Assignment 6 Report

EE615 : Embedded Systems Lab

Atharv Gade (210020004)
Yash Meshram (210020053)
22 September 2024

Aim:

a. Part 1

For the first part of the assignment, we were supposed to create a PWM waveform with a variable duty cycle. The duty cycle should be initiated at 50% and increased by 5% when one of the switches is pressed and decreased by 5% when the other switch is pressed.

b. Part 2

The problem statement is the same as part 1 but now we have to increase or decrease using only 1 switch. The duty cycle should increase for a short press and decrease for a long press.

Theory:

- **Switch 1 linked to GPIO port F interrupt :**

  - Program an interrupt to check if either Switch 1 or Switch 2 is pressed.

  - When the interrupt is triggered, determine which switch is pressed by comparing it with their 'on' values.

  - Based on the button pressed:

    - **SW1 (Switch 1)** : Decrease the value of `d`.

    - **SW2 (Switch 2)**: Increase the value of `d`.

- **Part 2 - Level Trigger for SW1** :

  - Set the interrupt to **level trigger** for Switch 1 (SW1).

  - Initialize the **Systick timer** on each button press to measure how long the button is held.

  - If the Systick measures a time greater than 1 second, it's considered a **long press**:

    - **Long press** : Increment the value of `d`.

    - **Short press** : Decrease the value of `d`.

- **Handling edge cases** :

  - When the value of `d = 0` (0% duty cycle), turn off the LED.

  - When `d = 100` (100% duty cycle), keep the LED constantly on.

- **Final step** :

  - Commit the code and submit it to the GitHub repository.

Code :

Part1 :

```
#include <stdint.h>
#include <stdbool.h>
#include "tm4c123gh6pm.h"
/**
 * main.c
 */

void GPIOInterrupt(void);
void INIT_SYS_CTRL_REGISTERS(void);
void INIT_GPIO_PORTF_REGISTERS(void);
void INIT_TIMER1_REGISTERS(void);

uint32_t PORTF_Interrupt = 0x00;
int duty = 80;
int main(void)
{
    INIT_SYS_CTRL_REGISTERS();
    INIT_GPIO_PORTF_REGISTERS();
    INIT_TIMER1_REGISTERS();

    while(1){
        NVIC_EN0_R = 0x40000000; // 30th bit controls PORTF
        GPIO_PORTF_IM_R = 0x11; // unmasking both switches
//        GPIO_PORTF_CR_R = 0x00;
        TIMER1_TBMATCHR_R = duty;
    }

    return 0;
}

void INIT_SYS_CTRL_REGISTERS(){
    SYSCTL_RCGC2_R |= 0x00000020;       /* enable clock to GPIOF */
    SYSCTL_RCGCTIMER_R = 0x02;
    SYSCTL_RCGCGPIO_R = 0x20;
}

void INIT_GPIO_PORTF_REGISTERS(){
    GPIO_PORTF_LOCK_R = 0x4C4F434B;     /* unlock commit register */
    GPIO_PORTF_CR_R = 0x1F;             /* make PORTF configurable */
    GPIO_PORTF_DEN_R = 0x01F;           /* set PORTF pins 4 : 0 pins */
    GPIO_PORTF_DIR_R = 0x0E;            /*  */
    GPIO_PORTF_PUR_R = 0x11;            /* PORTF0 and PORTF4 are pulled
up */
    GPIO_PORTF_AFSEL_R = 0x08;
    GPIO_PORTF_PCTL_R = 0x00007000;
```

```
    NVIC_EN0_R = 0x40000000; // 30th bit controls PORTF
    GPIO_PORTF_IS_R = 0x00; // interrupt sensitivity – edge
    GPIO_PORTF_IEV_R = 0x00;
    GPIO_PORTF_IM_R = 0x11; // unmasking both switches
}

void INIT_TIMER1_REGISTERS(){
    TIMER1_CTL_R = 0x00;
    TIMER1_CFG_R = 0x04;
    TIMER1_TBMR_R = 0x0A;
    TIMER1_TBILR_R= 160;
    TIMER1_TBMATCHR_R = duty;
    TIMER1_CTL_R = 0x0100;
}

void GPIOInterrupt(){
    PORTF_Interrupt = GPIO_PORTF_RIS_R & 0x11;

    NVIC_EN0_R = 0x00000000; // 30th bit controls PORTF
    GPIO_PORTF_IM_R = 0x00; // masking both switches
    if (PORTF_Interrupt == 0x01){
        GPIO_PORTF_ICR_R = 0x01;
        duty += 8;
        if (duty >= 152){ // saturating
            duty = 152;
        }
    }

    else if (PORTF_Interrupt == 0x10){
        GPIO_PORTF_ICR_R = 0x10;
            duty -= 8;
            if (duty <= 0){ // saturating
                duty = 0;
            }
        }


//    GPIO_PORTF_DATA_R ^= 0x02;


}
```

Part2:

```
#include <stdint.h>
#include <stdbool.h>
#include "tm4c123gh6pm.h"
/**
 * main.c
 */

void GPIOInterrupt(void);
void SystickInterrupt(void);
void INIT_SYS_CTRL_REGISTERS(void);
void INIT_GPIO_PORTF_REGISTERS(void);
void INIT_TIMER1_REGISTERS(void);

uint32_t PORTF_Interrupt = 0x00;
uint32_t SW_State = 0x00;
int duty = 80; // init, 50% duty cycle
int main(void)
```

```
{
    INIT_SYS_CTRL_REGISTERS(); // init system control registers
    INIT_GPIO_PORTF_REGISTERS();
    INIT_TIMER1_REGISTERS();

    while(1){
        NVIC_EN0_R = 0x40000000; // 30th bit controls PORTF
        GPIO_PORTF_IM_R = 0x01; // unmasking one switch
//        GPIO_PORTF_CR_R = 0x00;
        TIMER1_TBMATCHR_R = duty;// update the compar-value of TIMER1-B
    }

    return 0;
}

void INIT_SYS_CTRL_REGISTERS(){
    SYSCTL_RCGC2_R |= 0x00000020;       /* enable clock to GPIOF */
    SYSCTL_RCGCTIMER_R = 0x02; // enable clock to General Purpose Timer
1 Module
    SYSCTL_RCGCGPIO_R = 0x20; // enable clock to PORTF GPIO
}

void INIT_GPIO_PORTF_REGISTERS(){
    GPIO_PORTF_LOCK_R = 0x4C4F434B;     /* unlock commit register */
    GPIO_PORTF_CR_R = 0x1F;             /* make PORTF configurable */
    GPIO_PORTF_DEN_R = 0x0F;            /* set PORTF pins 4 : 0 pins */
    GPIO_PORTF_DIR_R = 0x0E;            /*  */
    GPIO_PORTF_PUR_R = 0x01;            /* PORTF0 and PORTF4 are pulled
up */
    GPIO_PORTF_AFSEL_R = 0x08; // Select PORTF3 (Green LED) for
Alternate Function:
                                // Green LED not driven by
GPIO_PORTF_DATA_R but instead by T1CCP1 (PWM Signal from GPT1)
    GPIO_PORTF_PCTL_R = 0x00007000; // connects the PWM output of GPTM1
to PORTF3

    NVIC_EN0_R = 0x40000000; // 30th bit controls PORTF
    GPIO_PORTF_IS_R = 0x00; // interrupt sensitivity - edge
    GPIO_PORTF_IEV_R = 0x00; // GPIO Interrupt triggered at negative
edge from Pulled-Up Switch
    GPIO_PORTF_IM_R = 0x01; // unmasking one switch (SW2)
}

void INIT_TIMER1_REGISTERS(){
    TIMER1_CTL_R = 0x00; // make sure TIMER1 is disabled before
configuring
    TIMER1_CFG_R = 0x04; // configures the timer in 16-bit mode
    TIMER1_TBMR_R = 0x0A; // configure the timer in periodic timer mode,
with PWM mode enabled
    TIMER1_TBILR_R= 160; // the reload value to achieve 10us (assuming
16MHz clock)
    TIMER1_TBMATCHR_R = duty; // the compare value for the timer
    TIMER1_CTL_R = 0x0100; // enable Timer B (the timer which we're
using)
}

void INIT_SYSTICK(){ // initializing systick
    NVIC_ST_RELOAD_R = 16000*500; // 500 ms
    NVIC_ST_CURRENT_R = 0x00;
    NVIC_ST_CTRL_R = 0x00000007;
}

void SystickInterrupt(){
```

```
    SW_State = (GPIO_PORTF_DATA_R & 0x01); // read the state of switch
    if (SW_State == 0x00){ // if it is still pressed, brighten
        duty -= 8;
        if (duty <= 0){ // saturating
            duty = 0;
        }
    }

    else{ // if it is released, stop systick- no need to brighten
anymore
        NVIC_ST_CURRENT_R = 0x00;
        NVIC_ST_CTRL_R = 0x00000000;
    }
}

void GPIOInterrupt(){
    PORTF_Interrupt = GPIO_PORTF_RIS_R & 0x01; // read which switch
caused the interrupt


    INIT_SYSTICK(); // in case this interrupt turns out to be a long-
press

    // for debouncing
    NVIC_EN0_R = 0x00000000; // 30th bit controls PORTF
    GPIO_PORTF_IM_R = 0x00; // masking both switches
    if (PORTF_Interrupt == 0x01){ // switch was pressed, reduce
brightness
        GPIO_PORTF_ICR_R = 0x01; // for edge-triggered interrupts,
necessary to clear the interrupt status
        duty += 8;
        if (duty >= 152){ // saturating
            duty = 152;
        }
    }



}
```

Result:

We observed that the interrupt handler successfully identified the correct switch interrupt and adjusted the duty cycle by either increasing or decreasing it accordingly.

We used github to commit changes to our project.

Conclusion:

We demonstrated the ability to use the interrupt handler to modify the duty cycle of the blinking LED.