Proyecto 2: Manejo de dimmer con PWM

Introducción

Se programará un regulador de luz (dimmer) con una secuencia de regulación preestablecida, utilizando el kit `STM32F4DISCOVERY (STM32F407G-DISC1)` y la placa de expansión `SDBoard`. Se emplearán los LED *rojo* y *verde* de la columna derecha (columna B) de esta placa para simular las luces reguladas.

El objetivo es introducir el uso y configuración de señales PWM (modulación por ancho de pulso) y retardos temporales, a través de un ejemplo práctico. Para ello, será necesario configurar los jumpers de la `SDBoard`, los pines de la placa `Discovery`, y desarrollar el código que defina el comportamiento esperado del sistema.

El funcionamiento del dimmer se define de la siguiente manera:

1. Al iniciar, las luces *roja* y *verde* se encienden a máxima intensidad (PWM al 100 %) durante dos segundos.
2. Luego, ambos se apagan completamente (PWM al 0 %).
3. Tras un segundo de espera, La luz *roja* se enciende a máxima intensidad, mientras que la verde permanece apagada.
4. A partir de ese momento:
    a. La luz *roja* comienza a disminuir progresiva yu paulatinamente su brillo hasta apagarse.
    b. Simultáneamente, la luz *verde* incrementa su brillo hasta alcanzar la máxima intensidad.
5. Al completarse esta transición, los roles se invierten: la luz *verde* comienza a disminuir su brillo, mientras que la luz *roja* lo aumenta.
6. Este ciclo de transición cruzada se repite indefinidamente.

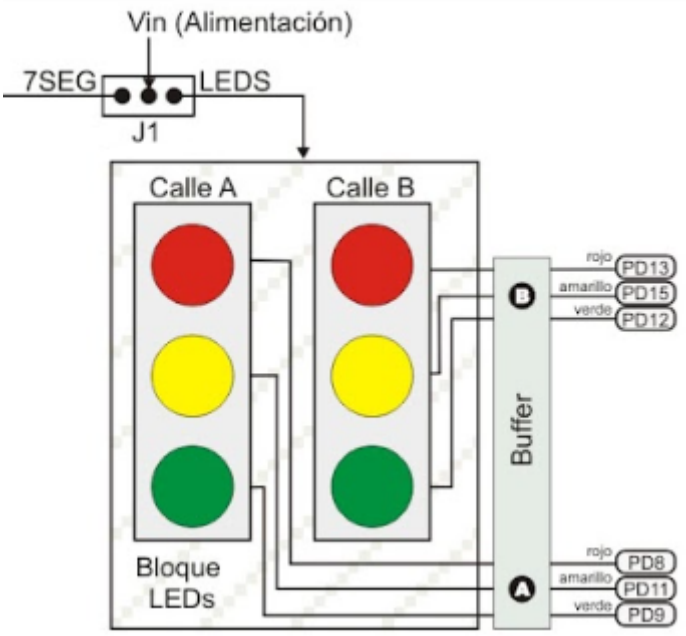
Instrucciones

- Configurar los jumpers de la placa `SDBoard` para habilitar el uso de los LED.
- Identificar y configurar adecuadamente los pines del microcontrolador de la placa `Discovery` conectados a los LED. Como así también, el temporizador asociado a los mismos, para la generación de señales PWM.
- Realizar un diagrama de flujo que represente el comportamiento del sistema.
- Escribir un algoritmo para el microcontrolador acorde al diagrama, estructurado en funciones y con los comentarios suficientes para que resulte fácil de entender y corregir/modificar.
- Realizar pruebas para verificar el comportamiento del sistema y reescribir el algoritmo de ser necesario.

Desarrollo

Configuración del hardware

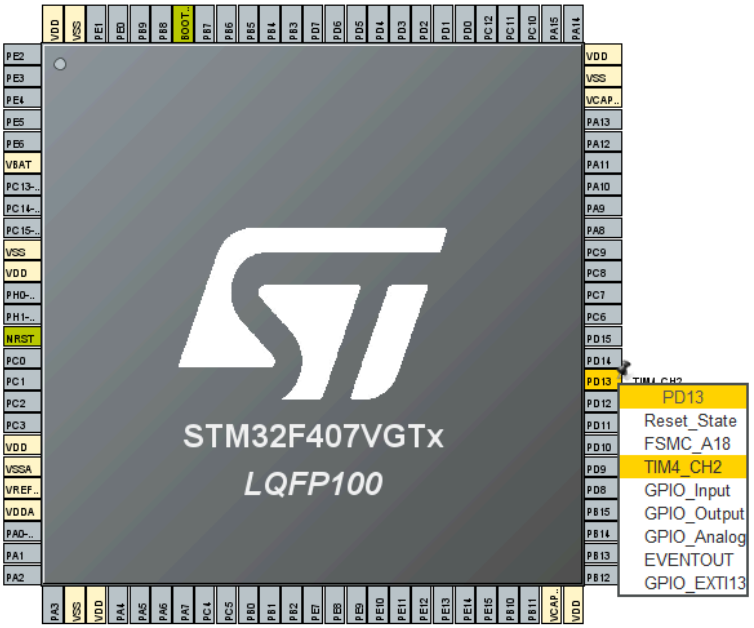
Al estudiar ambas placas se determina lo siguiente:



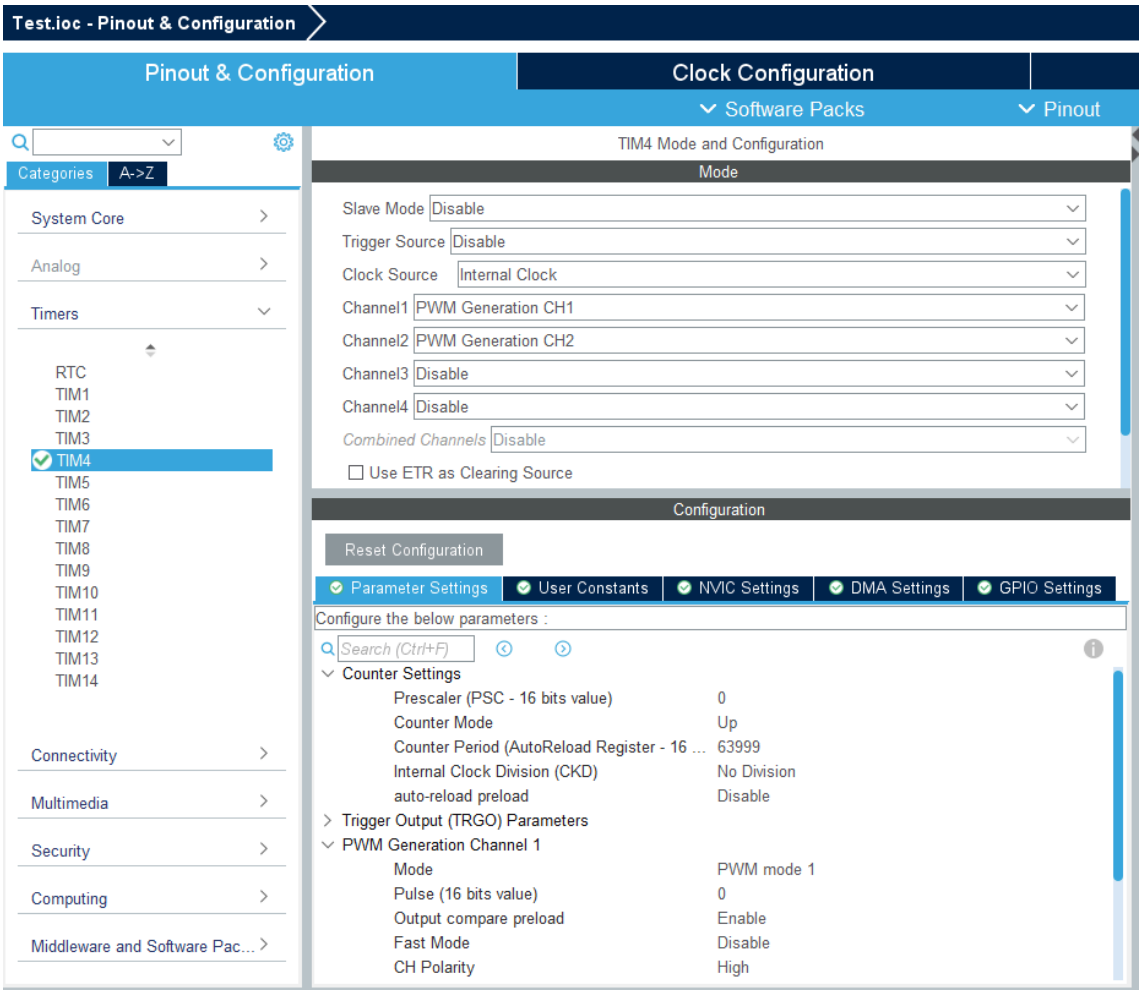
- El jumper `J1` debe conectarse en la posición `LEDS`.
- Los LED *rojo* y *verde* de la columna B son controlados respectivamente por los pines del microcontrolador correspondientes a los bits 13 y 12 del puerto D. Se encienden escribiendo un uno lógico en dichos bits y se apagan escribiendo un cero lógico. Los pines correspondientes deben asociarse a los canales de un temporizador que genere las señales PWM. En el microcontrolador de la `Discovery`, la única opción son los canales 2 y 1 del temporizador 4.

Por lo tanto, se debe asociar el pin `PD13` al canal 2 del temporizador 4 (`TIM4_CH2`), y el pin `PD12` al canal 1 del mismo (`TIM4_CH1`).

Para ello, en el `STM32CubeIDE` cree un nuevo proyecto para la placa `STM32F407G-DISC1` o para el microcontrolador `STM32F407VGT6`. Abra el archivo de configuración (el archivo con extensión .ioc), haga clic en cada uno de los pines mencionados y elija la opción correspondiente.



Luego, en el menú de la izquierda vaya a `Timers => TIM4` y realice la siguientes configuraciones:



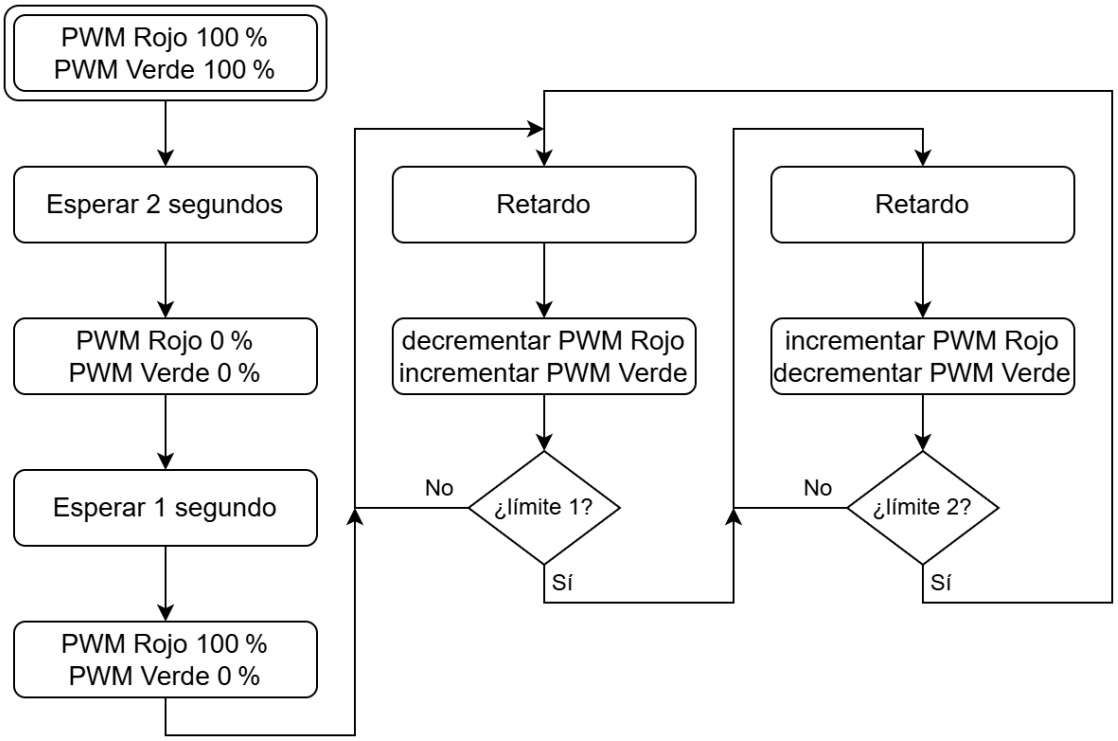
- En `Mode`:
    - Elija el reloj interno como fuente de reloj del temporizador (`Internal Clock` en `Clock Source`).
    - Para el canal 1, establezca la generación de señal PWM con salida (`PWM Generation CH1` en `Channel1`).
    - Repita para el canal 2 (`PWM Generation CH2` en `Channel2`).
- En `Configuration => Parameter Settings => Counter Settings`:
    - Establezca el valor del preescalador (`Prescaler, PSC`) en 0 para no dividir la frecuencia del reloj interno. Es decir, la frecuencia de reloj para el contador será la misma que la del reloj interno. La cual, para el temporizador 4, es `APB1 Time Clocks`[1].
    - Elija el modo ascendente para el contador (`Up` en `Counter Mode`).
    - Configure el periodo del contador (`Counter Period`) en 63 999. Esto resulta en 64.000 valores posibles de ciclo de trabajo (resolución de 0.0015625 % o 15.625 ppm) y una frecuencia[2] de 250 Hz en la señal PWM.
- En `Configuration => Parameter Settings => PWM Generation Channel 1`:
    - Elija el modo alineado a flancos, el cual es el funcionamiento típico de los PWM (`PWM mode 1` en `Mode`).
    - Establezca el ciclo de trabajo (`Pulse`) con el que iniciará la señal PWM en 0.
    - Active los cambios síncronos de los ciclos de trabajo (`Enabled` en `Output compare preload`).
    - Configure la primera parte del ciclo como la parte activa (`High` en `CH Polarity`).
- En `Configuration => Parameter Settings => PWM Generation Channel 2`:
    - Repita la configuración del punto anterior.

Diagrama de flujo

Se pueden idear diferentes algoritmos que realicen la función solicitada, como así también diferentes diagramas de flujo que modelen esos algoritmos. Un ejemplo es el siguiente.

[1] Por defecto, la frecuencia de APB1 Time Clocks es de 16 MHz. Para su configuración, vea la pestaña `Clock Configuration` del archivo .ioc.

[2] Frecuencia PWM = Frecuencia de reloj interno / ((Periodo del contador + 1) * (preescalador + 1))



Una manera de transcribir el diagrama a un algoritmo es analizando las acciones descritas en cada columna:

- Las acciones de la segunda columna representan una transición en la que la intensidad de la luz roja (verde) disminuye (aumenta) progresivamente hasta alcanzar un valor mínimo (máximo). Estas acciones pueden agruparse e implementarse en una función específica.
- De forma similar, las acciones de la tercera columna describen el proceso inverso: la intensidad de la luz roja (verde) aumenta (disminuye) hasta llegar al valor máximo (mínimo). Este comportamiento también puede encapsularse en una función aparte.
- Finalmente, las acciones de la primera columna corresponden a la fase de inicialización o prueba del sistema. Estas pueden organizarse en una función dedicada a dicha tarea.

Algoritmo

Una práctica común y recomendada en programación es utilizar y agrupar parámetros para facilitar el ajuste de valores. En este proyecto se pueden identificar cuatro parámetros relevantes:

1. El tiempo que permanecen encendidas las luces al iniciar el sistema.
2. El tiempo que permanecen apagadas antes de iniciar las transiciones.
3. El retardo entre cambios de los ciclos de trabajo de las señales PWM.
4. El valor de incremento o decremento aplicado a dichos ciclos.

Aunque los dos primeros corresponden a valores fijos en la implementación actual, es una buena práctica definirlos como parámetros, ya que podrían necesitar ajustes en el futuro o adaptarse a otras aplicaciones.

Se recomienda declarar estos parámetros en la sección `USER CODE BEGIN PD` del archivo `main.c`, lo que permitirá un acceso centralizado y organizado para futuras modificaciones.

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define TENCENDIDO 2000 // Tiempo en ms que permanecen encendidas las luces al
iniciar el sistema
#define TAPAGADO   1000 // Tiempo en ms que permanecen apagadas las luces antes
iniciar las transiciones
#define TCAMBIO      20 // Tiempo en ms entre cambios de los ciclos de trabajo de
las señales PWM
#define VALCAMBIO   640 // Valor (en LSB de resolución de PWM) para los cambios de
los ciclos de trabajo
/* USER CODE END PD */
```

Dado que la resolución configurada para las señales PWM es de 64 000, se estableció el valor de VALCAMBIO en 640 para que cada incremento (o decremento) del ciclo de trabajo represente un cambio del 1 %. Esto implica que se requieren 100 incrementos (o decrementos) para que el ciclo de trabajo pase del mínimo al máximo (o viceversa). Al definir TCAMBIO como 20 ms, se obtiene una duración total de 2 segundos para una transición completa (100 × 20 ms = 2 s).

Las funciones a utilizar pueden definirse en la sección `USER CODE BEGIN 4`.

```
/* Private user code ---------------------------------------------------------*/

/*...*/

/* USER CODE BEGIN 4 */

// Prueba inicial del sistema
void inicio(void){
 // Obtenemos el valor máximo de ciclo de trabajo de la configuración del hgardware
 uint16_t ciclomaximo = TIM4->ARR; // Valor máximo de ciclo de trabajo
 // Configuramos el PWM para que inicie con los LED encendidos
 TIM4->CCR1 = ciclomaximo; // LED verde al 100 %
 TIM4->CCR2 = ciclomaximo; // LED rojo al 100 %
 // Arrancamos el PWM
 HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1); // Inicio de la modulación PWM, LED
verde
 HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2); // Inicio de la modulación PWM, LED rojo
 // Mantenemos los LED encendidos por un tiempo
 HAL_Delay(TENCENDIDO); // Retardo de TENCENDIDO milisegundos
 // Apagamos los LED
 TIM4->CCR1 = 0; // LED verde al 0 %
 TIM4->CCR2 = 0; // LED rojo al 0 %
 // Mantenemos los LED apagados por un tiempo
 HAL_Delay(TAPAGADO); // Retardo de TAPAGADO milisegundos
 //Encendemos el LED rojo
 TIM4->CCR2 = ciclomaximo; // LED rojo al 100 %
}
```

```
// Disminuye rojo, aumenta verde
void transicion1(void){
 // Obtenemos valores de la configuración del hgardware
 uint16_t ciclomaximo = TIM4->ARR;  // Valor máximo de ciclo de trabajo
 uint16_t cicloverde  = TIM4->CCR1; // Valor actual de ciclo de trabajo en LED
verde
 uint16_t ciclorojo   = TIM4->CCR2; // Valor actual de ciclo de trabajo en LED rojo
 do{
       // Esperamos un tiempo antes de cambiar las intencidades de lso LED
   HAL_Delay(TCAMBIO); // Espera por TCAMBIO milisegndos
   // Disminuimos un poco la intensidad del LED rojo
   ciclorojo = (ciclorojo > VALCAMBIO) ? ciclorojo - VALCAMBIO : 0; // Decremento
con saturación en 0 %
   TIM4->CCR2 = ciclorojo; // Actualiza el ciclo de trabajo en LED rojo
   // Aumentaqmos un poco la intensidad del LED verde
   cicloverde = (ciclomaximo - cicloverde > VALCAMBIO) ? cicloverde + VALCAMBIO :
ciclomaximo; // Incremento con saturación en 100 %
   TIM4->CCR1 = cicloverde; // Actualiza el ciclo de trabajo en LED verde
   // Verficamos si repetimos o invertimos
 }while( ciclorojo > 0 && cicloverde < ciclomaximo ); // Repetimos mientras no se
alcance un límite
}

// Disminuye verde, aumenta rojo
void transicion2(void){
 // Obtenemos valores de la configuración del hgardware
 uint16_t ciclomaximo = TIM4->ARR;  // Valor máximo de ciclo de trabajo
 uint16_t cicloverde  = TIM4->CCR1; // Valor actual de ciclo de trabajo en LED
verde
 uint16_t ciclorojo   = TIM4->CCR2; // Valor actual de ciclo de trabajo en LED rojo
 do{
       // Esperamos un tiempo antes de cambiar las intencidades de lso LED
   HAL_Delay(TCAMBIO); // Espera por TCAMBIO milisegundos
   // Disminuimos un poco la intensidad del LED verde
   cicloverde = (cicloverde > VALCAMBIO) ? cicloverde - VALCAMBIO : 0; //
Decremento con saturación en 0 %
   TIM4->CCR1 = cicloverde; // Actualiza el ciclo de trabajo en LED rojo
   // Aumentaqmos un poco la intensidad del LED verde
   ciclorojo = (ciclomaximo - ciclorojo > VALCAMBIO) ? ciclorojo + VALCAMBIO :
ciclomaximo; // Incremento con saturación en 100 %
   TIM4->CCR2 = ciclorojo; // Actualiza el ciclo de trabajo en LED rojo
   // Verficamos si repetimos o invertimos
 }while( cicloverde > 0 && ciclorojo < ciclomaximo ); // Repetimos mientras no se
alcance un límite
}

/* USER CODE END 4 */
```

Acto seguido, en la sección `USER CODE BEGIN PFP`, declare los prototipos de las funciones definidas.

```
/* Private function prototypes -----------------------------------------------*/
/*...*/
/* USER CODE BEGIN PFP */
void inicio(void); // Prueba inicial del sistema
void transicion1(void); // Disminuye rojo, aumenta verde
void transicion2(void); // Disminuye verde, aumenta rojo
/* USER CODE END PFP */
```

Ahora sólo resta armar el código principal. Primero, en la sección `USER CODE BEGIN 2`, escriba las acciones que lleven el sistema al estado inicial.

```
/* USER CODE BEGIN 2 */
// Nos aseguramos de apagar las luces amarilla y verde, y de encender la roja
inicio(); // Prueba inicial del sistema
/* USER CODE END 2 */
```

Por último, escriba la rutina de ejecución permanente que realiza el sistema.

```
/* Infinite loop */
/*...*/
while (1)
/*...*/
  /* USER CODE BEGIN 3 */
   transicion1(); // Progresiva y gradualmente disminuye rojo, aumenta verde
   transicion2(); // Progresiva y gradualmente disminuye verde, aumenta rojo
}
/* USER CODE END 3 */
```

Verificación

Verifique el correcto funcionamiento del sistema en el hardware. De ser necesario, realice una depuración del sistema, ajuste las constantes de tiempo, reescriba el algoritmo hasta verificar que el comportamiento sea el deseado. Pruebe el algoritmo con distintos valores `TCAMBIO` y `VALCAMBIO` y observe cómo cambian las transiciones.

Desafío

Implemente un sistema que controle de manera progresiva y paulatina la intensidad de los LED de la columna B de la `SDBoard`, siguiendo la secuencia detallada a continuación:

1. Inicie con el LED *amarillo* encendido a máxima intensidad.
2. Aumente la intensidad del LED *rojo* hasta alcanzar su valor máximo.
3. Disminuya la intensidad del LED *amarillo* hasta alcanzar su valor mínimo.
4. Aumente la intensidad del LED *verde* hasta alcanzar su valor máximo.
5. Disminuya la intensidad del LED *rojo* hasta alcanzar su valor mínimo.
6. Aumente la intensidad del LED *amarillo* hasta alcanzar su valor máximo.
7. Disminuya la intensidad del LED *verde* hasta alcanzar su valor mínimo.
8. Repita indefinidamente los pasos del 2 al 7.