Lab 4 - Analog to Digital Converter

Matt Douglas, Priyanka Makin, Zach Passarelli, Savio Tran

ECEN 2020

October 21, 2016

The goal of this lab is to explore analog to digital conversion of electrical signals.This is the first lab where we use some Booster Pack functionalities along with the MSP432 board. We were given the task to start with converting analog signals from a temperature sensor found on the booster pack to digital values. Then we also use analog to digital conversion to output what quadrant the joystick on the Booster Pack is in.

Pre-Questions

a) $V_{resolution} = \frac{V_{ref,\,max} - V_{ref,\,min}}{2^{resolution}} = \frac{3.3 - 0}{2^{14}} = 0.2014\ mV$

b) $Error = \frac{V_{ref,\,max} - V_{ref,\,min}}{2^{resolution + 1}} = 0.1007\ mV$

c) $V_{saturation} = (V_{resolution})(2^{resolution} - 1) = 3.2998\ V$

d) $Temp_C = 0.52 * V_{adc};\ V_{adc} = \frac{N_{adc} * V_{ref}}{2^{resolution}};$

The coefficient 0.52 comes from plotting a line of best fit for sample data found in the documentation for the internal temperature sensor.

Temperature Sensor

To use the temperature sensor, we start with configuring UART communication like we did in Lab 3. We also make sure to include the functions “uart_putchar” and “uart_putchar_n” so that we can transmit our data onto the RealTerm terminal. Then we have to configure the ADC, analog-to-digital converter, as stated in the Lab 4 instructions. We enable a reference voltage of 1.2 V and configure the internal temperature sensor to be read by the ADC. When the reference generator is not busy, the ADC is ready to convert data and then trigger an interrupt.

In the ADC14_IRQHandler we convert the binary temperature readings to decimal values in Celsius, Fahrenheit, and Kelvin and display them on the RealTerm console. In pre-question d) we address how we convert the output of the temperature sensor to a temperature in celsius. We write and implement an int to ascii function because our “uart_putchar_n” function can only transmit a number up to 8 bits so we can’t transmit any temperature readings greater than 255. Essentially, with the int to ascii function we transmit each digit separately over UART guaranteeing no value will exceed 255.

Interfacing the Joystick

To interface the joystick to the MSP432 we first configure the correct ports. Pin 4.4 determines the x-direction and Pin 6.0 specifies the y-direction. In the ADC14_IRQHandler we write a function that transmits a string over UART specifying what quadrant the joystick is in. Since we have a 14-bit ADC resolution we can calculate the boundaries of the quadrants. The maximum in the x and y direction is given by

$$N_{adc} = 2^{resolution} - 1 = 2^{14} - 1 = 16383$$

So we know that the positive x-direction, the right quadrants, go from 8191 to 16383 N_adc. Likewise, we know that the positive y-direction, the upper quadrants go from 8191 to 16383 N_adc. However, the function we write doesn't work as expected, it can only distinguish between left and right but not up and down. The issue may come from the ADC interrupt handler trying to handle and read from two interrupts at once since both x and y directions are constantly making conversions.

Real World Applications

Analog to digital conversion plays an important role in allowing an embedded system to interact with the world around it. In this lab, we already saw that analog to digital conversion can be used in temperature sensors and for joystick controllers, but there is a world of other applications. Many other sensors require analog to digital conversion. For example, the light sensors commonly found in phones. Many smartphones can take an input from a light sensor so that they can adjust the screen brightness according to the phone's environment. This requires converting the analog data of ambient brightness into a discrete value so that the phone can adjust.

Another major application would be music recording. Music is recorded today by sampling different elements of a song, and converting it to a digital file. These are then filtered and compiled to make a song recording. So, analog to digital conversion acts as the interface between the real world and electronics.

Conclusion

This is a great lab to end the material with because it basically brings most of the important embedded systems topics together in one challenging exercise. This lab tests our pin configuration, interrupt enabling and handling, UART communication, circular buffer implementation, and problem solving capabilities.

This lab requires reading a lot of documentation. As we start working with more peripherals, more documentation is required. For example, while the technical manual contained information on pin configuration, we had to go through the temperature sensor document to find a table that provided binary N_ADC values that corresponded to the temperature. Using this allowed us to create a conversion to temperature. Coupling ADC with UART allowed us to print the temperature readings from the temperature sensor in degrees Celsius, Kelvin, and Fahrenheit onto RealTerm.

```c

#include "msp.h"
#include "core_cm4.h"
#include "CircBuff.h"
#include <stdint.h>
#include <stdlib.h>
#define SCB_SCR_ENABLE_SLEEPONEXIT (0x00000002)
#define BOUNDARY (8191) //Define Nadc boundary for joystick

#define temp
#define joystick

void configure_port();
void configure_ADC();
void configure_clock();
void configure_timer();
void configure_serial_port();

void P1_IRQHandler();
void ADC14_IRQHandler();
void TA0_IRQHandler();


void uart_putchar(uint8_t tx_data){ //Single character transmit via UART
    while(!(UCA0IFG & UCTXIFG)); //Wait for transmitter ready
    UCA0TXBUF = tx_data; //Load data onto buffer for transmission
}

//Multi-character transmitter via UART
void uart_putchar_n(uint8_t *data, uint32_t length){
    uint32_t i;
    for(i = 0; i< length;i++){ //Iterate through data
        //Transmit 1 character per iteration using putchar function
        uart_putchar(data[i]);
    }
}

void i2a(int16_t data){ //Int to ASCII converter
        uint8_t length;
        int16_t abdata;
        if(data < 0){ //Absolute value for determining length
            abdata = -data;
        }
        length = 1;
        if(abdata >= 10){
            length = 2;
        }
        if(abdata >= 100){
            length = 3;
        }
        if(abdata >= 1000){
            length = 4;
        }
        if(abdata >= 10000){
            length = 5;
        }

```

```c
        uint8_t digit_data, a[length+1];

        if(data < 0){ //Check if number negative
            a[0] = 45; //Add '-' character
            data = -data; //Absolute value for conversions
        }
        if(data >= 0){ //Number positive
            a[0] = 32; //Add ' ' character (placeholder)
        }
        int i = length;
        while(i>1){
            //Get least significant digit and subtract from data
            digit_data = data % (10^(length-i+1));
            data -= digit_data;
            digit_data /= (10^(length-i)); //Reduce to single digit
            a[i] = digit_data + 48; //single digit to ASCII
            i--;
        }
        a[1] = (data/(10^(length-1))) + 48;

        uart_putchar_n(a,(length+1));
}

void main(void){

    WDTCTL = WDTPW | WDTHOLD;  //Stop watchdog timer
    configure_port();
    configure_ADC();
    configure_clock();
    configure_serial_port();
    __enable_interrupt();

    //Wake up on exit from ISR
    SCB->SCR &= ~SCB_SCR_ENABLE_SLEEPONEXIT;

    while(1){
        ADC14->CTL0 |= ADC14_CTL0_SC; //Start conversion
        __sleep(); //Block until conversion finish
        __no_operation(); //No-operation

    }
}


void configure_port(void){
    //joystick: p4.4(y), p6.0(x)
    P6->SEL0 |= BIT0; //Primary mode
    P4->SELC |= BIT4; //Tertiary mode

}

void configure_ADC(void){
    //Initialize shared reference module
    while(REF_A->CTL0 & REF_A_CTL0_GENBUSY);
    //Enable internal 1.2v ref
    REF_A->CTL0 = REF_A_CTL0_VSEL_0 | REF_A_CTL0_ON;
    //Turn on Temperature sensor
```

```c
    REF_A->CTL0 &= ~REF_A_CTL0_TCOFF;

    //Configure ADC - Pulse sample mode; ADC14SC trigger
    //ADC ON, temperature sample period > 30us
    ADC14->CTL0 |= ADC14_CTL0_SHT0_5 | ADC14_CTL0_ON | ADC14_CTL0_SHP;
    ADC14->CTL0 |= ADC14_CTL0_CONSEQ_3;
    //Configure internal temp sensor channel, set res
    ADC14->CTL1 |= ADC14_CTL1_TCMAP | ADC14_CTL1_RES_3;
    //Map temp analog channel to MEM0/MCTL0, set 3.3v ref
    ADC14->MCTL[0] |= ADC14_MCTLN_INCH_22;
    //Map joystick analog channels to (x)MEM1/MCTL1 and (y)MEM2/MCTL2
    ADC14->MCTL[1] |= ADC14_MCTLN_INCH_15;
    ADC14->MCTL[2] |= ADC14_MCTLN_INCH_9 | ADC14_MCTLN_EOS;
#ifdef temp
    //Enable interrupts for temp sensor
    ADC14->IER0 = ADC14_IER0_IE0;
#endif
#ifdef joystick
    //Enable interrupts for joystick
    ADC14->IER0 |= ADC14_IER0_IE1;
    ADC14->IER0 |= ADC14_IER0_IE2;
#endif
    while(!(REF_A->CTL0 & REF_A_CTL0_GENRDY));
    ADC14->CTL0 |= ADC14_CTL0_ENC; //Enable conversions
    NVIC_EnableIRQ(ADC14_IRQn); //Enable ADC interrupts in NVIC
}

void configure_clock(void){
    //Configure clock
    CS->KEY = 0x695A; //Unlock CS module for register access
    CS->CTL0 = 0; //Reset tuning parameters
    CS->CTL0 = CS_CTL0_DCORSEL_1; //Setup DCO clock (3 MHz)
    //Select ACLK = REFO, SMCLK = MCLK = DCO
    CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;
    CS->KEY = 0; //Lock CS module for register access
}

void configure_timer(void){
    //Configure timer: Up mode, SMCLK src
    TIMER_A0->CTL |= TIMER_A_CTL_SSEL__SMCLK | TIMER_A_CTL_MC__UP;
    TIMER_A0->R = 0;
    TIMER_A0->CCR[0] = 40000; //Capture compare value
    //Enable capture compare interrupt
    TIMER_A0->CCTL[0] |= TIMER_A_CCTLN_CCIE;
    NVIC_EnableIRQ(TA0_0_IRQn);
}

void configure_serial_port(void) {
    //Configure UART pins: primary function
    P1SEL0 |= BIT2 | BIT3;
    //Configure UART: SMCLK src
    UCA0CTLW0 |= UCSWRST; //Put eUSCI in reset
    UCA0CTLW0 |= UCSSEL_2;
    UCA0BRW = 26; //Set Baud Rate (115200)
    UCA0CTLW0 &= ~UCSWRST; //Initialize eUSCI
    //Enable TX interrupts
    UCA0IE |= UCRXIE;
```

```c
    NVIC_EnableIRQ(EUSCIA0_IRQn);
}

void P1_IRQHandler(void){ //Button interrupt handler

}

void ADC14_IRQHandler(void){
    if(ADC14->IFGR0 & ADC14_IFGR0_IFG0){ //ADC results [0] handler
        uint32_t Nadc_temp = ADC14->MEM[0]; //Read data from ADC results
        int16_t C_temp = (0.000038*Nadc_temp) - 357; //voltage to Celcius conv.
        int16_t K_temp = C_temp + 273; //Celcius to Kelvin conv.
        int16_t F_temp = (C_temp*(9/5)) + 32; //Celcius to F. conv.
        i2a(C_temp); //Convert C temp to ASCII
        uint8_t C_label[10] = {" Degrees C"}; //Formatting
        uart_putchar_n(C_label,10);
        uart_putchar('\n');
        i2a(K_temp); //Convert K temp to ASCII
        uint8_t K_label[2] = {" K"}; //Formatting
        uart_putchar_n(K_label,2);
        uart_putchar('\n');
        i2a(F_temp); //Convert F temp to ASCII
        uint8_t T_label[10] = {" Degrees F"}; //Formatting
        uart_putchar_n(T_label,10);
        uart_putchar('\n');
        uart_putchar('\n');

    }
    if(ADC14->IFGR0 & ADC14_IFGR0_IFG1){ //ADC results [1] handler
        volatile uint32_t X_adc;
        uint8_t r_label[5] = {"Right"};
        uint8_t l_label[4] = {"Left"};
        X_adc = ADC14->MEM[1]; //Read data from ADC results
        if(X_adc > BOUNDARY){ //Joystick x in right region
            uart_putchar_n(r_label,5);
            uart_putchar('\n');
        }
        else{ //Joystick x in left region
            uart_putchar_n(l_label,4);
            uart_putchar('\n');
        }
    }
    if(ADC14->IFGR0 & ADC14_IFGR0_IFG2){ //ADC results [1] handler
        volatile uint32_t Y_adc;
        uint8_t up_label[5] = {"Upper"};
        uint8_t down_label[5] = {"Lower"};
        Y_adc = ADC14->MEM[2]; //Read data from ADC results
        if(Y_adc > BOUNDARY){ //Joystick y in upper region
            uart_putchar_n(up_label,5);
            uart_putchar('\n');
        }
        else{ //Joystick y in lower region
            uart_putchar_n(down_label,5);
            uart_putchar('\n');
        }
    }
}
```

```

void TA0_IRQHandler(void){ //Timer interrupt handler

}

```