Laboratory 6: Timers and Interrupts

Experiment Sheet

Purpose

The purpose of this laboratory is to learn how to configure and use timers and interrupts in ARM-based microcontrollers, specifically on the TM4C123GH6PM. The experiment involves setting up a timer to generate periodic interrupts, and handling those interrupts in an interrupt service routine (ISR) to toggle an LED. The focus will be on configuring the timer, enabling interrupts, and managing the interrupt controller.

Essential Knowledge

- **SYSCTL_RCGCTIMER**: System Control - Timer Clock Gating Control register, which enables or disables the clock to the timer.
- **TIMER0_CFG**: Timer configuration register for Timer0.
- **TIMER0_TAMR**: Timer A mode register, used to set the mode (e.g., periodic, one-shot).
- **TIMER0_TAILR**: Timer interval load register, used to load the interval for the timer (defines how frequently the interrupt occurs).
- **TIMER0_ICR**: Timer interrupt clear register, used to clear the interrupt flag after the interrupt has been handled.
- **TIMER0_IMR**: Timer interrupt mask register, used to enable or disable timer interrupts.
- **NVIC_EN0**: NVIC interrupt enable register, used to enable specific interrupts in the Nested Vector Interrupt Controller.
- **GPIOF_BASE**: Base address for GPIO Port F, used to control LEDs and switches.
- **NVIC_PRI4**: NVIC priority register for Timer0 interrupts, which determines the priority of the interrupt.

Timer Types

The TM4C123GH6PM has multiple types of timers:

1. **General-Purpose Timers (GPT)**:
    - Used for standard timing functions, such as generating periodic interrupts, measuring time intervals, or creating delays.
    - Example: Timer0, Timer1, Timer2, Timer3.
    - Mode: Can be configured in One-Shot, Periodic, PWM (Pulse Width Modulation), Capture, Compare, Watchdog, and Time Base modes.
2. **SysTick Timer**:
    - A special 24-bit timer dedicated to generating periodic interrupts at regular intervals. Typically used for system timekeeping, task scheduling, or managing delays.
    - Configurable in **Periodic** mode.
    - Example: SysTick timer in ARM Cortex-M processors.
3. **Watchdog Timer (WDT)**:
    - A safety feature used to reset the system if the software fails to reset the timer within a set interval (i.e., to prevent system lock-ups).
    - Not used in this experiment, but important in embedded systems for system reliability.

Interrupt Types

Interrupts can be classified into different types based on whether they can be masked or not, and the trigger condition:

1. **Maskable Interrupts**:
    - Interrupts that can be enabled or disabled by setting the appropriate bits in the interrupt mask registers.
    - Example: Timer interrupts, GPIO interrupts.
    - Triggered in either **Edge** or **Level** modes.
2. **Non-Maskable Interrupts (NMI)**:
    - Critical interrupts that cannot be disabled, often used for system fault or emergency conditions.
    - Example: A watchdog timer reset interrupt.

Interrupt Modes

Interrupts can operate in the following modes:

1. **Edge Triggered Interrupts**:
    - The interrupt is triggered when a signal changes state (rises or falls).
    - Example: GPIO pin interrupt when a button is pressed or released.
2. **Level Triggered Interrupts**:
    - The interrupt is triggered when a signal stays at a particular state (high or low) for a defined period.
    - Example: A sensor output that stays high when activated.

Timer and Interrupt Configuration for the Experiment

- **Timer Type**:
    - **General-Purpose Timer (GPT)**: Specifically, we will use **Timer0** in **Periodic Mode** for generating periodic interrupts to toggle an LED.
    - **SysTick Timer** can also be configured for periodic interrupts, but in this experiment, we focus on the general-purpose timer.
- **Interrupt Type**:
    - **Maskable Interrupt**: Timer0 interrupt is maskable, which means we can enable or disable it using the appropriate registers.
    - **Edge Triggered**: Timer interrupts are typically edge-triggered, based on the timer's internal counting mechanism when it overflows or reaches the defined interval.

Steps for Timer and Interrupt Setup

Timer Setup Steps

1. **Enable the Timer Clock:**
    - Enable the clock to Timer0 in the system control register (`SYSCTL_RCGCTIMER`).
    - This step ensures that Timer0 is powered on and ready to be configured.
2. **Configure Timer in Periodic Mode:**
    - Set Timer0 to Periodic Mode by configuring the **TIMER_TAMR** register.
    - In Periodic mode, the timer will generate interrupts after a set period and then restart, continuously generating interrupts at regular intervals.
3. **Set Timer Load Value:**

- Set the timer interval by writing the desired value to the **TIMER_TAILR** register.
- This value determines the time delay before the timer generates an interrupt, such as setting it to generate an interrupt every second.

4. **Enable Timer:**
   - Enable Timer0 by writing the appropriate value to the **TIMER_CTL** register.
   - This starts the timer and allows it to begin counting according to the interval set.

Interrupt Setup Steps

1. **Enable Timer Interrupt:**
   - Enable the interrupt for Timer0 by setting the corresponding bit in the **TIMER_IMR** (Interrupt Mask Register).
   - This allows the processor to respond when the timer generates an interrupt.
2. **Configure NVIC for Interrupt Handling:**
   - Enable the interrupt for Timer0 in the **Nested Vector Interrupt Controller (NVIC)** by setting the appropriate bit in the **NVIC_ISER** register.
   - Additionally, configure the interrupt priority using the **NVIC_IPR** register to specify the interrupt's importance relative to other interrupts.
3. **Write Interrupt Handler:**
   - Write an interrupt handler function (e.g., `Timer0_Handler`) that will be executed when the interrupt occurs.
   - This function is responsible for clearing the interrupt flag and handling any specific tasks triggered by the timer interrupt.
4. **Clear Interrupt Flag:**
   - In the interrupt handler, clear the interrupt flag by writing to the **TIMER_ICR** register.
   - This step ensures that the interrupt is properly acknowledged and ready for the next occurrence.

This setup will allow you to handle periodic interrupts triggered by Timer0 and use them to control the LED's state.

Task for Students:

1 **SW1 (PF4) Functionality**:
   - When SW1 is pressed, the LED color should cycle forward through the sequence: 0x02, 0x04, 0x06, 0x08.
   - After reaching 0x08, pressing SW1 again should wrap the color back to 0x02, continuing the cycle.

2 **SW2 (PF0) Functionality**:
   - When SW2 is pressed, the LED should revert to the previous color in the sequence:
     - If the current color is 0x06, pressing SW2 should change it to 0x04.
     - If the current color is 0x04, pressing SW2 should change it to 0x02.
     - If the current color is 0x02, pressing SW2 should change it to 0x08 (the last color in the cycle).