Proyecto 4: LED intermitentes con temporizador

Introducción

En este proyecto se programará un sistema de luces intermitentes utilizando el kit `STM32F4DISCOVERY (STM32F407G-DISC1)` junto con la placa de expansión `SDBoard`. Las señales luminosas serán visualizadas a través de los LED integrados en la `SDBoard`.

El objetivo principal es familiarizarse con el uso de `entradas/salidas de propósito general (GPIO)`, la gestión de interrupciones de periféricos y la implementación de retardos temporales, mediante el desarrollo de un caso práctico. Para ello, será necesario realizar una configuración adecuada de los jumpers de la `SDBoard`, los pines de la placa `Discovery` y desarrollar el código que defina el comportamiento deseado del sistema.

El funcionamiento del sistema será el siguiente:

1. Al encender el sistema, todos los LED se encenderán durante dos segundos.
2. A continuación, todos los LED se apagan completamente.
3. Después de una pausa de dos segundos, se iniciarán los siguientes procesos:
    a. Los LED de la columna B comenzarán a conmutar su estado (encendido/apagado) a las siguientes frecuencias, sincronizadas mediantes un temporizador:
        - *rojo* : 2 Hz.
        - *amarillo* : 5 Hz.
        - *verde* : 10 Hz.
    b. El LED *rojo* de la columna A conmutará su estado cada 1 segundo, utilizando retardos bloqueantes (HAL_Delay) desde el programa principal.

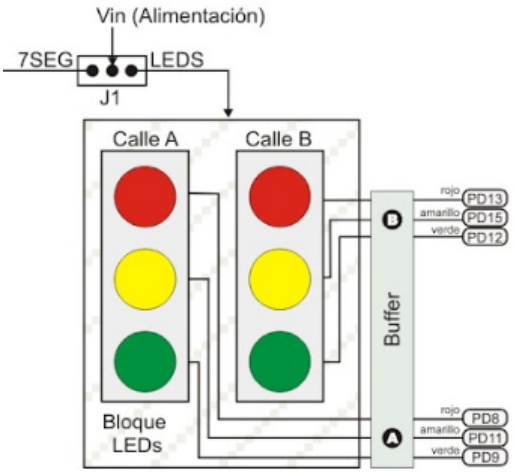
Instrucciones

- 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.
- 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 de jumpers y pines

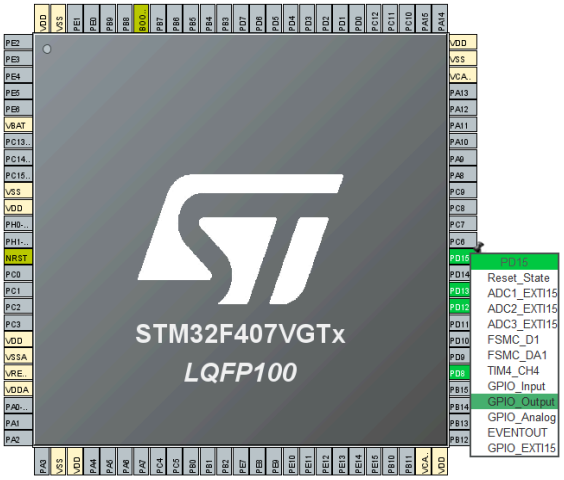
Al analizar ambas placas, se determina lo siguiente:



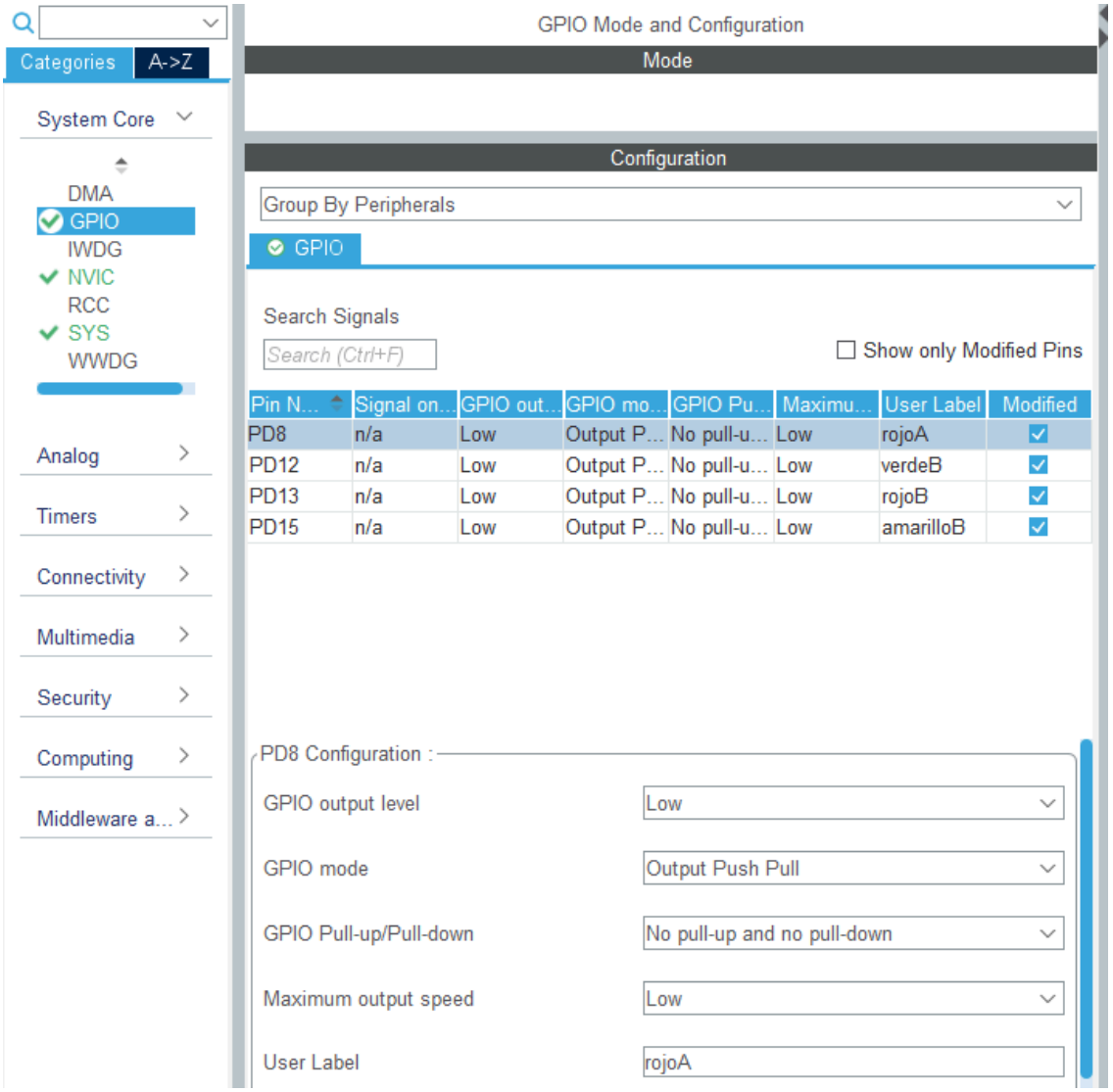
- El jumper `J1` debe conectarse en la posición `LEDS`.
- El LED rojo de la columna A, y los LED rojo, amarillo y verde de la columna B están controlados respectivamente por los pines del microcontrolador correspondientes a los bits 8, 13, 15 y 12 del puerto D (`PD8`, `PD13`, `PD15` y `PD12`). Estos LED se encienden al escribir un 1 lógico en los respectivos bits y se apagan al escribir un 0 lógico. Los pines mencionados deben configurarse como salidas en modo push-pull, sin resistencias de pull-up ni pull-down.

Por lo tanto, es necesario configurar los pines `PD8`, `PD12`, `PD13` y `PD15` como salidas digitales (`GPIO_Output`) en modo `push-pull`, asegurándose que no tengan resistencias de pull-up o pull-down activadas.

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 `System Config => GPIO` y en el listado de GPIO seleccione y configure uno a uno el modo de cada pin.



NOTAS:

- ☐ Dado que no existen requerimientos específicos sobre los tiempos de subida y bajada (slew rate) de las señales de salida,es recomendable configurar el parámetro `Maximum output speed` en `Low`. Esto ayuda a reducir el consumo de energía y a minimizar la emisión de interferencias electromagnéticas (EMI).
- ☐ El parámetro `GPIO output level` define el nivel lógico que tendrá el pin de salida inmediatamente después de un reinicio del microcontrolador.

Opcionalmente, en la misma sección se puede asignar una etiqueta personalizada (`User Label`) a cada pin. La herramienta generará automáticamente una directiva `#define` en el archivo `main.h`, asociando la etiqueta al nombre del pin correspondiente. Esto permite escribir el código de manera más clara y rápida, facilitando tanto su desarrollo como su mantenimiento. Por ejemplo, dado los siguientes `#define`

```
/* Private defines -----------------------------------------------------------*/
#define rojoA_Pin GPIO_PIN_8
#define rojoA_GPIO_Port GPIOD
#define verdeB_Pin GPIO_PIN_12
#define verdeB_GPIO_Port GPIOD
#define rojoB_Pin GPIO_PIN_13
#define rojoB_GPIO_Port GPIOD
#define amarilloB_Pin GPIO_PIN_15
#define amarilloB_GPIO_Port GPIOD
```

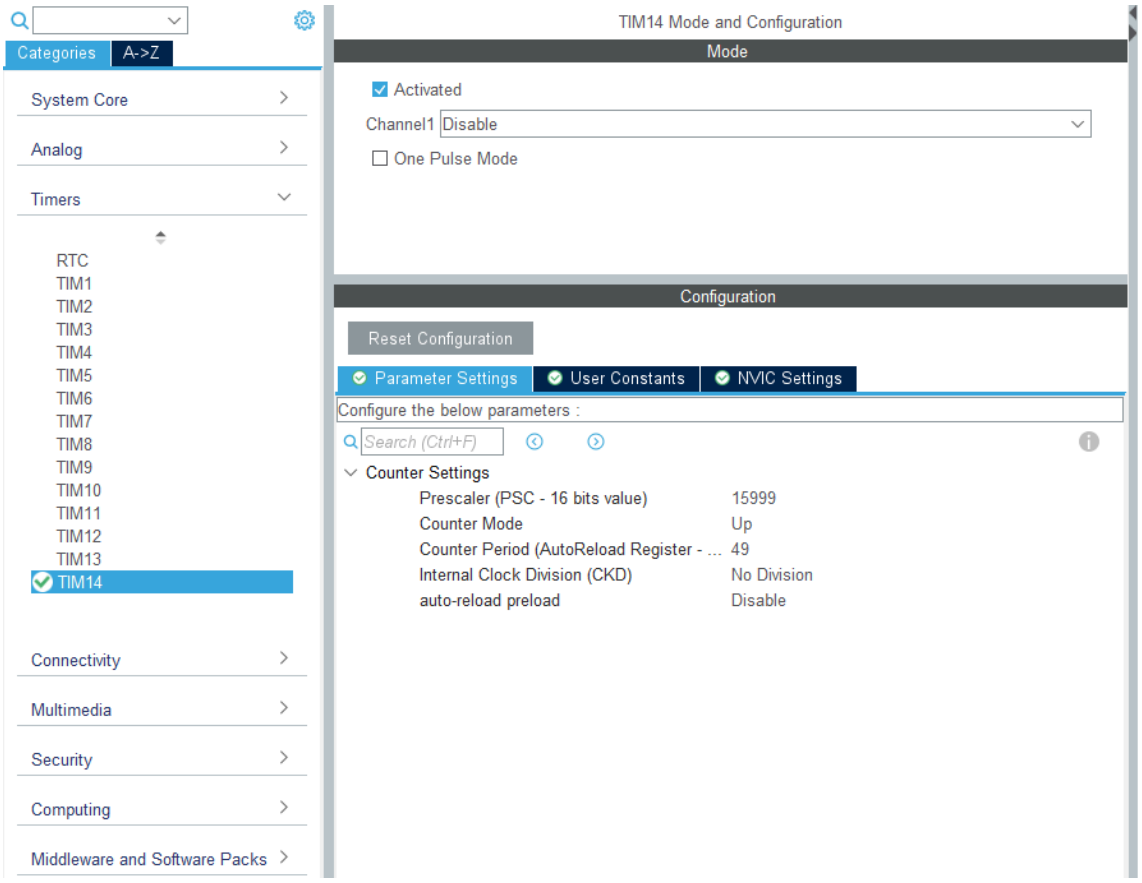
para encender el LED rojo de la columna A se puede escribir

```
HAL_GPIO_WritePin(rojoA_GPIO_Port, rojoA_Pin, GPIO_PIN_SET);
```

en lugar de

```
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, GPIO_PIN_SET);
```

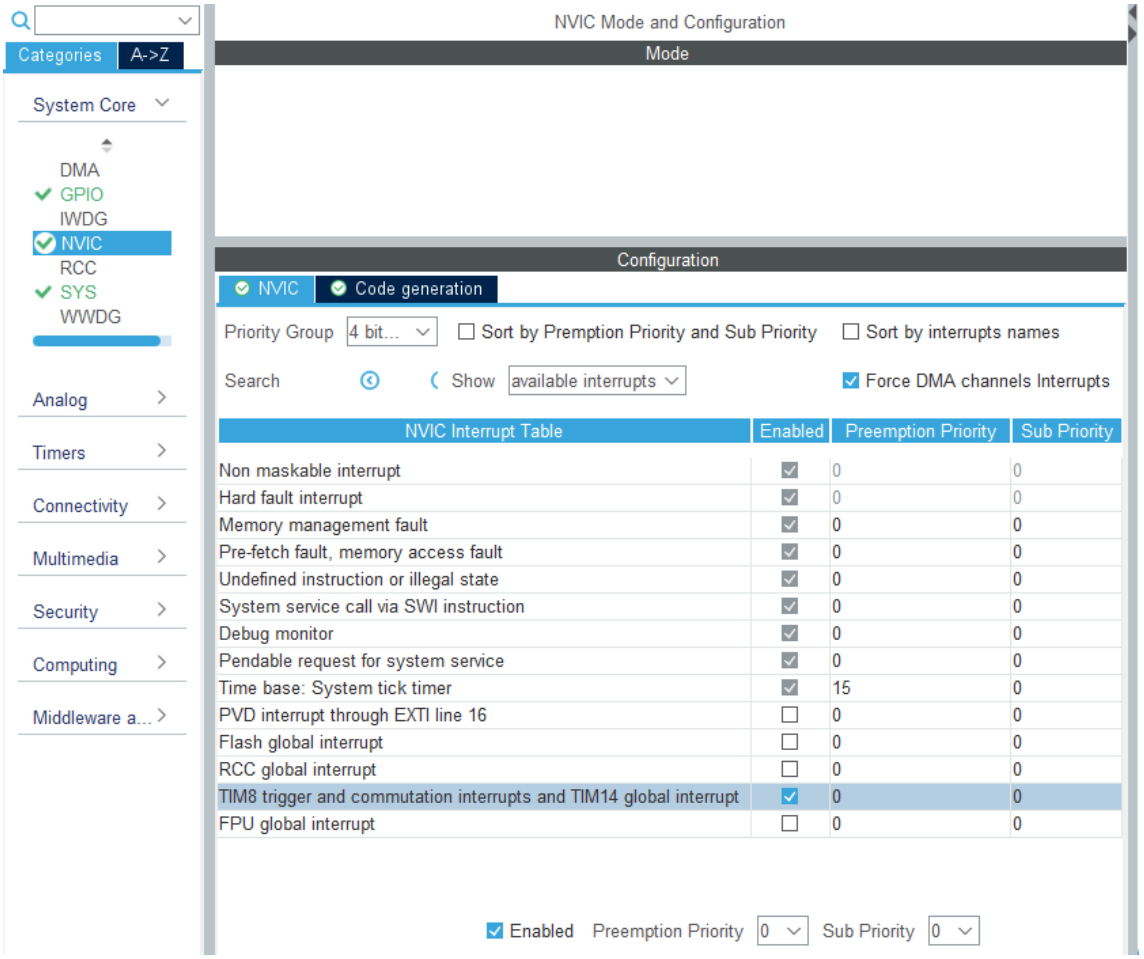
Luego, en el menú de la izquierda vaya a `Timers => TIM14` y realice las siguientes configuraciones:



- Habilite el temporizador marcando la casilla de activación (`Activated`).
- En `Configuration => Parameter Settings => Counter Settings`:

- Establezca el valor del preescalador (`Prescaler, PSC`) en 15999 para dividir la frecuencia del reloj interno llevándola a 1 kHz. Es decir, la frecuencia de reloj para el contador será 16.000 veces menor que la del reloj interno. La cual, para el temporizador 14, es `APB1 Time Clocks`[1].
- Elija el modo ascendente para el contador (`Up` en `Counter Mode`).
- Configure el periodo del contador (`Counter Period`) en 49. Esto hará que el temporizador genere una interrupción cada 50 ciclos, es decir, cada 50 milisegundos para la frecuencia configurada.

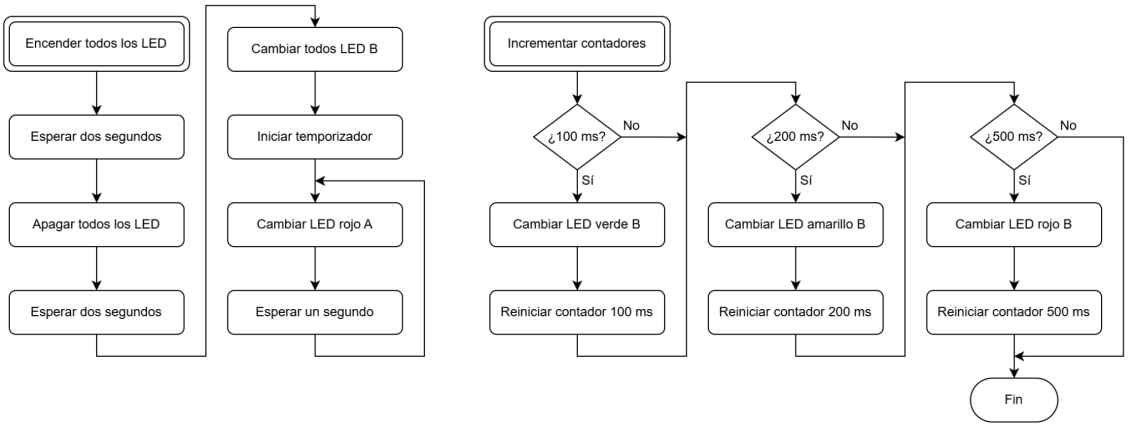
Para finalizar con la configuración resta habilitar la interrupción del temporizador y, opcionalmente, asignarle algún valor de prioridad. En el menú de la izquierda, vaya a `System Config => NVIC` y habilite la interrupción del temporizador 14 (`TIM8 trigger and commutation interrupts and TIM14 global interrupt`).



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

Diagrama de flujo

Existen diversas formas de implementar la funcionalidad solicitada, tanto a nivel de algoritmos como de diagramas de flujo que los representen. A continuación se presenta un ejemplo: el diagrama de la izquierda corresponde al programa principal, mientras que el diagrama de la derecha representa la rutina de atención a la interrupción del temporizador.



Una forma de transcribir el diagrama a un algoritmo consiste en analizar y agrupar las acciones descritas de la siguiente manera:

- En el diagrama de la izquierda:
    - Las acciones ubicadas en la primera columna, junto con las dos primeras acciones de la segunda columna, corresponden a la fase de prueba e inicialización del sistema. Estas pueden integrarse en una función específica encargada de llevar a cabo dicha tarea.
    - Las últimas acciones de la segunda columna implementan la intermitencia del LED rojo A. Si bien podrían agruparse en una función dedicada, dado que se trata únicamente de dos instrucciones de alto nivel, pueden escribirse directamente dentro del bucle principal del programa.
- En el diagrama de la derecha:
    - Las acciones relacionadas con el cambio de estado de los LED, la verificación del valor de los contadores y su reinicio pueden agruparse en una función que se ejecute dentro de la rutina de atención a la interrupción del temporizador.

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 seis parámetros relevantes:

1. El tiempo que permanecen encendidas las luces al iniciar el sistema.

2. El tiempo que permanecen apagadas antes de iniciar las intermitencias.
3. El tiempo entre conmutaciones del LED rojo A.
4. Cantidad de interrupciones para conmutar el LED rojo B.
5. Cantidad de interrupciones para conmutar el LED amarillo B.
6. Cantidad de interrupciones para conmutar el LED verde B.

Aunque todos 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 los parámetros en la sección `USER CODE BEGIN PD` del archivo `main.c`. Esto permite un acceso centralizado y organizado, facilitando futuras modificaciones.

```
/* Private typedef -----------------------------------------------------------*/
/*...*/
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
// Tiempo en ms que permanecen encendidas las luces al iniciar el sistema
#define TENCENDIDO 2000
// Tiempo en ms que permanecen apagadas las luces antes iniciar las transiciones
#define TAPAGADO   2000
// Tiempo en ms que en el que los rebotes son significativos
#define TROJOA     1000
// Cantidad de interrupciones que hay en 500 ms ( 2 Hz) (500 ms / 50 ms = 10)
#define CROJOB       10
// Cantidad de interrupciones que hay en 200 ms ( 5 Hz) (200 ms / 50 ms =  4)
#define CAMARILLOB    4
// Cantidad de interrupciones que hay en 100 ms (10 Hz) (100 ms / 50 ms =  2)
#define CVERDEB       2
/* USER CODE END PD */
```

Una forma de compartir datos entre funciones es mediante el uso de variables globales, especialmente cuando las funciones no son invocadas directamente por el programa principal (por ejemplo, las rutinas de atención a interrupciones). Se recomienda definir estas variables en la sección `USER CODE BEGIN PV`.

```
/* Private variables ---------------------------------------------------------*/
/*...*/
/* USER CODE BEGIN PV */
/* USER CODE BEGIN PV */
// Contador de interrupciones para conmutar LED rojo B
volatile uint8_t cntRojoB     = 0;
// Contador de interrupciones para conmutar LED amarillo B
volatile uint8_t cntAmarilloB = 0;
// Contador de interrupciones para conmutar LED verde B
volatile uint8_t cntVerdeB    = 0;
/* USER CODE END PV */
```

Las funciones a utilizar pueden definirse en la sección `USER CODE BEGIN 4`. Para facilitar la comprensión del algoritmo, se implementarán funciones auxiliares para encender todos los LED y apagar todos los LED. Es importante notar que la rutina de atención a interrupciones de temporizadores está declarada por la librería HAL como `void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)`, el programador

sólo debe definir el cuerpo de esta función para manejar la interrupción correspondiente.

```
/* Private user code ---------------------------------------------------------*/
/*...*/
/* USER CODE BEGIN 4 */
// Prueba e inicialización del sistema
void inicio(void){
 // Encendemos todos los LED y esperamos un tiempo
 encenderTodos();
 HAL_Delay(TENCENDIDO);
 // Apagamos todos los LED y esperamos un tiempo
 apagarTodos();
 HAL_Delay(TAPAGADO);
 // Hacemos el primer cambio de estado de los LED de la columna B
 HAL_GPIO_TogglePin(rojoB_GPIO_Port, rojoB_Pin);
 HAL_GPIO_TogglePin(amarilloB_GPIO_Port, amarilloB_Pin);
 HAL_GPIO_TogglePin(verdeB_GPIO_Port, verdeB_Pin);
 // Iniciamos el temporizador
 HAL_TIM_Base_Start_IT(&htim14);
}

// Apaga todos los LED
void apagarTodos(void){
 HAL_GPIO_WritePin(    rojoA_GPIO_Port,     rojoA_Pin, GPIO_PIN_RESET);
 HAL_GPIO_WritePin(   verdeB_GPIO_Port,    verdeB_Pin, GPIO_PIN_RESET);
 HAL_GPIO_WritePin(amarilloB_GPIO_Port, amarilloB_Pin, GPIO_PIN_RESET);
 HAL_GPIO_WritePin(    rojoB_GPIO_Port,     rojoB_Pin, GPIO_PIN_RESET);
}

// Enciende todos los LED
void encenderTodos(void){
 HAL_GPIO_WritePin(    rojoA_GPIO_Port,     rojoA_Pin, GPIO_PIN_SET);
 HAL_GPIO_WritePin(   verdeB_GPIO_Port,    verdeB_Pin, GPIO_PIN_SET);
 HAL_GPIO_WritePin(amarilloB_GPIO_Port, amarilloB_Pin, GPIO_PIN_SET);
 HAL_GPIO_WritePin(    rojoB_GPIO_Port,     rojoB_Pin, GPIO_PIN_SET);
}

// Comprueba los contadores, los reinicia y cambia el estado de los LED
void comprobarConmutar(void) {
 if (CVERDEB <= cntVerdeB){
  HAL_GPIO_TogglePin(verdeB_GPIO_Port, verdeB_Pin);
  cntVerdeB = 0;
 }
 if (CAMARILLOB <= cntAmarilloB){
  HAL_GPIO_TogglePin(amarilloB_GPIO_Port, amarilloB_Pin);
  cntAmarilloB = 0;
 }
 if (CROJOB <= cntRojoB){
  HAL_GPIO_TogglePin(rojoB_GPIO_Port, rojoB_Pin);
  cntRojoB = 0;
 }
}

// Rutina de atención a la interrupción del temporizador 14
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
 if (htim->Instance == TIM14) {
  cntVerdeB++;
  cntAmarilloB++;
  cntRojoB++;
  comprobarConmutar();
 }
}
/* 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 e inicialización del sistema
void apagarTodos(void); // Apaga todos los LED
void encenderTodos(void); // Enciende todos los LED
void comprobarConmutar(void); // Comprueba los contadores y conmuta los LED
/* 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 */
inicio(); // Prueba e inicialización 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 */
 // Conmutamos el estado del LED rojo A y esperamos un tiempo
 HAL_GPIO_TogglePin(rojoA_GPIO_Port, rojoA_Pin);
 HAL_Delay(TROJOA);
}
 /* 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.

Desafío

Repita el procedimiento previamente realizado para volver a implementar el sistema, esta vez incorporando la siguiente ampliación:

- El programa principal deberá conmutar el estado del LED verde de la columna A cada 15 segundos, intervalo que será indicado por el temporizador.