IV PWM Duty Cycle

Contreras R Orlando[1,2*] and Garcia B Iker[1,2*†]

[3]Departamento de Sistemas Electrónicos, UAA, Av. Universidad 940, Aguascalientes, 20131, Aguascalientes, México.

*Corresponding author(s). E-mail(s): {al348390,al307630}@edu.uaa.mx;

†Ambos autores contribuyeron de forma equitativa a este trabajo.

Abstract

Esta práctica tiene como objetivo poder controlar el ciclo de trabajo de un LED por medio de botones, todo esto por medio de interrupciones y PWM.

Keywords: ASM, ARM, Shift, Registros, PWM, Duty Cycle

1 Introducción

En esta práctica se busca implementar un sistema capaz de controlar el ciclo de trabajo (duty cycle) de un LED mediante el uso de botones, empleando técnicas de modulación por ancho de pulso (PWM) e interrupciones. El objetivo principal es explorar cómo la combinación de estos dos mecanismos permite una gestión eficiente y precisa del pulso del LED, sin necesidad de intervención constante por parte del procesador.

2 Objetivos

- Implementar un sistema de control de brillo de un LED utilizando modulación por ancho de pulso (PWM).
- Configurar y utilizar interrupciones para detectar la presión de botones sin necesidad de sondeo constante.
- Modificar dinámicamente el ciclo de trabajo del PWM en respuesta a eventos externos (presión de botones).

3 Pseudocódigo

A continuación, se presenta el pseudocódigo que representa la lógica del programa en ensamblador:

Algorithm 1 Manejo de interrupciones externas

Función *0Handler*:
- Guardar la dirección de retorno con `PUSH {LR}` **if** $R5 > R6$ **then**
  - Saltar a la etiqueta `Salto`
- **else**
  - $R5 \leftarrow R5 + 1200$
- **Salto:** Restaurar la dirección de retorno con `POP {LR}` Retornar con `BX LR`

Función *1Handler*:
- Guardar la dirección de retorno con `PUSH {LR}` **if** $R5 < 0x1$ **then**
  - Saltar a la etiqueta `Salto`
- **else**
  - $R5 \leftarrow R5 - 1200$
- **Salto:** Restaurar la dirección de retorno con `POP {LR}` Retornar con `BX LR`

Algorithm 2 Bucle infinito que actualiza el valor del CCR1

Etiqueta `loop()`:
- Cargar la dirección de `TIM2_CCR1` en `R0` Almacenar el valor de `R5` en la dirección apuntada por `R0` Saltar incondicionalmente a `loop`

Algorithm 3 Configuración del sistema de interrupciones externas (EXTI)

Función *_NVIC*:
- Cargar la dirección de `NVIC_ISER0` en `R0` Leer el valor actual en `R1` desde `[R0]` Habilitar interrupciones 6 y 7 con `ORR R1, #(1<<6)` y `ORR R1, #(1<<7)` Escribir el nuevo valor de `R1` en `[R0]` Retornar con `BX LR`

Función *_EXTI*:
- Cargar la dirección de `EXTI_IMR` en `R0` y leer su valor en `R1` Habilitar EXTI0 y EXTI1 con `ORR R1, R1, #0x03` Escribir `R1` de nuevo en `[R0]`
- Cargar la dirección de `EXTI_RTSR` en `R0` y leer su valor en `R1` Configurar flanco de subida para EXTI0 y EXTI1 con `ORR R1, R1, #0x03` Escribir `R1` en `[R0]`
- Retornar con `BX LR`

Función *_Syscfg*:
- Cargar la dirección de `SYSCFG_EXTICR1` en `R0` Configurar EXTI0 y EXTI1 para puerto B: `MOV R1, #0x0011` Escribir `R1` en `[R0]` Retornar con `BX LR`

GitHub Repository

4 Explicación del Código Ensamblador

Este es el código original en ensamblador que corresponde al pseudocódigo anterior:

```
__main
    BL      Config_RCC
    BL      Config_GPIO
        BL                  Config_TIM
        BL      Conf_NVIC
    BL      Config_EXTI
    BL      Config_Syscfg
    LDR     R5,=12000
    LDR     R6,=24000
loop
    LDR         R0, =TIM2_CCR1
    STR                 R5,[R0]

        B                   loop

Config_RCC
    LDR     R0,=RCC_AHB1
    LDR     R1,[R0]
    ORR     R1,#(0x3)
    STR     R1,[R0]


        ; TIM5  TIM4  TIM3  TIM2
        ; 0      0     0      1
    LDR     R0,=RCC_APB1
    LDR     R1,[R0]
    ORR     R1,#0x0001
    STR     R1,[R0]
    LDR     R0,=RCC_APB2
    LDR     R1, [R0]
    ORR     R1, R1, #(1 << 14)
    STR     R1, [R0]

    BX      LR

Config_GPIO
    LDR     R0, =GPIOA_MODER
    LDR     R1, [R0]
    LDR     R2, =0x02
    ORR     R1,R1,R2
    STR     R1,[R0]
    LDR     R0, =GPIOA_OSPEED
    LDR     R1, [R0]
    ORR     R1,R1, #0x03
    STR     R1,[R0]
    LDR     R0, =GPIOA_AFRL
    LDR     R1, [R0]
    ORR     R1,#(1 << 0)
    STR     R1,[R0]
        LDR         R0,= GPIOB_PUPDR
        LDR         R1, [R0]
        ORR         R1,R1,#0x0A
        STR         R1, [R0]

    BX      LR



Conf_NVIC
        LDR         R0,= NVIC_ISER0
        LDR         R1, [R0]
        ORR         R1, #(1<<6)
        ORR         R1, #(1<<7)
        STR         R1, [R0]
        BX                  LR

Config_EXTI
        LDR         R0,= EXTI_IMR
        LDR         R1, [R0]
        ORR         R1,R1, #0x03
        STR         R1, [R0]

        LDR         R0,= EXTI_RTSR
        LDR         R1, [R0]
        ORR         R1,R1, #0x03
        STR         R1, [R0]
        BX                  LR
Config_Syscfg
    LDR     R0, =SYSCFG_EXTICR1
    MOV     R1, #0x0011
    STR     R1, [R0]
    BX      LR


exti0Handler
        PUSH {LR}
    CMP     R5, R6
    BHI     salto
    ADD     R5,R5,#1200
salto
        POP {LR}
    BX      LR

exti1Handler
        PUSH {LR}
    CMP     R5,#0x1
    BLO    salto
    SUB     R5,R5,#1200
        POP {LR}
        BX      LR

Config_TIM
        LDR         R0, =TIM2_PSC
        MOV         R1, #999 ; Tim_Prescaler -->(1000-1)
        STR                 R1,[R0]

        LDR         R0, =TIM2_ARR
        MOV         R1, #23990 ; Tim_Period -->(24000-1)
        STR                 R1,[R0]

    LDR         R0, =TIM2_CCMR1
    LDR         R1, [R0]
        ORR         R1, #0x68
        STR                 R1,[R0]

    LDR         R0, =TIM2_CCER
        LDR         R1, [R0]
        ORR         R1, #(1 << 0)
        STR                 R1,[R0]


        LDR         R0, =TIM2_CR1
        LDR         R1, [R0]
        ORR         R1, #0x81
        STR                 R1,[R0]

        BX          LR
        align
        end
```

4.1 Direcciones de Registro

- **RCC_BASE**: Dirección base del sistema de control de reloj. Permite habilitar los relojes para los periféricos como GPIOA, GPIOB, SYSCFG y TIM2.
- **GPIOA_BASE y GPIOB_BASE**: Direcciones base de los puertos GPIO A (salida PWM) y B (entradas para botones).
- **Registros MODER, OTYPER, OSPEED, PUPDR**: Configuración de modo de operación, tipo de salida, velocidad y resistencias internas para los pines de los GPIO.
- **Registros IDR y ODR**: Entrada y salida de datos digitales en los pines de los puertos.
- **Registros del Timer (TIM2)**: Incluyen control del prescaler (PSC), ARR (auto-reload), y CCR1 (valor del ciclo de trabajo del PWM).
- **EXTI y NVIC**: Configuración de interrupciones externas y habilitación en el controlador de interrupciones (NVIC).
- **SYSCFG_EXTICR1**: Registro que enlaza las líneas EXTI con los pines PB0 y PB1.

4.2 Inicialización del Sistema

El sistema comienza ejecutando la subrutina `Config_RCC`, que se encarga de:

- Habilitar el reloj de los puertos GPIOA y GPIOB mediante el registro `RCC_AHB1ENR`.
- Activar el reloj del temporizador TIM2 en `RCC_APB1ENR`.
- Habilitar SYSCFG para permitir la conexión entre pines GPIO y las líneas EXTI.

4.3 Configuración de GPIO

- El pin **PA0** se configura en modo *función alternativa* (AF1) para usarse con la salida PWM del Timer 2 canal 1.
- Se establece una velocidad alta en PA0 para que soporte correctamente la señal PWM.
- Los pines **PB0** y **PB1** se configuran como entradas digitales con resistencias *pull-down* activadas, permitiendo detectar flancos de subida al presionar los botones.

4.4 Lógica de Control

- El valor del ciclo de trabajo (duty cycle) del PWM se almacena en el registro `TIM2_CCR1`.
- En el bucle principal, el código mantiene constante el valor del duty cycle, que solo se modifica mediante interrupciones:
  - **EXTI0 (PB0)** incrementa el duty cycle (duración del LED) en pasos.
  - **EXTI1 (PB1)** disminuye el duty cycle.
- Se implementan verificaciones para evitar que el valor sobrepase los límites mínimo (0) y máximo (valor definido por ARR).

4.5 Subrutinas

- `Config_RCC`: Activa los relojes de los periféricos necesarios (GPIOA, GPIOB, TIM2, SYSCFG).
- `Config_GPIO`: Configura PA0 como salida alternativa para PWM y PB0/PB1 como entradas con pull-down.
- `Config_TIM`: Inicializa el temporizador TIM2 para generar una señal PWM ajustable.
- `Conf_NVIC`: Habilita las interrupciones EXTI0 y EXTI1 en el NVIC.
- `Config_EXTI`: Configura las líneas EXTI0 y EXTI1 para que se activen con flancos de subida.
- `Config_Syscfg`: Asocia EXTI0 y EXTI1 con los pines PB0 y PB1 respectivamente.
- `exti0Handler`: Aumenta el ciclo de trabajo del PWM, limitado por un valor máximo.
- `exti1Handler`: Disminuye el ciclo de trabajo del PWM, limitado por un valor mínimo.

5 Conclusiones

El mayor problema que tuvimos fue configurar correctamente las interrupciones en el puerto B. Al principio no respondía como esperábamos, y eso nos hizo perder algo de tiempo revisando registros y configuraciones. Finalmente, entendimos qué bits debíamos ajustar y logramos que funcionara como queríamos.

Esta practica nos enseño el como a la hora de hacer código el entender el microcontrolador y entender el manual nos permite cambiar el funcionamiento del dispositivo a como se requiera.