EE128: Data Acquisition, Instrumentation and Process Control
University of California, Riverside

# Lab 3: Interrupt Handling and ADC

Fall 2019

## Objective

To verify concept and to become familiar with interrupt-handling techniques for the Kinetis K64F microcontroller, to improve skills on designing and debugging microcontroller-based systems, and to get familiar with the concept of A/D conversion and 7-segment displays.

## References

- Lecture 3, 4, 5 (on piazza)
- FRDM K64F Board User Guide: https://os.mbed.com/media/uploads/GregC/frdm-k64f_ug_rev0.1.pdf
- K64F MCU User Guide: https://os.mbed.com/media/uploads/GregC/k64f_ds_rev6.pdf
- K64F MCU Reference Manual: (MOST IMPORTANT RESOURCE) https://community.nxp.com/servlet/JiveServlet/download/581169-1-364797/K64P144M120SF5RM.pdf
- K64F Errata: http://cache.freescale.com/files/32bit/doc/errata/FRDMK64F_ERRATA.pdf

## Equipment

- PC running MS Windows
- Oscilloscope
- Digital Multi-Meter (DMM)
- FRDM K64F development board
- Function generator
- **Breadboard** (you need to bring one)

## Parts

- 2 each 7-segment display, common cathode
- 1 each 4-bit DIP switch (piano type)
- 2 each 390Ω x 8, independent resistor network (DIP)
- 1 each 10 kΩ potentiometer (or equivalent to create an adjustable voltage divider circuit)

## Software

- Freescale Kinetis Design Studio 3.2.0 (IDE)

## Background

### *Interrupt for GPIO*

For this lab, you will configure a GPIO pin to trigger an interrupt. GPIO pins can trigger interrupts with high input signal, low input signal, falling edge, rising edge, or either edge triggers. In this lab, we want to trigger our interrupts on a **falling edge**. To enable the GPIO pin to trigger an interrupt on a falling edge, you must configure PORTx_PCRn. For example:

```
PORTA_PCR1 = 0xA0100; /* Configure PORTA Pin 1 for GPIO & Interrupt on Falling-Edge */
```

Then to enable interrupts from Port A, you must unmask it with the following statement:

```
NVIC_EnableIRQ(PORTA_IRQn);
```

> **Note:** to figure out what this function does, right click it from the source code in KDS and press 'Open Declaration'. Then where it says NVIC->ISER, right click ISER and open declaration. Then find the corresponding register in the datasheet. Also look at NVIC (Nested Vector Interrupt Controller in the datasheet and the Cortex-M4 reference manual in piazza). Knowing what these are will help your understanding of interrupts.

Once this is set up, a falling edge on Port A Pin 1 causes a hardware interrupt to Kinetis K64F. A user *interrupt service routine* (ISR), specifically written for the Port A interrupt, will be invoked. The address of this routine should be stored in the Port A interrupt vector, specifically 0x0000_012C - 0x0000_012F in the Kinetis K64F addressing space. If the ISR is named following the CMSIS rules, e.g., **PORTA_IRQHandler()** for Port A ISR, the linker configures the interrupt vector automatically. Otherwise, you have to register the ISR address to the interrupt vector manually. See below for example code structure:

```
void main(void) {
        SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK; /* Enable Port A Clock Gate Control */
        PORTA_PCR1 = 0xA0100; /* Configure Port A pin 1 for GPIO and interrupt on falling edge */
        PORTA_ISFR = (1 << 1); /* Clear ISFR for Port A, Pin 1*/
        NVIC_EnableIRQ(PORTA_IRQn); /* Enable interrupts from Port A*/

        < main code (loop) >
}

void PORTA_IRQHandler(void)
{
        < interrupt code >
}
```

### *ADC*

In this lab, we will be working with the Kinetis K64F A/D converter. Please read the appropriate background material on this module, like the Lecture Note on Analog to Digital Conversion and the relevant sections of the K64F MCU reference manual.

## Lab Exercise

In this lab, we will design and implement an interrupt-driven two-digit display for bi-directional counter and ADC output on the FRDM K64F board:

1) A "clock tick" circuit, which is conceptually a flip-flop Q driven by the hardware signal CLK. This clock tick will be displayed using a single ***decimal point*** on a 7-segment display. For every clock tick received, the decimal point should pulse on and off (Use set up in Figure 1).

2) A two-digit bi-directional counter and ADC output display (MAX 3.3V) driven by the same CLK signal will be created. The operation of the display is determined by the following two digital signals:

- MODE_SW: 0 for ADC mode (read voltage from ADC and display)
  1 for counter mode (update a counter and display)
- CNT_DIR: 0 for upward counting (count up until 99 and roll over to 0 after 99)
  1 for downward counting (count down until 0 and roll over to 99 after 0)
  CNT_DIR is effective only when MODE_SW == 1 (counter mode)

The value of the counter or ADC output will be displayed using two 7-segment LEDs. See Figure 1.

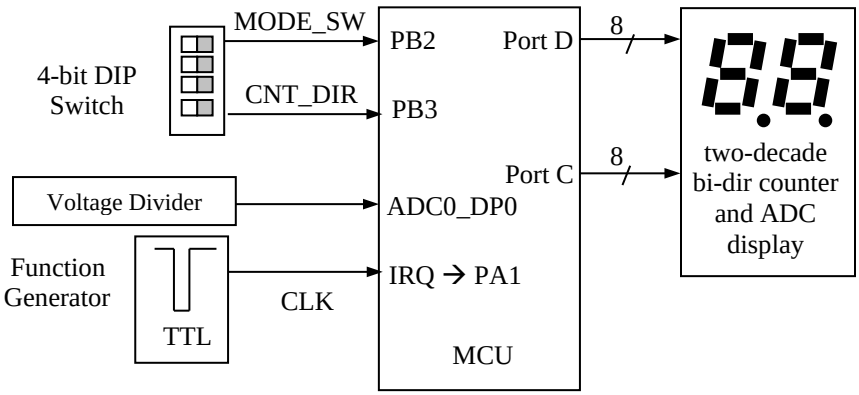


**Figure 1**

## Design Considerations

**DO NOT APPLY MORE THAN 3.3V TO ADC INPUT OR ANY GPIO PIN. THIS MAY PERMANENTLY DAMAGE THE BOARD!**

### *Resource Allocation*

- The seven segment displays can be driven through Ports C and D. Make sure you use current limiting resistors.
- The control signals, MODE_SW and CNT_DIR, can be read in through Port B. In Figure 1, PB2 means Port B Pin 2 and PB3 means Port B Pin 3.
- The external CLK can be introduced into the system through any GPIO pin, if configured to generate an interrupt request. For this lab, we will use **Port A Pin 1** (bit 1, not bit 0) to generate an interrupt.
- We will create a simple voltage divider circuit using the potentiometer, and then measure the voltage with the ADC module on the K64F. The ADC module should be configured for a range of 0 to 3.3 volts. Dedicated power supply and ground are required for the ADC in order to isolate the

sensitive analog circuitry from the normal levels of noise present on the MCU. Use ADC channel DAD0 which is connected to **ADC0_DP0** pin on the dev board.

### *Programming*

Make sure you are able to write the following programs based on your hardware resource allocation:

- Update and display the clock tick (decimal point of 7-segment display);
- Update and display the ADC voltage based on MODE_SW;
- Update and display the counter value based on MODE_SW and CNT_DIR;

The control program contains three parts:

#### *1. Configuration and initialization*

This part of the program sets up the interrupts and ADC.

#### *2. Interrupt service routine (ISR)*

Invoked upon each interrupt arrival, the ISR performs the following operations: (1) flips the conceptual flip-flop for the clock-tick, (2) read from Port B (where your switches are), (3) read from the ADC or update the counter based on switch input, and (4) display the voltage or the counter value.

In summary, your ISR should implement the following pseudo-code:

```
void PORTA_IRQHandler(void)
{
        Toggle ones place decimal point;
        Read Port B
        if (MODE_SW is 0)     /* ADC Mode */
              Read from ADC and convert to decimal value; (e.g., ADC reads 0xFF, voltage is 1.6)
        if(MODE_SW is 1)      /* Count Mode */
            if (CNT_DIR is 0)   /* Count Direction */
                count up to 99 and roll over to 0
            if(CNT_DIR is 1)
                count down to 0 and roll over to 99
        Display ADC value or Counter value based on MODE_SW; /* PORT C and D to seven segments */
        Clear ISFR
}
```

#### *3. Main program*

The functionality of the main program after initialization is to do nothing. But your configuration/initialization code should resemble the following:

```
void main (void)
{
          Enable Port A, B, C, D, and ADC0 clock gating.
          Configure PA1, PB[3:2], PC[8:7, 5:0], PD[7:0] for GPIO.
          Configure PA1 to trigger interrupts on falling edge input.
          Clear PORTA ISFR
          Configure ADC for 16 bits, and to use bus clock.
          Disable the ADC module;
          Set PB[3:2] for input;
```

```
        Set PD[7:0] and PC[8:7,5:0] for output;
        Enable Port A IRQ interrupts;
        for (;;){}
}
```

Some **important** tips for this lab:

- Read the lecture slides for GPIO, interrupts and ADC. You may use example code in those slides.
- Read the datasheet resources listed in the slides.

## Pre-lab

1. Draw a schematic diagram for the experiment.
2. Write the required C program code for the experiment.

*Do you want to complete the experiment during lab hours?* Draw a schematic diagram and write program code before attending the lab session.

For the 7-segment display, check the pin assignment first.

## Procedures

1. Design the required circuit for the experiment.
2. Build your source program(s) with the compiler to generate executable file.
3. Download and debug/run your programs. Continue only if your program works correctly.

## Questions

1. Measure the time duration from the falling edge of the CLK signal to the change of the decimal point (from on to off, or vice versa) on a 7-segment display. (Use oscilloscope)
2. For this lab, we use a falling-edge generated interrupt. If you use level-sensitive interrupts, what extra steps (in hardware and/or software) would you need? (don't write the entire code or schematic; answer in a brief manner)
3. What are the factors contributing to the interrupt latency? Answer briefly.
4. What is the resolution (minimum distinguishable input voltage) of the ADC implemented in this lab?

## Lab Demo (50 points)

Demonstrate your working system to the TA and get a confirmation of completion.

Check if the system works as expected according to MODE_SW and CNT_DIR settings. The update rate of the decimal point should change with the function generator's frequency. For ADC, change the potentiometer by hand.

## Lab Report (50 points)

Make sure you include the following in your report:

1. Abstract (a short paragraph stating the objectives and accomplishments)
2. Experiment system specification (what has been designed and implemented) – **10 points**
   - Flowchart diagram (show how your system works)
   - Photos of your boards and circuits
3. Hardware design – **10 points**
   - Draw schematics of your own; do not copy and paste from the handout
4. Software design – **10 points**
   - High level description of the software
   - Program listing (including comments)
5. Technical problems encountered, and how they are solved
6. Answers to the questions – **10 points**
7. Conclusion
   - Very short concluding remark
   - Quick summary of the contributions of each member