`

SYSC3310 Lab 5 – Using the SysTick

Fall 2018

Objectives:

- Use of polled SysTick
- Use of interrupt-driven SysTick
- Vary the experience in technical reading by using **Texas Instruments Training Material**

Submission: Lab5b.c, Lab5c.c and Lab5d.c

Equipment:

- MSP432 P401R LaunchPad Development Kit

References and Reading Material

- Posted Source: TI Training Material's **Module 9, Lab 9 : SysTick Timer** (Also posted on CULearn)
  - This is part of Valvano's TI Robotics System Learning Kit (TI-RSLK)
  - Associated Video on SysTick theory
  - Associated Video on Lab 9's Section 9.1
  - Associated Video on Lab 9's Section 9.2

- Posted Code: CortexM.h, Cortex.c (Available within Valvano's Board Support Package)
  - See Lab 4 on setting up the "inc" folder. These files should be in this "inc" folder.

Part A: Background Preparation

When using SysTick in DRA style, there are complications with the included header files.

Before:

```
#include <msp.h>
```

Now:

```
#include "../inc/msp432p401r.h"
#include "../inc/CortexM.h"          // And add CortexM.c to your project
```

The relative pathname in the code above **(../inc/)** assumes that you have used this folder structure. Please see Lab 4 for instructions.

`

Part B: Polled use of SysTick

- *Your solution must be stored in a project and program called Lab5b*

You are to complete TI's Module 9, Lab 9: Sections 9.4.1, 9.4.2. Read the whole lab, but ignore the portions about the use of a logic analyser and of adding new circuitry. If you don't understand the discussion, please watch the videos that go along with this lab (listed above).

- The instructions make use of SRA-style of GPIO. You must convert this to the DRA style.

Clarifications on Section 9.4.1:

- The program described in this section is essentially a re-structured version (using nice functions) of the demonstration program called Demo-SysTickPolled, posted on CULearn. Feel free to copy-paste code from there into your new program.
- You are given the code for the function called SysTick_Init(). It is given in SRA-style. You need to translate this to DRA-style.
  - Please see your lecture notes
  - **Please study the TRM, Section 4.11.**
- You are only given the call signature for the function called `SysTick_Wait1us()` is given. You are then given a description of this function's implementation, "Sequence of Steps" given at the top of Page 5. You have to figure out the code that matches these 3 steps.
  - You will have to study the TRM, Section 4.11. In particular, you will have to find out what the COUNT bit (Is it bit 0 or is it bit 14? You need to know this first, before figuring out the masks to use in your bit manipulations).
- You are given the code for the main program, called **Program9_1** (in SRA-style). Rename it to `main()`
  - We won't use the Clock_Init48MHz() function. We will use the default clock frequency of 3 MHz. However that will mean that you will have to modify (increase) the values used for delays.
  - You will have to write the LaunchPad_Init() function. It should configure the red LED's port (as we've done before). Again, feel free to copy-paste from the demonstration program.
- Run your program before continuing to the next section. Make sure that the light is blinking as described, 75% on, 25% off. Sometimes if your LED is solid RED, it may actually be blinking, but at a rate that is too quick for our eyes.
  - Try setting breakpoints on the lines that turn the LED on and off. If you arrive at this line more than once, you know that the code is working, and the LED is just blinking too quickly.
  - Try changing your delay parameters to larger numbers.

`

Clarifications on Section 9.4.2:

- You are given the array `PulseBuf[]`. You can actually copy-paste from the PDF to your code. Re-build to make sure that it compiles (i.e. not error during the copy-paste)
- You must implement the steps 1..7
- Run your program and observe. You should see the rate of blinking change, from the LED being almost always on, to being barely on.

Part C: Interrupt-driven use of SysTick

- *Your solution must be stored in a project and program called Lab5c*

You are to go beyond TI's Lab 9, and transform your solution for Part B into an interrupt-driven solution. Your solution must not contain any code that polls the SysTick flags. When you get it working, it should behave the same as Part B from the user's perspective.

- Within SysTick_Init(), you must tweak the initialisation of SysTick to now include enabling of interrupts
- You must install the SysTick ISR (**Suggestion: Use the demo program from the lectures**), including globally enabling interrupts.
- You must move all logic from the main()'s loop into the ISR, re-working it to remove all polling and all loops. Your main() loop should be empty.
    - ISRs do not generally contain loops. Transform the while-loop into if-statements, based on state variables. "State variables" are global variables used by an ISR to govern its logic. In this part, the ISR needs to know whether the pulse should be H (high) or whether it should be L (low). Introduce a new global variable that has either one of two values, HIGH or LOW. Every execution of the ISR should alternate between the two values. You will also need a global variable to index through the Pulse_Buf[] array.

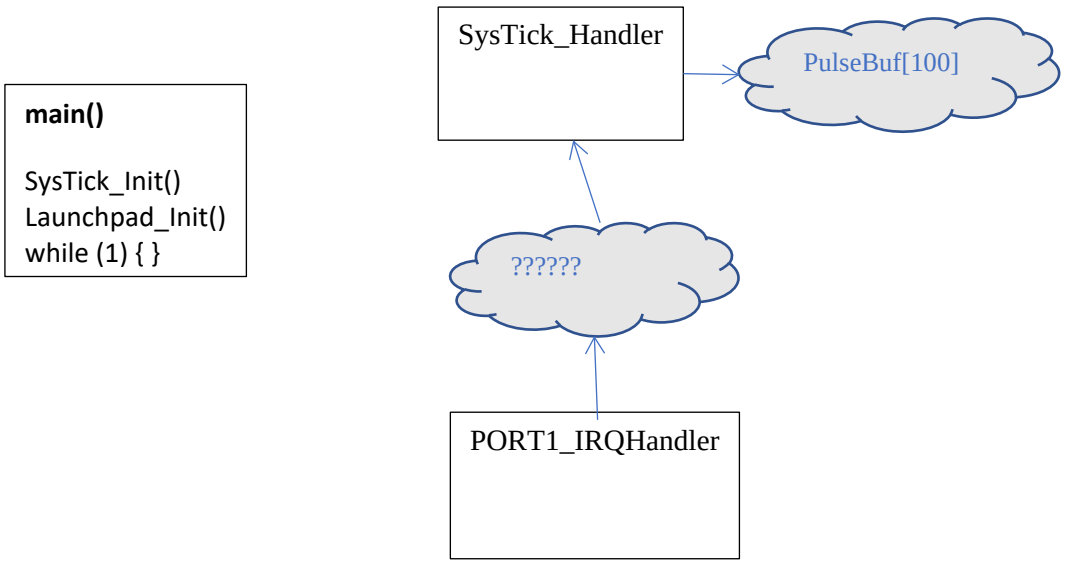
`

Part D: Multiple Interrupt Sources

- *Your solution must be stored in a project and program called Lab5d*

Let's return to TI's Lab 9. You are to now complete **Section 9.4.3** using your interrupt-driven solution. The described use of the Launchpad's switches must also be interrupt-driven.

- You are welcome to re-use the code from our previous lab on interrupt-driven GPIO, merging it into your existing code.
- As in Part C, part of the solution will lie in the introduction of another global variable that encodes the current state of the heartbeat, either STARTED or STOPPED. This global variable should be written by the switch's PORT1_IRQHandler and read by the SysTick_IRQHandler (which uses it within another if-statement). It will act as a SEMAPHORE between the two ISRs.



`

Marking Scheme: Total = 6 + + 5 + 7 = 18 marks

Part A: No submission needed. For learning only.

Part B: Total = 6

Demonstration: 1 mark

- Red LED should be blinking at varying intensity, repeatedly

Inspection: 1 mark each, Total = 5

1. The code uses DRA-style code.
2. Code polls the SysTick's COUNT bit to implement the delays
3. The code is well-structured, using the instruction from Section 9.4.1 (SysTick_Init and LaunchPad_Init)
4. The Pulse_Buf[] array is used, and an counter is used to index through the array to set the SysTick delays at varying values.
5. Overall Style

Part C: Total = 5 marks

Demonstration: 1 mark

- The behaviour should be similar to Part B, perhaps at a different rate but with varying intensities.

Inspection: 1 mark each, Total = 4

1. Installation of SysTick ISR, including local enable of SysTick interrupts
2. The main() loop is completely empty
3. The ISR contains no loop, and only if-statements based on a global variable that records whether the delay is for a HIGH pulse or a LOW pulse
4. Overall Style – NOTE: Program must be a CLEAN merge of the 2 demo programs (InputOuput and Interrupt)

Part D: Total = 7

Demonstration: 1 mark

- Left alone, the behaviour should be identical to Part C. When switches are pressed, the LED sequence should start/stop.

Inspection: 1 mark each, total = 6

1. Installation of switch's ISR
2. Additional code within Launchpad_Init() to configure the switches
3. The main() loop is completely empty
4. The switch's ISR is a single if-statement that sets/clears a global variable based on which switch was pressed
5. The SysTick ISR is largely the same, except for an outer if-statement that enables the logic only when the heartbeat is ON (by reading the switch's global variable)
6. Overall Style – NOTE: Program must be a CLEAN merge of the 2 demo programs (InputOuput and Interrupt)

Overall marks for style (To be used for all labs in this course):

`

- Comments, indentation, and well-named functions and variables
- Removal of all extra code (no commented out sections, no unused code leftover from some other example)