EE128
Interrupt Handling and ADC

Lab Report #3

Ana Cardenas Beltran
Alison Gonzales

Abstract

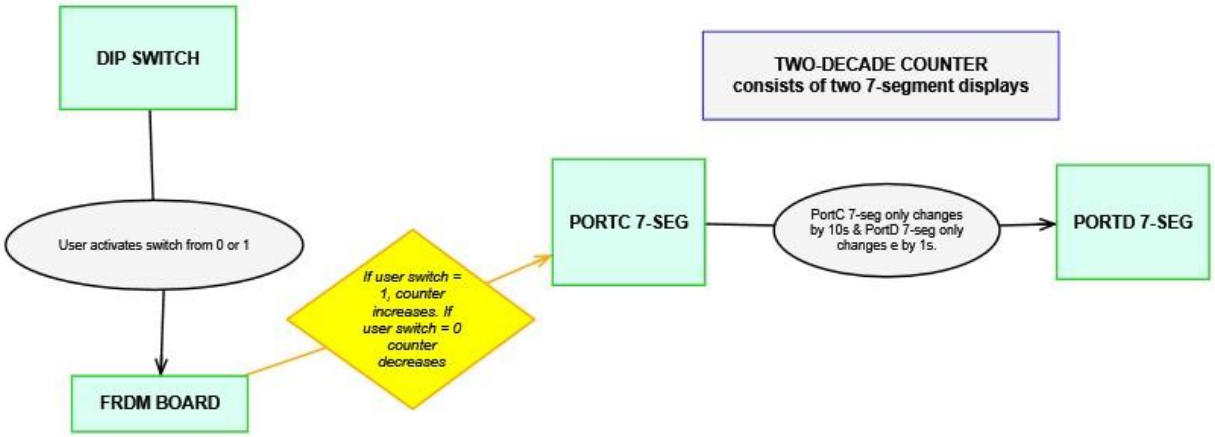
The objective of this lab is to configure a GPIO pin to trigger various interrupts - high input signal, low input signal, falling edge, rising edge, or either edge triggers. This lab specifically tests the falling edge interrupt by enabling the GPIO pin using the configuration PORTx_PCRn. All of these will be tested using the Kinetis K64F A/D converter.

Students accomplished the necessary skills to build an analog to digital converter circuit with various interrupts using the Kinetis K64F. By utilizing the system clock of the FRDM board, students successfully created an interrupt-driven two-digit display.

Experiment System Specification

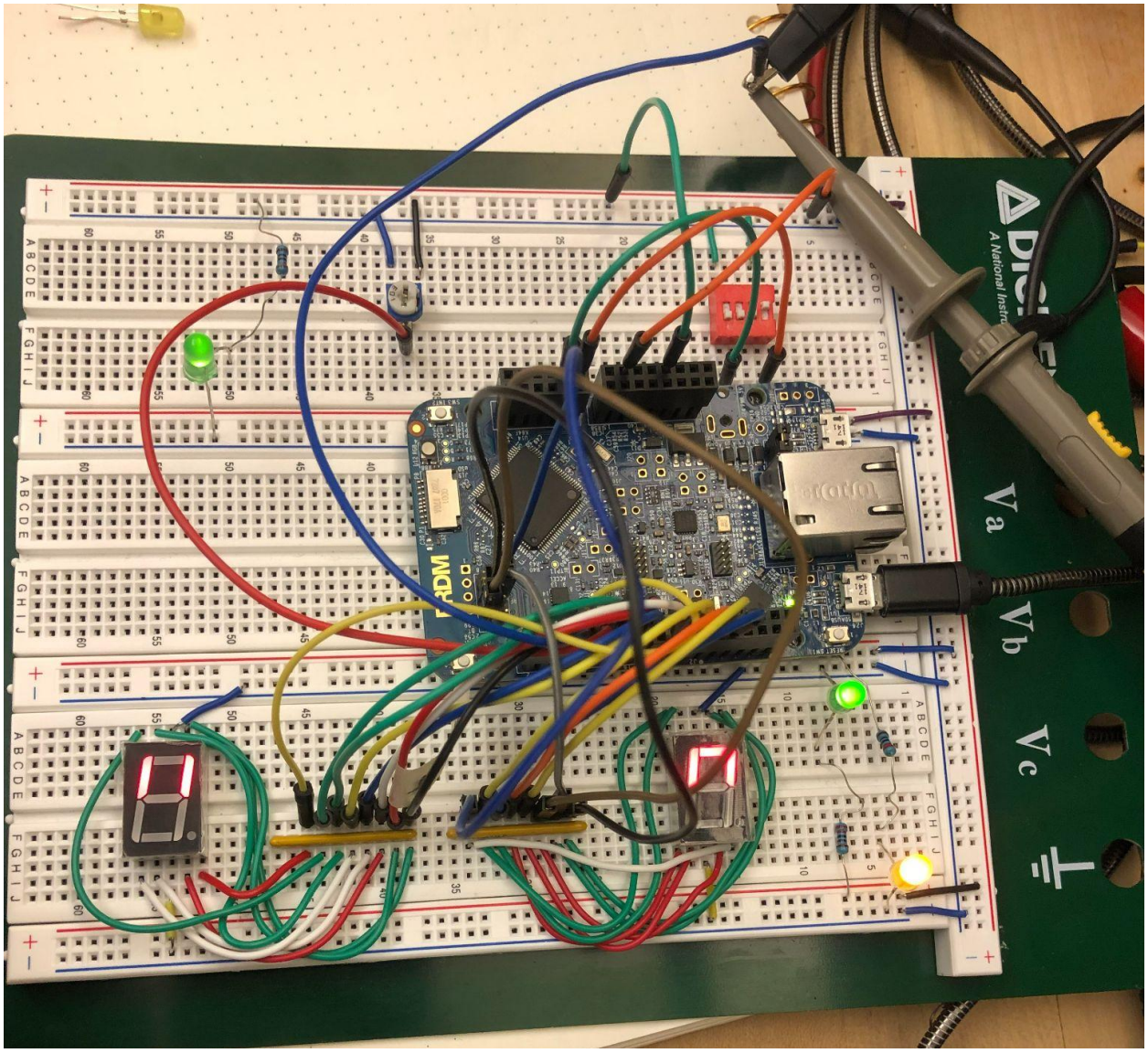
In this lab, Freescale Kinetis Design Studio (KDS) 3.2.0 IDE was used as the main work environment for programming the Kinetis K64F development board to create an interrupt-driven two-digit display. Multiple circuit elements: 7-segment display, common cathode, 4-bit DIP switch (piano type), 390 ohms x 8 independent resistor network (DIP), and 10k Ohms were used to test the ADC output on the FRDM board. The CLK signal is configured based on the two-digit bi-directional counter and ADC output display using a voltage maxim of 3.3V.

Flowchart Diagram

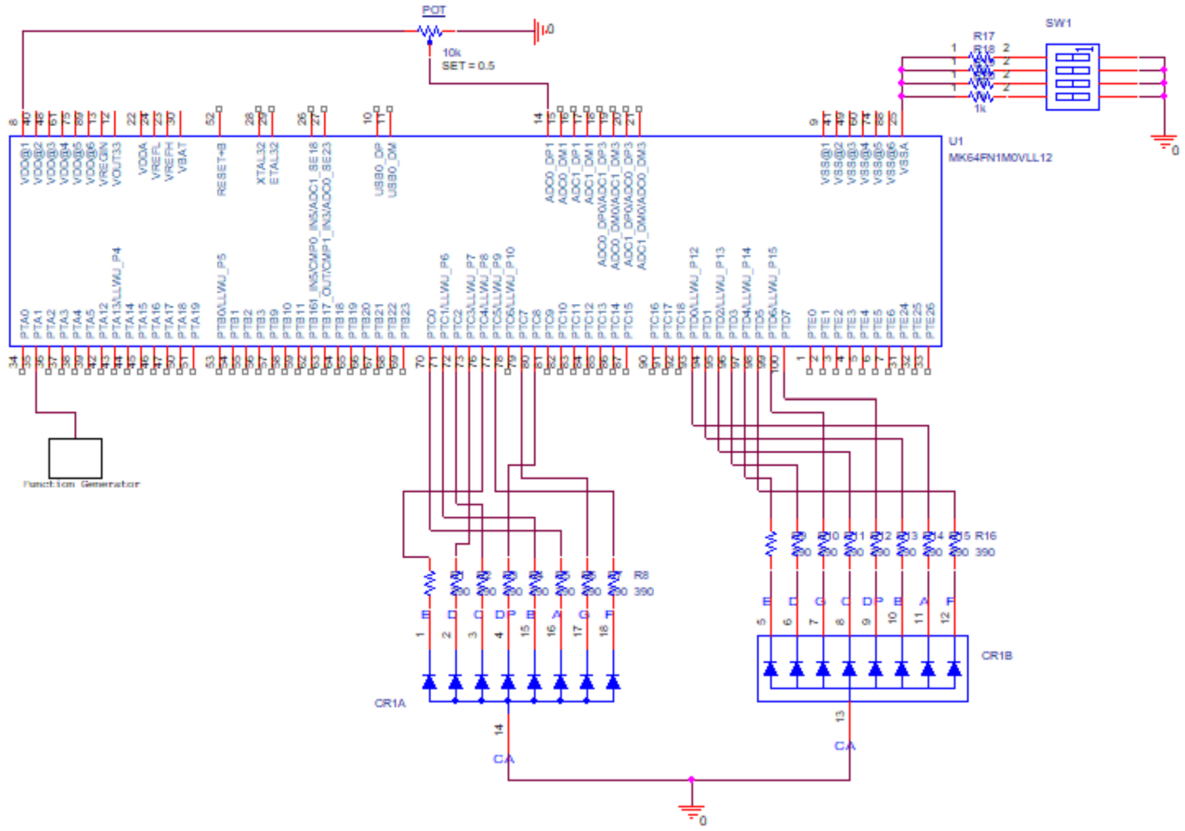


Boards & Circuits

Power Supply was not used in this circuit.



Hardware Design



Software Design

```
/*high level software description:
This code detects a 0 or 1 from the DIP Switch connected to PORTB and based on that, either
counts up or counts down on 2 7-segment LEDs, connected to PORTD and PORTC. We implemented an
interrupt service routine (ISR) that allows our hardware to trigger interrupts.
*/

/*
 * EE128 Lab 3
 * Ana Sofia Cardenas
 * Alison Gonzales
 *
`*
 * Copyright (c) 2015, Freescale Semiconductor, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
```

```
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of Freescale Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include "fsl_device_registers.h"

unsigned int i = 0;
long MODE_SW, CNT_DIR;
unsigned int counter1, counter2 = 0; //counter1 -> 7seg (PortC) ; counter2 -> (PortD)
unsigned int val1, val2 = 0;

unsigned char decoderD[10] = {
               0x7E, 0x30, 0x6D, 0x79, 0x33, 0x5B, 0x5F, 0x70, 0x7F, 0x7B }; //hex values from 0
to 9 on 7 seg display
unsigned char decoderC[10] = {0xBE, 0x30, 0xAD, 0xB9, 0x33, 0x9B, 0x9F, 0xB0, 0xBF, 0xBB};

unsigned short ADC_read16b(void);
void PORTA_IRQHandler(void);

void main(void)
{
       SIM_SCGC5 |= SIM_SCGC5_PORTA_MASK; /*Enable Port A Clock Gate Control*/
       SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK; /*Enable Port B Clock Gate Control*/
       SIM_SCGC5 |= SIM_SCGC5_PORTC_MASK; /*Enable Port C Clock Gate Control*/
       SIM_SCGC5 |= SIM_SCGC5_PORTD_MASK; /*Enable Port D Clock Gate Control*/
       SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK;  // 0x8000000u; Enable ADC0 Clock

       //Configure PA1,  PB[3:2],  PC[8:7, 5:0],   PD[7:0] for GPIO.
       PORTB_GPCLR = 0x000C0100;
       PORTC_GPCLR = 0x01BF0100;
       PORTD_GPCLR = 0x00FF0100;

       PORTA_PCR1 = 0xA0100; //Configure PA1 to trigger interrupts on falling edge input.
```

```
	PORTA_PCR4 = 0x100;

	ADC0_CFG1 = 0x0C; //Configure ADC for 16 bits, and to use bus clock.
	ADC0_SC1A = 0x1F; //Disable the ADC module;

	GPIOB_PDDR = 0x00000000;	//Set PB[3:2]for input;
	GPIOA_PDDR |= (0 << 1);		//Configure PortA, Pin1 for input
	GPIOD_PDDR = 0x000000FF;	//PORTD [7:0] output
	GPIOC_PDDR = 0x000001BF;	//Configure Port C Pins 0-5 and 7-8 for Output;  //0001 1011
1111 <- PINS 0-5, 7-8

	GPIOC_PCOR = 0xFF; // clears output on PortD[0:7]
	GPIOC_PSOR = 0x1BF;

	PORTA_ISFR = (1 << 1); //Clear PORTA ISFR, Pin 1
	NVIC_EnableIRQ(PORTA_IRQn);

	for (;;){
	}

	return 0;
}

unsigned short ADC_read16b(void){
	ADC0_SC1A=0x00;
	while(ADC0_SC2 & ADC_SC2_ADACT_MASK);
	while(!(ADC0_SC1A & ADC_SC1_COCO_MASK));
	return ADC0_RA;
}

void PORTA_IRQHandler(void){
	NVIC_ClearPendingIRQ(PORTA_IRQn); //Clear pending Interrupt
	GPIOD_PTOR |= 0x80; //Toggle decimal point
	unsigned int data = 0;

	CNT_DIR = GPIOB_PDIR & 0x08; // PORTB3
	MODE_SW = GPIOB_PDIR & 0x04; // PORTB2

	if (MODE_SW == 0x00) {
		//Read from ADC and convert to decimal value; e.g ADC reads 0xFF, voltage is 1.6
		data = ((unsigned int)ADC_read16b()*33)/(0xFFFF);
		val1 = data / 10; //gets 10s place
		val2 = data % 10; //gets 1s place

		GPIOC_PCOR = 0xFF; //clear output on PORTC
		GPIOC_PSOR = (unsigned int) decoderC[val1]; //print the 10s place

		GPIOD_PCOR = 0xFF; // clears output on PortD[0:7]
		GPIOD_PSOR = (unsigned int) decoderD[val2]; //print the 1s place


	}else if (MODE_SW == 0x04) { //if (MODE is 1) ADC Mode
		if (CNT_DIR == 0x00) {
```

```
                        //if (CNT_DIR is 0) Count Direction count up to 99 and roll over to 0
                        if ((counter2 == 9) && (counter1 < 9)) { // PD 7-seg is in ones, PC 7-seg
is in 10s
                                counter1++; //only increase when counter2 reach 9;
                                counter2 = 0; //counter2 resets to 0;
                        }else if (counter2 == 9 && counter1 == 9)  { //if 99, reset to 0
                                counter1 = 0;
                                counter2 = 0;
                        }else { //counter 2 is only increasing because of ones place
                                counter2++;
                        }
                }else if (CNT_DIR == 0x08){
                        // if (CNT_DIR is 1) Count Direction count down to 0 and roll over to 99
                        if (counter2 == 0 && counter1 > 0) { // PD 7-seg is in ones, PC 7-seg is in
10s
                                counter1--; //only decrease when counter2 reach 9;
                                counter2 = 9; //counter2 resets to 9;
                        }else if (counter2 == 0 && counter1 == 0)  { //if 99, reset to 0
                                counter1 = 9;
                                counter2 = 9;
                        }else { //counter 2 is only increasing because of ones place
                                counter2--;
                        }
                }
                //PORTD Display
                GPIOD_PCOR = 0xFF; // clears output on PortD[0:7]
                GPIOD_PSOR = (unsigned int) decoderD[counter2];
                //PORTC Display
                GPIOC_PCOR = 0xFF; //Check PORTC pins availability
                GPIOC_PSOR = (unsigned int) decoderC[counter1]; //access decoder array to display
number in PORTC;

                for (i= 0; i < 1000000; i++); //Delay
        }

        PORTA_ISFR = (1 << 1); //Clear ISFR

}
```

Technical Problems

The biggest problem we encountered was with the 7-segment LED Displays. The 2 7-segment displays we had were not working properly, so we were unable to tell whether our code was functioning or not. We ended up connecting the 7-segment displays separately by connecting com to ground and then adding voltage to each pin to see if they would light up. From this, we found that the 7-segment displays

were not working. We ended up receiving 2 more 7-segment displays but they also were not properly working. Only the top half of the displays was working. Another problem we encountered was we tried using a power supply to measure the voltage that was being supplied to the breadboard and we were getting incorrect results. The voltage it was measuring was 6 V when we were only supplying 3 V.

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)*
   The time duration from the falling edge of the CLK signal to change of the decimal point has a 2 ms period.
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)*
   To use a falling-edge generated interrupt, we must use a function generator in hardware. For software we need to use NVIC_EnabledIRQ(PortA_IRQn). By using both these on hardware and software, falling-edge generated interrupt can be used.
3. *What are the factors contributing to the interrupt latency? Answer briefly.*
   Some factors contributing to the interrupt latency are the time imputed by the user. In our case of the two-decade bi-dir counter, the system have a interrupt latency of counting in 10s and 1s. The 7-segment display on the 10s position should not count parallel to the 1s 7-segment display along with a user input controlling the decrease and increase of the counter.. The factors contributing to this feature is that there is time-variance in counting to both 7-segments and also user input.
4. *What is the resolution (minimum distinguishable input voltage) of the ADC implemented in this lab?*
   The minimum resolution was 0.1V.

Conclusion

In this lab, we learned how to use 7-segment LED Displays and how to configure GPIO pins to trigger interrupts. We also became more familiar with A/D converting. We had issues with our 7-segment LED Displays, but we were able to use other methods of debugging with different EE Equipment such as oscilloscope and multimeter.

Alison worked on the code, hardware, and other debugging parts of the code. Sofia worked on setting up the hardware, debugging it, and debugging our code.