UNIVERSIDAD NACIONAL DE COLOMBIA

Guía
Configuración de interrupciones externas en C

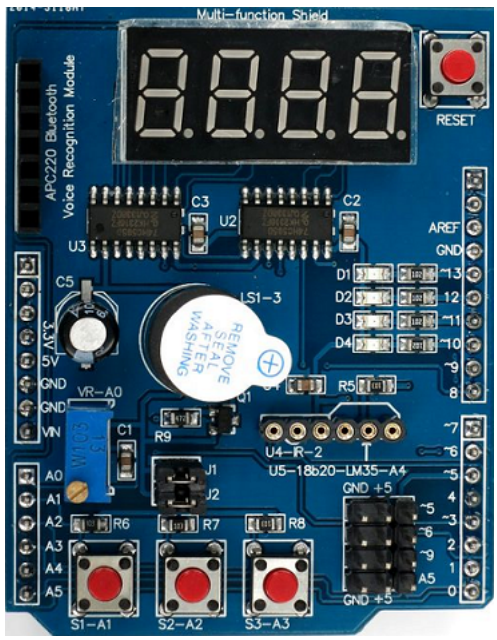
Alejandro Ramírez Jaramillo

Universidad Nacional de Colombia sede Manizales Email: alramirezja@unal.edu.co

Equipo

La Multi Function Shield es un shield que se usará en conjunto con la tarjeta STM32L476RG, para poder llevar a cabo nuevos ejercicios. Este Shield cuenta con periféricos que pueden ser usados al ensamblarse con la tarjeta, algunos de estos son:

- Botones (S1, S2, S3 y Reset): Cuenta con tres botones que pueden ser usados por el usuario en la parte inferior de la tarjeta, más un cuarto botón en la esquina superior derecha que se usar para reset.
- Led (D1, D2, D3, D4): Cuatro Leds, cada uno conectado en serie con una resistencia de 510 Ohm, los cuatro son programados por el usuario.
- Display 7 segmentos (3641BH): Cuatro display 7 segmentos de ánodo común, manejados por dos drivers 74HC595.
- Potenciometro de 10k: Un potenciometro de 10 k Ohm conectado a dos pines del shield, el usuario puede controlar el valor del potenciometro manualmente.
- Pines de conexión de módulos: Hay pines para conectar directamente un sensor de temperatura (DS18B20), un sensor infrarrojo (1838B Infrared IR receiver) y un módulo bluetooth (APC220), no incluye los módulos, solo los pines para conectarlos.



1 GPIO

Los cuatro leds del shield nos permitirán crear secuencias de luces para probar nuestro manejo de los GPIO, sin la necesidad de conectar algún otro elemento a la tarjeta. A continuación se muestra como es la conexión interna de los leds y como se ve en la tarjeta:

En la imagen se evidencia que los pines que se usarán para manejar los pines son los pines 13, 12, 11 y 10 de la shield, que están conectados a los pones PA5, PA6, PA7 y PB6 respectivamente. Dada la conexión entre los leds y los pines sabemos que para encender el led es necesario apagar el pin, cerrando el circuito, mientras que al encender el pin el voltaje entre los terminales del led no es suficiente para encenderlo, de manera que ya sabemos como tenemos que programar en el pin para encender y apagar el led.
Para programar los pines primero es necesario inicializarlos, para lo cual usaremos los registros que aprendimos en la Guía 2 de Estructuras Computacionales que se puede encontar en el siguiente link.

Guía 2 LED_ON.

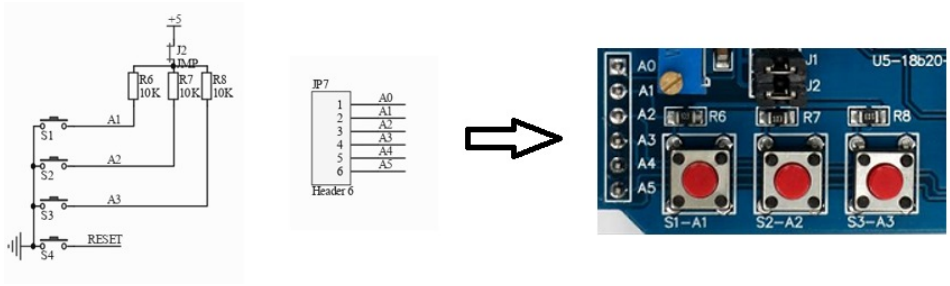
Después de programar todos los pines, se vería algo así:

```
void GPIO_init(void){
  RCC->AHB2ENR |= 0x00000007;
  // Enable GPIOA and GPIOC Peripheral Clock (bit 0 and 2 in AHB2ENR register)
  // Make GPIOA Pin5 as output pin (bits 1:0 in MODER register)
  GPIOA->MODER &= 0xABFFFFFF;   // Clear bits 11, 10 for P5
  GPIOA->MODER &= 0xFFFF57FF;   // Write 01 to bits 11, 10 for P5
  GPIOA->MODER &= 0xFFFFFCF3;
  GPIOA->ODR =0x00E0;
  GPIOB->MODER &= 0xFFFFFEBF;   // Clear bits 11, 10 for P5
  GPIOB->MODER &= 0xFFFFDFFF;   // Write 01 to bits 11, 10 for P5
  GPIOB->MODER &= 0xFFFFFFFC;
  GPIOB->ODR =0x00E0;
  // Make GPIOD Pin13 as input pin (bits 27:26 in MODER register)
  GPIOC->MODER &= 0xFFFFFFFF;   // Clear bits 27, 26 for P13
  GPIOC->MODER &= 0xF3FFFFFF;   // Write 00 to bits 27, 26 for P13
}
```

En este mismo bloque de código se encuentra la inicialización de los botones, tanto de la SHIELD como de la tarjeta, pero esto se explicará más a profundidad en la siguiente sección. Con esta configuración se puede usar el registro `GPIOX->ODR` para encender y apagar los leds.

2 Botones

Los tres botones del SHIELD nos permitirán interactuar directamente con el usuario, pues al presionar un botón el microcontrolador detectará ese cambio y podrá reaccionar a el si se programa para eso, además los tres botones esta conectados a diferentes pines por lo que el microcontrolador puede diferenciar cual de los tres botones esta siendo presionado e incluso detectar cambios en el estado de los tres al mismo tiempo.



En la imagen se evidencia que los pines en los que se recibe la señal del botón son A1, A2 y A3, que en la tarjeta STM32L476RG corresponden a los pines PA1, PA4 y PB0. Si analizamos la conexión de los botones que se muestra en la imagen podemos concluir que cuando el botón se encuentra presionado, el pin al que esta conectado quedará en corto con tierra (0V), mientras que cuando el botón no esta presionado el pin estará conectado a una tensión diferente de cero (+5V). Sabiendo esto podremos determinar el estado del botón al leer el pin correspondiente. Para el caso del botón de la tarjeta de desarrollo, este se encuentra conectado al pin PC13.

Para este caso se seguirá el mismo procedimiento que para la configuración de los leds, pues ambos son gpio, con la diferencia de que esta vez se configurarán los pines como entradas en lugar de salidas. Esta nueva configuración nos permitirá leer el estado del pin, que se encuentra conectado al botón.

```
void GPIO_init(void){
  RCC->AHB2ENR |= 0x00000007;
  // Enable GPIOA and GPIOC Peripheral Clock (bit 0 and 2 in AHB2ENR register)
  // Make GPIOA Pin5 as output pin (bits 1:0 in MODER register)
  GPIOA->MODER &= 0xABFFFFFF;   // Clear bits 11, 10 for P5
  GPIOA->MODER &= 0xFFFF57FF;   // Write 01 to bits 11, 10 for P5
  GPIOA->MODER &= 0xFFFFFCF3;
  GPIOA->ODR =0x00E0;
  GPIOB->MODER &= 0xFFFFFEBF;   // Clear bits 11, 10 for P5
  GPIOB->MODER &= 0xFFFFDFFF;   // Write 01 to bits 11, 10 for P5
  GPIOB->MODER &= 0xFFFFFFFC;
  GPIOB->ODR =0x00E0;
  // Make GPIOD Pin13 as input pin (bits 27:26 in MODER register)
  GPIOC->MODER &= 0xFFFFFFFF;   // Clear bits 27, 26 for P13
  GPIOC->MODER &= 0xF3FFFFFF;   // Write 00 to bits 27, 26 for P13
}
```

Cabe explicar que la razón por lo que los botones S1 y S2 se inicializan juntos (en código), mientras que el botón S3 se inicializa por separado es que estos pertenecen a diferentes puertos.

Ahora que se han configurado pasamos a usar los botones, para lo cual vamos a leer el estado del botón usando un registro que contiene un 1 o 0 dependiendo del voltaje que se vea en el pin, este valor se almacenará en una variable para poder usarlo después, aunque también podría usarse la función directamente en condicionales o programar una interrupción externa, pero el modo más simple de uso y el que explicaremos en la guía es la lectura y almacenamiento en una variable. El registro en el que están guardados estos estados es `GPIOX->IDR`.

La lectura de los botones se puede hacer de dos maneras: pooling o interrupción externa. El pooling consiste en estar revisando constantemente el estado de los botones usando el registro `GPIOX->IDR`, mientras que la interrupción externa es una interrupción que se activa cuando se detecta un cambio en el pin de entrada, mientras esto no ocurra el resto del código se ejecuta con normalidad.

3 Interrupción externa

Se van a programar un total de 4 interrupciones externas, una por cada botón que se inicializó, cada una reaccionará solo a los cambios de estado de su botón correspondiente. Para esto usaremos los registros que se explicaron en la guía 3 de Interrupciones externas que se puede encontrar en el siguiente link:

Guía 3 External Interruption.

Primero comenzamos inicializando las 4 interrupciones externas, esto es simplemente repetir la configuración de la interrupción del botón de la tarjeta de desarrollo pero usando los valores para los nuevos pines de entrada:

```
void Ext_init(void){
    // enable SYSCFG clock
      RCC->APB2ENR |= 0x1;
      // Writing a 0b0010 to pin13 location ties PC13 to EXT4
      SYSCFG->EXTICR[3] |= 0x20; // Write 0002 to map PC13 to EXTI4
      SYSCFG->EXTICR[0]|=0x01;  // Write 01 to map PA1 and PB0 to EXTI1
      SYSCFG->EXTICR[1]&=0x00;  // Write 00 to map PA4 to EXTI2
      // Choose either rising edge trigger (RTSR1) or falling edge trigger (FTSR1)
      EXTI->RTSR1 |= 0x2013;   // Enable rising edge trigger on EXTI4
      // Mask the used external interrupt numbers.
      EXTI->IMR1 |= 0x2013;    // Mask EXTI4
      // Set Priority for each interrupt request
      NVIC->IP[EXTI15_10_IRQn] = 0x10; // Priority level 1
      NVIC->IP[EXTI0_IRQn] = 0x11;
      NVIC->IP[EXTI1_IRQn] = 0x12;
      NVIC->IP[EXTI4_IRQn] = 0x13;
      // enable EXT0 IRQ from NVIC
      NVIC_EnableIRQ(EXTI0_IRQn);
      // enable EXT1 IRQ from NVIC
      NVIC_EnableIRQ(EXTI1_IRQn);
      // enable EXT4 IRQ from NVIC
      NVIC_EnableIRQ(EXTI4_IRQn);
      // enable EXT15_10 IRQ from NVIC
      NVIC_EnableIRQ(EXTI15_10_IRQn);
}
```

Con la aparición de nuevas interrupciones es necesario habilitar cada una de estas y crear sus manejadores, en primer lugar debemos identificar el acónimo de la interrupción que vamos a usar, por ejemplo, cuando usamos el pin 13 del puerto C como entrada para la interrupción, el acrónimo es `EXTI15_10`, pues es el que corresponde a las entradas de pines entre el 10 y el 15 para todos los puertos. Los acrónimos se encuentran en la página 396 del Reference Manual.

0	7	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	8	settable	PVD_PVM	PVD/PVM1/PVM2/PVM3/PVM4 through EXTI lines 16/35/36/37/38 interrupts	0x0000 0044
2	9	settable	RTC_TAMP_STAMP /CSS_LSE	RTC Tamper or TimeStamp /CSS on LSE through EXTI line 19 interrupts	0x0000 0048
3	10	settable	RTC_WKUP	RTC Wakeup timer through EXTI line 20 interrupt	0x0000 004C
4	11	settable	FLASH	Flash global interrupt	0x0000 0050
5	12	settable	RCC	RCC global interrupt	0x0000 0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000 0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000 0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000 0068
11	18	settable	DMA1_CH1	DMA1 channel 1 interrupt	0x0000 006C

Los señalados en rojo son los acrónimos correspondientes a las interrupciones que usan los pines 0, 1 y 4 como entradas. En código se ve así:

```
      // enable EXT0 IRQ from NVIC
      NVIC_EnableIRQ(EXTI0_IRQn);
      // enable EXT1 IRQ from NVIC
      NVIC_EnableIRQ(EXTI1_IRQn);
      // enable EXT4 IRQ from NVIC
      NVIC_EnableIRQ(EXTI4_IRQn);
      // enable EXT15_10 IRQ from NVIC
      NVIC_EnableIRQ(EXTI15_10_IRQn);
```

Después de esto se crean los manejadores, los cuales contendrán el código que se ejecutará cuando se active la interrupción.

```
void EXTI15_10_IRQHandler(void)
{
  //Check if the interrupt came from exti13
  if(EXTI->PR1 & (1 <<13)) {
      boton=0;
    // Clear pending bit
    EXTI->PR1 = 0x00002000;
  }
}
void EXTI0_IRQHandler(void)
{
  //Check if the interrupt came from exti0
  if(EXTI->PR1 & (1 <<0)) {
      boton=0;
    // Clear pending bit
    EXTI->PR1 = 0x00000001;
  }
}
void EXTI1_IRQHandler(void)
{
  //Check if the interrupt came from exti1
  if(EXTI->PR1 & (1 <<1)) {
      boton=0;
    // Clear pending bit
    EXTI->PR1 = 0x00000002;
  }
}
void EXTI4_IRQHandler(void)
{
  //Check if the interrupt came from exti4
  if(EXTI->PR1 & (1 <<4)) {
      boton=0;
    // Clear pending bit
    EXTI->PR1 = 0x00000010;
  }
}
```

4 Referencias

+ PM0214 Programming manual, STM32 Cortex-M4 MCUs and MPUs programming manual.
+ RM0351 Reference manual, STM32L4x5 and STM32L4x6 advanced Arm-based 32-bit MCUs.
+ STM32l476rg datasheet.
+Time-optimal Real-Time Test Case Generation using UPPAAL