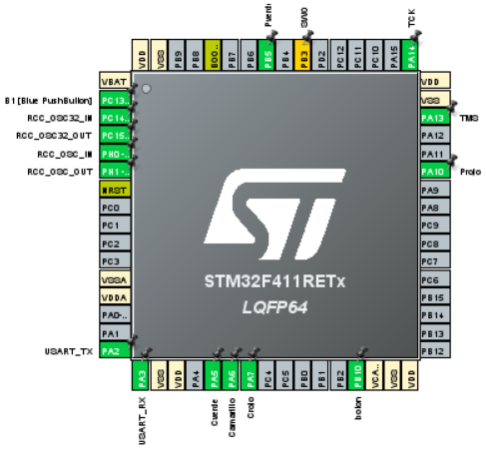
PRÁCTICA 3: Adding Interrupts

Finalidad

El objetivo de esta práctica es la continuación de la práctica 2. El objetivo principal es cambiar el control del pulsador introduciendo interrupciones.

Esquema STM32 CubeX

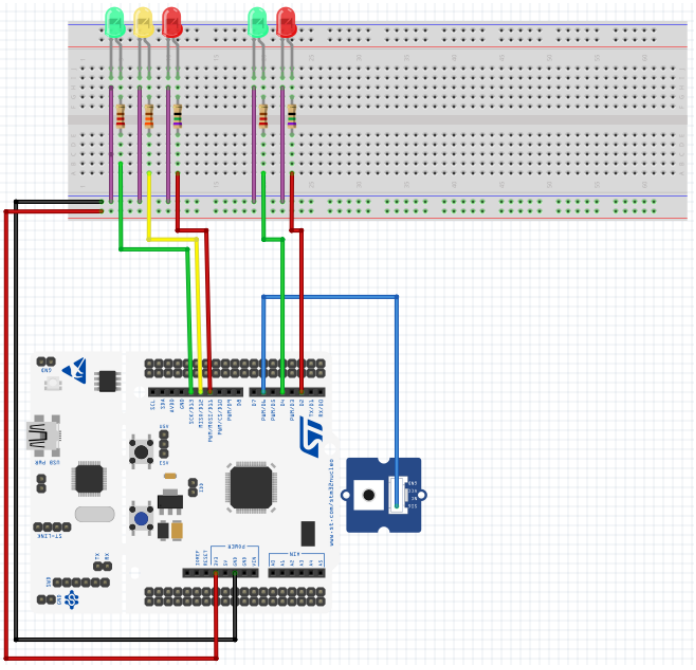


Pin Name	Sig...	GPI...	GPIO mode	GPIO Pull-...	Maximum ...	User Label	Modified
PA5	n/a	Low	Output Push Pull	No pull-up ...	Low	Cverde	☑
PA6	n/a	Low	Output Push Pull	No pull-up ...	Low	Camarillo	☑
PA7	n/a	Low	Output Push Pull	No pull-up ...	Low	Crojo	☑
PA10	n/a	Low	Output Push Pull	No pull-up ...	Low	Projo	☑
PB5	n/a	Low	Output Push Pull	No pull-up ...	Low	Pverde	☑
PB10	n/a	n/a	External Interrupt Mode with Falling edge trigger detection	No pull-up ...	n/a	boton	☑
PC13-ANTI_TAMP	n/a	n/a	External Interrupt Mode with Falling edge trigger detection	No pull-up ...	n/a	B1 [Blue Pus...	☑

En esta ocasión, hemos configurado el pin del botón como EXTI (External Interrupt Mode). Además, hemos habilitado las interrupciones en esa línea (sección NVIC).

No entraremos en más detalle, ya que tanto la configuración de los pines como los componentes del circuito han sido explicados en la práctica anterior.

Esquema del circuito



Solución propuesta

En esta ocasión, a diferencia de la práctica anterior, hemos organizado la secuencia del programa mediante un switch para una mayor limpieza y organización. Hemos mantenido el acceso a registros ya que nos resulta más práctico poder modificar distintos pines simultáneamente, a pesar de ser "menos legible".

En primer lugar, declaramos todos los posibles estados fuera de la sección main. Esto lo realizamos ya que serán utilizados en distintas funciones como la de callback.

```
/* USER CODE BEGIN PM */

//Declaramos los posibles estados del programa
static enum {ST_Inicial, ST_CAmarillo, ST_PVerde, ST_PARPADEO} next_state = ST_Inicial;
/* USER CODE END PM */
```

Declaramos además las dos funciones que utilizaremos para cumplir la lógica del programa.

```
//Declaramos las dos funciones con las que contará el programa
void secuencia(int new_state);
void task_parpadeo(void);
```

```
while (1)
{
  /* USER CODE END WHILE */

  secuencia(next_state);

  /* USER CODE BEGIN 3 */
}
```

En la parte principal del programa únicamente se llama a la función de secuencia, a la que pasamos como parámetro el siguiente estado.

Dicha función consiste en lo siguiente:

```
void secuencia(int new_state){

  switch (new_state)
  {
  case ST_Inicial:
    GPIOA -> ODR = GPIO_ODR_OD5_Msk | GPIO_ODR_OD10_Msk; //coche verde y peaton rojo activos
    break;
  case ST_CAmarillo:
    HAL_Delay(3000); //Esperamos los 3 segundos que dura el estado inicial
    GPIOA->ODR = GPIO_ODR_OD6_Msk | GPIO_ODR_OD10_Msk; //Coche amarillo y peaton rojo activos
    HAL_Delay(3000);
    next_state=ST_PVerde;
    break;
  case ST_PVerde:
    GPIOA->ODR = GPIO_ODR_OD7_Msk; //Coche rojo activo
    GPIOB -> ODR = GPIO_ODR_OD5_Msk; //Peaton verde activo
    HAL_Delay(5000);
    next_state=ST_PARPADEO;
    break;
  case ST_PARPADEO:
    task_parpadeo();
    next_state= ST_Inicial;
    break;
  }
}
```

El funcionamiento es el siguiente: el estado inicial se corresponde con el semáforo de vehículos verde y el de peatones rojo. Esto se mantendrá hasta que se habilite la interrupción (pulsar el botón).

Cuando se pulsa el botón, accedemos a la función callback:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == boton_Pin)
  {
    next_state = ST_CAmarillo; //El siguiente estado del programa será ST_CAmarillo(1)
  }

}
```

La pulsación del botón conlleva a la activación de la interrupción. Aquí, se modifica el siguiente estado al estado amarillo.

Tras esto, se continua con la ejecución del método secuencia, cuyo funcionamiento es idéntico al de la práctica anterior.

En el estado parpadeo se llama a la propia función (explicada en la práctica 2):

```
void task_parpadeo(void){ //coche rojo activo y peaton verde parpadea
    for (size_t i = 0; i < 3; i++)
    {
      HAL_GPIO_WritePin(Pverde_GPIO_Port,Pverde_Pin,GPIO_PIN_RESET);
      HAL_Delay(500);
      HAL_GPIO_WritePin(Pverde_GPIO_Port,Pverde_Pin,GPIO_PIN_SET);
      HAL_Delay(500);
      HAL_GPIO_WritePin(Pverde_GPIO_Port,Pverde_Pin,GPIO_PIN_RESET);
    }
}
```

En esta sección del código se declara las distintas interrupciones así como su prioridad.

```
  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}
```