`

SYSC3310 Lab 7 – Using the ADC on the MKII's Joystick

Fall 2018

Objectives:

- ADC using the Joystick with Valvano's BSP.
- Interrupt-driven ADC, by evolving the BSP into Interrupt-driven BSP
- Another practice in GPIO Interrupts using the MKII's Joystick's Select button
- Demonstrating Learning Transfer using the Accelerometer

Equipment:

- MSP432 P401R LaunchPad Development Kit
- Educational BoosterPack MKII – A peripheral board of sensors and actuators

References and Reading Material

- Valvano's Board Support Package (BSP) – Posted on CULearn
- Demonstration Program: **Demo-MKII-JoystickandSysTick-PolledADC**

Part A: First experience of the Joystick using an existing program

- *Create a project called Lab7a. No submission needed (because it is a demo program)*

Copy and build the demonstration program covered in class, called **Demo-MKII-JoystickandSysTick-PolledADC**. When running this program, you will be able to use your joystick with its <x,y> settings displayed in real-time on the LCD as well as the number of times that you push down on the joystick (known as the **SELECT**).

To demonstrate that you have not only run the program, but have taken the time to study the code, answer the following questions:

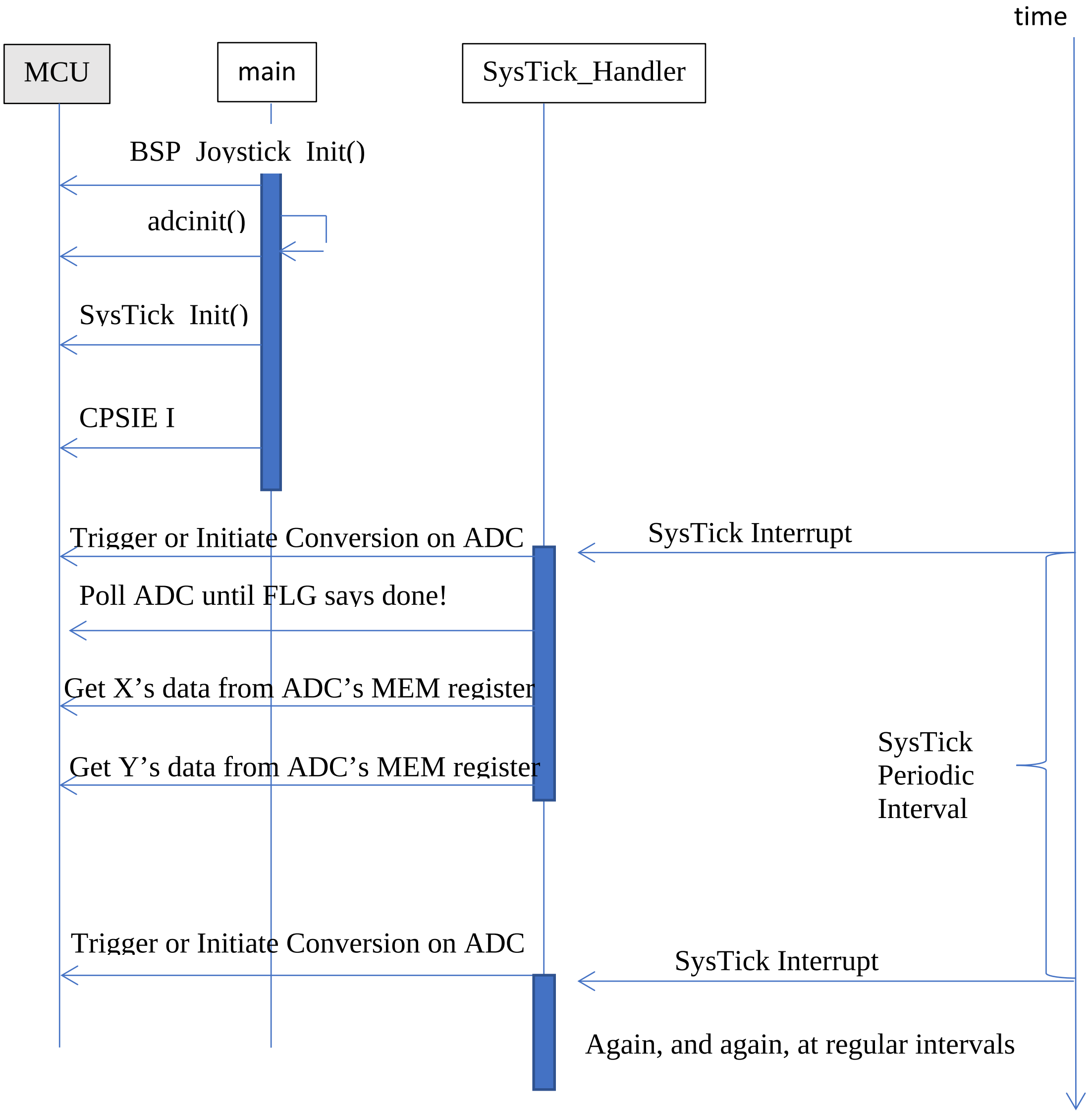
1) How many conversion bits is the ADC programmed for? Look in `BSP_Joystick_Init().`
2) What are the default origins of X,Y (when you aren't touching the controller)?
3) What are the minimum (left) and maximum (right) values of x?
4) What are the minimum (down) and maximum(up) values of y?
5) There's a SysTick ISR in the program, yet the name of the program says that there is **polledADC**. Explain how you can have both interrupts and polling in this same program.
6) What is the first line of code in BP_`Joystick_Init()`? It is a call to a function: `adcinit`(). It is a `static` function.
   a. Why is this function declared as `static`? What does `static` mean in C?
   b. Where else is this function called? What is shared in common with all the code that calls this function?

`

Part B: Using ADC interrupts

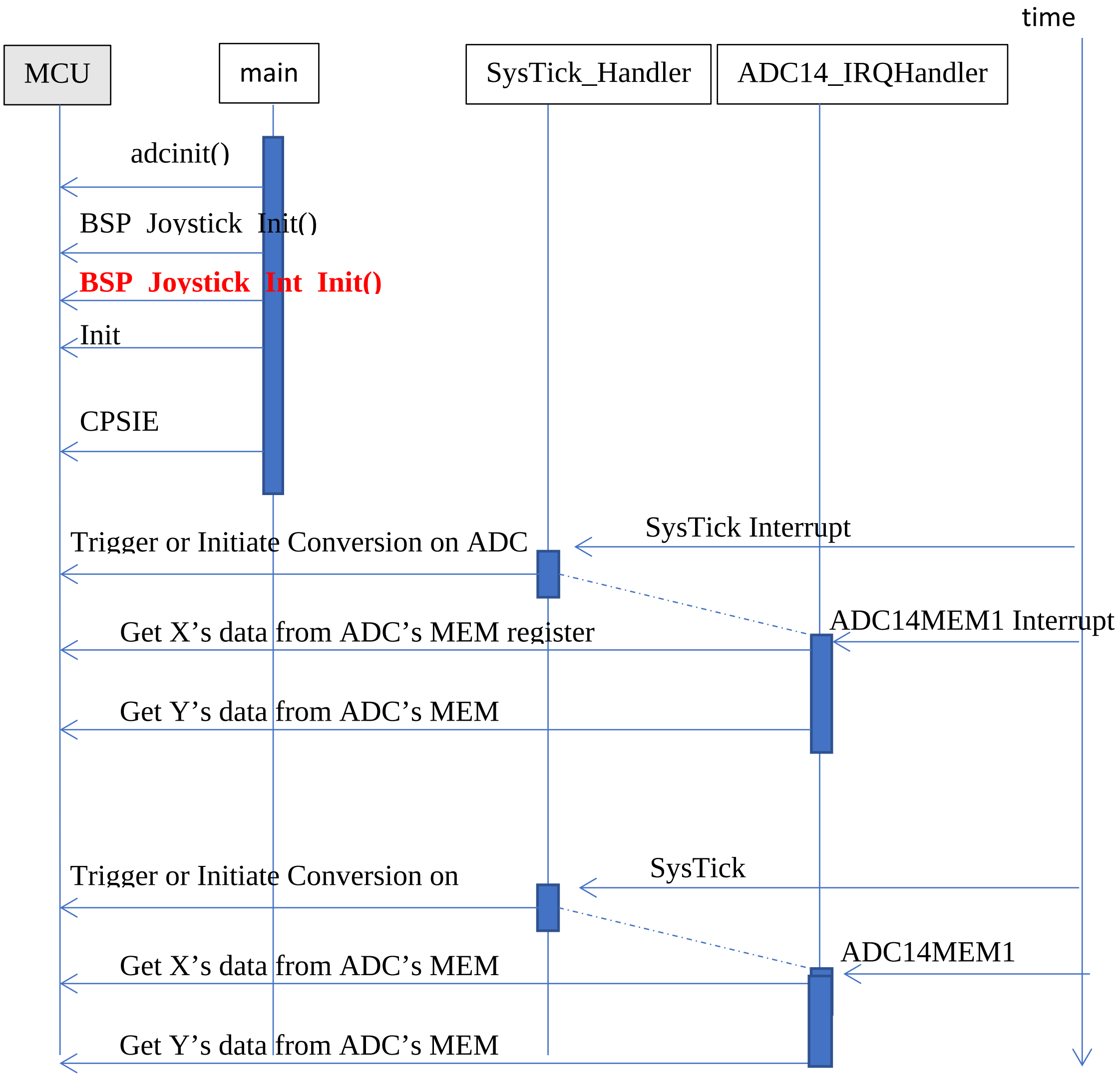
- *Use a project called Lab7b. Submit only the .c file*

The thread diagram below shows the timing analysis for the polled version that you just ran. Take the time to learn how to read these sequence diagrams. main() is responsible only for initialization; after that, you don't see main() execute anymore (no more thick blue line). SysTick_Handler then runs periodically (shown twice in this diagram) in response to a SysTick Interrupt. During each execution, SysTick_Handler initiates an ADC conversion AND waits until it is complete in order to display the updated <x,y> values.


time
MCU
main
SysTick_Handler
BSP Joystick Init()
adcinit()
SysTick Init()
CPSIE I
Trigger or Initiate Conversion on ADC
SysTick Interrupt
Poll ADC until FLG says done!
Get X's data from ADC's MEM register
Get Y's data from ADC's MEM register
SysTick
Periodic
Interval
Trigger or Initiate Conversion on ADC
SysTick Interrupt
Again, and again, at regular intervals

`

The second thread diagram below shows the timing analysis for the interrupt-driven version that you must now develop – your code's execution must match this diagram. The `SysTick_Handler` now initiates an ADC conversion but it does not wait around until the conversion finishes; instead it simply initiates the ADC conversion ***sequence***. When that conversion ***sequence*** is complete, the ADC14 interrupt will occur and that will launch the ACD14_IRQHandler (if you've installed it correctly). Your ACD14_IRHandler then reads the ready <x,y> values and displays them.


time
MCU
main
SysTick_Handler
ADC14_IRQHandler
adcinit()
BSP Joystick Init()
BSP Joystick Int Init()
Init
CPSIE
Trigger or Initiate Conversion on ADC
SysTick Interrupt
ADC14MEM1 Interrupt
Get X's data from ADC's MEM register
Get Y's data from ADC's MEM
Trigger or Initiate Conversion on
SysTick
ADC14MEM1
Get X's data from ADC's MEM
Get Y's data from ADC's MEM

`

The following steps will lead you through the development of the interrupt-driven version of the program. The steps assume that you have created a new project called **Lab7b** which is a copy of the working program you executed in Part A. The steps work incrementally. Feel free to try it by yourself.

1. Prepare! Before you can write code, you must know what hardware you are using. **Using both the TRM, the sample code in BSP.c as well as the lecture notes**, answer the following questions and keep the answers handy.

    - For both the joystick's **x** and **y**, what are the two analog channels in use?
    - For the joystick's SELECT button, what digital port is in use?
    - For both the joystick's **x** and **y**, to what are the two ADC14MEMn buffers are these analog channels mapped? These two ADC14MEMx buffers will be converted together in one sequence.
    - Within any Keil project, click on **Device** and open up startup_msp432p401r_uvision.s startup file.
        - What is the **IRQ level** of the ADC14 interrupt? You will need this to program the NVIC.
        - What is the required name of the ADC14 ISR?
        - While you are here, what is the required name of the ISR for the joystick's SELECT button, and what is its IRQ level?

2. We need to configure the ADC14. For the most part, the configuration is the same, so you still want to call **BSP_Joystick_Init**() (from Valvano's BSP.c package). However, we now also want to enable ADC14 interrupts. For this, we will write an additional function called **BSP_Joystick_Int_Init()** in our project. Within this function, we simply add the lines of code to enable interrupts on the ADC14:

    - Locally enable the ADC14 Interrupts (i.e. using ADC14IERx)
    - Enable NVIC interrupts for the ADC14 interrupt (i.e. using NVIC_ISER0/1)
    - Set the NVIC priority level for the ADC14 interrupt (i.e. using NVIC_IPRx)
    - Repeat these three steps for the joystick's SELECT button.

    Remember to write "friendly" code so that you don't undo any of the configuration of the ADC14 done in the original function BSP_Joystick_Init().

    Within main(), add a call to your new function **BSP_Joystick_Int_Init**().

2. We need to write and install the two ISRs. To begin, your goal will be just to prove that the ISR is installed correctly and all of your programming for the previous step is correct. You want to see interrupts happen! To begin, write **empty ISRs** for both the ADC14 interrupt and for the SELECT button's ISR.

    - *With empty ISRs, you MAY be able to only generate one occurrence of each type of interrupt. That's okay. To get multiple occurrences, see the next step.*

`

- You can immediately test the installation of the SELECT button's ISR. Run the program using the debugger, putting a breakpoint on the first line in the button's ISR. Push on the SELECT button. If the breakpoint is reached, you know that your configuration is correct. If the breakpoint is not reached, you must go back to step 2 and check your programming at each minute detail. Good luck.
- To test the installation of the ADC14's ISR, you have to first modify the original SysTick_Handler. Instead of initiating a conversion, waiting and then displaying the results, the SysTick_Hander should simply initiate a conversion. To do so, it must (1) set the STARTADD (the first memory buffer to be converted) and (2) enable interrupts. The code is given for you (assuming you've programmed the correct MEMx buffer)

```
ADC14CTL1 = (ADC14CTL1&~0x001F0000) |  (0 << 16);
ADC14CTL0 |= 0x00000003;
```

3. You are ready for this next stage, only if you have proven in the previous step – using debugging breakpoints – that both ISRs will run if triggered. The next small step is to write the most-basic ISR. All ISRs must minimally clear the hardware flag that triggered their interrupt.
    - For the SELECT button's ISR, you must clear the appropriate bit in the PxIFG. If you need help, study the sample code for Demo-Interrupt-GPIO.
    - For the ADC's ISR, you must clear the appropriate bits in the ADC14IFGR0 register. Conveniently, these flags are automatically cleared if you simply read the associated ADC14MEMx buffers (i.e. you read the converted results). For example:

```
uint16_t dummy;
dummy = ADC14MEMn    (where n is the buffer number you are using
dummy = ADC14MEMn+1  (because you must clear all flags in the
sequence!)
```

Now, you should be able to see multiple interrupts generated, time and again. When you hit the breakpoint, hit RUN again to let the program run to the next interrupt.

4. You are now ready to complete the project. Complete both ISRs so that they update the LCD with their data. Your final program should work identically to the polled version in Part A.
    - The ISR should only print out the updated values for x and y. Print out the header information once, in the main(), as part of the initialization ritual.

Example: x = 555 The header string "x = " can be printed once, during initialization. The value 555 is the updated value, printed within the ISR.

Discussion: Do you get one interrupt or two, per tick? Two – Channel 0 then 1, then 0 and 1, then 0 and 1

`

PART C: Using the Accelerometer

- *Use a project called Lab7c. Submit only the .c file*

Make your own demonstration program for the accelerometer so that the LCD displays the current values of <x,y,z> of the accelerometer. The easiest way to do this is to make a copy of your program for Part B and modify it, changing all references of JoyStick to Accelerometer. This part essentially tests whether you understood what you did in Part C (and whether you can repeat it without help).

Marking Scheme: Total Marks of 19

Part A: No demonstration or inspection is needed, because it is simply running a demo program. The TA may however ask the given questions during any of the demos of the other parts.

Part B:

Demonstration: (3 marks)

- 0 : Not working
- 1: The program runs, but consists only of the empty or minimal ISRs. Some of the incremental steps have been achieved but the program is incomplete.
- 2: Works as described, with either the <x,y> or the SELECT button being displayed in real time.
- 3: Works as described, with both <x,y> and the SELECT button being displayed in real time

Inspection: 1 mark each (0, ½ or 1), for a total of 7

1. Structure of program follows interrupt-driven architecture (all code is within **three** ISRs and global/static variables, except for the initialisation; main loop is empty).
2. BSP_Joystick_Int_Init() contains all code related to the installation and enabling of interrupts – local enable, NVIC enable and priority.
3. The SysTick_Handler contains only code for initiating ADC conversions on the joystick, using the proper STARTADDR (All extraneous code is removed)
4. The SELECT ISR uses a global/static variable to count – and display – the number of presses. The ISR only prints the changing portion of the LCD display.
5. The ADC ISR accesses MEM<0,1> to get the <x,y> results and displays it on the LCD. The ISR only prints the changing portion of the LCD display.
6. Main() prints the static portion of the LCD messages once, during initialization, and then does nothing.
7. Overall Style – As usual

Part C:

Demonstration: 2 marks

- 0 : Not working
- 1: The program runs, but consists only of the empty or minimal ISRs. Some of the incremental steps have been achieved but the program is incomplete.
- 2: Works as described, with the <x,y,z> being displayed in real time.

`

Inspection: 1 mark each (0, ½ or 1) Total of 7 marks

1. Structure of program follows interrupt-driven architecture (all code is within **two** ISRs and global/static variables, except for the initialisation; main loop is empty)
2. BSP_Accelerometer_Int_Init() contains all code related to the installation and enabling of interrupts – local enable, NVIC enable and priority. No extra code related to the joystick is left over. Only code for the accelerometer. Check the value used for ADC14IER0.
3. The SysTick_Handler contains only code for initiating ADC conversions on the accelerometer, using the proper STARTADDR (All extraneous code is removed)
4. The ADC ISR accesses MEM<2,3,4> to get the <x,y,z> results and displays it on the LCD. The ISR only prints the changing portion of the LCD display.
5. Main() prints the static portion of the LCD messages once, during initialization, and then does nothing.
6. Overall Style

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)
- Friendly code when configuring peripherals