University of Texas at El Paso
Electrical and Computer Engineering Department

EE 3176 – Laboratory for Microprocessors I

Spring 2022

LAB 03
SysTick Timer and Interrupts

Goals:

- Learn about the SysTickTimer
- Design and build a kitchen timer.
- Compare and contrast using software-based delays vs. timer-based delays.

Pre Lab Questions:

- What is the purpose of a computing delay – an explicit delay in an algorithm?
- What is a microcontroller timer?
- What is the difference between software-based and timer-based delays?
- What is the MSP432 SysTick timer and how do you use it?
- If a processor operates at 3 MHz, how many cycles does it complete in 0.333 s, 0.5 s, and 30 seconds?
- What is an interrupt?

Timer Interrupts
Lab Guide

Microcontrollers usually have internal oscillators to drive their timers for designs including time tracking. In addition to internal oscillators, which are inexpensive and convenient but not accurate, external crystal-based oscillators are used when design require accurate oscillators for clock signal sources. As a quick and dirty approximation to timing, software loops may be used to introduce explicit processing delays. However, accurate delays are difficult to achieve with this approach because actual delays will be a function of executed instructions and will change when the processor or its clock frequency is modified. For instance, the default processor clock of the MSP432 is the digitally controlled oscillator (DCO), which has a default frequency of 3 MHz ±0.5%.

SysTick Timer

The SysTick timer is a simple 24-bit down counter that can be used to generate time delays and periodic interrupts. The SysTick timer runs at bus clock frequency, which is driven by default by the DCO. This timer is available in all ARM Cortex-M microcontrollers, which makes code using it portable across a variety of microcontrollers. Just like GPIO ports are configured and used for binary input/output operations through a set of registers, the SysTick timer is configured and used by accessing registers in the table below.

Address/bits	31-24	23-17	16	15-3	2	1	0	Name
0xE000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	CTRL
0xE000E014	0	24-bit RELOAD value						LOAD
0xE000E018	0	24-BIT CURRENT value of SysTick timer counter						VAL

Code in the next page is a simple example of how to use the SysTick timer.

```c
//******************************************************************************
//  MSP432P401 Demo - SysTick timer used in interval mode
//
//  Description: SysTick is configured to count down from 0x60000 to 0. At 0,
//  the SysTick count gets reloaded to the original 0x60000 value to repeat the
//  process. An interrupt is also configured to trigger when the SysTick count
//  gets down to 0. P1.0 LED is toggled in the SysTick interrupt service routine.
//
//           MSP432P401x
//         ---------------
//     /|\|               |
//      | |               |
//      --|RST            |
//        |           P1.0|-->LED
//
//   William Goh
//   Texas Instruments Inc.
//   Oct 2016 (updated) | November 2013 (created)
//   Built with CCSv6.1, IAR, Keil, GCC
//******************************************************************************
#include "msp.h"
int main(void) {
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD; // Stop WDT

    // Configure GPIO
    P1->DIR |= BIT0;
    P1->OUT &= ~BIT0;

    // Enable & configure the SysTick Timer Module
    SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk;
    SysTick->LOAD = 0x60000 - 1;     // Period = 0x60000
    SysTick->VAL = 0x01;             // Clear value by writing dummy value
    SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;  // Enable interrupt

    // Enable global interrupt
    __enable_irq();

    SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;  // Sleep on exit from ISR
    __DSB(); // Ensure SLEEPONEXIT takes effect immediately

    while (1)
      __sleep();
} // End of main

// Interrupt service routine (ISR) invoked when SysTick down counter reaches 0.
void SysTick_Handler(void)
{
    P1->OUT ^= BIT0; // Toggle P1.0 LED
}
```

System Design

- Create an empty C project for the MSP432 Launchpad.
- Enter and execute the SysTick demo code above.
- Using a voltmeter and then an oscilloscope see the output value and signal produced by the code. Write down the voltmeter measurement. Draw the output signal in your lab notebook annotating it with values of high voltage, low voltage, and cycle length.
- Modify the code to produce a 30 Hz signal. What was your modification and how did you decide to make it? Can you measure it with your voltmeter? What amplitude and period values is it showing? Why? Verify your modification and analysis by now producing a 45 Hz signal.
- Use this signal to design a kitchen count-down timer (the “KTimer”) that you can set, start/continue, stop, and clear.
- Have your KTimer display remaining time in minutes and seconds in binary.
- When it must be set, the KTimer must blink a red LED
- When the KTimer starts it must turn on a blue LED.
- When the KTimer stops it must turn on a red LED.
- While measuring time, the KTimer must blink a green LED.
- Draw a circuit schematic diagram and the algorithm flowchart of your final design.
- Once you are done, show your lab notes and demonstrate the KTimer functions to the Teaching Assistant for lab credit.

Frequently Asked Questions

Can I manage two or more timings at once?

Yes, you can manage two or more different timings. The MSP432 has one a number of internal oscillators to drive clock sources for the processor and peripherals that require timing signals.

Will timer modules interfere with each other?

Timer modules and clock sources are independent, but they have different priorities. Considering this, a high-priority timer module will be processed before a low-priority one. Thus, while this is a form of high-priority modules interfering with low priority ones, this behavior provides better control on time-based processing and more predictable real-time response. See the MSP432 User Guide for additional information on timers and interrupt processing.