STM32 Course Report: Weeks 4-5

Cao Zhengyang
Department of Electronics and Electrical Engineering
12110623@mail.sustech.edu.cn

I. Lecture 4

A. Summary

This lecture delved into advanced STM32 programming with FWLib, covering modular LED control, key detection techniques for LED interaction, and practical considerations like code portability and jitter elimination. We learned to build modular LED modules, utilize key presses to control LED state, and configure input ports for accurate key detection, equipping we to create sophisticated and adaptable programs for our STM32 microcontroller.

B. LED Looping

```
void LED_RGBlooping(void){
    delay_init();
    int gapTime = 1000;
    while(1){
        GPIO_ResetBits(LED_GPIO_PORT,
        LED_GPIO_Pin_R);
        delay_ms(gapTime);
        GPIO_SetBits(LED_GPIO_PORT,
        LED_GPIO_Pin_R);
        delay_ms(1);

        GPIO_ResetBits(LED_GPIO_PORT,
        LED_GPIO_Pin_G);
        delay_ms(gapTime);
        GPIO_SetBits(LED_GPIO_PORT,
        LED_GPIO_Pin_G);
        delay_ms(1);

        GPIO_ResetBits(LED_GPIO_PORT,
        LED_GPIO_Pin_B);
        delay_ms(gapTime);
        GPIO_SetBits(LED_GPIO_PORT,
        LED_GPIO_Pin_B);
        delay_ms(1);
    }
}
```

This code makes the RGB LED change colors in a loop. It sets a delay, switches on and off the Red, Green, and Blue parts one after another, and keeps looping forever. It's like a continuous color show for the LED.

C. LED Switching Using Switch

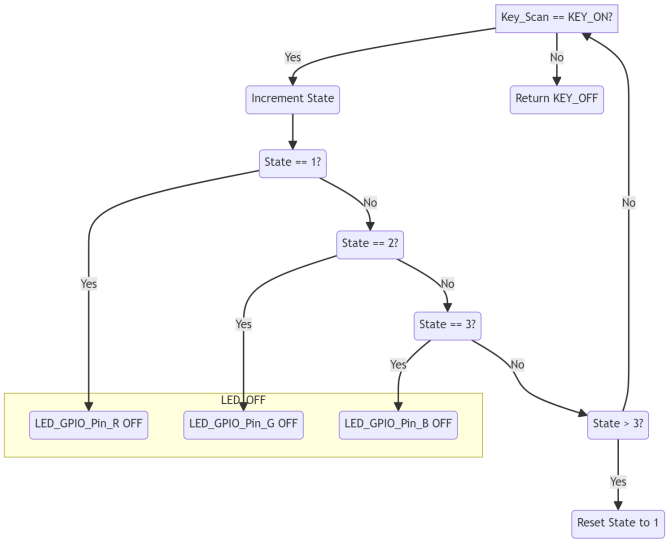

Fig. 1. LED Switching Flow Chart

```
#define LED_GPIO_CLK    RCC_APB2Periph_GPIOB
#define LED_GPIO_Pin    GPIO_Pin_5
#define LED_GPIO_PORT    GPIOB

#define LED_GPIO_Pin_R    GPIO_Pin_5
#define LED_GPIO_Pin_G    GPIO_Pin_0
#define LED_GPIO_Pin_B    GPIO_Pin_1

//switch related define
#define SWITCH_GPIO_CLK     RCC_APB2Periph_GPIOA
#define SWT_GPIO_PORT    GPIOA
#define K1_GPIO_Pin    GPIO_Pin_0
uint8_t Key_Scan
(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin) {
    if (GPIO_ReadInputDataBit(GPIOx, GPIO_Pin) ==
    KEY_ON)
    {
        while (GPIO_ReadInputDataBit(GPIOx,
        GPIO_Pin) == KEY_ON);
        return KEY_ON;
    } else {
        return KEY_OFF;
    }
}
```

```
void LED_switchControl2(void) {
    delay_init();
    int state = 0;
    while (1) {
        if (Key_Scan(SWT_GPIO_PORT,
        K1_GPIO_Pin) == KEY_ON) {
            state++;
            delay_ms(10);
        }
        if (state == 1) {
            GPIO_SetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_R);
            GPIO_SetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_G);
            GPIO_SetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_B);
            GPIO_ResetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_R);
        } else if (state == 2) {
            GPIO_SetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_R);
            GPIO_SetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_G);
            GPIO_SetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_B);
            GPIO_ResetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_G);
        } else if (state == 3) {
            GPIO_SetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_R);
            GPIO_SetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_G);
            GPIO_SetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_B);
            GPIO_ResetBits(LED_GPIO_PORT,
            LED_GPIO_Pin_B);
        } else if (state > 3) {
            state = 1;
        }
    }
}
```

This code is like a traffic cop for a button and an LED. The first part checks if a button connected to a specific GPIO pin is pressed. If it is, it waits until the button is released.

The second part controls an LED based on the button press. It starts by initializing a delay function and a state variable. In a never-ending loop, it checks if the button is pressed. If so, it increases a counter and waits a bit. Depending on the counter value, it turns on and off different colors of an RGB LED connected to different GPIO pins. The loop keeps going, cycling through LED colors and resetting the counter when it goes beyond 3. It's like a dance routine for the LED triggered by the button.

II. Lecture 5

A. Summary

This lecture covered three main topics: debug methods for finding and fixing program errors, clock system configuration for managing microcontroller timing, and NVIC interrupt control for handling external events. Through simulations and hands-on exercises, we learned to configure clocks to 72MHz, prioritize interrupts, and modify interrupt service routines for LED blinking and timer control.

B. the process of NVIC configuration

1) Set Priority Grouping:
   - Use NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup) to define how priority bits are allocated for preemption and sub-priority.
   - Choose a grouping that suits application's interrupt hierarchy needs.
2) Call NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct) to set
   - NVIC_IRQChannel
   - NVIC_IRQChannelPreemptionPriority
   - NVIC_IRQChannelSubPriority
3) Enable the NVIC
   - NVIC_Init(&NVIC_InitStruct);
   - NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;

C. Modify the project

This is a program which need

```
#include "sys.h"

int main(void)
{
        LED_GPIO_Config();
        delay_init();
        TIM3_Init(9999, 7199);
        static uint8_t i = 0;
        while (1)
        {
                KEY_Scan();
                LED_Change();
        }
}
```