॥ सा विद्या या विमुक्तये ॥
भारतीय प्रौद्योगिकी संस्थान धारवाड
Indian Institute of Technology Dharwad

EE690: Embedded Systems Lab

Lab Assignment 4

Group 11
EE23DP001 Aditya Shirodkar
EE23MS005 Prasanth Pithanisetty

EK-TM4C123: Implementation of Interrupts

1 Aim:

To implement interrupts for blinking an onboard LED for 1 second every time a user activates the onboard switch.

2 Procedure:

1. Enable clock to the necessary GPIO port and make the port configurable
2. Configure the onboard LEDs as digital outputs and onboard switches as digital inputs.
3. Define functions for GPIO pins and systick interrupts to toggle LED for 1 second using systick upon activation of onboard user switch.

3 Documents Referred:

1. TM4C123GH6PM microcontroller datasheet

4 Algorithm:

The algorithm for switch based LED toggling using interrupts is as follows:

1. The GPIO port F is configured such that LEDs are digital outputs and switches are digital inputs.
2. Interrupts for Port F are enabled
3. An interrupt function is written for handling hardware interrupts caused by pressing of user switch. Whenever the on-board switch is activated, the following occurs:
   (a) The on-board LED is toggled ON
   (b) Interrupts are enabled for Systick
   (c) Systick is loaded with the desired delay and enabled
4. An interrupt function is written for handling Systick interrupt, for when the systick timer runs out. This is described below:

(a) The on-board LED is toggled ON

(b) The interrupt flag for port F is cleared

5. NVIC is configured to allot priority 1 to the hardware interrupt

6. A While(1) loop is included in the main.c code for perpetual running of the program.

5 Code:

Lines added to tm4c123gh6pm_startup_ccs.c:

```
Line 58: void SysTickHandler(void);
Line 59: void IntPortFHandler(void);
Line 87: SysTickHandler,     // The SysTick handler
Line 118:IntPortFHandler,    // GPIO Port F
```

Main.c Code:

```
/*
 * This code uses interrupts to blink an LED for one second
 * whenever a user switch is pressed
 * It uses 2 interrupt services:
 * PortFHandler; for handling hardware interrupt when switch is pressed
 * SysTickHandler; for when systick timer runs out
 */

#include <stdint.h>
#include <stdbool.h>
#include <tm4c123gh6pm.h>

//Control and Status Register
#define STCTRL *((volatile long *) 0xE000E010)
//SysTick Reload Value Register
#define STRELOAD *((volatile long *) 0xE000E014)
//SysTick Current Value Register
#define STCURRENT *((volatile long *) 0xE000E018)

//Definitions to configure systick CSR(Control and Status Register)
//bit 0 of CSR enables systick counter
#define ENABLE (1<<0)
//bit 1 of CSR to generate interrupt to the NVIC when
//SysTick counts to 0
#define INT_EN (1<<1)
//bit 2 of CSR to select system clock
#define Clk_SRC (1<<2)
```

```
//bit 16 of CSR; The SysTick timer has counted to 0 since
//the last time this bit was read.

#define COUNT_FLAG (1<<16)

#define Mask_Bits 0x11

//Configure PortF; Enable clock to PortF; LEDs as digital output,
//Switches as digital input
void PortFConfig(void)
{
    //enable clock to GPIOF
    SYSCTL_RCGC2_R |= 0x00000020;
    //Unlock PortF register
    GPIO_PORTF_LOCK_R = 0x4C4F434B;
    //Enable Commit function
    GPIO_PORTF_CR_R = 0x1F;

    //Enable all pins on port F
    GPIO_PORTF_DEN_R = 0x1F;
    //Set LEDs as outputs and Switches as inputs
    GPIO_PORTF_DIR_R = 0x0E;
    //Pull-up for user switches
    GPIO_PORTF_PUR_R = 0x11;
}


void IntPortFHandler(void)
{
    //Clear any previous interrupts on port F
    GPIO_PORTF_ICR_R = Mask_Bits;
    GPIO_PORTF_IM_R &= ~Mask_Bits;
    //Reinitialise Systick Counter to Zero
    STCURRENT=0x00;

    STRELOAD = 16*1000000;
    //Enable Systick, Enable Interrupt Generation,
    //Enable system clock (80MHz) as source
    STCTRL |= (ENABLE | INT_EN | Clk_SRC);
    GPIO_PORTF_DATA_R = 0x0E;
}

void SysTickHandler(void)
{
    GPIO_PORTF_DATA_R = 0x11;
    //mask, clear and unmask gpio interrupt
```

```
    GPIO_PORTF_IM_R &= ~Mask_Bits;
    GPIO_PORTF_ICR_R = Mask_Bits;
    GPIO_PORTF_IM_R |= Mask_Bits;
}


int main(void)
{
    PortFConfig();

    //PortF Interrupt Configurations:
    //User Sw should trigger hardware interrupt

    //Edge trigger detected
    GPIO_PORTF_IS_R &= ~Mask_Bits;
    //Trigger interrupt according to GPIOIEV
    GPIO_PORTF_IBE_R &= ~Mask_Bits;
    //Trigger interrupt on falling edge
    GPIO_PORTF_IEV_R &= ~Mask_Bits;
    //Mask interrupt bits
    GPIO_PORTF_IM_R &= ~Mask_Bits;
    //clear any prior interrupts
    GPIO_PORTF_ICR_R |= Mask_Bits;
    //enable interrupts for bits corresponding to Mask_Bits
    GPIO_PORTF_IM_R |= Mask_Bits;

    //NVIC Configuration
    //PortF interrupts correspond to
    //interrupt 30 (EN0 and PRI7 registers)
    //Interrupts enabled for port F
    NVIC_EN0_R |= (1<<30);
     //Interrupt Priority 1 to Port F
    NVIC_PRI7_R &= 0xFF3FFFFF;
    //Reinitialise Systick Counter to Zero
    STCURRENT=0x00;

    while(1);

}
```

6 Results:

On activating the user switch, a hardware interrupt is triggered, which turns on the onboard LED and enables the Systick timer to run for 1 second. Upon completion of 1 second, Systick generates an interrupt of its own, which is used for clearing the interrupt flag and toggling the

LED off. This can be seen in Fig. 1 to 3.

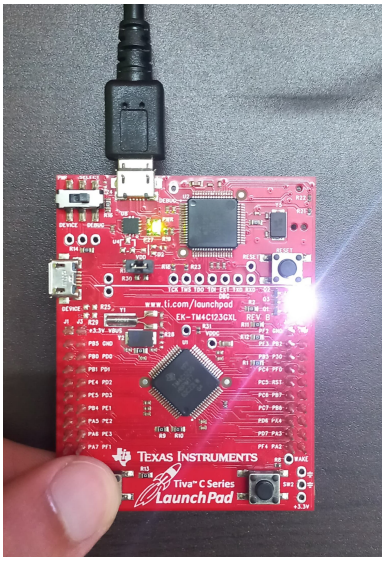

Figure 1: Hardware Interrupt Triggerred

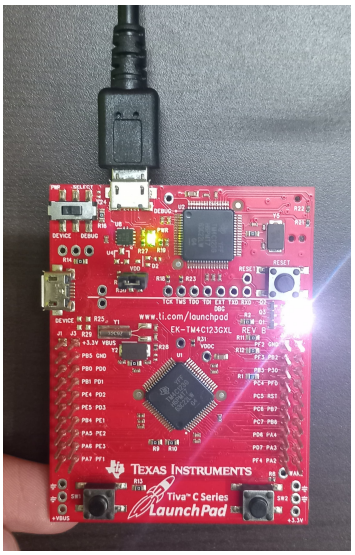

Figure 2: LED remains on as Systick counts for 1 second

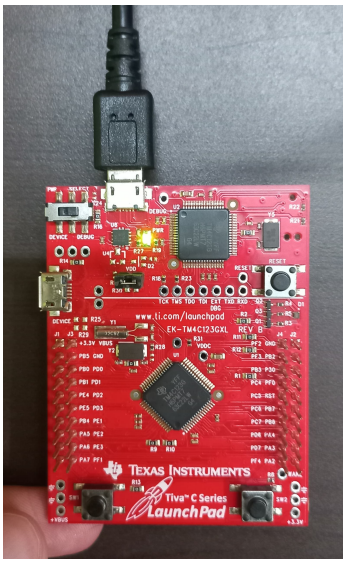

Figure 3: Systick interrupt triggered

The above operation can be verified using an oscilloscope, as is shown in Fig. 4.

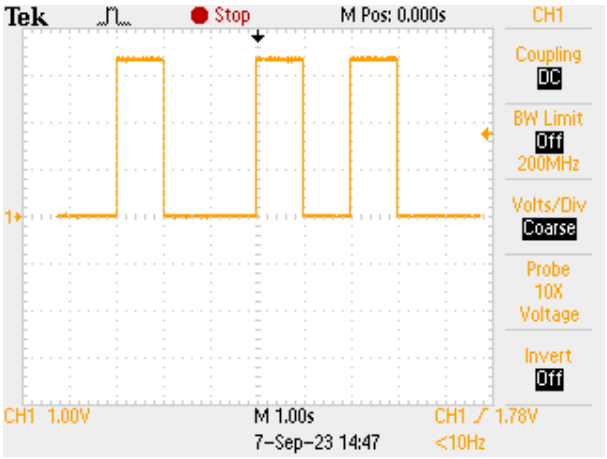

Figure 4: Oscilloscope Data for 1 second interrupt based LED toggling upon activation of user switch

7 Conclusion

Interrupts prove to be a critical function in any microprocessor based application. The incorporation of hardware interrupt for detection of user switch activation, as well as systick interrupt for timer countdown allows the microprocessor to perform other necessary tasks, rather than periodically polling the user switch and timer, leading to better utilisation of processor resources.