19 September 2024

Lab 6 Report

Group 9

Ashis Pradhan 210020003

Gurvir Singh Bhatti 210020012

Aim:

Part 1:

Create a PWM waveform with frequency = 100KHz and variable duty cycle.

The program should begin with d = 50%.

On pressing one switch the duty should be increased by 5% and on pressing other switch it should be decreased by 5%.

Part 2:

Implement the same but using only 1 switch (SW1 OR SW2) – short press for d increase and long press for decrease.

Theory:

The switch 1 press is directly linked to the GPIO port F interrupt. We could easily program the interrupt for checking if the switch 1 is pressed of switch 2 pressed for the first part. Whenever the interrupt is triggered we check which switch is pressed by comparing it with the their ‘on’ values, after which, in either case we can program the d to be changed on each press of the buttons to either increase (SW2), or to decrease (SW1).

For part 2 we set the interrupt to level trigger for the SW1, and initialise the Systick on each button press, to measure the amount of time the button is pressed, if the systick measures time greater than 1 second, we have considered that as long press, hence we increment the value of d, otherwise, we decrease the value of d on each short press. To normalise the edge cases, whenever the value of d = 0 (0 Duty cycle), we turn off the LED, and in case of d = 100, the LED is constantly switched on.

After this we commit the files to the GitHub repository for submission.

Code:

Lab 6 part 1:

```
33
34 void GPIOPortF_Handler(void) {
35     // Check if interrupt occurred on PF4 (Switch)
36     if (GPIO_PORTF_RIS_R & (1 << 4)) {
37         GPIO_PORTF_ICR_R |= (1 << 4);  // Clear the interrupt flag
38         if (d<100){
39             d += 5;  //Increase duty cycle
40         }
41     }
42     if (GPIO_PORTF_RIS_R & (1 << 0)) {
43         GPIO_PORTF_ICR_R |= (1 << 0);  // Clear the interrupt flag for PF0
44         if (d > 0) {
45             d -= 5;  // Decrease duty cycle
```

Lab 6 part 2:

```
int main(void) {
    // Enable clock for GPIO Port F
    SYSCTL_RCGCGPIO_R |= (1 << 5); // Enable clock for Port F
    while ((SYSCTL_PRGPIO_R & (1 << 5)) == 0);  // Wait until Port F is ready

    // Unlock PF4 (switch)
    GPIO_PORTF_LOCK_R = 0x4C4F434B;  // Unlock GPIO Port F
    GPIO_PORTF_CR_R |= 0x10;      // Allow changes to PF4

    // Configure PF4 (Switch) as input and PF1 (LED) as output
    GPIO_PORTF_DIR_R &= ~(0x10);   // Set PF4 as input (switch)
    GPIO_PORTF_DIR_R |= 0x02;     // Set PF1 as output (LED)
    GPIO_PORTF_DEN_R |= 0x13; // Enable digital for PF4 and PF1
    GPIO_PORTF_PUR_R |= 0x10;     // Enable pull-up on PF4

    // Disable interrupt for PF4 during configuration
    GPIO_PORTF_IM_R &= ~(0x10);

    // Set PF4 to trigger interrupt on falling edge (button press)
    GPIO_PORTF_IS_R |= 0x10;
    GPIO_PORTF_IEV_R &= ~(0x10);

    // Clear any prior interrupt on PF4
    GPIO_PORTF_ICR_R |= 0x10;

    // Enable interrupt on PF4
    GPIO_PORTF_IM_R |= 0x10;

    // Enable the interrupt in NVIC (IRQ30 for GPIO Port F)
    NVIC_EN0_R |= (1 << 30);

    // Enable global interrupts
    __asm("     CPSIE I");

    // Initial state: turn off the LED
    GPIO_PORTF_DATA_R &= ~(1 << 1);

    while(1) {
            if (d==0){
                GPIO_PORTF_DATA_R &= ~(0x02);
            }else if (d==100){
                GPIO_PORTF_DATA_R |= 0x02;
            }else{
                GPIO_PORTF_DATA_R &= ~(0x02);
                Delay(d);
                Delay(100-d);
            }
        }
}
```

-:Please find the attached startup file for the interrupt handling and better understanding of the implementation:-

Output:

We observed that the interrupt handler was able to correctly determine the switch interrupt and appropriately increase of decrease the duty cycle.

Conclusion:

We demonstrated that we can utilise the interrupt handler to change the duty cycle of the blinking LED.