

Una librería para animar nuestras maquetas de forma simple



Muchas veces queremos añadir algún elemento a nuestras maquetas con algún tipo de efecto o pequeña automatización y consultando con nuestros amigos como implementarlo siempre hay alguien que dice: *“Eso con Arduino es más sencillo de hacer!”*.

Puede ser más sencillo pero tiene el pequeño inconveniente que hay que aprender a programar en lenguaje Arduino para sacar provecho de todo su potencial.

Os presento **simplex.h** una librería para Arduino para realizar animaciones y algo más en nuestras maquetas con comandos simples y compatibles con la programación tradicional del lenguaje Arduino.

1. Introducción a simplex.h

simplex.h es una librería para Arduino que aprovechando la potencia del preprocesador del lenguaje Arduino proporciona comandos simples para automatizar nuestras maquetas y es plenamente integrable con el lenguaje Arduino.

La librería permite dar un nombre (**SET_NAME**) y definir los pines de Arduino como salidas simples (**PIN_OUT**) por ejemplo para usar relés o LED que se pueden activar normalmente o de forma invertida, o que pueden parpadear. Para activarlos se usa **SET** y para desactivarlos **RESET**.

También se pueden definir como salidas para LED con efectos de iluminación (**PIN_EFFECT**) (encendido/apagado lento, soldadura, fluorescentes,...) o por impulsos (**PIN_COIL**) para activar las bobinas de los desvíos o para controlar servos (**PIN_SERVO**).

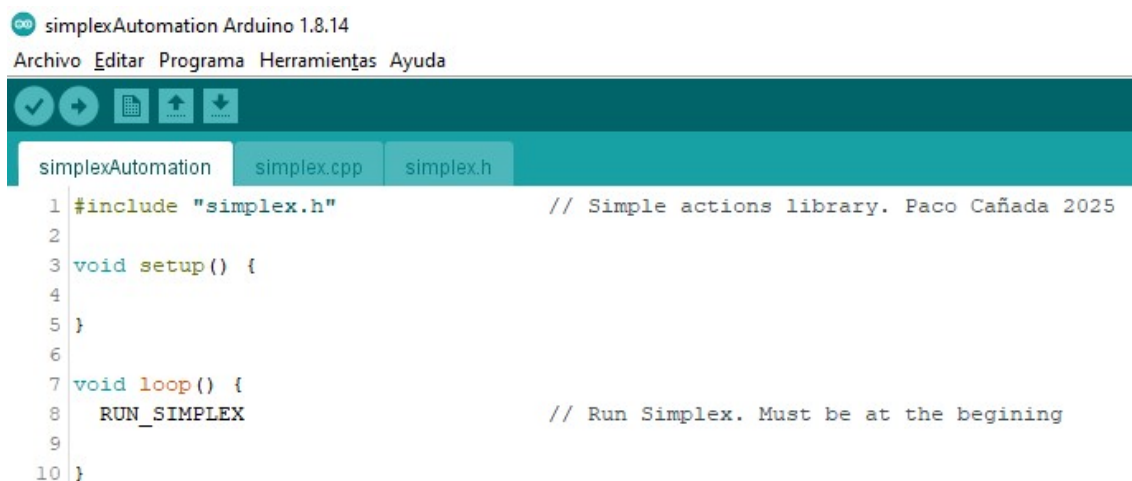
Los pines también se pueden definir como entradas (**PIN_INPUT**) para señales digitales o para botones (**PIN_BUTTON**) o sensores (**PIN_SENSOR**).

Podemos definir esperas (**WAIT**) y algunos temporizadores (**SET_TIMER**) así como repetir acciones (**REPEAT**) y llamar (**CALL**) a funciones (**FUNCTION**) que realizarán más acciones.

Se puede definir de forma simple una maquina de estados finitos (**FSM_STATE**) y cambiar entre estados (**FSM_GO**).

Añadiendo el hardware adecuado podemos realizar un decodificador de accesorios que realice acciones al cambiar un accesorio a verde (**DCC_ACC_GREEN**) o a rojo (**DCC_ACC_RED**).

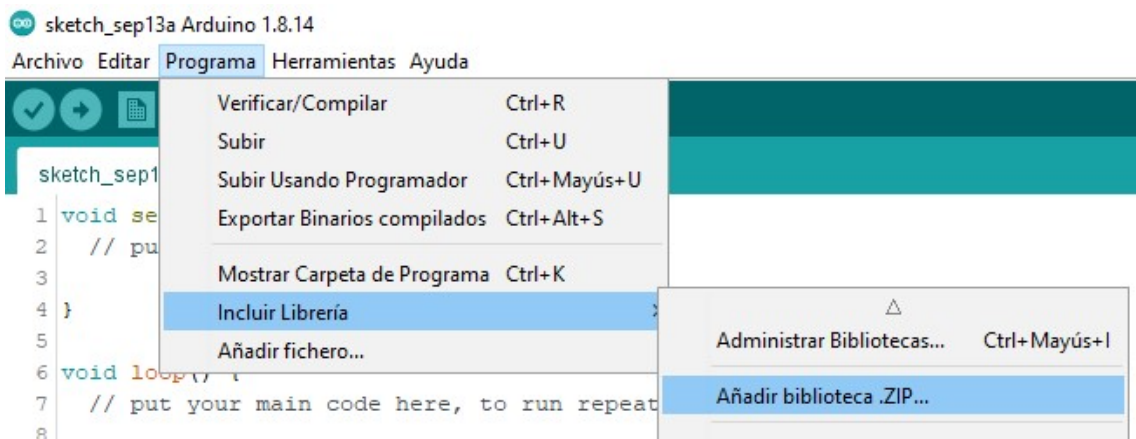
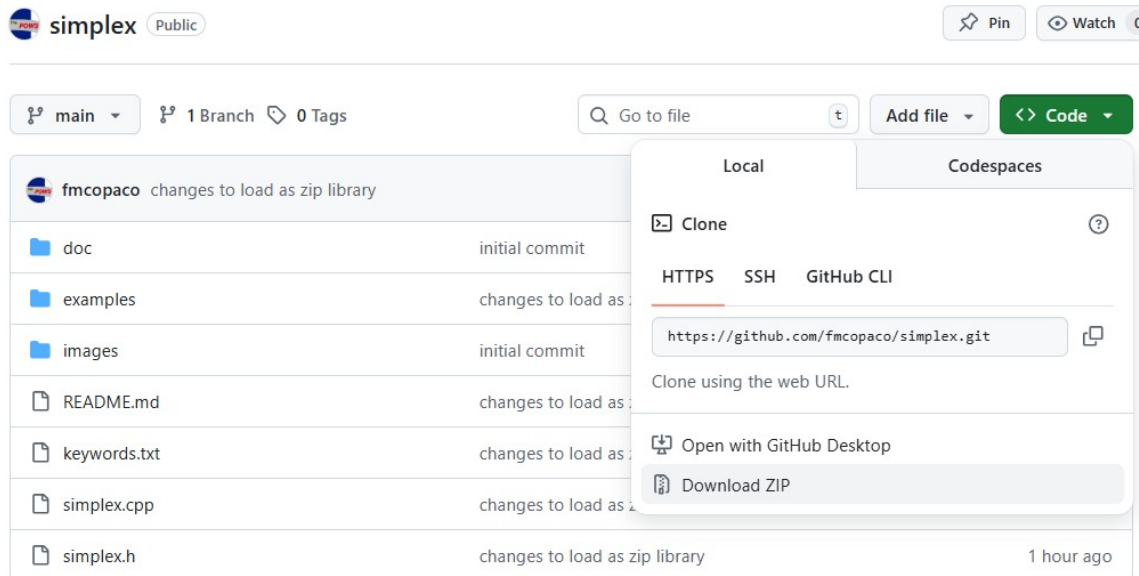
Para usar la librería sólo hay que copiar los archivos **simplex.h** y **simplex.cpp** al directorio donde se encuentra vuestro *sketch*, incluir la librería y en el **loop()** poner **RUN_SIMPLEX** al inicio:



```
simplexAutomation Arduino 1.8.14
Archivo Editar Programa Herramientas Ayuda

simplexAutomation simplex.cpp simplex.h
1 #include "simplex.h" // Simple actions library. Paco Cañada 2025
2
3 void setup() {
4
5 }
6
7 void loop() {
8   RUN_SIMPLEX // Run Simplex. Must be at the begining
9 }
10 }
```

Otra opción es descargar esta librería como archivo .zip y añadirla en el Arduino IDE desde el menú Programa -> Incluir Librería -> **Añadir biblioteca .ZIP ...**, además también estarán disponibles los ejemplos en el menú **Archivo** y no necesitareis copiar los archivos **simplex.h** y **simplex.cpp** al directorio donde se encuentra vuestro sketch.



Comandos de la librería simplex.h

IMPORTANTE Entre el comando y el (no deben escribirse espacios ya que sino el preprocesador generará un error al compilar el código.

Comandos básicos:

Comando	Descripción
RUN_SIMPLEX	Procesa las acciones de simplex.h
SET_NAME(n,p)	Asigna un valor (0...255) a un nombre de constante (pin, etc.,...)
VAR_NAME(n)	Define el nombre de una variable
SET_VAR(n,p)	Dar un valor a una variable (entre -32768 a 32767)
PIN_OUT(p)	Define un pin como salida (relé, luces,...)
PIN_COIL(p)	Define un pin como salida de pulso (Bobina desvío,...)
PIN_EFFECT(p)	Define un pin como efecto de luz
PIN_SERVO(p)	Define un pin como servo (máx. 12: _MAX_SERVO)
PIN_INPUT(p)	Define un pin como entrada
PIN_BUTTON(p)	Define un pin como entrada anti rebote (Pulsador, final carrera...)
PIN_SENSOR(p)	Define un pin como una entrada de sensor
PIN_STEPPER4(p)	Define 4 pins consecutivos para control motor paso a paso (tipo 28BYJ-48)
PIN_STEPPER2(p)	Define 2 pins consecutivos para control motor paso a paso (tipo driver A4988 y similares, p: STEP, p+1: DIR)
PIN_MODE(p,n)	Establece el modo de un pin (tipo PIN_OUT/ PIN_EFFECT/PIN_STEPPER)
PIN_TIME(p,n)	Establece el tiempo para un pin o la velocidad de un servo o motor paso a paso
SET(p)	Activa pin de salida (tipo PIN_OUT/ PIN_EFFECT/ PIN_COIL)
RESET(p)	Desactiva pin de salida (tipo PIN_OUT/ PIN_EFFECT/ PIN_COIL)
SERVO(p,n)	Mover el servo a una posición (0..180)
STEPPER_SET(p,n)	Establece el paso actual (posición) del motor paso a paso
STEPPER_GO(p,n)	Mover el motor paso a paso a la posición absoluta
STEPPER_CW(p,n)	Mover el motor paso a paso a la posición relativa (sentido horario)
STEPPER_CCW(p,n)	Mover el motor paso a paso a la posición relativa (sentido antihorario)
ARRIVED(p)	Comprobar si el servo llegó a la posición
PRESSED(p)	Comprobar si se pulsó el botón
ACTIVE(p)	Comprobar si el sensor esta activo
FREE(p)	Comprobar si el sensor esta libre
FSM_NEW(n)	FSM, define el nombre de una nueva FSM (Multi FSM)
FSM_USE(n)	FSM, usar los estados correspondientes al FSM (Multi FSM)
FSM_NAME(n)	FSM, define el nombre de un nuevo estado
FSM_STATE(n)	FSM, comprobar estado actual
FSM_GO(n)	FSM, transición a un nuevo estado
SET_TIMER(p,n)	Arrancar un temporizador (máx. 4: _MAX_TIMERS)
TIMEOUT(p)	Comprobar si el temporizador acabó (máx. 4: _MAX_TIMERS)
WAIT(n)	Espera un tiempo procesando acciones simplex
WAIT_SERVO(p)	Espera que un servo llegue a la posición procesando acciones simplex
WAIT_RELEASE(p)	Espera a que se suelte el botón procesando acciones simplex
WAIT_STEPPER(p)	Espera que un motor paso a paso llegue a la posición procesando acciones simplex
FUNCTION(n)	Define una función
CALL(n)	Llamar a una función
REPEAT(n)	Repetir comandos (2..65535 veces, máx. 4 bucles: _MAX_REPEAT)
AND(x,y)	Comprobar si ambas condiciones son verdaderas

Comando	Descripción
OR(x,y)	Comprobar si al menos una condición es verdadera
NOT(x)	Negación lógica
EQUAL(x,y)	Comprobar si dos valores son iguales

Comandos DCC:

Comando	Descripción
PIN_DCC()	Usar pin de entrada DCC (_DCC_PIN)
DCC_ACC_RED(n)	Comprobar si el número de accesorio recibido está en rojo
DCC_ACC_GREEN(n)	Comprobar si el número de accesorio recibido está en verde

Comandos Xpressnet:

Comando	Descripción
XNET_ADDR(n)	Inicializa interface Xpressnet (MAX485). Dirección válida: 1..31
XNET_RED(n)	Mover el accesorio a rojo (1..1024)
XNET_GREEN(n)	Mover el accesorio a verde (1..1024)
XNET_TOGGLE(n)	Mover accesorio a la otra posición (1..512)
XNET_ACTIVE(n,p)	Comprobar si la entrada p del módulo RS n está ocupada
XNET_FREE(n,p)	Comprobar si la entrada p del módulo RS n está libre
XNET_CHANGED(n)	Comprobar si hay cambios en el módulo RS n

NOTA Con el Arduino Uno/Nano si usamos [Serial](#) en nuestro programa no podremos usar los comandos Xpressnet. En el archivo **simplex.h** modifique la siguiente línea como sigue:

```
#define USE_XNET    false
```

2. Ejemplo básico: Blink

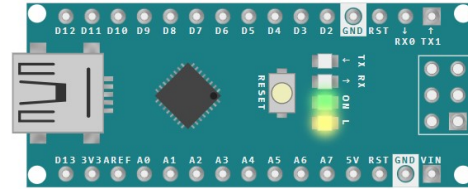
El ejemplo básico de hacer parpadear el LED de la placa del Arduino en el pin 13 con la librería **simplex.h** podría ser así:

```
#include "simplex.h"           // Librería de acciones simples de Paco Cañada 2025

SET_NAME(LED, 13)            // Damos el nombre LED el valor 13

void setup() {
  PIN_OUT(LED)                // Definimos el pin como una salida
  PIN_MODE(LED, FLASH)        // La salida será parpadeante
  PIN_TIME(LED, 1000)          // El tiempo de parpadeo será 1 segundo
  SET(LED)                    // activamos la salida
}

void loop() {
  RUN_SIMPLEX                  // Run Simplex must be at the beginning
}
```



SET_NAME nos permite definir un nombre en este caso para un pin, como parámetros necesita el nombre y el valor (entre 0 y 255) a asociar a ese nombre, así cuando tengamos diferentes salidas podremos dirigirnos a ellas por su nombre en lugar de por su número si así lo deseamos.

PIN_OUT permite definir una salida digital, necesita un parámetro que es el pin.

PIN_MODE permite cambiar el modo de cómo se comporta una salida (**NORMAL**, **INVERT** o **FLASH**), necesita dos parámetros el pin y el modo, en este caso **FLASH**.

PIN_TIME permite definir el tiempo de parpadeo, necesita dos parámetros, el pin y el tiempo en milisegundos, en este caso un segundo encendido y un segundo apagado (1 segundo = 1000ms).

SET activa la salida, si el modo es **NORMAL** tendremos 5V en el pin, si el modo es **INVERT** tendremos 0V y si es **FLASH** parpadeará con el tiempo definido por **PIN_TIME**.

En el **loop()** sólo necesitamos incluir **RUN_SIMPLEX** que se encargará de gestionar las salidas y los tiempos.

No necesitamos nada más, la librería hará su trabajo y pondrá a parpadear el pin como se ha definido.

3. El perrito

Alfred tiene en su módulo un perrito que al accionar el público un pulsador sale de su caseta y ladra. El perrito se mueve con la ayuda de un servo y el ladrido lo consigue con un módulo con el sonido pregrabado que se activa al dar un pulso en una entrada. En su día lo hizo usando mi decodificador **KDaktion** (https://usuarios.tinet.cat/fmco/dccacc_sp.html#kdaktion). Con la librería **simplex.h** sería así:



[link](#)

```
#include "simplex.h" // Librería de acciones simples de Paco Cañada 2025

// Define los nombres aquí
SET_NAME(PERRITO, 11) // Nombre para el pin del servo
SET_NAME(PULSADOR, 10) // Nombre para el pin del pulsador
SET_NAME(SONIDO, 9) // Nombre para el pin del módulo de sonido
SET_NAME(LED, 13) // Nombre para el pin del LED

void setup() {
    // Define los pines aquí
    PIN_SERVO(PERRITO) // servo que mueve el perrito
    PIN_TIME(PERRITO, 80) // velocidad del servo
    PIN_BUTTON(PULSADOR) // pulsador del publico
    PIN_COIL(SONIDO) // salida impulso para modulo sonido
    PIN_TIME(SONIDO, 500) // tiempo del pulso para modulo de sonido
    PIN_OUT(LED) // LED para el publico

    // Estado inicial
    RESET(SONIDO) // no activar sonido
    SET(LED) // enciende LED para indicar listo
    SERVO(PERRITO, 0) // mover el perrito a la caseta
}

void loop() {
    RUN_SIMPLEX // Run Simplex debe estar al principio

    PRESSED(PULSADOR) {
        PIN_MODE(LED, FLASH) // parpadeo del LED para indicar ocupado
        SERVO(PERRITO, 100) // mover el perrito fuera de la caseta
        WAIT_SERVO(PERRITO) // esperar a que llegue
        SET(SONIDO) // activar el ladrido
        WAIT(6000) // esperar 6 segundos
        SERVO(PERRITO, 0) // mover el perrito a la caseta
        WAIT_SERVO(PERRITO) // esperar a que llegue
        WAIT(10000) // esperar 10 segundos para otros espectadores
        PIN_MODE(LED, NORMAL) // modo LED fijo
        SET(LED) // enciende LED para indicar listo
    }
}
```

Inicialmente con **SET_NAME** definimos los nombres de los pins, no es necesario pero luego el código queda más legible, no siempre recordamos que **SET(9)** es activar la salida de sonido.

En el **setup()** definimos qué función tiene cada pin y modificamos los parámetros necesarios para que haga lo que queremos. Con **PIN_SERVO** definimos la salida para un servo y con **PIN_TIME** le decimos la velocidad a la que se moverá.

PIN_BUTTON lo usamos para definir la entrada de un pulsador conectado entre el pin y GND, por defecto tiene activada el antirebote y el *pull-up* por lo que no es necesaria una resistencia externa entre el pin y +5V a no ser que los cables sean muy largos.

PIN_COIL es una salida que dará un pulso cuando se active, con **PIN_TIME** definimos en este caso la duración de ese pulso en **ms**. En esta salida puede que tengamos que usar hardware adicional para activar el modulo de sonido dependiendo del tipo de módulo.

PIN_OUT es una simple salida digital para activar un LED para que el público sepa cuándo puede pulsar el botón.

Después de definir los pines ponemos las condiciones iniciales, con **RESET** desactivamos la salida y con **SET** la activamos. Con **SERVO** hacemos que el correspondiente ángulo del servo coloque al perrito dentro de su caseta. Por defecto la salida del LED es fija así que el LED se encenderá para indicar al público que puede pulsar el botón.

Luego en el **loop()** que se repetirá indefinidamente, ponemos al principio **RUN_SIMPLEX** para que la librería se encargue de todo y ponemos las acciones para cuando se pulse el botón.

Para saber cuándo se pulsa el botón usamos **PRESSED** que ejecutará las acciones que hay entre las **{ }** cuando el botón del pin indicado se pulse.

Al pulsar el botón, con **PIN_MODE** hacemos que el LED parpadee (**FLASH**) para indicar que el perrito está ocupado, por defecto parpadea con un ritmo de 500ms pero lo podemos modificar con **PIN_TIME**.

Hacemos que el perrito salga de la caseta moviendo el servo un ángulo hasta que llegue al patio con **SERVO** y esperamos a que llegue con **WAIT_SERVO** antes de activar el sonido del ladrido con **SET**, esta salida al estar definida como pulso se desactivará automáticamente.

Luego esperamos un tiempo con **WAIT** a que ladre. Es importante usar **WAIT** aquí ya que si se usa **delay()** del lenguaje Arduino entonces no se estaría ejecutando la librería **simplex.h** y las temporizaciones del LED y la salida de sonido se verían afectadas.

Movemos con **SERVO** y esperamos con **WAIT_SERVO** a que el perrito entre en la caseta y esperamos un tiempo con **WAIT** para que el público deje paso a otra persona que anime nuevamente al perrito a salir.

Finalmente hacemos que el LED deje de parpadear poniendo el LED en modo **NORMAL** con **PIN_MODE**.

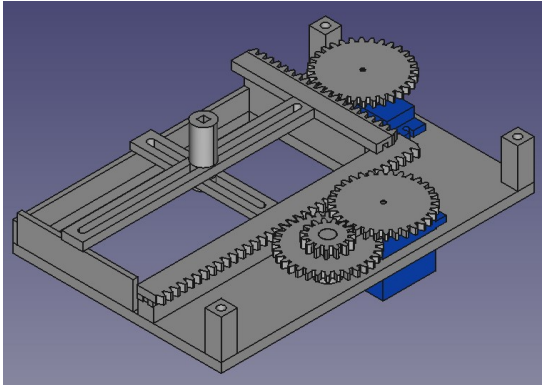
Como el **loop()** se repetirá la próxima vez que se pulse el botón se ejecutará de nuevo nuestra secuencia.

4. El Almacén

En mi módulo del Almacén de la estación del Carrilet de Salou he colocado dentro una carretilla con un pequeño imán que se mueve mediante un mecanismo accionado por dos servos para obtener movimiento en el eje X y en el eje Y. Al pulsar el botón se enciende la luz del almacén y aparece la carretilla dese el fondo, luego se mueve la grúa del muelle de carga para mover una carga al vagón



<https://youtu.be/o2QX1iSs8tc>



```
#include "simplex.h" // Simple actions library. Paco Cañada 2025

// Define los nombres aquí
SET_NAME(CARRETILLA_X, 5)
SET_NAME(CARRETILLA_Y, 6)
SET_NAME(GRUA, 9)
SET_NAME(PULSADOR, A4)
SET_NAME(LUZ, A5)
SET_NAME(LED, 2)

void setup() {
    // Define los pines aquí
    PIN_OUT(LUZ)
    PIN_OUT(LED)
    PIN_BUTTON(PULSADOR)
    PIN_SERVO(CARRETILLA_X)
    PIN_SERVO(CARRETILLA_Y)
    PIN_SERVO(GRUA)
    PIN_TIME(CARRETILLA_Y, 60)
    PIN_TIME(GRUA, 80)

    // Estado inicial
    RESET(LUZ)
    SET(LED)
    SERVO(CARRETILLA_X, 90)
    SERVO(CARRETILLA_Y, 90)
    SERVO(GRUA, 135)
}

void loop() {
    RUN_SIMPLEX // Run Simplex debe estar al principio

    PRESSED(PULSADOR) {
        // Cuando se pulse el botón
        PIN_MODE(LED, FLASH) // parpadeo del LED para indicar ocupado
        SET(LUZ) // encender luz del almacén
        WAIT(2000)
        SERVO(CARRETILLA_Y, 60) // mover la carretilla
        WAIT_SERVO(CARRETILLA_Y)
        PIN_TIME(CARRETILLA_X, 60)
        SERVO(CARRETILLA_X, 10)
        WAIT_SERVO(CARRETILLA_X)
        WAIT(2000)
        SERVO(GRUA, 150) // mover grúa a recoger carga
        WAIT_SERVO(GRUA)
        WAIT(1500)
        SERVO(GRUA, 35) // mover carga al vagón
        WAIT_SERVO(GRUA)
        WAIT(2500)
        SERVO(GRUA, 135) // mover grúa a posición inicial
        WAIT_SERVO(GRUA)
        WAIT(3000)
        PIN_TIME(CARRETILLA_X, 30) // mover carretilla a posición inicial
        SERVO(CARRETILLA_X, 90)
        SERVO(CARRETILLA_Y, 90)
        WAIT_SERVO(CARRETILLA_Y)
        RESET(LUZ)
        WAIT(5000) // esperar 5 segundos para espectadores
        PIN_MODE(LED, NORMAL) // modo LED fijo
        SET(LED) // enciende LED para indicar listo
    }
}
```

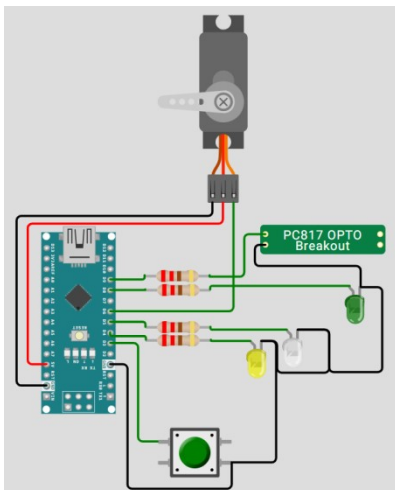
Como en el ejemplo anterior aquí movemos los servos con **SERVO** y variamos su velocidad con **PIN_TIME**, si queremos movimientos rectos usamos **WAIT_SERVO**, sino serán movimientos en diagonal.

5. Ermita

Otra animación que Alfred tiene en sus módulos es una ermita con una campana funcional que al pulsar el público un botón se enciende la luz de la ermita, la campana se mueve con un servo y suena gracias a un módulo de sonido, al acabar las campanadas se enciende una vela en la capilla y después de un tiempo se apagan todas las luces. Como principiante en programación en lenguaje Arduino fue una tarea ardua ya que no tenía a su disposición la librería **simplex.h**:



[link](#)



```
#include "simplex.h" // Librería de acciones simples de Paco Cañada 2025

// Define los nombres aquí
SET_NAME(CAMPANA, 6) // Nombre para el pin del servo
SET_NAME(PULSADOR, 3) // Nombre para el pin del pulsador
SET_NAME(VELA, 4) // Nombre para el pin del LED amarillo
SET_NAME(LUZ, 5) // Nombre para el pin del LED blanco
SET_NAME(SONIDO, 9) // Nombre para el pin del módulo de sonido
SET_NAME(LED, 8) // Nombre para el pin del LED verde público

void setup() {
    // Define los pines aquí
    PIN_SERVO(CAMPANA) // servo que mueve la campana
    PIN_TIME(CAMPANA, 20) // velocidad del servo
    PIN_BUTTON(PULSADOR) // pulsador del publico
    PIN_COIL(SONIDO) // salida impulso para modulo sonido
    PIN_TIME(SONIDO, 500) // tiempo del pulso para modulo de sonido
    PIN_OUT(LED) // LED verde para el publico
    PIN_OUT(LUZ) // LED blanco luz ermita
    PIN_EFFECT(VELA) // LED amarillo vela
    PIN_MODE(VELA, CANDLE) // Efecto de vela

    // Define el estado inicial aquí
    RESET(SONIDO) // no activar sonido
    SET(LED) // enciende LED para indicar listo
    SERVO(CAMPANA, 80) // campana colgando
}

void loop() {
    RUN_SIMPLEX // Run Simplex debe estar al principio

    PRESSED(PULSADOR) {
        // Cuando se pulse el botón
        PIN_MODE(LED, FLASH) // hacer parpadear el LED para indicar ocupado
        SET(LUZ) // enciende luz de la ermita
        WAIT(2000) // espera al capellan
        SET(SONIDO) // activar el sonido
        REPEAT(6) {
            // repetir 6 veces
            SERVO(CAMPANA, 140) // mover campana a un extremo
            WAIT_SERVO(CAMPANA) // esperar a que llegue
            SERVO(CAMPANA, 20) // mover campana al otro extremo
            WAIT_SERVO(CAMPANA) // esperar a que llegue
        }
        SERVO(CAMPANA, 80) // campana colgando
        SET(VELA) // enciende vela
        WAIT(5000) // espera 5 segundos
        RESET(VELA) // apaga vela
        RESET(LUZ) // apaga luz
        PIN_MODE(LED, NORMAL) // modo LED fijo
        SET(LED) // enciende LED para indicar listo
    }
}
```

Esta animación igual que las anteriores espera a que se pulse el botón, hace parpadear el LED del público y enciende la luz de la ermita, espera un tiempo y activa el sonido de la campana y la mueve. En este caso para activar el sonido lo hace activando la entrada del módulo de sonido través de un opto conectado a una salida del Arduino.

Para mover la campana repite 6 veces el movimiento del servo de un extremo a otro de la campana. **REPEAT** repetirá las veces indicadas las acciones que hay entre las **{ }** finalmente volvemos a dejar la campana colgando.

Luego enciende la vela, esta salida la hemos definido como **PIN_EFFECT** y le hemos asignado el efecto **CANDLE** con **PIN_MODE** con lo que la librería **simplex.h** hará el efecto de luz temblorosa como una vela en esta salida. Estos son los efectos que se pueden conseguir con **simplex.h**

<i>Efecto</i>	<i>Descripción</i>
DIMMER	Encendido / apagado lento
FLASH	Parpadeo con encendido/apagado lento
CANDLE	Vela
FIRE	Fuego
FLUORESCENT	Luz fluorescente
WELDING	Soldadura
PWM	Intensidad 0..15 (62.5Hz)

Las salidas están pensadas para conectar un LED con su resistencia entre la salida y GND. El efecto **FLASH** permite usar **INVERT** para tener dos salidas parpadeantes alternantes, para ello use primero **INVERT** y luego **FLASH** con **PIN_MODE** con el mismo tiempo de parpadeo para ambas y active con **SET** las dos salidas a la vez.

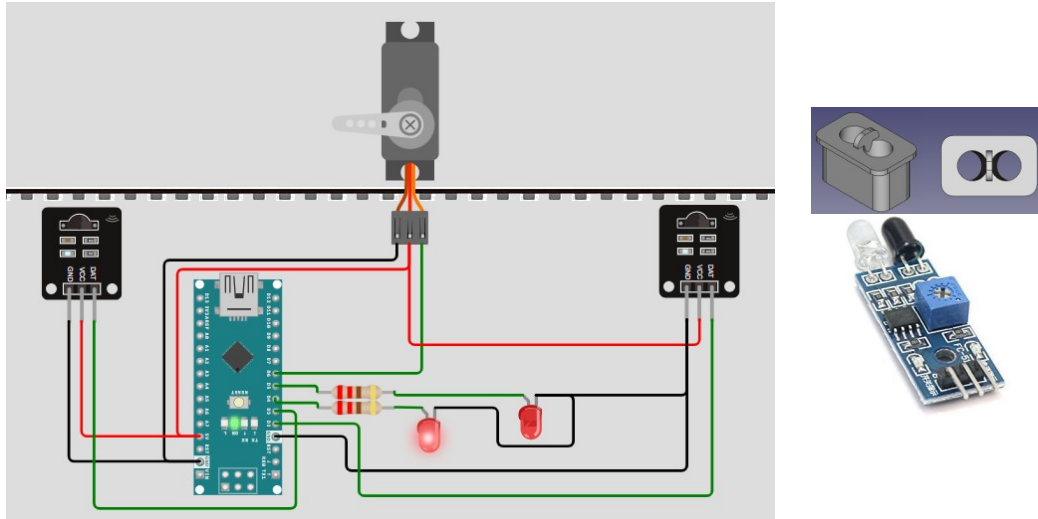
```
PIN_EFFECT(11)           // pin 11 con efecto flash
PIN_MODE(11, FLASH)
PIN_TIME(11, 1200)
PIN_EFFECT(10)           // pin 10 con efecto flash invertido
PIN_MODE(10, INVERT)
PIN_MODE(10, FLASH)
PIN_TIME(10, 1200)       // usar el mismo tiempo de parpadeo

SET(10)                  // activar ambas salidas a la vez
SET(11)
```

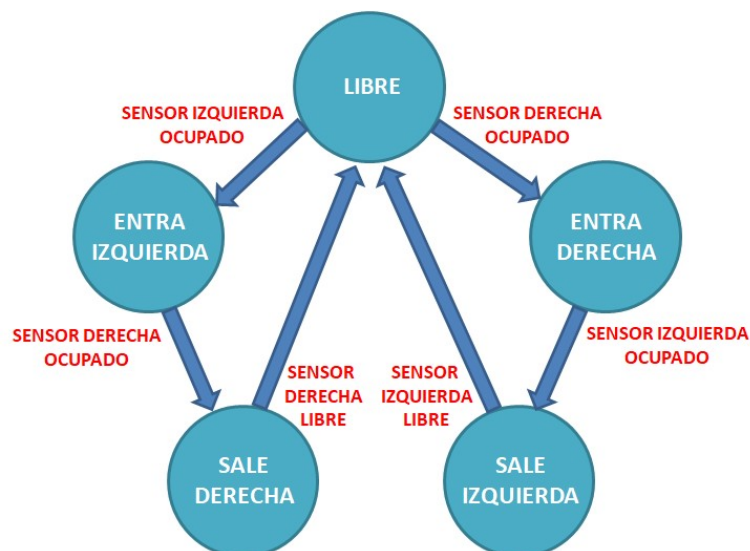
El efecto **PWM** es para controlar la intensidad de un LED o la velocidad de un motor, este último necesitaría un transistor por la gran corriente que consume. El valor de la intensidad/velocidad se define con **PIN_TIME** con un valor entre 0 y 15. La frecuencia PWM es bastante baja (62.5 Hz).

6. Paso a nivel

Un automatismo de un paso a nivel en el que se controlen las barreras con un servo y las luces intermitentes alternantes en el que bajen las barreras al llegar un tren y que suban cuando ya ha pasado todo el convoy mediante un par de detectores no es una tarea normalmente sencilla en lenguaje Arduino. Este tipo de automatizaciones se puede implementar con una máquina de estados finitos.



La librería **simplex.h** proporciona unos comandos básicos para la programación mediante una máquina de estados finitos (FSM). En este tipo de programación se definen unos estados en los que el sistema permanece estable y pasa de uno a otro al darse alguna condición, durante esta transición se ejecutan las acciones necesarias para cambiar de estado.



En este caso lo podemos implementar con 5 estados como se ve en el gráfico. En el estado LIBRE no hay tren circulando en el tramo del paso a nivel, las barreras están levantadas y la señal luminosa está apagada.

Cuando un tren llega a uno de los detectores colocados a una distancia que dé tiempo a bajar las barreras, se bajará la barrera y se activará la señal luminosa intermitente y dependiendo del sensor que se active se pasará al estado `ENTRA_DERECHA` o `ENTRA_IZQUIERDA`.

Ahora hay que esperar a que llegue al otro detector para que pasemos al estado `SALE_IZQUIERDA` o `SALE_DERECHA`. Finalmente, en estos últimos estados hay que esperar a que este libre este segundo detector para subir la barrera y apagar la señal y pasar de nuevo al estado `LIBRE`.

Con la librería **simplex.h** lo podemos implementar así:

```
#include "simplex.h" // Librería de acciones simples de Paco Cañada 2025

// Define los nombres aquí
SET_NAME(BARRERA, 6) // Nombre para el pin del servo
SET_NAME(IZQUIERDA, 3) // Nombre para el pin del sensor izquierda
SET_NAME(DERECHA, 2) // Nombre para el pin del sensor derecha
SET_NAME(LUZ1, 4) // Nombre para el pin del LED rojo
SET_NAME(LUZ2, 5) // Nombre para el pin del LED rojo

// Define los estados aquí
FSM_NAME(LIBRE) // paso a nivel libre
FSM_NAME(ENTRA_DERECHA) // tren entrando por la derecha
FSM_NAME(ENTRA_IZQUIERDA) // tren entrando por la izquierda
FSM_NAME(SALE_DERECHA) // tren saliendo por la derecha
FSM_NAME(SALE_IZQUIERDA) // tren saliendo por la izquierda

// Define funciones adicionales aquí
FUNCTION(BajarBarrera) {
  SET(LUZ1)
  SET(LUZ2)
  SERVO(BARRERA, 0)
}

FUNCTION(SubirBarrera) {
  RESET(LUZ1)
  RESET(LUZ2)
  SERVO(BARRERA, 90)
}
```

Con `SET_NAME` definimos un nombre para cada pin, con `FSM_NAME` definimos un nombre para cada uno de los estados.

Para acciones que se repitan podemos definir pequeñas funciones con `FUNCTION` a las que daremos un nombre para después llamarlas con `CALL` cuando queramos que se ejecuten. En este caso definimos la función *BajarBarrera* para que el servo baje la barrera y la señal intermitente se encienda y la función *SubirBarrera* para que el servo suba la barrera y la señal intermitente se apague.

```
void setup() {
  // Define los pines aquí
  PIN_SERVO(BARRERA) // servo que mueve la campana
  PIN_TIME(BARRERA, 10) // velocidad del servo
  PIN_EFFECT(LUZ1) // LED rojo señal
  PIN_MODE(LUZ1, INVERT)
  PIN_MODE(LUZ1, FLASH)
  PIN_TIME(LUZ1, 1000)
  PIN_EFFECT(LUZ2) // otro LED rojo señal
  PIN_MODE(LUZ2, FLASH)
  PIN_TIME(LUZ2, 1000)
  PIN_SENSOR(DERECHA) // sensor derecha
  PIN_TIME(DERECHA, 2000) // para sensor infrarrojo aumentar tiempo
  PIN_SENSOR(IZQUIERDA) // sensor izquierda
  PIN_TIME(IZQUIERDA, 2000) // para sensor infrarrojo aumentar tiempo

  // Define el estado inicial aquí
  CALL(SubirBarrera)

  FSM_GO(LIBRE) // estado inicial
}
```

En el `setup()` definimos la función de cada uno de los pins tanto de salida como de los sensores de entrada.

Los sensores (infrarrojos, Reed, consumo,...) se definen con `PIN_SENSOR`. Estos se conectan entre el pin de entrada y GND, `simplex.h` por defecto hace que la ocupación se extienda 500ms más después de desactivarse, con `PIN_TIME` podemos variar este tiempo. En este caso al usar detectores de infrarrojos lo extendemos hasta 2 segundos para evitar que se informe como libre entre vagón y vagón.

Después ponemos el estado inicial del sistema para empezar por el estado LIBRE. Como estas acciones las hemos definido en la función *SubirBarrera* simplemente la llamamos con `CALL` y establecemos el estado inicial con `FSM_GO` a LIBRE.

```
void loop() {
  RUN_SIMPLEX // Run Simplex debe estar al principio

  FSM_STATE(LIBRE) {
    ACTIVE(DERECHA) {
      CALL(BajarBarrera)
      FSM_GO(ENTRA_DERECHA) // estado inicial
    }
    ACTIVE(IZQUIERDA) {
      CALL(BajarBarrera)
      FSM_GO(ENTRA_IZQUIERDA) // estado inicial
    }
  }

  FSM_STATE(ENTRA_DERECHA) {
    ACTIVE(IZQUIERDA) {
      FSM_GO(SALE_IZQUIERDA) // estado inicial
    }
  }

  FSM_STATE(ENTRA_IZQUIERDA) {
    ACTIVE(DERECHA) {
      FSM_GO(SALE_DERECHA) // estado inicial
    }
  }

  FSM_STATE(SALE_DERECHA) {
    FREE(DERECHA) {
      CALL(SubirBarrera)
      FSM_GO(LIBRE) // estado inicial
    }
  }

  FSM_STATE(SALE_IZQUIERDA) {
    FREE(IZQUIERDA) {
      CALL(SubirBarrera)
      FSM_GO(LIBRE) // estado inicial
    }
  }
}
```

En el `loop()` implementamos la máquina de estados, por cada estado usamos `FSM_STATE` y entre `{ }` definimos que ocurre en ese estado.

En el estado LIBRE esperamos a que se active un sensor con `ACTIVE` y entre `{ }` realizamos las acciones para pasar al siguiente estado cuando se active ese sensor. Si se activa el sensor DERECHA ejecutamos las acciones para bajar las barreras y pasamos con `FSM_GO` al estado ENTRA_DERECHA, si es el izquierdo tras bajar las barreras pasaríamos al estado ENTRA_IZQUIERDA.

En los estados ENTRA_DERECHA y ENTRA_IZQUIERDA esperamos con `ACTIVE` a que se active el sensor IZQUIERDA o DERECHA respectivamente para saber que el tren está saliendo del paso a nivel y podamos pasar al siguiente estado SALE_IZQUIERDA o SALE_DERECHA.

En estos últimos estados esperamos con **FREE** a que el sensor este libre, que será dos segundos después de pasar el último vagón ya que hemos definido con **PIN_TIME** ese tiempo de espera para evitar que se informe de libre entre vagón y vagón.

Cuando se informe de que esta libre el sensor llamamos a la función para subir la barrera y apagar la señal y pasamos de nuevo al estado LIBRE para una nueva secuencia.

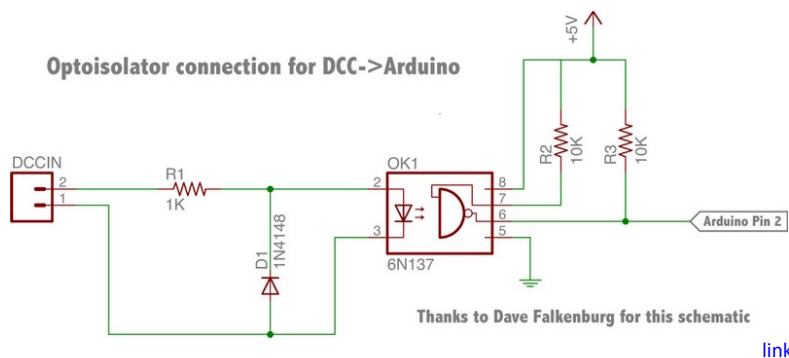
Si en lugar de detectores infrarrojos usáramos Reed para una detección efectiva no deberíamos circular a velocidades excesivamente elevadas ya que la librería **simplex.h** además de la detección realiza muchas otras tareas por lo que con pulsos muy cortos en la entrada del sensor podría no responder como esperamos.

Extra

En los ejemplos encontrarás una posible implementación de un paso a nivel con doble vía siguiendo este mismo ejemplo.

7. Decodificador de accesorios DCC

simplex.h facilita la realización de sencillos decodificadores de accesorios DCC de una forma directa y sin tratar con CV o con funciones de librerías complejas. Para ello necesitamos añadir un pequeño circuito con un opto acoplador que nos proteja de las tensiones de la señal DCC y nos la proporcione en el pin 2 del Arduino.



Para una pequeña estación secundaria haremos un decodificador para una señal de tres focos y su avanzada, otra señal de dos focos y una señal mecánica accionada por un servo:

Semaforos	Señal 1				Avanzada			Señal 2		Servo	
Accesorio	20		21					25		28	
Posicion	ROJO	VERDE	ROJO	VERDE				ROJO	VERDE	ROJO	VERDE

La relación entre la señal de entrada a la estación y su avanzada es esta:

Señal entrada estación	Señal avanzada

Como siempre definimos los nombres de los pines para cada una de las señales para facilitar la lectura del programa:

```
#include "simplex.h" // Librería de acciones simples de Paco Cañada 2025

// Define los nombres aquí
SET_NAME(ROJO_1, 3) // Nombres para los pines de la señal de tres focos
SET_NAME(VERDE_1, 4)
SET_NAME(AMARILLO_1, 5)
SET_NAME(VERDE_AV, 6) // Nombres para los pines de la señal avanzada
SET_NAME(AMARILLO_AV, 7)
SET_NAME(ROJO_2, 8) // Nombres para los pines de la señal de dos focos
SET_NAME(VERDE_2, 9)
SET_NAME(MECANICA, 10) // Nombre para el pin del servo de la señal mecánica
```

En el `setup()` Definimos el tipo de cada pin, los de las señales luminosas con `PIN_EFFECT` y efecto `DIMMER` para un encendido y apagado progresivo, el de la señal mecánica como `PIN_SERVO` y el pin de entrada DCC como `PIN_DCC` además ponemos todas las señales en aspecto rojo al iniciar el decodificador.

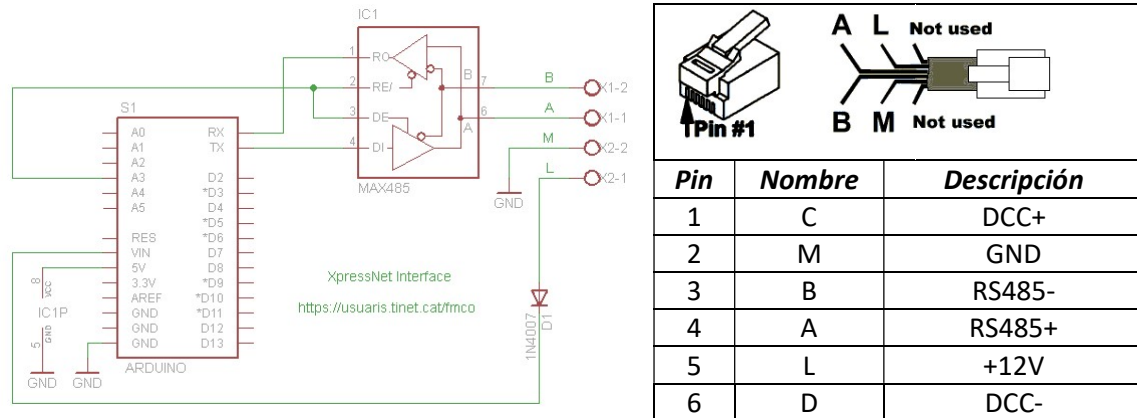
```
void setup() {  
    // Define los pines aquí  
    PIN_EFFECT(ROJO_1)           // pines de salida de luces con efecto  
    PIN_EFFECT(VERDE_1)  
    PIN_EFFECT(AMARILLO_1)  
    PIN_EFFECT(VERDE_AV)  
    PIN_EFFECT(AMARILLO_AV)  
    PIN_EFFECT(ROJO_2)  
    PIN_EFFECT(VERDE_2)  
    PIN_MODE(ROJO_1, DIMMER)      // efecto de encendido y apagado lento  
    PIN_MODE(VERDE_1, DIMMER)  
    PIN_MODE(AMARILLO_1, DIMMER)  
    PIN_MODE(VERDE_AV, DIMMER)  
    PIN_MODE(AMARILLO_AV, DIMMER)  
    PIN_MODE(ROJO_2, DIMMER)  
    PIN_MODE(VERDE_2, DIMMER)  
    PIN_SERVO(MECANICA)           // pin del servo de la señal mecánica  
    PIN_DCC()                     // pin de entrada DCC  
    // Define el estado inicial aquí  
    SET(ROJO_1)                  // señal principal y avanzada  
    SET(AMARILLO_AV)  
    SET(ROJO_2)                  // señal de dos focos  
    SERVO(MECANICA, 45)          // señal mecánica, ángulo para aspecto de parada  
}
```

En el `loop()` usamos `DCC_ACC_RED` y `DCC_ACC_GREEN` con la dirección de accesorio a la que queremos que obedezca y entre `{ }` las acciones que queremos realizar cuando se reciba esa orden DCC de movimiento de accesorio.

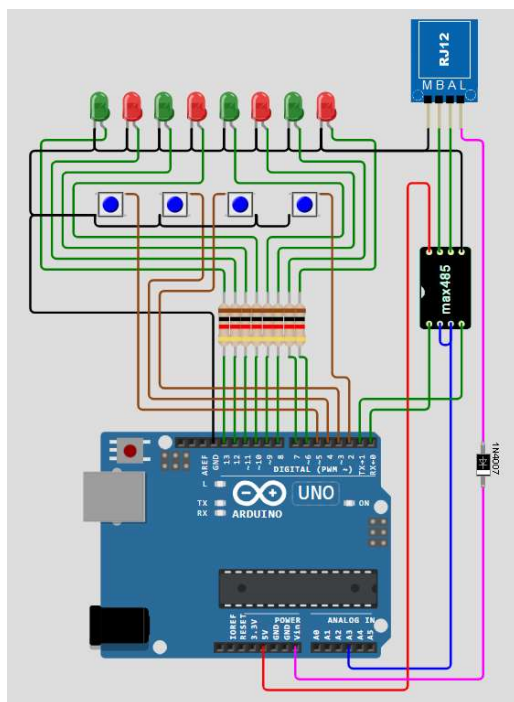
```
void loop() {  
    RUN_SIMPLEX                  // Run Simplex debe estar al principio  
  
    DCC_ACC_RED(20) {            // accesorio 20 ROJO: principal parada, avanzada anuncio parada  
        SET(ROJO_1)  
        RESET(VERDE_1)  
        RESET(AMARILLO_1)  
        RESET(VERDE_AV)  
        SET(AMARILLO_AV)  
    }  
    DCC_ACC_GREEN(20) {          // accesorio 20 VERDE: principal vía libre, avanzada vía libre  
        RESET(ROJO_1)  
        SET(VERDE_1)  
        RESET(AMARILLO_1)  
        SET(VERDE_AV)  
        RESET(AMARILLO_AV)  
    }  
    DCC_ACC_RED(21) {            // accesorio 21 ROJO: principal anuncio parada, avanzada anuncio precaución  
        RESET(ROJO_1)  
        RESET(VERDE_1)  
        SET(AMARILLO_1)  
        SET(VERDE_AV)  
        SET(AMARILLO_AV)  
    }  
    DCC_ACC_GREEN(21) {          // accesorio 21 VERDE: principal anuncio precaución, avanzada anuncio precaución  
        RESET(ROJO_1)  
        SET(VERDE_1)  
        SET(AMARILLO_1)  
        SET(VERDE_AV)  
        SET(AMARILLO_AV)  
    }  
    DCC_ACC_RED(25) {            // accesorio 25 ROJO: señal 2: parada  
        SET(ROJO_2)  
        RESET(VERDE_2)  
    }  
    DCC_ACC_GREEN(25) {          // accesorio 25 VERDE: señal 2: vía libre  
        RESET(ROJO_2)  
        SET(VERDE_2)  
    }  
    DCC_ACC_RED(28) {            // accesorio 28 ROJO: señal mecánica: parada  
        SERVO(MECANICA, 45)  
    }  
    DCC_ACC_GREEN(28) {          // accesorio 28 VERDE: señal mecánica: vía libre  
        SERVO(MECANICA, 135)  
    }  
}
```

8. Pequeño TCO Xpressnet

Con la librería **simplex.h** es bastante fácil realizar un TCO (Tablero de Control Óptico) para el bus Xpressnet. Necesitaremos un circuito con el MAX485 como interface que se conecta a TX, RX y A3:



Nuestro TCO tendrá cuatro pulsadores para mover los desvíos 1 al 4 y LEDs de color rojo y verde para cada desvío para mostrar la posición. Cuando pulsemos un botón se enviará la orden por Xpressnet para cambiar la posición del desvío. Para usar el interface Xpressnet con **simplex.h** tenemos que añadir **XNET_ADDR** en el **setup()** con la dirección del bus de nuestro TCO entre 1 y 31 que no coincida con la de otro dispositivo conectado al bus.



```
#include "simplex.h" // Librería de acciones simples de Paco Cañada 2025

// Define los nombres aquí
SET_NAME(PULSADOR1, 2)
SET_NAME(PULSADOR2, 3)
SET_NAME(PULSADOR3, 4)
SET_NAME(PULSADOR4, 5)

SET_NAME(ROJO1, 6)
SET_NAME(VERDE1, 7)
SET_NAME(ROJO2, 8)
SET_NAME(VERDE2, 9)
SET_NAME(ROJO3, 10)
SET_NAME(VERDE3, 11)
SET_NAME(ROJO4, 12)
SET_NAME(VERDE4, 13)

// Define los estados aquí

// Define las variables aquí

// Define funciones adicionales aquí

void setup() {
    // Define los pines aquí

    PIN_OUT(ROJO1)
    PIN_OUT(VERDE1)
    PIN_OUT(ROJO2)
    PIN_OUT(VERDE3)
    PIN_OUT(ROJO3)
    PIN_OUT(VERDE3)
    PIN_OUT(ROJO4)
    PIN_OUT(VERDE4)
    PIN_BUTTON(PULSADOR1)
    PIN_BUTTON(PULSADOR2)
    PIN_BUTTON(PULSADOR3)
    PIN_BUTTON(PULSADOR4)

    XNET_ADDR(5) // Dirección del dispositivo en el bus Xpressnet

    // Define el estado inicial aquí
}
```

En el `loop()` cuando se pulse un botón con `PRESSED` mandamos la orden de cambiar la posición del desvío correspondiente con `XNET_TOGGLE` y esperamos a soltar el botón con `WAIT_RELEASE` para que no vaya cambiando la posición del desvío continuamente.

Con un solo botón no sabríamos a priori la posición en la que está para encender o apagar los LED correspondientes pero la central nos informará del cambio de posición del desvío tanto si lo movemos nosotros como si se mueve desde otro mando mediante la retro señalización por lo que consultamos si ha habido cambios con `XNET_CHANGED` en el módulo correspondiente a los desvíos y comprobamos la posición de cada desvío con `XNET_ACTIVE` para encender o apagar el LED de cada posición.

```
void loop() {
  RUN_SIMPLEX           // Run Simplex debe estar al principio

  PRESSED(PULSADOR1) {
    XNET_TOGGLE(1)       // Cambia desvío numero 1
    WAIT_RELEASE(PULSADOR1)
  }
  PRESSED(PULSADOR2) {
    XNET_TOGGLE(2)       // Cambia desvío numero 2
    WAIT_RELEASE(PULSADOR2)
  }
  PRESSED(PULSADOR3) {
    XNET_TOGGLE(3)       // Cambia desvío numero 3
    WAIT_RELEASE(PULSADOR3)
  }
  PRESSED(PULSADOR4) {
    XNET_TOGGLE(4)       // Cambia desvío numero 4
    WAIT_RELEASE(PULSADOR4)
  }

  XNET_CHANGED(1) {      // Modulo RS 1, corresponde a los desvíos 1 a 4
    XNET_ACTIVE(1, 1) {   // Modulo RS 1, entradas 1 y 2 corresponden al desvío numero 1
      SET(ROJ01)
      RESET(VERDE1)
    }
    XNET_ACTIVE(1, 2) {
      RESET(ROJ01)
      SET(VERDE1)
    }
    XNET_ACTIVE(1, 3) {   // Modulo RS 1, entradas 3 y 4 corresponden al desvío numero 2
      SET(ROJ02)
      RESET(VERDE2)
    }
    XNET_ACTIVE(1, 4) {
      RESET(ROJ02)
      SET(VERDE2)
    }
    XNET_ACTIVE(1, 5) {   // Modulo RS 1, entradas 5 y 6 corresponden al desvío numero 3
      SET(ROJ03)
      RESET(VERDE3)
    }
    XNET_ACTIVE(1, 6) {
      RESET(ROJ03)
      SET(VERDE3)
    }
    XNET_ACTIVE(1, 7) {   // Modulo RS 1, entradas 7 y 8 corresponden al desvío numero 4
      SET(ROJ04)
      RESET(VERDE4)
    }
    XNET_ACTIVE(1, 8) {
      RESET(ROJ04)
      SET(VERDE4)
    }
  }
}
```

9. Motores paso a paso

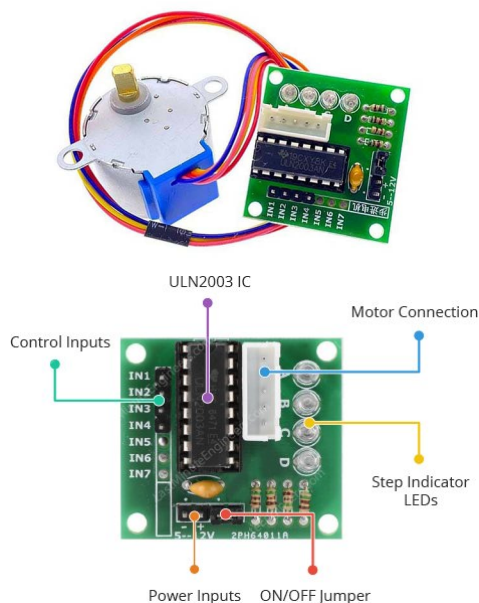
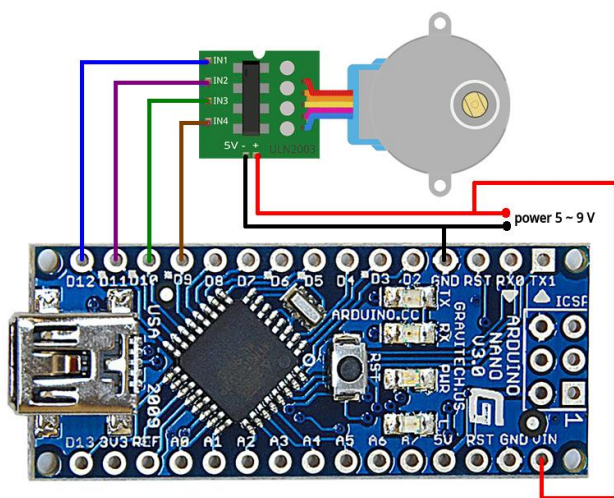
Con la librería **simplex.h** también se pueden controlar motores paso a paso unipolares tipo 28BYJ-48 o bipolar tipo NEMA con el controlador adecuado.

Para usar los unipolares necesitamos 4 pines consecutivos para controlarlos y en **PIN_STEPPER4** indicamos el primer pin, para los bipolares necesitamos 2 pines consecutivos para controlarlos y en **PIN_STEPPER2** indicamos el primer pin que es la señal STEP, el siguiente pin corresponde a la señal DIR en caso de usar un driver A4988 o similar.

Para su control con **STEPPER_SET** indicamos el paso actual (posición), por defecto es el paso 0. Luego con **STEPPER_GO** podemos mover el motor al paso especificado (posición absoluta). Si lo que queremos es movernos desde el paso actual (posición relativa) usaremos **STEPPER_CW** para movernos la cantidad de pasos indicada en sentido horario y **STEPPER_CCW** para movernos en sentido anti horario.

Con **WAIT_STEPPER** podemos esperar a que el motor paso a paso llegue a la posición indicada, la velocidad se puede cambiar con **PIN_TIME**, cuanto más bajo más rápido se moverá.

Los populares 28BYJ-48 en montajes de Arduino usan un controlador simple con un ULN2003 y se controlan con 4 cables. Son de 32 pasos y una reducción de un factor de 64 lo que da un total de 2048 pasos por vuelta, se suelen encontrar para tensiones de 5V y de 12V.



```
#include "simplex.h"           // Librería de acciones simples
de Paco Cañada 2025

SET_NAME(MOTOR, 9)           // 4 pins: 9, 10, 11, 12

void setup() {
  PIN_STEPPER4(MOTOR)
  PIN_TIME(MOTOR, 5)
  PIN_MODE(MOTOR, FULL_STEP)

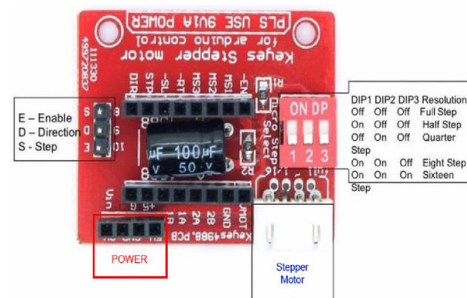
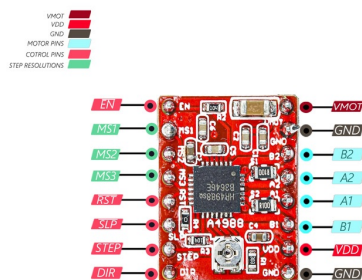
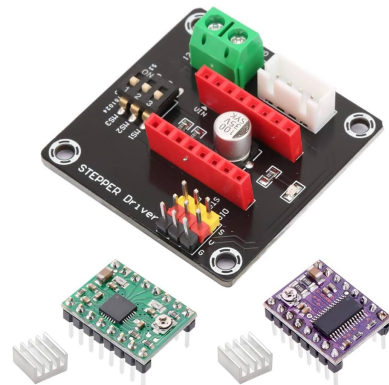
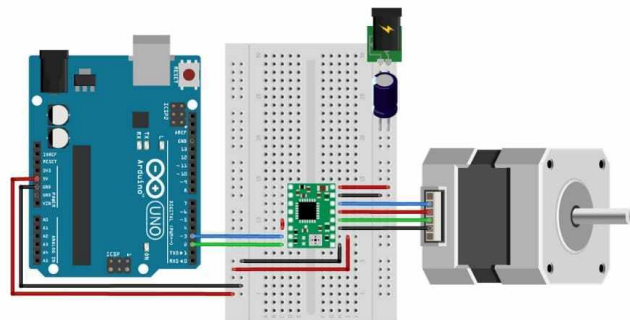
  STEPPER_CW(MOTOR, 2048)    // una vuelta completa
  WAIT_STEPPER(MOTOR)
  WAIT(100)
  STEPPER_CCW(MOTOR, 2048)
  WAIT_STEPPER(MOTOR)
}

void loop() {
  RUN_SIMPLEX
}
```

Estos motores se pueden controlar en pasos completos o también con medios pasos con lo que tendremos 4096 pasos por vuelta a costa de perder algo de fuerza.

Para usar un modo u otro usamos **PIN_MODE** con el valor **FULL_STEP** para pasos completos (por defecto) o **HALF_STEP** para medios pasos.

Los motores paso a paso bipolares tipo NEMA, normalmente utilizados en impresoras 3D, necesitan un driver para controlarlos, hay diferentes tipos A4988, DRV8825, etc. que permiten usarlos con solo dos pines, STEP para mover un paso y DIR para indicar el sentido de movimiento, según el cableado puede haber un pin ENABLE para activar o desactivar las bobinas del motor. También tienen unas entradas, normalmente conectadas a unos interruptores DIP, para seleccionar si se mueve a pasos completos, medios pasos y hasta dieciseisavos de paso. Tienen un potenciómetro para regular la corriente que les llega a las bobinas, si es demasiado baja perderá pasos y si es demasiado alta se calentará el motor. Por suerte existen diferentes placas donde pinchar estos controladores con todo lo necesario.



```
#include "simplex.h" // Librería de acciones simples
de Paco Cañada 2025

SET_NAME(MOTOR, 2) // 2 pins, STEP: 2, DIR: 3
SET_NAME(ENABLE, 4) // opcional, según driver

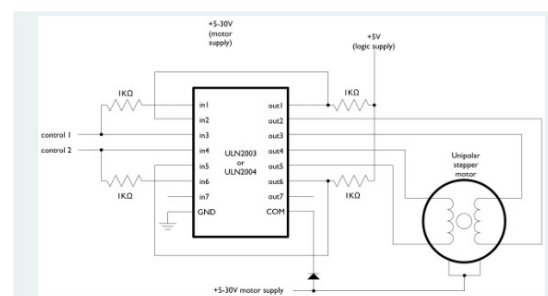
void setup() {
  PIN_OUT(ENABLE)
  PIN_STEPPER2(MOTOR)
  PIN_TIME(MOTOR, 5)
  PIN_MODE(MOTOR, DRIVER)

  RESET(ENABLE) // según driver, activar motor
  STEPPER_CW(MOTOR, 200) // una vuelta completa
  WAIT_STEPPER(MOTOR)
  WAIT(100)
  STEPPER_CCW(MOTOR, 200)
  WAIT_STEPPER(MOTOR)
  SET(ENABLE) // opcional, desactivar motor
}

void loop() {
  RUN_SIMPLEX // Run Simplex debe estar al principio
}
```

Con **STEPPER2** podemos seleccionar, dos modos de funcionamiento en **PIN_MODE**, el modo **DRIVER** (por defecto) para los drivers tipo A4988, DRV8825, etc. y el modo **FULL_STEP** para otro tipo de control:

Two Pins



NOTA

En el caso de que los cables del motor no estén en el orden correcto, el motor no girará o girará al revés, se puede probar a intercambiar la conexión de dos cables de una de las bobinas.