`

SYSC3310 Lab 8 – Using the Timers

Fall 2018

Objectives:

- Programming of TIMER_A
- Use of the Temperature and Light Sensor on the MKII
- Multiple ISRs

Equipment:

- MSP432 P401R LaunchPad Development Kit

Submission: Lab9b.c and Lab9c.c.

References and Reading Material

- Demo programs from the lectures:
  - Demo-Interrupt-TimerA0-Continuous (Uses TA0_N_IRQHandler)
  - Demo-Interrupt-TimerA0-Continuous-DividedClock (Uses TA0_N_IRQHandler)
  - Demo-Interrupt-TimerA0-Upmode (Uses TA0_N_IRQHandler)
  - Demo-Interrupt-TimerA0-UpDownMode (Uses TA0_N_IRQHandler&TA0_0_IRQHandler)
  -

Part A: Exploring an Timer Program

- *You are to use a project called **Lab8a** for your solution.*

In the lectures notes, you have an example of a Timer program called **Demo-Interrupt-TimerA0-Continuous-ClockDivided**. As the name suggests, the program uses TimerA0 in continuous mode to toggle the LED at a fixed rate.

Sample Question: What interrupt source is triggered by Continuous Mode? Alternatively, for each timer module, there are two interrupt sources possible: TAx_N_IRQHandler and Tax_0_IRQHandler. Which one is used in this program for continuous mode?

1. As a start, you should create a project called **Lab9a** and copy-paste this code into its main program, and run the program to see that it works.
2. Demonstrate your understanding of continuous mode by changing the rate at which the LED blinks. What single value must be changed?

No demonstration or submission. For learning purposes only.

`

Part B: A Step Towards Part C

- *You are to use a project called **Lab8b** for your solution.*

You're going to demonstrate your comfort with learning new material on your own by changing your program to now make use of the MK-II's temperature sensor and light sensor. Both are devices supported by Valvano's BSP software – in other words, use the functions in BSP.h to read values from the temperature sensor and the light sensor.

Specifically, at a regular period interval, your program must

(1) Toggle the Red LED
(2) Display the updated light level
(3) Display the updated temperature.

1. Copy your working program from Part A into a new project called **Lab9b**.
2. In the main() function, initialize the light sensor and the temperature sensor (using the proper functions that you've found in BSP.h)
3. In the TA0_N_IRQHandler ISR, add code to (1) read the temperature sensor and (2) display that value on the LCD. Once that is working, repeat for the light sensor.
   - I am purposely not helping you with this code to see if you can figure out the code by yourself. There are actually 5 functions for the temperature and you have to figure out which one to call.
     - It is an exercise in reading code, and reading comments for the code.
     - Have you learned about static methods?
     - Have you learned about pass-by-reference parameters?
     - Have you learned about ISR programming – ISRs should NOT do any action that is time-consuming!
   - The temperature value that is produced is a signed value. To convert the value to degrees-Celsius, you must divide the value by 100000.
   - The light value is also a signed value. It must be divided by 100.

Once you have the program working, you can play around. Shine your phone's flashlight at the light sensor, or cover it up. Press your finger on the temperature sensor and hold it there for a minute.

Helpful Tip and a Lesson about Using the LCD: **Read if the values being displayed are weird**

Suppose the light is really bright when you first run your program, giving a light value of 9876 to be displayed on the LCD using BSP_LCD_OutDec() at cursor (8,0) (Character 8 in Row 0)

Light: 9876

Now you cover the light sensor with your finger so that the light value is reduced to a value of 12 to be displayed on the LCD using BSP_LCD_OutDec() at cursor (8,0) (Character 8 in Row 0)

`

Light: 1276

The last two digits 76 of the previous large value are still present on the LCD display, corrupting the value being displayed!

Lesson: Every time you re-write numbers within a row on the LCD, you need to clear out all the spaces from the previous value.

Part C: Modifying the Timer Program to use Multiple Timers

- *You are to use a project called **Lab8c** for your solution.*

As mostly an exercise in installing ISRS and initialization rituals, you will modify the existing code from Part B to make use of three separate timers:

(1) Toggle the Red LED at 1 Hz, using Timer_A0
(2) Display the updated light level at 2 Hz, using Timer_A1
(3) Display the updated temperature at 0.5 Hz, using Timer_A2

Marking Scheme – Total of 15 marks

Part A: 1 mark for Demonstration. No submission needed
Part B: Demonstration: (3 marks)

- 0 : Not working
- 1: The program monitors one sensor but not both, with errors in the display (big numbers hide small numbers)
- 2: The programs monitors both sensors, with errors in the display
- 3: The program monitors both sensors and there are no errors in the display

Inspection: 1 mark each (0, ½ or 1), for a total of 6

1. Structure of program follows interrupt-driven architecture (all code is within **one** ISRs and global/static variables, except for the initialisation; main loop is empty).
2. (2 marks) ISR does not have undue delays. Do not use the BSP_xxx_Input functions. Instead, use the BSP_xxx_End() functions and check the return value to check the status of the conversion.
3. Temperature and Light values are scaled appropriately.
4. The LCD is cleared before displaying a new number, in case the new number is smaller than the previous number.
5. Overall Style – As usual

Part C: Demonstration: (1 marks)

- 0 : Not working

`

- 1: The program works identically to Part B.

Inspection: 1 mark each (0, ½ or 1), for a total of 4

1. Program now contains three ISRs – TimerA0, A1 and A2.
2. Each ISR has only code for one device (LED, temperature or sensor)
3. The NVIC is configured separately for each of the three interrupts, all enabled and all set at priority 2.
4. Each Timer is programmed to a different frequency by using different clock dividers. (Bits 7:6 in TAxCTL must be different in TA0CTL and TA1CTL). The rate of 0.5 Hz in TA2CTL requires a software divide (using a global variable).

Overall marks for style

- Comments, indentation, and well-named functions and variables
- Removal of all extra code (no commented out sections)

`

Part D: Problem Solving with an Applied Problem [Optional. No marks. To be used in Fall2019]

- *This section is written in the style of a sample exam question*
- *The problem refers back to a problem in a previous lab. It is similar but there are significant differences in these requirements that take precedence over the previous requirements.*
- *You are to use a project called **Lab9d** for your solution.*

You have been hired to build a traffic light using the MSP432P401r Launchpad. The traffic light will be implemented using a mixture of Launchpad's and the MKII's devices:

- The Traffic Light is the RGB on the MKII
- The pedestrian button – is B1 on the MKII
- The arrival car sensor (dug into the pavement under the stop line) – is B2 on the MKII
- The pedestrian sign – is the LCD on the MKII
  - Print "WAIT" – When the pedestrian should stop (i.e. when GREEN)
  - Print 10…0 countdown – When the pedestrian is permitted to walk (i.e. when RED or YELLOW)

The system will begin with the light GREEN (so cars can drive through). It will cycle through a regular schedule of GREEN (10 seconds) – YELLOW (2 seconds) – RED (10 seconds). If a pedestrian presses the pedestrian button while the light is GREEN, the light should immediately turn YELLOW. If a car arrives while the light is RED, the light should turn GREEN no more than 2 seconds later.

Your program will consist of four threads-of-control made up solely of interrupt foreground threads; there is no background main thread in this system (loop in the `main`() function is empty). Remember that the key to unlocking the logic in interrupt-driven solutions lies in the global variables through which the threads communicate with each other. Typically, one thread will write a global variable that is read by the other thread, thereby effectively sending a message. In the system below, it is suggested that the `timeRemaining` (0 to 10 seconds) be used as one of those global variables. For example, when the pedestrian button is pushed, the `timeRemaining` is set to 2, if it is not already less than two.

The system could be written using only one timer (and hence, 3 threads) because all the frequencies are equals (all 1 Hz). The use of two timers is strictly arbitrary, done for teaching-and-learning purposes, to give you practice in rituals and ISR installations.

Global variables
(add more as needed)

uint8_t colour = GREEN // GREEN, YELLOW, RED

uint32_t timeRemaining= 10

int main() {

// Initialisation Ritual

while (1) {}

void PORT?_IRQHandler () {

// Acknowledge interrupt

// Handle pedestrian button

void PORT?_IRQHandler () {

// Acknowledge interrupt

// Handle car arrival

void TA0_0_IRQHandler () {

// Acknowledge interrupt

// Manage Traffic Light

void TA1_0_IRQHandler () {

// Acknowledge interrupt

// Manage Pedestrian Signal