Parpadeo de LED mediante SysTick e Interrupciones en el lenguaje C

Autor

David Ramírez Betancourth (C.C 1002636667)

Fecha

2 de diciembre de 2024

Resumen

Este documento detalla el desarrollo de un programa en lenguaje C para controlar el parpadeo de un LED utilizando el temporizador SysTick en el microcontrolador STM32L476RG. Se aborda la configuración de los periféricos, la implementación del código fuente y la verificación del funcionamiento del sistema.

Introducción

Microcontrolador STM32L476RG

Vamos a utilizar un Microcontrolador STM32l476RG que es un componente electrónico que ya posee varios módulos necesarios:

- Módulo depurador en tiempo real st-link

- Módulo de conectores varios
- Botón de usuario (USER, B1)
- LED interno (LD2)
- Núcleo ARM-4 (Cortex-M4)

Objetivo

- Implementar un parpadeo de LED usando SysTick como temporizador

Desarrollo

Inicilalización de Systick

```
// Base de SysTick y registros
#define SYSTICK_BASE 0xE000E010U
#define SYSTICK_CTRL (*(volatile uint32_t *)(SYSTICK_BASE + 0x0))
#define SYSTICK_LOAD (*(volatile uint32_t *)(SYSTICK_BASE + 0x4))
#define SYSTICK_VAL (*(volatile uint32_t *)(SYSTICK_BASE + 0x8))
#define SYSTICK_CALIB (*(volatile uint32_t *)(SYSTICK_BASE + 0xC))

// Bits de control en el registro SysTick_CTRL
#define SYSTICK_CTRL_ENABLE (1U << 0)  // Bit de habilitación
#define SYSTICK_CTRL_TICKINT (1U << 1) // Bit de interrupción
#define SYSTICK_CTRL_CLKSRC (1U << 2)  // Fuente de reloj (1: procesador)
```

Primero se da la dirección base de systick para hacer su offset preestablecido a cada componente en memoria del systick (ctrl, load, val, calib), por lo tanto la dirección en memoria base es `0xE000E010U` y se le suma los offset necesarios (`0x#`)

- `SysTick_CTRL (0xE000E010)`: Controla el habilitado del temporizador, las interrupciones y la fuente de reloj.
- `SysTick_LOAD (0xE000E014)`: Configura el valor inicial desde el cual el

contador comienza a decrementar.

- `SysTick_VAL (0xE000E018)`: Almacena el valor actual del contador decreciente y permite reiniciarlo escribiendo cualquier valor en él.
- `SysTick_CALIB (0xE000E01C)`: Proporciona información de calibración, como la cantidad de ciclos necesarios para generar una interrupción en un período fijo (si está implementado)

Inicilalización de GPIO A y C

```
// Base addresses para GPIO
#define GPIOA_BASE 0x48000000U
#define GPIOC_BASE 0x48000800U

// Registros GPIO
#define GPIO_MODER_OFFSET 0x00U
#define GPIO_ODR_OFFSET 0x14U

#define GPIO_MODER(gpio) (*(volatile uint32_t *)((gpio) + GPIO_MODER_OFFSET))
#define GPIO_ODR(gpio) (*(volatile uint32_t *)((gpio) + GPIO_ODR_OFFSET))
```

Lo mismo pasa con el GPIO. Hay una dirección base en memoria, y cada gpio con sus respectivas configuraciones tienen un offset (MODER, ODR)

`GPIO_MODER_OFFSET (0x00U)`: Offset del registro de configuración de modo (MODER). Este registro define si un pin GPIO actúa como entrada, salida, función alternativa, o entrada analógica.

`GPIO_ODR_OFFSET (0x14U)`: Offset del registro de salida de datos (ODR). Este registro controla el valor de salida de los pines configurados como salidas.

Inicializar APB2

APB2 (Advanced Peripheral bus 2) es uno de los buses internos del microcontrolador que conecta el nucleo principal del procesador con los periféricos (TIM1, TIM8, EXTI)

```
#define RCC_BASE 0x40021000U
#define RCC_APB2ENR (*(volatile uint32_t *)(RCC_BASE + 0x60))
#define RCC_GPIOA_ENABLE (1U << 0)
#define RCC_GPIOC_ENABLE (1U << 2)

void RCC_EnableAPB2(uint32_t peripherals) {
    RCC_APB2ENR |= peripherals;
}
```

Interrupción de SysTick_Handler

`SysTick_Handler` es una palabra reservada que está definida para el nucleo l476rg, se utiliza para correro código cada que el Handler de SysTick aparezca, esto se da en los flancos de bajada del reloj interno del núcleo de 8MHz, En este programa lo estamos utilizando para tener un contador, para saber el tiempo transcurrido en el programa, con esto en mente podemos cambiar de estado el GPIOA pin de placa (PC13) que corresponde a LD2.
Así podemos hacer parpadear el led.

```
void SysTick_Handler(void)
{
    system_ticks++;
}
```

Programa Principal

El programa configura SysTick y GPIO para alternar el estado del LED interno cada 500 ms:

```c
#include <stdint.h>

int main(void) {
    // Inicializar SysTick
    SysTick_Init(3999); // Configura SysTick para generar interrupciones cada 1 ms

    // Habilitar GPIOA
    RCC_EnableAPB2(RCC_GPIOA_ENABLE);

    // Configurar pin 5 como salida
    GPIO_SetPinAsOutput(GPIOA_BASE, 5);

    uint32_t last_tick = 0;

    while (1) {
        if ((SysTick_GetTick() - last_tick) >= 500) {
            last_tick = SysTick_GetTick();
            GPIO_TogglePin(GPIOA_BASE, 5); // Alterna el estado del LED
        }
    }

    return 0;
}
```

Conclusión

El parpadeo del LED fue implementado exitosamente utilizando el temporizador SysTick y las interrupciones. Este enfoque permite un control eficiente y preciso del tiempo, demostrando la utilidad de SysTick en sistemas embebidos. El código es modular y puede adaptarse a otros proyectos con mínimos cambios.