Proyecto 0 - Semáforo de puerta con GPIO

Introducción

En este proyecto se programará un semáforo con transición de estado bajo demanda, utilizando el kit `STM32F4DISCOVERY (STM32F407G-DISC1)` y la placa de expansión `SDBoard`. La columna izquierda de LEDs de esta última (columna A) se utilizará para simular las luces del semáforo, mientras que el pulsador `SW3` de la misma permitirá comandar el cambio de estado.

El objetivo de este proyecto es aprender a trabajar con `entradas/salidas de propósito general (GPIO)`, implementar retardos y aplicar técnicas de filtrado de rebotes, todo mediante la programación de un sencillo ejemplo. 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 semáforo se define de la siguiente manera:

1. Al iniciar, se enciende únicamente la luz *roja* y se espera la orden de transición.
2. Al recibir el comando, se encienden las luces *roja y amarilla* durante dos segundos como advertencia del cambio de estado.
3. Transcurrido este tiempo, se apagan las luces roja y amarilla, se enciende la *verde* y se espera una nueva orden de cambio.
4. Tras recibirla, se apaga la luz verde y se enciende la luz *amarilla* durante dos segundos para advertir la próxima transición.
5. Pasado este periodo, se apaga la luz amarilla, se enciende nuevamente la luz *roja*, y el sistema regresa al estado inicial, quedando a la espera de una nueva solicitud.

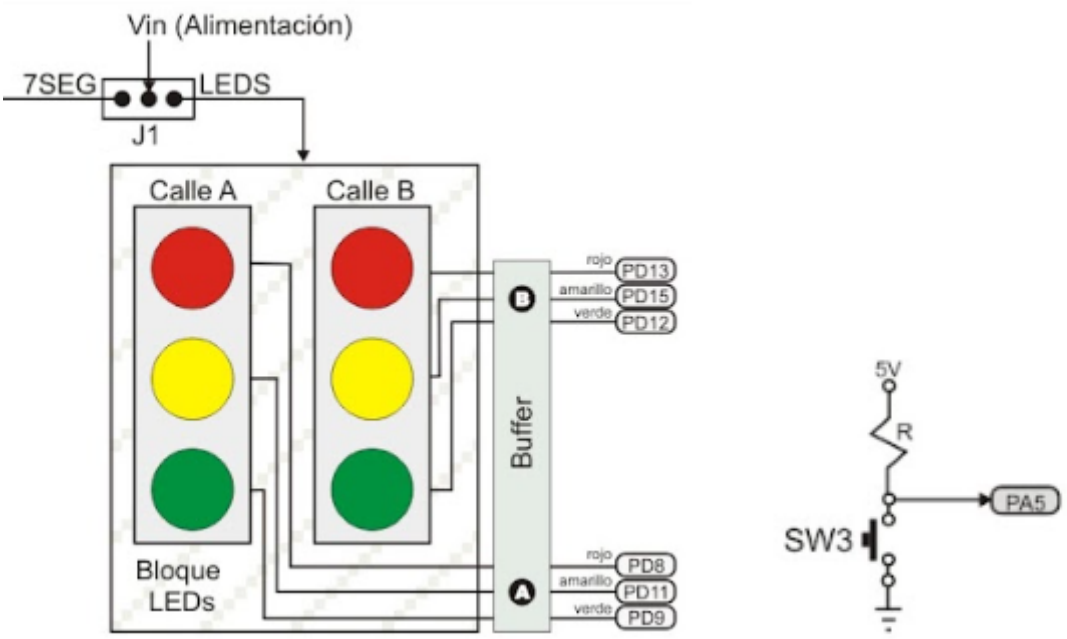
Instrucciones

- Configurar los jumpers de la placa `SDBoard` para habilitar el uso de los LEDs.
- Identificar y configurar adecuadamente los pines del microcontrolador de la placa `Discovery` conectados a los LEDs y al botón `SW3`.
- 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

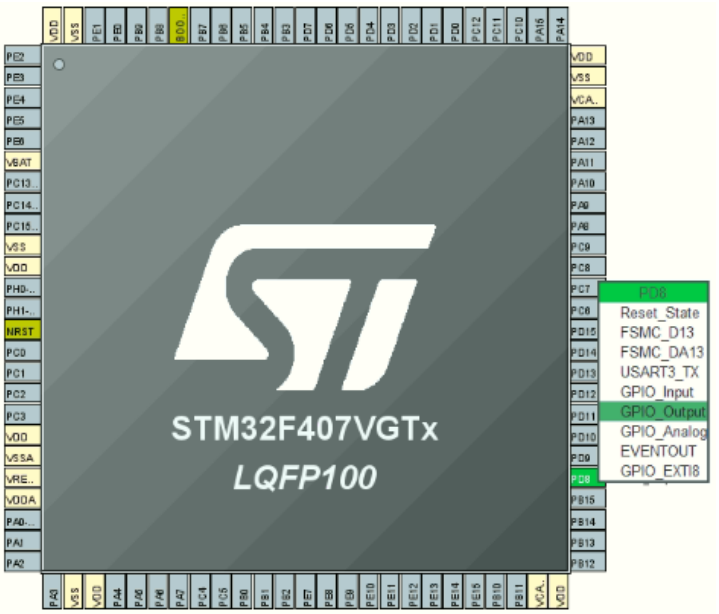
Al estudiar ambas placas se determina lo siguiente:



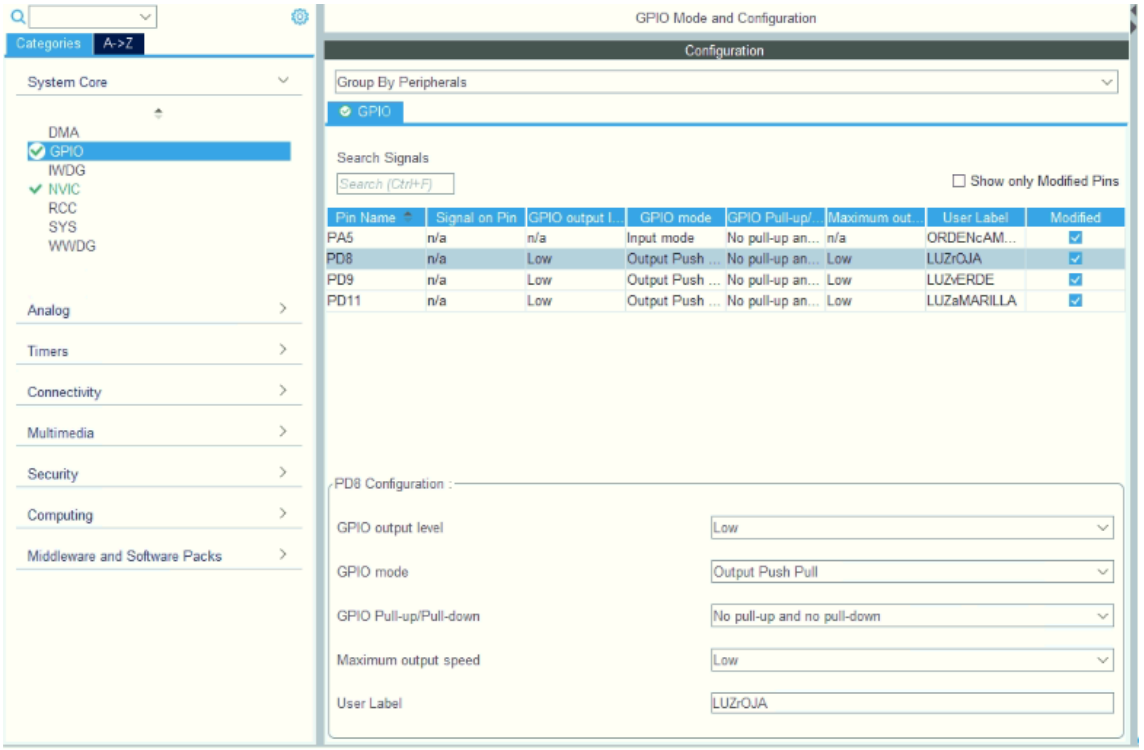
- El jumper `J1` debe conectarse en la posición `LEDS`.
- Los LED rojo, amarillo y verde de la columna A son controlados respectivamente por los pines del microcontrolador correspondientes a los bits 8, 11 y 9 del puerto D. Se encienden escribiendo un uno lógico en dichos bits y se apagan escribiendo un cero lógico. Los pines correspondientes deben configurarse en el modo push-pull y sin resistencias de pull-up o pull-down.
- El botón `SW3` está conectado al bit 5 del puerto A del microcontrolador. El pin correspondiente no requiere una resistencia de pull-up ni pull-down, ya que recibe un uno lógico mientras el botón no está presionado. No se ha implementado un circuito para el filtrado de los rebotes.

Por lo tanto, se deben configurar como entrada (`GPIO_Input`) el pin `PA5`, y como salida (`GPIO_Output`) en modo `push-pull` los pines `PD8`, `PD9` y `PD11`; todos ellos sin resistencias de pull-up o pull-down.

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 BOTONCAMBIO_Pin GPIO_PIN_5
#define BOTONCAMBIO_GPIO_Port GPIOA
#define LUZROJA_Pin GPIO_PIN_8
#define LUZROJA_GPIO_Port GPIOD
#define LUZVERDE_Pin GPIO_PIN_9
#define LUZVERDE_GPIO_Port GPIOD
#define LUZAMARILLA_Pin GPIO_PIN_11
#define LUZAMARILLA_GPIO_Port GPIOD
```

para encender el LED rojo se puede escribir

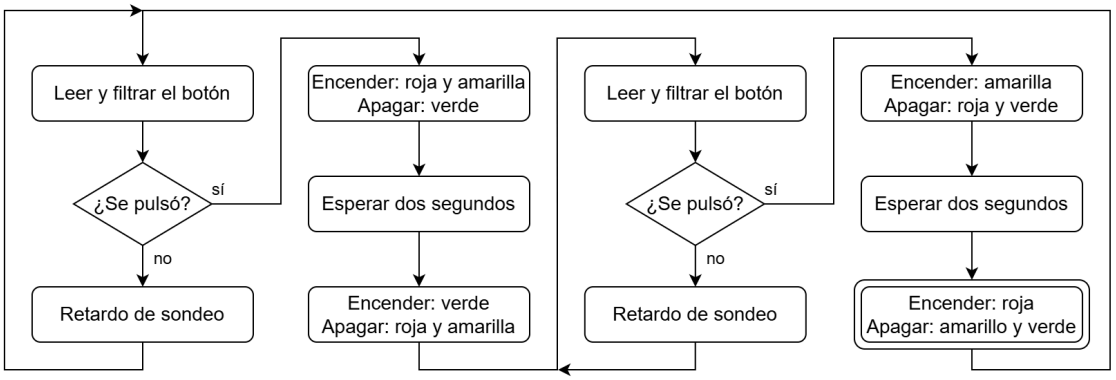
```
HAL_GPIO_WritePin(LUZROJA_GPIO_Port, LUZROJA_Pin, GPIO_PIN_SET);
```

en lugar de

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

Diagrama de flujo

Se pueden idear diferentes algoritmos que realicen la función solicitada, así como también diferentes diagramas de flujo que modelen esos algoritmos. Un ejemplo es el siguiente, donde el estado del botón `SW3` se lee por sondeo (pulling).



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

- Las acciones de la segunda columna representan la transición del estado con luz verde al estado con luz roja. Estas pueden agruparse y definirse en una función específica.
- De manera similar, las acciones de la cuarta columna generan el cambio inverso, es decir, de luz roja a luz verde, y también pueden implementarse en una función aparte.

- Por su parte, las acciones de la primera y tercera columna corresponden a la espera por una solicitud de cambio de estado. Se puede escribir una función que agrupe la lectura y filtrado, con la acción de determinar si hubo una solicitud de cambio de estado.

Aunque existen diversos algoritmos para realizar el filtrado por software los rebotes, aquí se presenta uno básico pero efectivo a los fines de este proyecto. El algoritmo consiste en leer el nivel lógico del pin conectado al botón, esperar un intervalo de tiempo suficiente para que los rebotes dejen de ser significativos (usualmente algunas decenas de milisegundos), y luego realizar una segunda lectura del mismo pin. Si ambas lecturas coinciden, se considera que el botón está en un estado estable: liberado si el nivel lógico es alto (debido a la resistencia de pull-up en el botón `SW3`), o presionado si el nivel es bajo.

Para determinar si hubo una solicitud de cambio, basta con comprobar en dos lecturas válidas y consecutivas (aunque no necesariamente contiguas) que el estado del botón haya cambiado de liberado a presionado.

Algoritmo

Una práctica usual y recomendada en programación, es usar y agrupar parámetros para facilitar el ajuste de valores. En este proyecto se pueden identificar tres, el periodo de espera entre las transiciones de estado, el retardo de sondeo y el intervalo de rebotes. Defina estos parámetros en la sección `USER CODE BEGIN PD` del archivo `main.c`.

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define TESPERA  2000 // periodo de espera en ms entre las transiciones
#define TREBOTES   20 // intervalo de rebotes en ms
#define TSONDEO   200 // retardo de sondeo en ms
/* USER CODE END PD */
```

Dado que una variable booleana es adecuada para indicar si hubo una solicitud de transición, incluya la librería `stdbool.h` en la sección `USER CODE BEGIN Includes`.

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
/*...*/
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdbool.h> // variable booleanas
/* USER CODE END Includes */
```

Luego, en la sección `USER CODE BEGIN 4`, defina las funciones a utilizar.

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

```c
/* USER CODE BEGIN 4 */
void CambiarRojoVerde(void){ // Transición de rojo a verde
 //La luz roja se encuentra encendia. Encendemos la amarilla.
 HAL_GPIO_WritePin(LUZAMARILLA_GPIO_Port, LUZAMARILLA_Pin, GPIO_PIN_SET);
 // Esperamos un timepo
 HAL_Delay(TESPERA); // Retardo de TESPERA milisegundos
 // Apagamos las luces roja y amarilla, y encendemos la verde
 HAL_GPIO_WritePin(LUZROJA_GPIO_Port, LUZROJA_Pin, GPIO_PIN_RESET);
 HAL_GPIO_WritePin(LUZAMARILLA_GPIO_Port, LUZAMARILLA_Pin, GPIO_PIN_RESET);
 HAL_GPIO_WritePin(LUZVERDE_GPIO_Port, LUZVERDE_Pin, GPIO_PIN_SET);
}

void CambiarVerdeRojo(void){ // Transición de verde a rojo
 // La luz verde se encuentra encendida. La apagamos, y encendemos la amarilla
 HAL_GPIO_WritePin(LUZVERDE_GPIO_Port, LUZVERDE_Pin, GPIO_PIN_RESET);
 HAL_GPIO_WritePin(LUZAMARILLA_GPIO_Port, LUZAMARILLA_Pin, GPIO_PIN_SET);
 // Esperamos un tiempo
 HAL_Delay(TESPERA); // Retardo de TESPERA milisegundos
 // Apagamos la luz amarilla y encendemos la roja
 HAL_GPIO_WritePin(LUZAMARILLA_GPIO_Port, LUZAMARILLA_Pin, GPIO_PIN_RESET);
 HAL_GPIO_WritePin(LUZROJA_GPIO_Port, LUZROJA_Pin, GPIO_PIN_SET);
}

// Determinar si el botón fue pulsado
bool VerificarPedido(void){
 // Se crea e inicializa un registro de lectura válida del botón
 static GPIO_PinState ultima_lectura_valida = GPIO_PIN_SET; //botón liberado
 // Se crean variables para lecturas intermedias
 GPIO_PinState lectura1, lectura2;
 // Se crea una variable booleana para indicar si hay un pedido
 bool pedido = false; // No hay pedido hasta que se pulsa el botón
 // Se lee el estado del botón
 lectura1 = HAL_GPIO_ReadPin(BOTONCAMBIO_GPIO_Port, BOTONCAMBIO_Pin);
 // Si hubo un cambio
 if (lectura1 != ultima_lectura_valida){
   // Se espera un tiempo para filtrar los rebotes
   HAL_Delay(TREBOTES); // Retardo de TREBOTES milisegundos
   // Se lee nuevamente el estado del botón
   lectura2 = HAL_GPIO_ReadPin(BOTONCAMBIO_GPIO_Port, BOTONCAMBIO_Pin);
   // Si ambas lecturas son iguales, se considera una lectura válida
   if(lectura2 == lectura1)
     ultima_lectura_valida = lectura2;
   // Si el botón pasó de liberado a pulsado (1-->0), hubo un pedido de cambio de
estado
   if (ultima_lectura_valida == GPIO_PIN_RESET)
     pedido = true;
 }
 return pedido;
}
/* USER CODE END 4 */
```

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

```c
/* Private function prototypes -----------------------------------------------*/
/*...*/
/* USER CODE BEGIN PFP */
void CambiarRojoVerde(void); // Transición de rojo a verde
void CambiarVerdeRojo(void); // Transición de verde a rojo
bool VerificarPedido(void); // Determinar si el botón fue pulsado
/* 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
HAL_GPIO_WritePin(LUZAMARILLA_GPIO_Port, LUZAMARILLA_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(LUZVERDE_GPIO_Port, LUZVERDE_Pin, GPIO_PIN_RESET);
HAL_GPIO_WritePin(LUZROJA_GPIO_Port, LUZROJA_Pin, GPIO_PIN_SET);
/* USER CODE END 2 */
```

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

```
/* Infinite loop */
/*...*/
while (1)
/*...*/
  /* USER CODE BEGIN 3 */
  // Esperamos por una solicitud de cambio de estado.
  while(!VerificarPedido()) // Comprobamos si hay una solicitud haciendo una
lectura filtrada del botón
    HAL_Delay(TSONDEO); // Si no la hay, esperamos por un tiempo y volvemos a
comprobar
  // Al recibir la solicitud, cambiamos de estado de ROJO a VERDE
  CambiarRojoVerde();
  // Esperamos por una solicitud de cambio de estado.
  while(!VerificarPedido()) // Comprobamos si hay una solicitud haciendo una
lectura filtrada del botón
    HAL_Delay(TSONDEO); // Si no la hay, esperamos por un tiempo y volvemos a
comprobar
  // Al recibir la solicitud, cambiamos de estado de VERDE a ROJO
  CambiarVerdeRojo();
}
/* 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 visto para realizar un contador cíclico en el rango de enteros [0, 63] que:

- Se inicie con el valor 0.
- Se incremente con el botón `SW3`.
- Se decremente con el botón `SW1`.
- Se reinicie a 0 con el `SW2`.
- Muestre su valor en binario en los 6 LED de la `SDBoard`.