arduino para otros propósitos, como una botonera para manejarlo). Además, se independiza totalmente la alimentación de los LEDs de la del propio Arduino.

Finalmente, las placas PCA9685 son baratas. Así que no suponen un sobre coste inasumible, particularmente cuando se combinan con el también barato Arduino Nano (o si se comparan con el coste de un Arduino Mega que permitiera conectar un número similar de semáforos).

2 REQUISITOS

Esta guía asume unos conocimientos básicos de Arduino y su programación. También será necesaria una mínima experiencia en electrónica (conocimiento de componentes y circuitos elementales, así como de montajes).

Requisitos necesarios:

- Placa Arduino (una Nano será suficiente en la mayoría de los casos)
- Placa(s) PCA9685, tantas como se requiera (normalmente un máximo de 3)
- Alimentador de 5 V de corriente continua (un viejo cargador USB de móvil puede valer)
- Circuito(s) de conversión de cátodo común a ánodo común (si las señales a instalar son de ánodo común). Ver Apéndice 2.
- Método de entrada: decodificador, pulsadores, automatismo, etc. (ver 3.1)

Requisitos recomendados:

- En el caso de usar control digital, es muy recomendable que tanto la central como (en su caso) el programa de control por ordenador o “tablet”, soporten el protocolo de paquetes **“Signal Aspect”** para manejar señales multiaspecto (ver Capítulo 4). Ello simplificará notablemente la configuración del sistema, y es el único soportado en este documento.

El programa requiere además las librerías públicas **Adafruit_PWMServoDriver.h** para controlar las placas PCA9685 y **Wire.h** para permitir la comunicación por el bus I2C, ambas disponibles como estándar en la versión actual del Arduino IDE oficial.

3 INSTALACIÓN

3.1 ESQUEMA GENERAL

El programa es un controlador de semáforos, que se encarga de mostrar el aspecto correcto de entre los definidos posibles, con los parpadeos y atenuaciones adecuadas. Será necesario por lo tanto un método de entrada de datos que le dé al programa las órdenes pertinentes para cada señal.

El esquema básico de funcionamiento se muestra en la Figura 1:

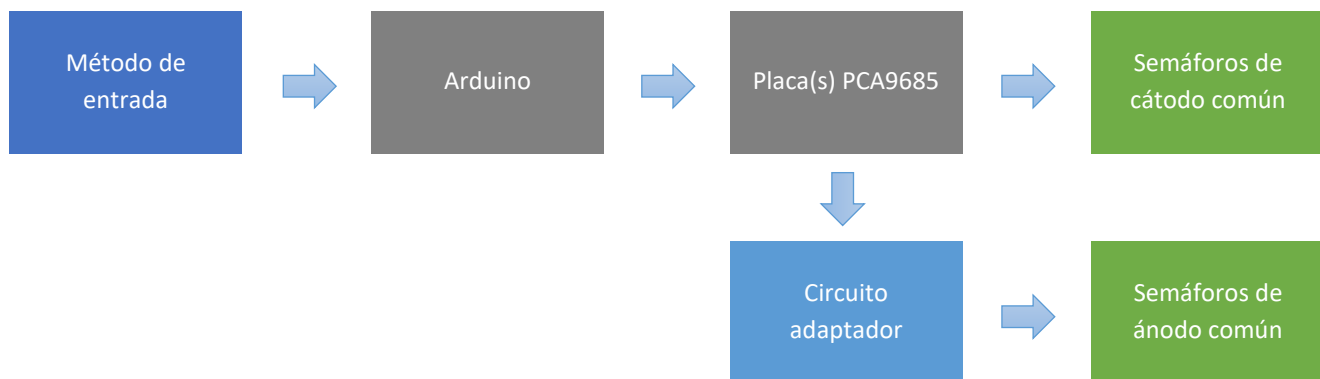


Figura 1: Esquema general

Para el método de entrada habrá varias opciones: en una maqueta digital se usará generalmente un decodificador que haga de interfaz entre la central y el Arduino, mientras que en una maqueta analógica normalmente se usarán pulsadores conectados directamente, o bien entradas de los automatismos instalados (ampollas reed, relés, etc.)

En cualquiera de los casos, el Arduino recibirá las órdenes del método de entrada, las procesará, asignará el aspecto correcto a cada semáforo en cada momento, y controlará las placas PCA9685 que alimentarán a los semáforos. Si estos son de ánodo común, será necesario interponer un circuito adaptador (ver sección 3.3.2).

3.1.1 SISTEMA DIGITAL

En un sistema digital, el método de entrada será un decodificador que transforme las órdenes de la central digital a un “array” o lista de comando con los aspectos de todos los semáforos (con el formato definido en el Capítulo 4). La mayoría de los decodificadores comerciales **no servirá** para este propósito: generalmente, este decodificador se hará de manera barata y sencilla con otra placa de Arduino, comunicando ambas placas mediante el puerto serie.

En la Figura 2 se muestra un esquema de conexiones. El Arduino Nano en la parte superior hace las veces de decodificador, y se comunica con el que está en la parte inferior, que sirve de placa de control, y a su vez se une a las placas PCA9685. Notar que los cables que comunican los pines RX0 y TX1 de cada Arduino se cruzan, de forma que se unen el TX1 (transmisor) con el RX0 (receptor) de la placa homóloga y viceversa.



Figura 2: Configuración general en una maqueta digital

Es técnicamente posible integrar la función de decodificador en la propia placa Arduino que controla los semáforos, evitando tener que usar otra placa para ello, pero **no es recomendable** por varias razones:

- 1) Al integrar la función de decodificador, la memoria de una placa de Arduino Nano queda al límite, haciéndolo inestable.
- 2) El típico código manejado por interrupciones necesario para decodificar una señal DCC aumenta la carga del procesador: con un número elevado de semáforos (más de 8) puede fallar esporádicamente al recibir las órdenes de la central, tardar en cambiar de aspecto, o afectar a las atenuaciones y parpadeos. El uso de un Arduino Mega (que dispone de más memoria pero similar capacidad de proceso) no soluciona estos problemas.
- 3) La instalación es más “limpia”, flexible y ordenada con el decodificador separado, y el incremento en precio de comprar otra placa de Arduino Nano no es determinante. Es mejor centralizar todos los decodificadores (no sólo para semáforos, sino también para desvíos u otros accesorios) en una placa dedicada, y enviar los comandos a las placas de actuación correspondientes.

El programa tal y como está preparado asume que la decodificación se realizará aparte. Ver en el Apéndice 1 las instrucciones para hacer un decodificador compatible.

3.1.2 SISTEMA ANALÓGICO

En un sistema analógico, el conexionado y el uso dependerán de la configuración de la maqueta y de su grado de automatismo. En este documento sólo se cubrirá el modo manual.

3.1.2.1 MODO MANUAL

En este caso usaremos los pines libres del Arduino para conectar unos pulsadores como método de entrada y manejar las señales (ver Figura 3). A priori, el aspecto de las señales no tendrá influencia sobre la circulación de los trenes, cuya interpretación quedará a cargo del “maquinista”. Es la forma más sencilla de conectar señales complejas de muchos aspectos.

Cada pin libre de entrada/salida en el Arduino podrá manejar un pulsador (la otra patilla de cada pulsador irá al conector GND común del Arduino). No sólo se podrán usar los 14 pines digitales (D0 a D13, numerados en el código del 0 al 13), sino que también los 8 analógicos (marcados en la placa del A0 al A7 y numerados en el código del 14 al 21) se pueden utilizar como digitales. Las excepciones que no se podrán usar son los pines del bus I2C, que en el Nano son el A4 (18) y el A5 (19) -consultar el manual en el caso de utilizar otro modelo de Arduino-.

La forma más sencilla de emplear los pulsadores, que es la utilizada en este programa, es dedicar algunos a seleccionar una serie de aspectos, y el resto a seleccionar los semáforos. De esta forma accionaríamos primero el pulsador de un semáforo, y seguidamente el del aspecto que le quisiéramos asignar (vía libre, parada, anuncio de precaución, etc.) Así optimizaríamos en limitado número de pines disponibles.

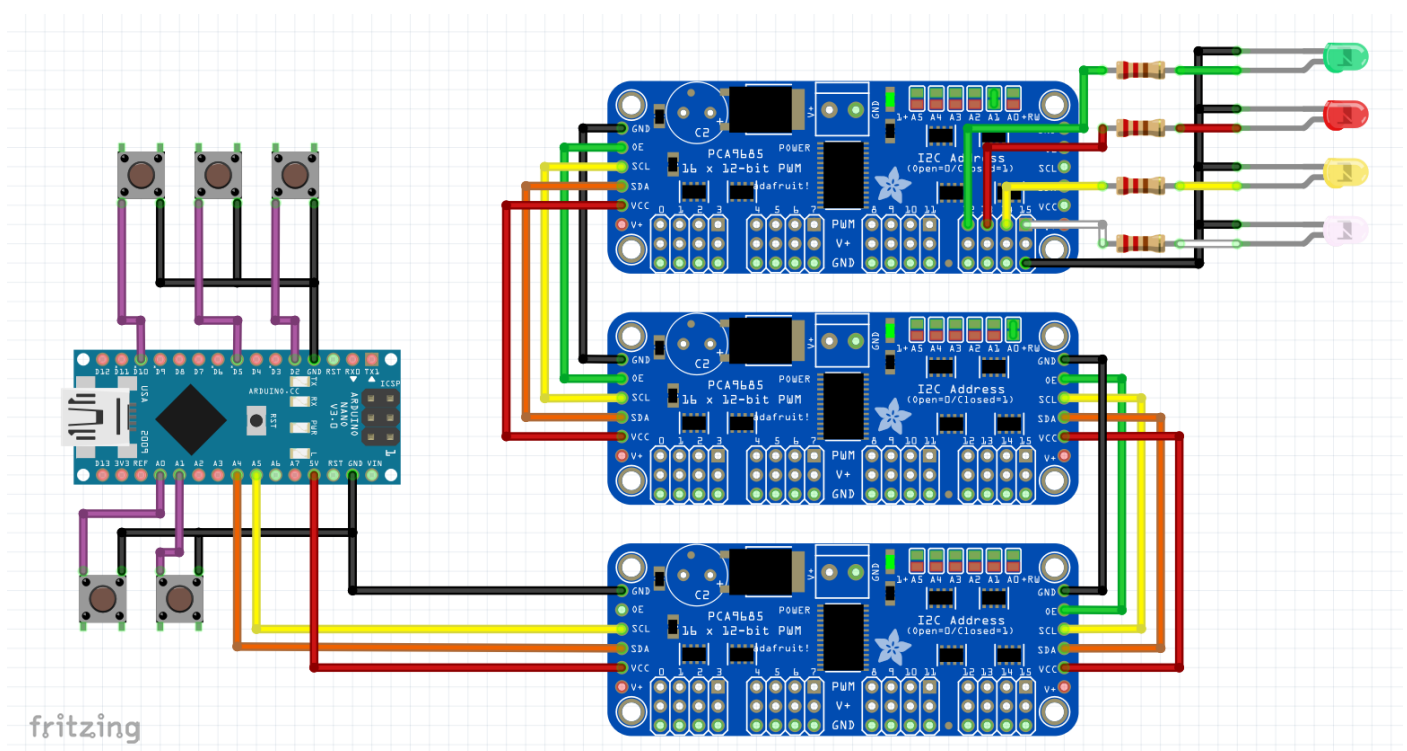


Figura 3: Configuración general en una maqueta analógica

La única restricción vendrá dada por el número de pulsadores libres, como es evidente. Como en principio sólo ocuparemos los dos cables del bus I2C para comunicarnos con las placas PCA9685, del total de 22 pines de entrada/salida en un Arduino Nano (14 digitales y 8 analógicos) nos quedarían 20 libres: ese será nuestro límite. Si, por ejemplo, necesitáramos un total de 10 aspectos posibles (vía libre, parada, anuncio de precaución, etc.) necesitaríamos 10 pines para los pulsadores de aspecto y nos quedarían otros 10 libres para controlar hasta un total de 10 semáforos. Si por el contrario necesitáramos controlar 12 semáforos, tendríamos un máximo de 8 aspectos seleccionables que habría que elegir entre todos los posibles.

Normalmente cualquier combinación posible (compatible con el máximo de 12 semáforos) cubrirá la gran mayoría de necesidades. Si necesitáramos una combinación que se saliese del límite de 20 pulsadores habría dos opciones:

- a) Usar un Arduino Mega como placa de control en vez del Nano, para ampliar el número de pines disponibles. Se llegaría así a un máximo de 68 pines libres después de descontar los dos requeridos para el bus I2C. Sobraría incluso teniendo en cuenta todos los aspectos posibles de cualquier señal de un operador ferroviario como RENFE o ADIF (que no suelen superar los 20). Lógicamente un Arduino Mega es más caro que un Nano, y seguiría pudiendo controlar sólo 12 semáforos, por lo que hay que hacer la cuenta del número de aspectos distintos que se van a necesitar y del número de semáforos deseado para tomar la decisión más económica: puede que salga mejor usar varios Arduino Nano separando en grupos de señales con diferentes aspectos que un solo Mega controlándolo todo.
- b) Conectar un [teclado numérico o "keypad"](#) que permita seleccionar una señal mediante un número que la identifique y luego un aspecto de manera simple, economizando el número de pines utilizados. Es probablemente la mejor opción, y la que más simplifica el cableado (más teniendo en cuenta lo que suele complicarse este aspecto en una maqueta analógica). En ese caso es recomendable acompañarlo de un pequeño display LCD para poder ver lo que estamos tecleando.

Para la opción a) bastará con asignar en el programa los pines conectados, como se verá después en el apartado 3.5. Si se elige la opción b) sería necesario modificar el código del programa, lo cual es sencillo con unos mínimos conocimientos básicos de Arduino (este documento no cubre esta opción).

3.1.2.2 MODO AUTOMÁTICO

En modo automático, el comando del sistema dependerá del accionamiento a través de ampollas reed o cualquier otro detector de presencia de los trenes, así como del sistema de control de la circulación de los trenes (acantonamientos y enclavamientos) a través de relés u otros automatismos.

Por ejemplo, para el accionamiento de una señal al paso por puntos concretos se pueden conectar ampollas reed a los pines de Arduino de manera similar a la expuesta en la Figura 3 con pulsadores, y programarlo en consecuencia. La limitación de pines disponibles sería la misma a la descrita en el apartado anterior, dependiente del tipo de Arduino utilizado.

Debido a la variedad de las posibles instalaciones, la programación de los controles de accionamiento, cantones y enclavamientos, así como de la posible regulación de los trenes dependiendo del aspecto de las señales, se deja al usuario. Si son complejos es posible que sea necesario programarlos en una placa adicional de Arduino, y mandar el comando a través del puerto serie (de manera similar a como funciona en sistema digital, ver 4.1)

3.2 CONEXIÓN DEL ARDUINO Y LAS PLACAS PCA9685

Las placas PCA9685 se conectan al Arduino a través del bus I2C, usando los pines marcados como "SDA" (datos) y "SCL" (reloj) en ambos dispositivos, como se puede ver en la Figura 4. En el Arduino Nano, el pin "SDA" es el "A4" y el pin "SCL" es el "A5".

NOTA: Consultar la documentación si se usa otro modelo de Arduino distinto al Nano, ya que la asignación de estos pines puede variar.

La longitud máxima de las conexiones con bus I2C es de unos 20 o 30 cm, es decir, las placas PCA9685 deberán situarse cerca del Arduino y cerca entre sí, llevando después los cables a los semáforos con la longitud requerida.

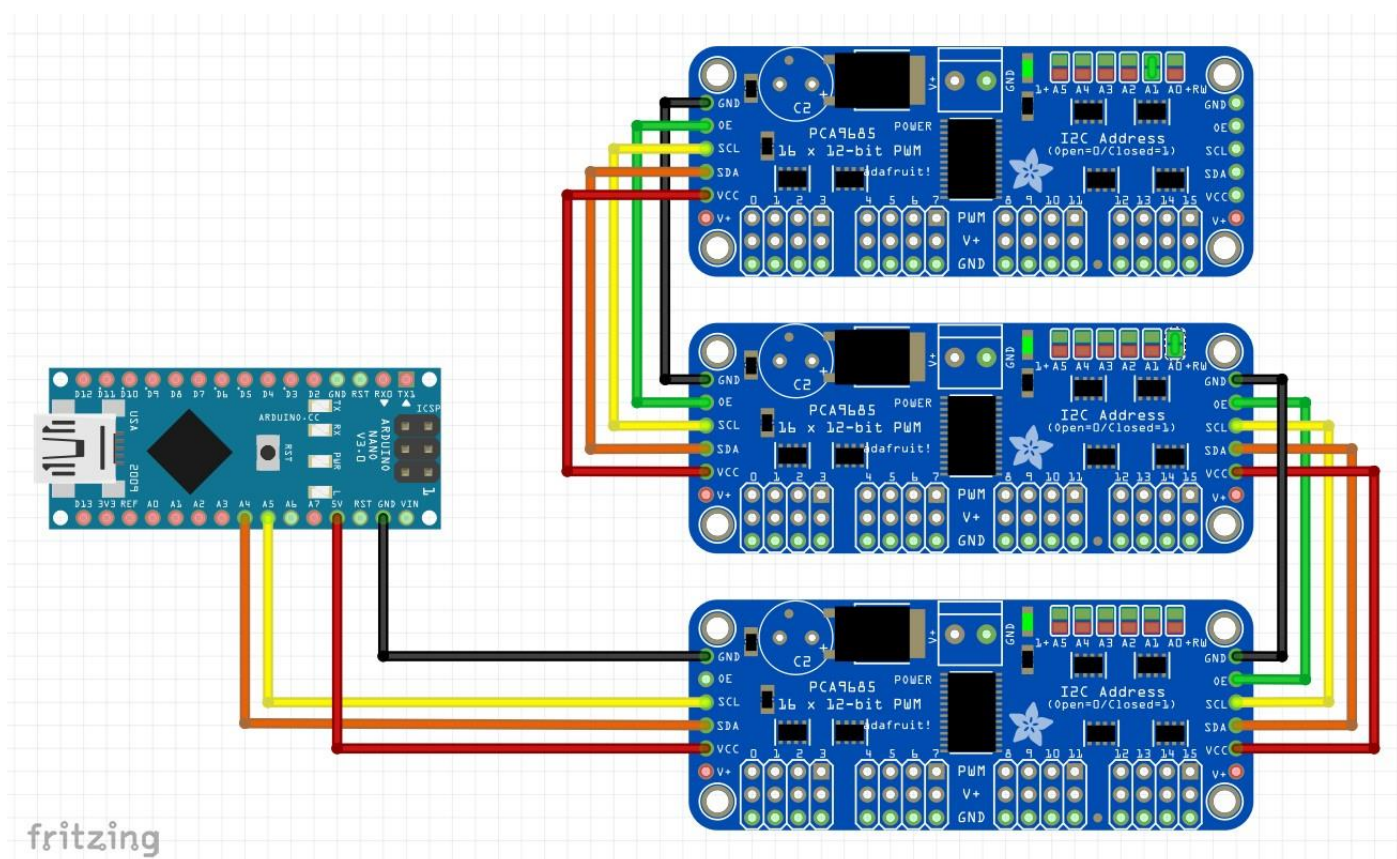


Figura 4: Unión de las placas PCA9685 al Arduino

En caso de necesitar más de una placa PCA9685, se pueden encadenar como se aprecia también en la Figura 4, utilizando las conexiones de sus extremos y uniéndolas con latiguillos. Conectaremos entre ellas también los pines "OE", aunque no formen parte de la conexión a Arduino. La única precaución es que **hay que asignar una dirección distinta a cada una de las placas**, y ello hay que hacerlo soldando unos pequeños "pads" que actúan como interruptores soldables, y que definirán la dirección con la que se llamará a esa placa concreta. Por defecto vienen de fábrica con la dirección "0x40", así que si sólo necesitáramos una placa no tendríamos que hacer nada. Si necesitásemos más, casi con toda seguridad no serían más de tres, ya que alcanzaríamos los límites de luces y señales del sistema. En ese caso podemos asignar la dirección "0x41" a la segunda placa soldando en ella el pad marcado como "A0", y la dirección "0x42" a la tercera, soldando en esta última únicamente el pad marcado como "A1". La Figura 5 muestra una placa con dirección "0x42" a modo de ejemplo (ver el círculo rojo sobre el pad "A1"):

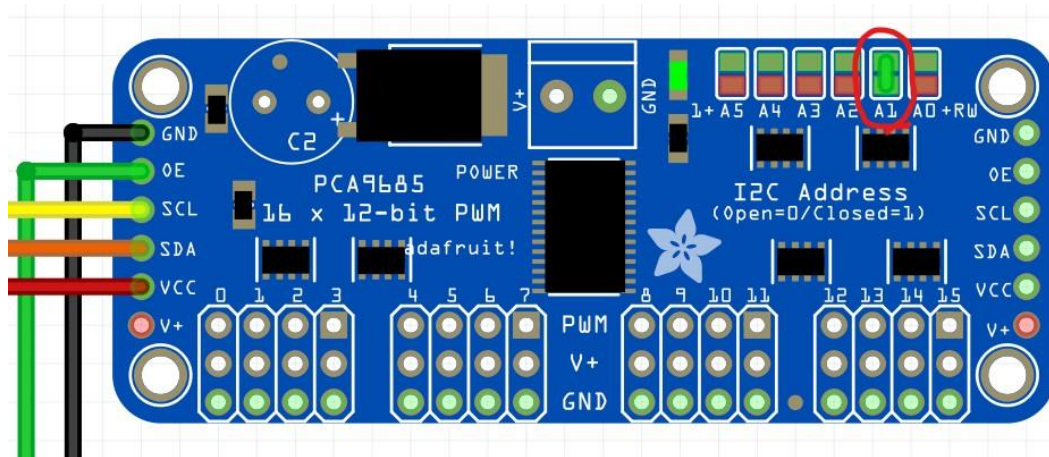


Figura 5: Soldadura de pin A1 para asignar la dirección "0x42"

Para más información, el proceso de conexión se explica [aquí](#) (en inglés, pero tanto el esquema de conexión como el proceso de soldadura de pads para asignar direcciones son fáciles de interpretar visualmente).

3.3 CONEXIÓN DE LAS SEÑALES

Ahora se conectará cada semáforo a las salidas de las placas PCA9685.

Cada placa PCA9685 tiene 16 salidas, numeradas de 0 a 15. Cada foco de cada señal se conectará a los pines "PWM" y "GND" de una de las salidas de la placa. No es necesario que todos los focos de una señal estén conectados a la misma placa PCA9685: se hará un "mapa" de distribución de cada foco y cada señal al realizar los ajustes del programa (ver 3.5.1).

Aquí llega la distinción entre señales LED de cátodo común y ánodo común:

- Las señales LED de cátodo común tienen el cable negativo común y un cable positivo para cada foco.
- Las señales LED de ánodo común tienen el cable positivo común y un cable negativo para cada foco.

Se pueden conectar señales de los dos tipos indistintamente a una misma placa, pero teniendo en cuenta sus particularidades, comentadas a continuación.

3.3.1 SEÑALES DE CÁTODO COMÚN

La placa PCA9685, como muchos otros microcontroladores (incluyendo Arduino), da una salida en configuración de cátodo común, ya que las patillas "GND" son comunes a todas las salidas (aunque por comodidad estén repetidas). Si tenemos un semáforo de ese tipo la conexión es directa y sencilla, como se puede ver en la Figura 6: los cátodos de los LEDs se unirán a las conexiones "GND" de la placa (o a una sola de ellas), y los ánodos a una resistencia de valor adecuado a su color, y a su vez al pin "PWM" correspondiente, según hayamos distribuido los colores. Esta será la configuración más típica en semáforos de construcción casera.

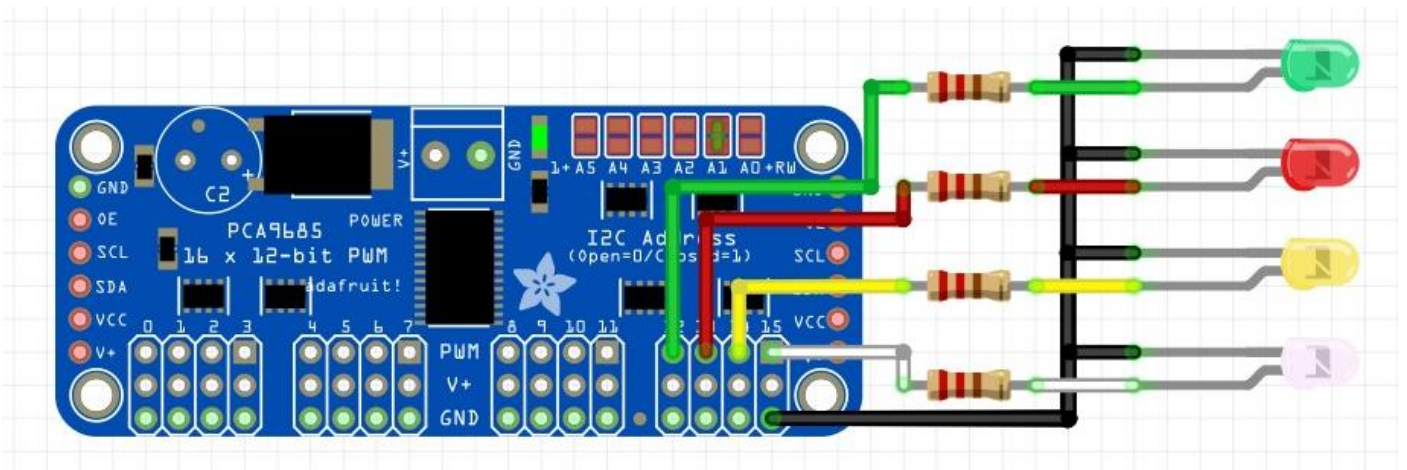


Figura 6: Ejemplo de señal de 4 focos con conexión de cátodo común

Si bien es técnicamente posible conectar directamente los LEDs a estas placas sin intercalar una resistencia (ya que emiten corriente PWM limitada a 10 mA que evitaría fundir los tipos de LED más comunes) es siempre una buena práctica ponerlas, y además nos permitirá ajustar el brillo en el caso de que resulte excesivo. Hay que elegir el valor adecuado para cada color y tipo de LED, consultando en su documentación su tensión de trabajo. Para hacer el cálculo hay que tener en cuenta que la tensión de salida de la placa PCA9685 será de 5 V, con el mencionado límite a 10 mA.

3.3.2 SEÑALES DE ÁNODO COMÚN

La mayoría de los fabricantes de señales para trenes a escala usa la configuración de ánodo común. Si el semáforo es este tipo, necesitaremos **intercalar un circuito adaptador** entre el semáforo y la placa PCA9685 (ver Figura 7). Se necesitará uno de estos circuitos por cada semáforo, y deberá tener tantas ramas como focos tenga la señal. Ver el Apéndice 2 para un esquema detallado del circuito y el material necesario.

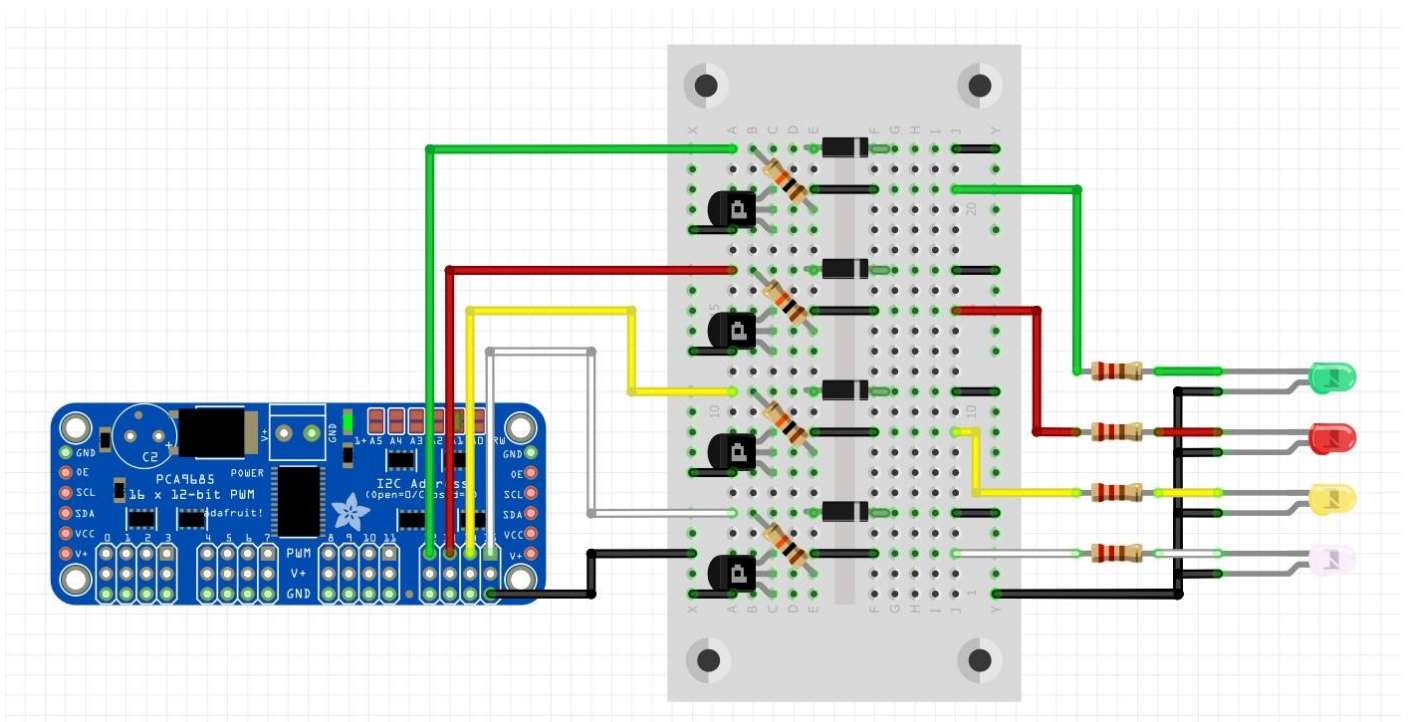


Figura 7: Ejemplo de señal de 4 focos con conexión de ánodo común y circuito adaptador

Estos circuitos adaptadores *invierten el funcionamiento de la señal*. Esto significa que en un semáforo de cátodo común podemos conectar el semáforo directamente a la PCA9685 y encender el LED verde simplemente dando corriente a su pin, dejando el resto de focos sin corriente. Sin embargo, en uno de ánodo común con uno de estos adaptadores conectado a un microcontrolador como el nuestro, *habría que dar corriente a todos los pines de ese semáforo excepto al verde para encender este último*. Este proceso se realiza automáticamente en el código, facilitado por el uso de las placas PCA9685 y sus librerías, y no requiere intervención del usuario. Además, las posibles diferencias en brillo que se pudieran producir dependiendo del número de focos encendidos (ya que damos corriente a uno o varios canales para encender el resto o unos pocos) se compensan automáticamente mediante una rutina en el programa: lo que habrá que tener en cuenta es que esta rutina **dividirá el brillo de cada foco por 3** (comparado con una señal de cátodo común) por lo que habrá que ajustar las resistencias de los LEDs en consecuencia (a valores menores que los que se usarían en configuración de cátodo común).

Con respecto a las resistencias de los LEDs, los semáforos comerciales suelen traerlas ya montadas, pero estas suelen ser adecuadas para tensiones de alimentación típicas de 12 o 16 V, por lo que si las mantenemos tal y como están, el brillo obtenido será muy bajo. Será recomendable reemplazarlas por unas adecuadas a la nueva corriente de salida (5 V con un máximo de 10 mA) y teniendo en cuenta que el semáforo brillará con un tercio de la intensidad lumínica esperada (debido al código mencionado). Una buena idea es partir de los valores de las resistencias originales del semáforo para cada color (ya que el fabricante habrá puesto las adecuadas), y hacer el cálculo de las nuevas en consecuencia, ya que cada tipo de LED de cada semáforo y fabricante puede tener requisitos distintos.

3.4 ALIMENTACIÓN ELÉCTRICA

Se alimentará eléctricamente el Arduino de la forma preferida (USB, pin Vin o cualquier otro método). **IMPORTANTE:** Tener en cuenta las restricciones aplicables de tensión de entrada y polaridad para cada método.

Las placas PCA9685 necesitarán alimentación eléctrica de 5 V en corriente continua, a través de sus pines extremos "Vcc" (positivo) y "GND" (negativo). No utilizaremos las conexiones "V+", ya que están pensadas para su uso con servos, y no tienen influencia alguna en la alimentación de los LEDs.

Es recomendable que la alimentación para estas placas se obtenga de una fuente externa y estable de 5 V (que resultará imprescindible si tenemos un número importante de luces y señales). Por ejemplo, un cargador USB de algún teléfono móvil viejo. Esta fuente puede ser la misma que alimenta al Arduino (si es que lo alimentamos por el pin de 5 voltios).

NOTA IMPORTANTE: Si la fuente externa para alimentar las placas es distinta de la que alimenta el Arduino, los polos negativos (o "ground") de las placas y del Arduino tienen que estar conectados entre sí.

3.5 CONFIGURACIÓN DEL PROGRAMA

Antes de subir el programa al Arduino es necesario modificar en el código los parámetros de funcionamiento para adaptarlos al sistema concreto a controlar. Se suministran dos códigos, uno para funcionamiento digital, en la carpeta "digital_version" y otro para funcionamiento analógico (en modo manual) en la carpeta "analog_version". Los ajustes necesarios serán los siguientes:

3.5.1 PARÁMETROS COMUNES A LAS DOS VERSIONES

Número de placas y semáforos

- Número de placas PCA9685 conectadas al Arduino (valor entero entre 1 y 62, normalmente no serán más de 3)

```
const uint8_t Numero_placas = 1;
```

- Número de semáforos conectados a través de las placas PCA9685 (valor entero entre 1 y 12):

```
const uint8_t Numero_semaforos = 2;
```

Parámetros lumínicos comunes

- Frecuencia (valor entero, en hertzios) de la corriente de salida a los LEDs. Se puede ajustar entre 40 y 10000 Hz. Un valor razonable es 200 Hz; reduciéndolo si se escuchara un ruido agudo al encenderse, y aumentándolo si se apreciara algún efecto de parpadeo. No afecta a la intensidad del brillo.

```
const uint16_t frec = 200;
```

- Tiempo de atenuado/encendido (valor entero en milisegundos, máximo 65535). Define el tiempo que tarda el LED en alcanzar su brillo o en apagarse. Dependiendo del tipo de señal real que se quiera representar y de su tipo de lámpara original, se pondrá un valor más alto o más bajo. 120 ms es un buen valor de partida para semáforos clásicos.

```
const uint16_t encendido_ms = 120;
```

- Duración de cada fase de parpadeo (valor entero, en milisegundos). Tiempo durante el cual estará encendido (o apagado) el foco durante el parpadeo, no incluyendo el tiempo de atenuado o encendido definido en la línea anterior.

```
unsigned long periodo = 425;
```

El tiempo total en milisegundos que el foco permanecerá encendido en cada parpadeo vendrá dado entonces por:

$$(2 \cdot \text{encendido_ms}) + \text{periodo}$$

- Pausa en el cambio de aspecto (valor entero en milisegundos, máximo 65535). Define el tiempo que la señal estará apagada durante un cambio de aspecto, para hacer el cambio más estético. 100 ms es un valor de partida adecuado.

```
const uint16_t pausa_aspectos = 100;
```

Parámetros de las placas PCA9685

Hay tres grupos, uno por cada una de las tres posibles placas conectadas. Los objetos se denominan "placa[]" y están numerados de 0 a 2 (placa[0], placa[1] y placa[2]). Ejemplo para la placa 2:

```
// Placa 2 / Board 2
placa[2] = Adafruit_PWMServoDriver(0x42);
placa[2].begin();
placa[2].setPWMFreq(frec);
```

IMPORTANTE: Por defecto están las tres placas activadas, si no se necesitara alguna, comentar la sección correspondiente a esa placa añadiendo dos barras (//) delante de cada línea.

Si fuera necesario conectar más de tres placas, copiar y pegar un grupo y seguir la numeración del array "placa[]".

Aparte de eso, el único parámetro que será necesario introducir es la dirección física de cada placa (ver el apartado 3.2) que viene dada por la función "Adafruit_PWMServoDriver()". En el ejemplo a continuación, la primera placa (0) lleva la dirección "0x40":

```
placa[0] = Adafruit_PWMServoDriver(0x40);
```

Parámetros de los semáforos

Aquí se introducen los datos de todos los semáforos instalados. Se numeran empezando por el 0 hasta un máximo de 11, siendo el número más alto coherente con el número máximo de semáforos definido anteriormente. Dado que se empiezan a numerar por 0, el número más alto corresponderá con Numero_semaforos - 1.

Sólo se utilizarán los grupos que correspondan con semáforos conectados, el resto deben dejarse comentados con dos barras (//). Si por ejemplo tenemos 6 semáforos conectados, sólo utilizaremos los bloques del 0 al 5 y comentaremos el resto. Por defecto están todos activados.

Cada grupo o bloque contendrá los datos de un semáforo, de la forma siguiente:

```
// Semaforo 0 / Signal 0
// Referencia / Reference: Mafen 4131.15 (de salida con 4 focos)

anodo_comun[0] = true;
pin_dummy[0] = false;

semaforo[0][0][0] = 1; // Placa foco verde
```

```

semaforo[0][0][1] = 4;           // Pin foco verde

semaforo[0][1][0] = 0;           // Placa foco rojo
semaforo[0][1][1] = 9;           // Pin foco rojo

semaforo[0][2][0] = 0;           // Placa foco amarillo
semaforo[0][2][1] = 10;          // Pin foco amarillo

semaforo[0][3][0] = 1;           // Placa foco blanco
semaforo[0][3][1] = 5;           // Pin foco blanco

semaforo[0][4][0] = 99;          // Placa foco azul
semaforo[0][4][1] = 99;          // Pin foco azul

comando[0] = 13;                 // Estado inicial de este semáforo

```

Primero se define si la señal es de ánodo común o de cátodo común, ajustando el parámetro “anodo_comun” a “true” o “false”, respectivamente. El ajuste “pin_dummy” no tiene influencia por ahora y se dejará en valor “false” (está previsto para posibles ampliaciones y mejoras).

Después se definen la placa PCA9685 y el pin de esta al que está conectado cada foco, mediante el array “semaforo[][][]”, que está estructurado de la forma siguiente:

semaforo[Número del semáforo][Número del foco][Elemento]

El campo “Elemento” podrá tomar únicamente dos valores posibles: 0 (si lo que vamos a definir es la placa PCA9685 a la que se conecta) o 1 (si lo que vamos a indicar es el pin).

El número del foco se asigna del 0 al 4 para los cinco focos posibles por semáforo. Por comodidad, para ayudar a identificarlos, se han comentado con colores típicos de semáforos: verde (foco 0), rojo (foco 1), amarillo (foco 2), blanco (foco 3) y azul (foco 4), si bien no es estrictamente necesario que se correspondan con los colores reales.

IMPORTANTE: En el caso de que uno de los focos de un semáforo no se use, no hay que comentarlo: simplemente poner su placa y su pin a valor 99. Sólo comentaremos el semáforo entero si no se usa ninguno de sus focos.

Notar que los focos de un mismo semáforo podrán estar conectados a placas distintas, aprovechando así al máximo el número de pines de cada placa.

Finalmente, el array comando[] definirá el aspecto inicial del semáforo al arrancar el sistema, según la definición de aspectos (ver Capítulo 4).

En el ejemplo de arriba, el semáforo es el primero de la lista (número de identificación 0) y tiene cuatro focos. Algunos focos están conectados a la placa 0 y otros a la 1: el foco verde estaría conectado al pin 4 de la placa 1, y el foco blanco al pin 5 de la misma placa. Pero el foco rojo estaría conectado al pin 9 de la placa 0 y el amarillo al pin 10 de esta. No tiene foco azul, así que sus valores son 99 tanto para la placa como para el pin. El comando inicial es el número 13, que correspondería con el aspecto de movimiento autorizado (ver Tabla 1, Capítulo 4).

3.5.2 PARÁMETROS PROPIOS DE LA VERSIÓN DIGITAL

Comunicaciones

Ajustar aquí la velocidad de comunicación del puerto serie del Arduino (en baudios):

```
Serial.begin(9600);
```

Evidentemente ha de ser la misma que en el emisor, ya sea otro Arduino, un PC, o cualquier otro dispositivo. En el ejemplo está ajustada a 9600 baudios.

NOTA: El código presupone que se usa el puerto serie principal del Arduino. Si se emula otro por software o se usa alguno de los alternativos en un Arduino Mega, no olvidar modificarlo aquí y en la función “repcion_comando”.

3.5.3 PARÁMETROS PROPIOS DE LA VERSIÓN ANALÓGICA

En esta guía se cubre únicamente el modo manual (ver 3.1.2).

Número de pulsadores de aspecto

Aquí hay que indicar el número total de aspectos que podremos seleccionar mediante sus correspondientes pulsadores:

```
const uint8_t Num_estados_analog = 8;
```

Parámetros de los semáforos

Además de los parámetros comunes descritos en 3.5.1, habrá que indicar para cada semáforo el pin al que está conectado su pulsador de selección, mediante el parámetro:

```
pin_control[0] = 7;
```

En el ejemplo para el semáforo 0, su pulsador de selección estaría conectado al pin 7 del Arduino.

Pulsadores de aspecto

Hay que indicar los pines a los que estarán conectados los pulsadores que permitirán seleccionar el aspecto, mediante el array “pin_estado[]”. El número total de entradas en este array ha de ser coherente con el valor indicado anteriormente en “Num_estados_analog”:

```
pin_estado[0] = 14;    // Pin para el pulsador "Parada"
pin_estado[1] = 15;    // Pin para el pulsador "Vía libre"
pin_estado[2] = 16;    // Pin para el pulsador "Vía libre condicional"
pin_estado[3] = 17;    // Pin para el pulsador "Anuncio de precaucion"
pin_estado[4] = 20;    // Pin para el pulsador "Anuncio de parada"
pin_estado[5] = 21;    // Pin para el pulsador "Anuncio de parada inmediata"
pin_estado[6] = 0;     // Pin para el pulsador "Rebase autorizado (con parada)"
pin_estado[7] = 1;     // Pin para el pulsador "Movimiento autorizado"
```

Ver el Capítulo 4 para la definición de los valores de cada aspecto.

4 FUNCIONAMIENTO E INSTRUCCIONES DE USO

El programa empieza a funcionar cuando recibe una orden de entrada y construye el array “comando[]”, que tiene tantos elementos como semáforos, cada uno de los cuales es un número entero que define el aspecto del semáforo. Por ejemplo:

comando[4] = 8

asignaría el aspecto “Parada” al semáforo con número 4, según el esquema de la Tabla 1. Esta tabla contiene los principales aspectos utilizados en las señales luminosas de RENFE/ADIF desde la Época IV hasta la actualidad, si bien muchos de los aspectos existen también en señales más antiguas. Los aspectos de “Parada diferida” e “Indicación de salida” no están definidos en el programa (están comentados) pero se ha previsto su posible utilización.

Algunas señales requieren el uso de canales nombrados con otro color para encender un cartelón u otro foco de un color ya existente: esto no representa ningún problema. Un ejemplo es el “preanuncio de parada”, en el que el canal azul encendería el cartelón, o las indicaciones de vía directa o vía desviada, en las que se repite el color blanco usando otro canal.

Aspecto	Número de aspecto asignado	Observaciones
Semáforo apagado	0	
Vía libre	1	
Vía libre condicional	2	
Anuncio de precaución	3	
Preanuncio de parada	4	Cartelón conectado al circuito azul
Anuncio de parada	5	
Anuncio de parada inmediata	6	
Parada diferida	7	No definido
Parada	8	
Rebase autorizado (con parada)	9	
Rebase autorizado (sin parada)	10	
Parada selectiva (foco azul fijo)	11	
Parada selectiva (foco azul parpadeando)	12	
Movimiento autorizado (en señales con foco blanco)	13	
Movimiento autorizado (en señales con foco azul)	14	
Indicación de entrada a vía directa	15	Foco blanco superior conectado al circuito verde
Indicación de entrada a vía desviada	16	Foco blanco lateral conectado al circuito amarillo
Indicación de salida	17	No definido
Parada (en semáforos bajos de dos focos rojos)	18	Foco rojo superior conectado al circuito verde

Tabla 1: Definición de aspectos (señales RENFE/ADIF)

Ver el Apéndice 3 en caso de querer reemplazar, modificar o extender esta distribución de aspectos.

La construcción del array “comando[]” dependerá del sistema que estemos utilizando (digital o analógico, ver el apartado 3.1) y se detalla a continuación.

4.1 VERSIÓN DIGITAL

Funcionamiento interno

En digital, el Arduino recibe las órdenes por defecto como un array a través del puerto serie principal (ver nota en 3.5.2) llamado “comando_recibido”, que contendrá un elemento por cada semáforo. Con el fin de facilitar la expansión con funciones adicionales, este array es en realidad un array de una estructura de datos denominada “comando_semaforos”, que por ahora sólo incluye un valor entero denominado estado con el número de aspecto (según la Tabla 1):

```
struct comando_semaforos
{
    uint8_t estado;
};

comando_semaforos comando_recibido[Numero_semaforos];
```

(Así en un futuro podrá enviarse información adicional al aspecto del semáforo sin cambiar de forma dramática la forma de procesar el programa.)

Después, el programa comparará la orden con el estado actual, y construirá el array “comando[]” para aplicar los nuevos aspectos, de la siguiente forma:

$$\text{comando}[i] = \text{comando_recibido}[i].\text{estado}$$

(Siendo “i” el número que identifica a cada semáforo).

Por lo tanto sólo será necesario que el decodificador mande el array “comando_recibido” con el formato correcto.

Central y decodificadores recomendados

Para aprovechar las capacidades de este sistema (manejo de una gran variedad de tipos de señales con una gran cantidad de aspectos) es **más que recomendable que tanto la central digital como el decodificador soporten el protocolo de paquetes “Signal Aspect”**. Este protocolo permite el manejo de señales multiaspecto (ver *Extended Accessory Decoder Control Packet Format* en la norma [NRMA S-9.2.1](#)). La configuración de otros protocolos existentes, aunque factible, puede ser mucho más compleja y limitada, y no está soportada en este manual.

En cuanto al decodificador, en el Apéndice 1 se muestra un diseño compatible con este estándar, utilizando una placa de Arduino.

Para la central, una opción barata y sencilla que soporta este protocolo es una minicentral DCC con Arduino (hay varios sitios dedicados a ello en Internet). También es posible comprar una comercial a bajo precio de la gama SPROG (pensadas para su manejo con ordenadores y aplicaciones móviles). Si se usa además un programa de control por ordenador que soporte varias centrales, una de estas minicentrales baratas podría usarse únicamente para el manejo de la señalización, dejando a la central principal para el control de los trenes (evitando además así problemas de consumo eléctrico en instalaciones grandes).

Uso con programas de ordenador

También es muy recomendable que el manejo y configuración se realicen desde una aplicación de ordenador o “tablet”, compatible también con el protocolo mencionado (Rocrail, JMRI, etc.) ya que facilitará muchísimo tanto la configuración inicial como el control de señales complejas.

En caso de utilizar Rocrail como programa de control, se incluye en el paquete la carpeta “Rocrail_SVG”, que contiene un archivo ZIP con una carpeta con imágenes SVG para representar en el panel de control los semáforos más comunes de RENFE/ADIF y sus aspectos, además de un archivo “léeme” con el prefijo a utilizar para cada señal y las instrucciones de instalación. La representación de estas señales es puramente simbólica (esto es, no pretende ser un dibujo fiel a la realidad), pero permite visualizar en el panel cada señal con sus aspectos correctos. En la Figura 8 se muestra un semáforo principal de tres focos mostrando el aspecto “Anuncio de precaución”.

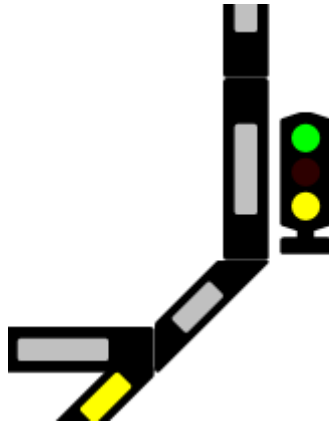


Figura 8: Representación SVG de un semáforo en Rocrail

Para usar estos archivos SVG en Rocrail hay que copiar la carpeta contenida en el ZIP a la carpeta “/svg/themes/”. Seguidamente, habrá que modificar unos parámetros en la configuración del programa. Primero, en la pestaña “SVG” de “Rocview Properties” indicaremos la ruta de la carpeta que hemos copiado. Debemos **añadir** la carpeta a las ya existentes, sin reemplazar ninguna de las que vienen por defecto (ver Figura 9).

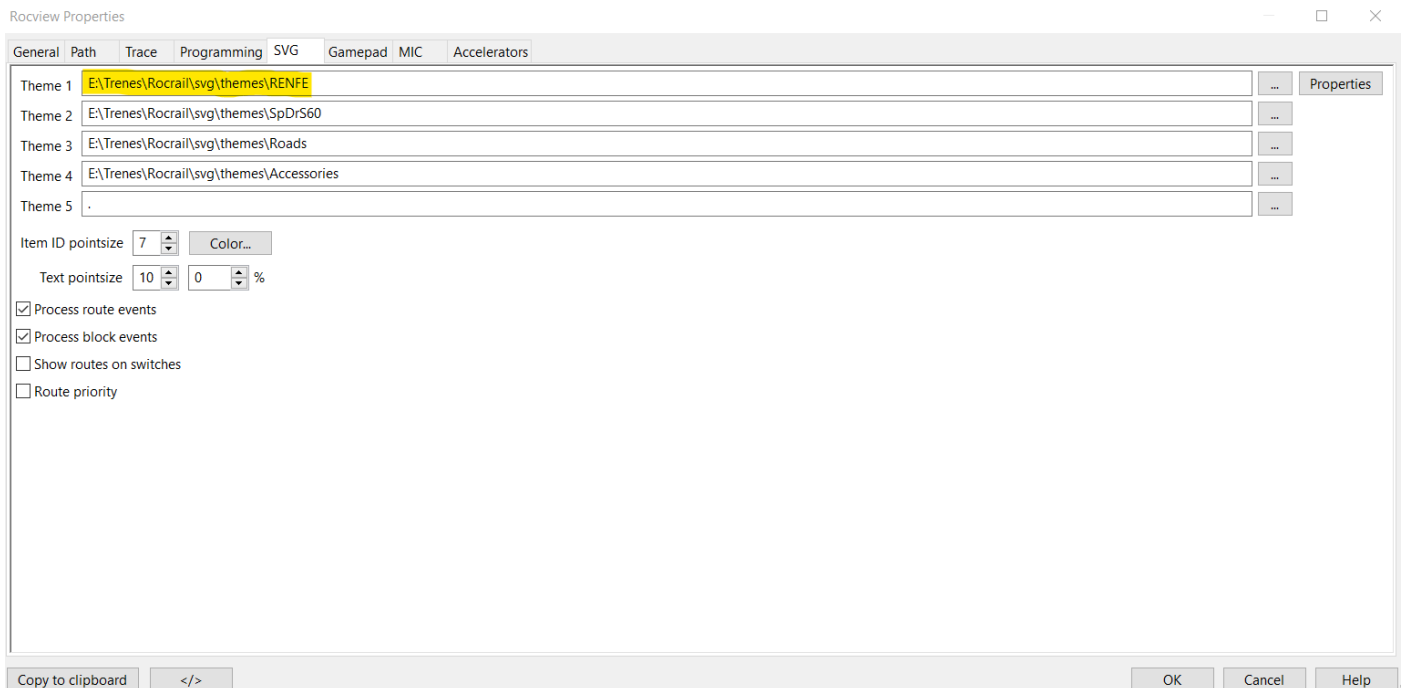


Figura 9: Configuración de Rocrail

Si usamos un control por web o por tablet/teléfono, además habrá que indicar la ruta de la carpeta en la pestaña “RocWeb” de “Rocrail Properties” (ver Figura 10).

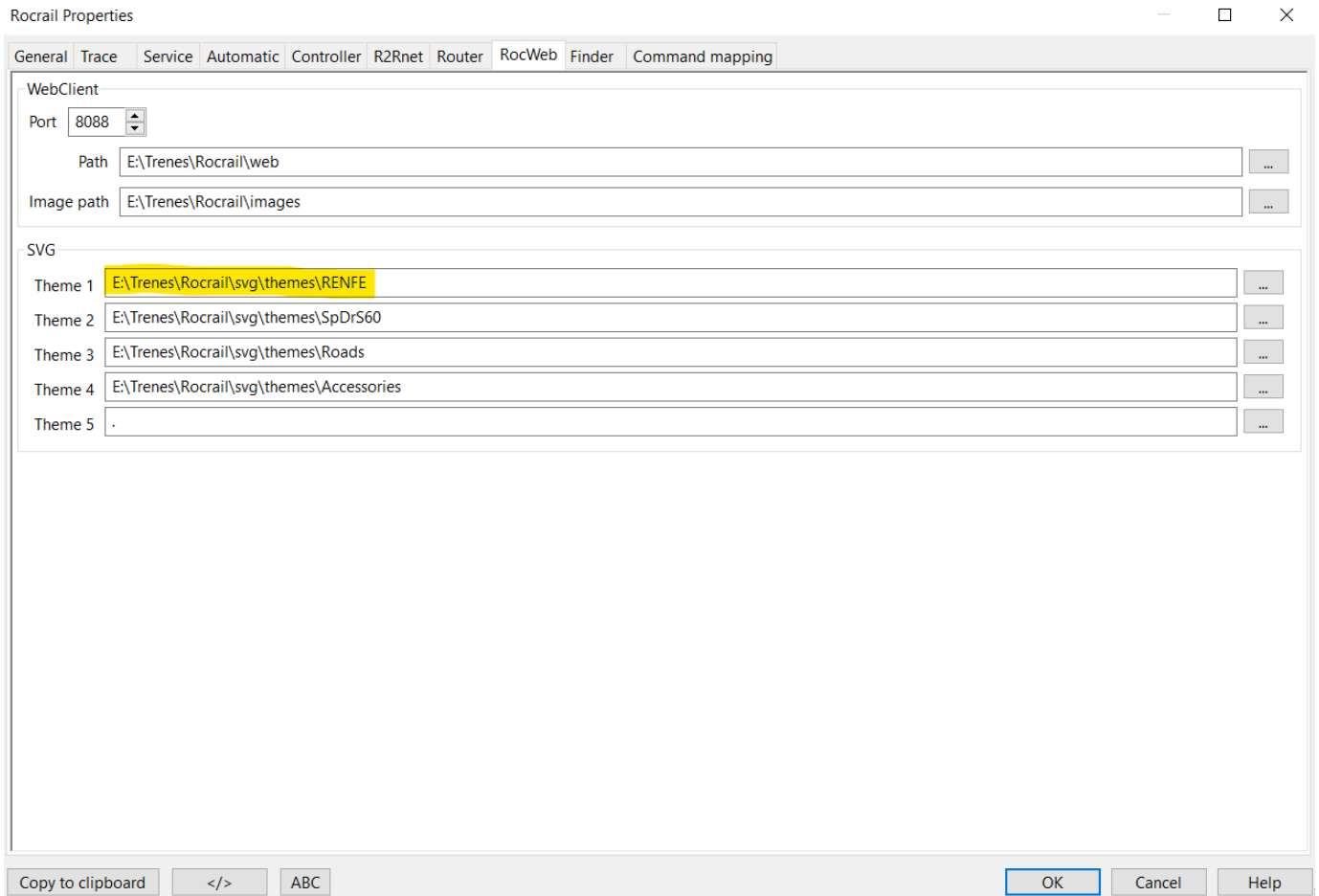


Figura 10: Configuración de Rocview para web/tablet/teléfono

A continuación hay que modificar las propiedades de cada semáforo en nuestro panel. En el apartado “Interface” de la configuración de la señal (Figura 11) es necesario indicar el control tipo “Aspect numbers” y marcar la casilla “Accessory”, así como una sola dirección DCC en el campo “RED” (dejando “Port” al valor 1 en este color). El resto de colores quedarán con 0 en todos los campos.

Ahora, en la pestaña “Details” (Figura 12), y fijándonos en el archivo “léeme” que acompaña a los archivos SVG, definiremos los parámetros para cada señal según su tipo. Indicaremos el número de aspectos posibles que puede representar en el campo “Aspects”. Señalaremos la casilla “Use prefix”, y en “Prefix” indicaremos el prefijo correspondiente al tipo de señal, según se indica en el “léeme”. Si la señal es baja (sin mástil, también llamada “mono”) marcaremos la opción “Dwarf signal”. Finalmente, en “Aspect names” pondremos los nombres de todos los aspectos en orden, tal y como aparecen en el archivo “léeme”, separados por comas.

Por último, si la señal incluye aspectos con luces parpadeantes (aparecerá especificado en el archivo “léeme”), volvemos al panel, pinchamos con botón derecho en la señal, y seleccionamos “Enable alternative SVG”.

Signal Semáforo 2 (4/11)

Interface ID: SPROG DCS

Node ID: 0 0x00000000 UID-Name:

RED Address: 7 Port: 1 ☒ red ☐ green

GREEN Address: 0 Port: 0 ☒ red ☐ green

YELLOW Address: 0 Port: 0 ☒ red ☐ green

WHITE Address: 0 Port: 0 ☒ red ☐ green

Protocol: Default

Dim: 10

Brightness: 100

Parameter: 0

Control:

- ☐ Default
- ☐ Patterns
- ☒ Aspect numbers
- ☐ Linear
- ☐ Binary
- ☐ Function

☒ Accessory Type

- ☒ Output
- ☐ Lights
- ☐ Servo
- ☐ Sound
- ☐ Motor
- ☐ Analog
- ☐ Macro
- ☐ Backlight
- ☐ LED

☒ Invert ☐ Pair gates ☒ Switch ☐ Switch time: 0 ms

Command time: 0 ms

< > </> + ABC OK Cancel Apply Help

Figura 11: Configuración de un semáforo en Rocrail (1)

Signal Semáforo 2 (4/11)

Index General Interface Wiring Details Usage

Signal type:

- ☒ Semaphore signal
- ☐ Light signal

Aspects: 7

Prefix: m_s3_1-

Signification:

- ☐ Distant signal
- ☒ Main signal
- ☐ Shunting signal
- ☐ Block state

☐ Dwarf signal

☒ Use prefix

Patterns / Aspects

	RED Address	GREEN Address	YELLOW Address	NumberValue	NumberValue
RED	<input checked="" type="radio"/> R1 <input type="radio"/> G1 <input type="radio"/> N	<input checked="" type="radio"/> R2 <input type="radio"/> G2 <input type="radio"/> N	<input checked="" type="radio"/> R3 <input type="radio"/> G3 <input type="radio"/> N	0 0 0 0	0 0 0 0
GREEN	<input checked="" type="radio"/> R1 <input type="radio"/> G1 <input type="radio"/> N	<input checked="" type="radio"/> R2 <input type="radio"/> G2 <input type="radio"/> N	<input checked="" type="radio"/> R3 <input type="radio"/> G3 <input type="radio"/> N	0 0 0 0	0 0 0 0
YELLOW	<input checked="" type="radio"/> R1 <input type="radio"/> G1 <input type="radio"/> N	<input checked="" type="radio"/> R2 <input type="radio"/> G2 <input type="radio"/> N	<input checked="" type="radio"/> R3 <input type="radio"/> G3 <input type="radio"/> N	0 0 0 0	0 0 0 0
WHITE	<input checked="" type="radio"/> R1 <input type="radio"/> G1 <input type="radio"/> N	<input checked="" type="radio"/> R2 <input type="radio"/> G2 <input type="radio"/> N	<input checked="" type="radio"/> R3 <input type="radio"/> G3 <input type="radio"/> N	0 0 0 0	0 0 0 0
BLANK	<input checked="" type="radio"/> R1 <input type="radio"/> G1 <input type="radio"/> N	<input checked="" type="radio"/> R2 <input type="radio"/> G2 <input type="radio"/> N	<input checked="" type="radio"/> R3 <input type="radio"/> G3 <input type="radio"/> N	0 0 0 0	0 0 0 0

Aspect names: Parada,Via libre,Via libre condicional,Anuncio de precaucion,Anuncio de parada,Anuncio de parada inmediata,Apagado

< > </> + ABC OK Cancel Apply Help

Figura 12: Configuración de un semáforo en Rocrail (2)

4.2 VERSIÓN ANALÓGICA

En un sistema analógico todo dependerá del tipo de método de entrada que hayamos instalado y del nivel de automatismo, según se vio en la sección 3.1.2.

Si se usa el modo manual con el sistema usado por defecto en el programa (ver 3.1.2.1), se pulsará primero un pulsador de selección de semáforo, y seguidamente otro que le asignará un estado concreto. Así por ejemplo, supongamos una instalación con 12 semáforos y 8 aspectos posibles. Cada uno de los doce semáforos tendría asignado un pulsador, y cada uno de los ocho aspectos tendría asignado otro (etiquetados por ejemplo como: parada, vía libre, vía libre condicional, anuncio de precaución, anuncio de parada, anuncio de parada inmediata, rebase autorizado, y movimiento autorizado). Si quisiéramos poner en vía libre el semáforo número 7, pulsáramos el pulsador correspondiente a esa señal, y seguidamente el de aspecto de vía libre.

Cualquier otro modo de funcionamiento dependerá de la configuración específica y automatismos de cada maqueta, y requerirá una programación adecuada.

5 SOLUCIÓN DE PROBLEMAS MÁS COMUNES

En general:

- Comprobar que el cableado está de acuerdo con el código (cada cable está en el pin correcto).
- Comprobar que los parámetros de número de placas y semáforos concuerdan con los realmente instalados.
- Comprobar que los parámetros de número de placas y semáforos concuerdan con los listados de propiedades de cada uno de ellos.

En digital:

- Comprobar que el número de semáforos en el array enviado por el decodificador coincide con el que espera la placa Arduino de control, en número de elementos y orden de estos, así como en numeración de estados.
- Comprobar que la comunicación por el puerto serie está correctamente establecida, que los cables están cruzados (entre "TX1" y "RX0" si el decodificador es otro Arduino) y que la tasa de transferencia en baudios es la misma entre decodificador y placa de control de los semáforos.

En analógico (modo manual):

- Comprobar que los parámetros del número de pulsadores para selección de señales y aspectos concuerdan con los realmente instalados.
- Comprobar que los parámetros de número de pulsadores para selección de señales y aspectos concuerdan con los listados de propiedades de cada uno de ellos.

APÉNDICE 1: DECODIFICADOR DCC CON ARDUINO

Para decodificar la señal DCC con un Arduino hay que:

- 1) Instalar un circuito optoacoplador entre la señal DCC (generalmente de las vías) y el Arduino.
- 2) Instalar un programa decodificador en el Arduino.

Un Arduino Nano será suficiente para este propósito.



Figura 13: Esquema de conexión del decodificador

NOTA: Es recomendable que el Arduino decodificador sea distinto e independiente del Arduino que controle los semáforos. Es posible juntar ambos en uno, pero hay una serie de limitaciones, como se vio en el Capítulo 3.1.1.

Circuito optoacoplador

El circuito optoacoplador es necesario porque la tensión DCC procedente de las vías o la central digital es mucho más alta (típicamente 12-18 V) de la que puede admitir un Arduino (5 V). Así, el circuito aislará eléctricamente las dos partes pero permitirá el paso de la señal al Arduino sin dañarlo. Se pueden encontrar muchos esquemas en Internet (por ejemplo [aquí](#)), todos son muy similares. Más detalles [en esta otra página](#) (en inglés).

Si bien se puede montar en una placa de prototipado rápido (como se puede ver en los enlaces de arriba) en la carpeta “PCB” hay un diseño sencillo en formato Gerber (“DCC_adaptor-Gerber.zip”) para construir una placa impresa de uno de estos circuitos.

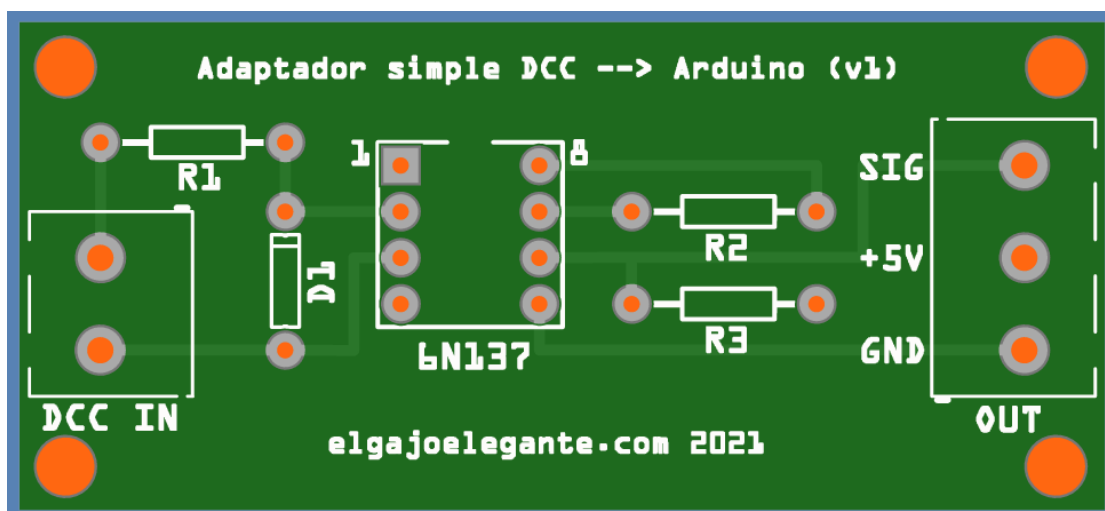


Figura 14: Vista previa de la placa de montaje del circuito optoacoplador

En ella, los componentes necesarios serían:

- D1: Diodo LED (si queremos que haga de testigo visual de tensión en las vías) o 1N4148 (si no es necesario)

- R1: Resistencia de valor dependiente de la tensión de la vía y tipo de central. Generalmente 1 k Ω es un valor adecuado para tensiones de vía típicas (9-14 V); usar 2,2 k Ω para tensiones más altas (16 o 18 V)
- 6N137: Circuito integrado optoacoplador 6N137
- R2: Resistencia de 10 k Ω
- R3: Resistencia de 10 k Ω
- 1 conector de 2 pines de 5,08 mm de paso, para la entrada DCC
- 1 conector de 3 pines de 5,08 mm de paso, para la salida al Arduino

Las vías se conectarán a la entrada “DCC IN”, y el Arduino a la conexión “OUT” de la siguiente forma:

- “SIG” (señal) se unirá al pin de entrada del Arduino decodificador (generalmente el 2, ver más abajo)
- “+5V” se unirá al pin 5V del Arduino decodificador
- “GND” se unirá al pin “GND” del Arduino decodificador

Programa para el Arduino decodificador

Aquí supondremos que se utiliza un Arduino independiente para decodificar la señal DCC.

En cuanto al programa para el Arduino, hay muchos circulando basados en distintas librerías, cada uno con sus ventajas e inconvenientes. En la carpeta “decoder” hay uno adaptado, basado en la librería “NmraDcc.h” (librería que se puede encontrar y descargar [aquí](#)).

Este programa utiliza la interrupción 0 de Arduino, que está asociada generalmente al pin 2, por lo que este será el pin de entrada de la señal procedente del circuito optoacoplador.

Los parámetros por configurar antes de cargar el programa en el Arduino serán los siguientes:

Número de aspectos y semáforos

- Número de aspectos posibles que pueden adoptar los semáforos definidos (valor entero entre 1 y 32). Debe ser coherente con la función “diccionario_tipos” (ver después) y con la definición de los semáforos en las placas de control:

```
const uint8_t Numero_estados = 19;
```

- Número de semáforos a controlar (valor entero; en principio sin límite de 12 porque sería técnicamente posible conectar varias placas de control a un mismo decodificador, usando distintos puertos serie):

```
const uint8_t Numero_semaforos = 12;
```

Parámetros de la función DCC

- Modo de dirección DCC. Hay que elegir uno de los dos tipos **y comentar la otra línea**. La primera es para el modo “Board Addressing” (una dirección DCC con varias subdirecciones para cada pareja de focos), y la segunda para el modo “Output addressing” (dirección DCC única para cada pareja de focos):

```
Dcc.init( MAN_ID_DIY, 10, CV29_ACCESSORY_DECODER, 0 );
Dcc.init( MAN_ID_DIY, 10, CV29_ACCESSORY_DECODER | CV29_OUTPUT_ADDRESS_MODE, 0 );
```

En el caso de usar el modo “Signal Aspect” (que es el modo soportado por defecto en el sistema de control) este ajuste no tendrá efecto, pero hay que dejar comentada una de las líneas de todas formas. Por defecto está comentada la primera, estando seleccionado el modo “Output addressing”.

Parámetros de los semáforos

Los parámetros de los semáforos instalados se introducirán en el array de estructura “semaforo[]”, incluyendo el tipo de semáforo, su dirección DCC asignada, y su aspecto inicial, para cada semáforo. El tipo de semáforo (.tipo) es un número entero coherente con la definición en la función “diccionario_tipos” (ver más abajo). La dirección DCC (.direccion_DCC) es también un número entero, que coincidirá con el que asignemos a la señal en la central digital. Notar que la dirección DCC única por señal asume que estamos en modo “Signal Aspect”. Y finalmente, el aspecto inicial (.estado_inicial) que mostrará la señal al encender el sistema, habrá de ser coherente con la definición de aspectos en la placa de control.

Así, para cada elemento “semaforo[]” contando desde 0, habrá un bloque como el siguiente:

```
// Semaforo 7: / Signal 7:
// Referencia / Reference: Mafen 4131.11 (RENFE principal con 3 focos, tipo 2)

semaforo[7].direccion_DCC = 19;
semaforo[7].tipo = 2;
semaforo[7].estado_inicial = 2;
```

Deberá haber tantos bloques de estos como el número de semáforos definido en el parámetro “Numero_semaforos”, por lo que habrá que comentar/borrar o añadir estos bloques según sea necesario.

En el ejemplo, el semáforo número 7 tendría dirección DCC 19, sería del tipo 2 (semáforo principal de tres focos), y su aspecto inicial 2 correspondería con “vía libre condicional”.

Diccionario de tipos de semáforos

El estándar “Signal Packet” se basa en definir una dirección DCC única para cada señal, mandando a esa dirección un único número entero entre 0 y 31 que le indica al decodificador de la señal qué aspecto tiene que mostrar. Muchas centrales o programas de control que soportan el estándar “Signal Packet” no permiten elegir el número para cada aspecto de una señal, de forma que si un semáforo tiene siete aspectos posibles, numerarán siempre los aspectos del 0 al 6. Así ocurre por ejemplo en Rocrail.

El problema es que, si tenemos varios tipos de señales mezcladas, puede ser imposible configurar para cada una una lista de aspectos consecutivos que coincidan con los de las otras. Por ejemplo, si tenemos una señal capaz de mostrar tres aspectos (parada, vía libre y anuncio de precaución), estos irán numerados del 0 al 2 (en principio en el orden que queramos). Pero si esta señal coexiste con otra capaz de mostrar dos aspectos (como por ejemplo anuncio de precaución y movimiento autorizado) estos se numerarán del 0 al 1, y no habrá forma de hacerlos coincidir con la otra señal. Además, el estándar NMRA indica que siempre que sea posible, el aspecto de “parada absoluta” deberá ocupar el primer lugar (0), lo cual complica todavía más la situación.

La solución es introducir una función en el programa del decodificador que haga de “diccionario” entre el comando recibido de la central digital, y el comando que se va a mandar a la placa de control con el número identificativo del aspecto concreto de cada señal, que será único en el sistema de señalización definido (ver ejemplo en Tabla 1). Esta función se denomina “diccionario_tipos”. Los valores definidos aquí permitirán además indicar el tipo de semáforo (como se vio en el párrafo anterior): cada modelo de semáforo corresponde con un valor case de esta función.

La función viene ya definida para las señales típicas de RENFE/ADIF, en coherencia con el resto del sistema, y no será necesario tocarla salvo que se quiera cambiar el sistema de señalización al de otra compañía, o añadir/quitar/modificar tipos de señales. Ver instrucciones para hacerlo en el Apéndice 3.

APÉNDICE 2: CIRCUITO ADAPTADOR A ÁNODO COMÚN

Se necesitará uno de estos circuitos por cada semáforo, y deberá tener tantas ramas como focos tenga la señal. El material necesario para cada señal será:

- 1 diodo 1N4148 por cada foco de la señal
- 1 transistor BC557 por cada foco de la señal
- 1 resistencia de 10 k Ω por cada foco de la señal

En la Figura 15 se muestra un ejemplo de circuito adaptador para una señal de tres focos (ramas roja, verde y amarilla, numeradas del 1 al 3) pero se puede extender o reducir como se desee según el número de focos siguiendo el mismo patrón, añadiendo o quitando tantas ramas como sean necesarias. Notar que la entrada negativa es una única entrada común, aunque por claridad en el esquema se ha representado por separado como una conexión “a tierra” para cada rama. La salida de la placa PCA9685 se conectaría a las entradas de este circuito (“GND” a la entrada común y “PWR” a cada una de las entradas positivas). El semáforo se conectaría a las salidas marcadas en el esquema **sin olvidar añadir las resistencias adecuadas (o modificar las existentes)** para cada LED en las salidas negativas (-), ya que este circuito no reduce la tensión ni la intensidad de salida.

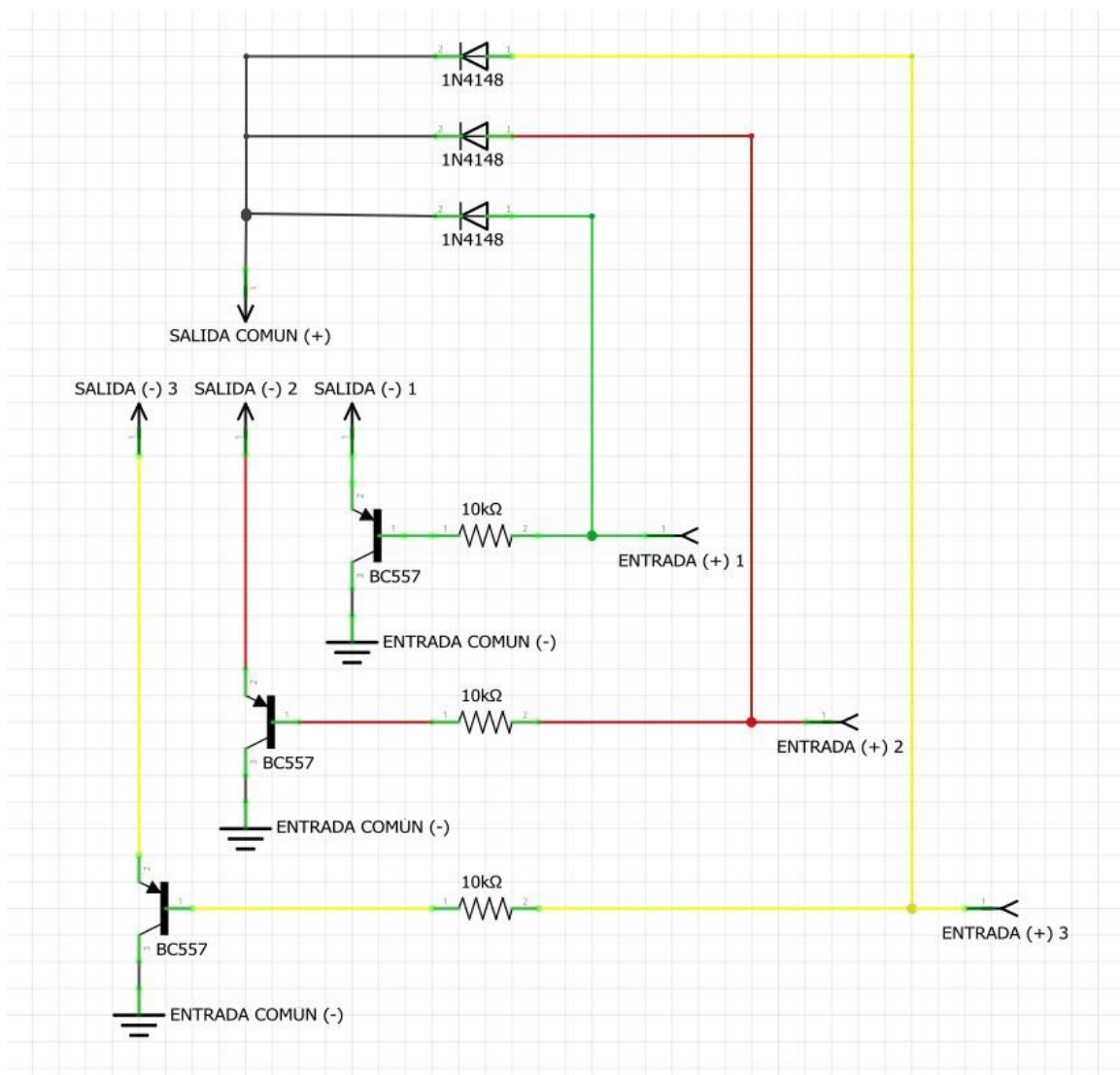


Figura 15: Esquema de un adaptador para ánodo común

Este sencillo adaptador se puede montar sobre una placa de prototipado rápido, o sobre una PCB hecha a medida.

En la carpeta "PCB" se han incluido unos diseños de placa impresa creados con un programa gratuito (KiCad) para estos adaptadores: hay tres modelos para señales de 2, 3 y 4 focos, denominados "Adaptador_anodo-Anode_adaptor". En estas placas se han numerado los focos de 1 a 4 para evitar confusiones con los distintos colores que puede tener una señal.

Estos diseños incluyen previsión para situar las resistencias de los LED necesarias para cada foco, hasta un máximo de dos en serie (nombradas "a" y "b") lo que cubrirá cualquier necesidad. En el caso más común en el que sólo pondremos una resistencia por foco, se podrá conectar en diagonal siguiendo la línea "R eq", como se puede ver en la Figura 16.

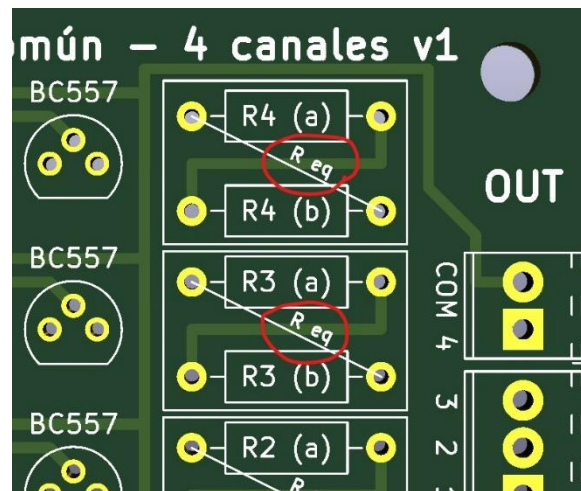


Figura 16: Detalle de la conexión de resistencias de salida en el diseño de PCB suministrado

APÉNDICE 3: CAMBIO DEL SISTEMA DE SEÑALIZACIÓN

En el caso de querer modificar los aspectos existentes de RENFE/ADIF o querer representar un sistema de señalización distinto (de otra administración ferroviaria) habrá que hacer las modificaciones siguientes, tanto en el programa de control como en el del decodificador:

Modificaciones a realizar en el programa de control

Será necesario reescribir la función “nuevo_estado” del programa, definiendo los nuevos aspectos. El programa soporta hasta 255 aspectos distintos.

La estructura es la de una función switch() con tantos casos como número de aspectos (case 0, case 1, etc.) que indican a cada foco su estado según el valor de las variables “control” y “flag_parpadeo” en la Tabla 2. El argumento “índice” es una variable interna, y por lo tanto no se toca.

Valor de “control[índice][foco]”	Valor de “flag_parpadeo”	Estado del foco
0	false	Apagado
5	false	Encendido
5	true	Encendido parpadeando

Tabla 2: Variables de control de cada foco

Finalmente, al terminar de definir los focos de cada estado habría que añadir el valor de “focos_activos”, indicando el número total de focos activos (encendidos o encendidos en parpadeo) en la señal.

A continuación se muestra un ejemplo de definición de un aspecto, con número asignado 10, en el que el foco rojo (1) está iluminado y el blanco (3) parpadeando, con el resto apagados. Por lo tanto, el número de focos activos sería de dos:

```
case 10:      // --- Rebase autorizado (sin parada) ---

    control[indice][1]      = 5;          // Foco rojo encendido
    flag_parpadeo[indice][1] = false;
    control[indice][3]      = 5;          // Foco blanco parpadeando
    flag_parpadeo[indice][3] = true;

    focos_activos[indice] = 2;

    break;
```

Modificaciones a realizar en el programa del decodificador

En el programa del decodificador, primero habrá que modificar el parámetro “Numero_estados” con el nuevo número total de estados posibles.

Después habrá que modificar la función “diccionario_tipos” para definir los nuevos tipos de semáforo y su traducción: de la lista consecutiva de aspectos que mandará la central (ver Apéndice 1) a los aspectos definidos en el programa de control (ver arriba).

La estructura de “diccionario_tipos” es la de una función switch() con tantos casos como número de tipos de señales distintas se vayan a definir (case 0, case 1, etc.). En cada caso indicaremos primero en la variable “aspectos” el número total de aspectos que puede representar la señal, seguido de una lista con la traducción de la numeración de la central a la que será capaz de interpretar el programa de control (según una tabla definida de forma similar a la Tabla 1).

```
case 8:

    // Semaforo bajo RENFE/ADIF con 2 focos: rojo, blanco
    // Ejemplo: Mafen 4141.02
    // 5 aspectos posibles

    aspectos = 5;

    semaforo[id_semaforo].aspecto[0] = 8; // Parada
    semaforo[id_semaforo].aspecto[1] = 9; // Rebase autorizado (con parada)
    semaforo[id_semaforo].aspecto[2] = 10; // Rebase autorizado (sin parada)
    semaforo[id_semaforo].aspecto[3] = 13; // Movimiento autorizado
    semaforo[id_semaforo].aspecto[4] = 0; // Apagado

    break;
```

En el ejemplo, se muestra la definición de un semáforo con 5 aspectos posibles. La central digital numerará estos aspectos de forma consecutiva del 0 al 4, así que pondremos cinco líneas del array “semaforo[id_semaforo].aspecto[]”, seguidas del valor que tomarían en la tabla del programa de control. Así, el tercer aspecto de esta señal (Rebase autorizado sin parada) sería el número 2 enviado por la central digital, y correspondería con el número 10 en el programa de control (Tabla 1).