Lab 8 – ADC

Youssef Samwel

yo800238@ucf.edu

EEL4742C Embedded Systems

Prof. Dr. Zakhia Abichar - Section 00419

3/28/2024

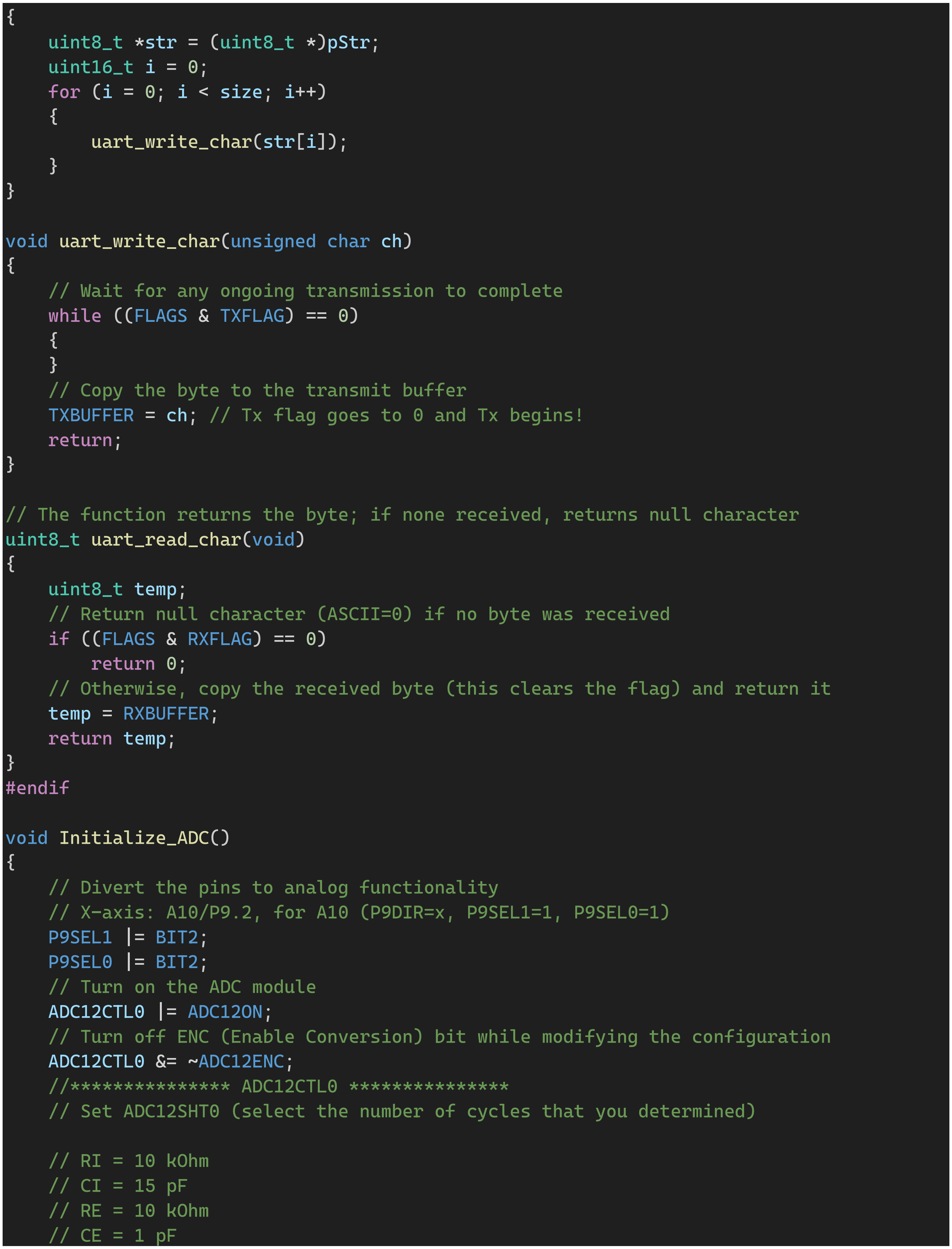
1.0 Project Description

In this lab, we will learn using the Analog-to-Digital Converter (ADC) of type Successive Approximation Register (SAR) with Charge Redistribution. We'll use the ADC to interface the two-dimensional joystick that's on the Educational BoosterPack.

2.0 Experiment Code

This code initializes the ADC module of the MSP430 microcontroller to perform analog-to-digital conversions for two channels: A10 (P9.2) for the X-axis and A4 (P8.7) for the Y-axis. It configures the ADC settings such as sample-and-hold time, resolution, and reference voltages. In the main function, after initializing the UART module for serial communication, it continuously reads the analog values of both channels using the ADC, compares them with previous readings, and if the change exceeds a certain threshold, it sends the updated values over UART.

```c
#ifndef __MAIN_INCLUDE__
#define __MAIN_INCLUDE__
#include <msp430fr6989.h>
#include <stdint.h>
#include <stdbool.h>

#define FLAGS UCA1IFG      // Contains the transmit & receive flags
#define RXFLAG UCRXIFG     // Receive flag
#define TXFLAG UCTXIFG     // Transmit flag
#define TXBUFFER UCA1TXBUF // Transmit buffer
#define RXBUFFER UCA1RXBUF // Receive buffer

#define BUTTON1 BIT1
#define BUTTON2 BIT2

#define MAX(x, y) (((x) > (y)) ? (x) : (y))
#define MIN(x, y) (((x) < (y)) ? (x) : (y))

// Configure UART to the popular configuration
// 9600 baud, 8-bit data, LSB first, no parity bits, 1 stop bit
// no flow control, oversampling reception
// Clock: SMCLK @ 1 MHz (1,000,000 Hz)
void Initialize_UART(void)
{
    // Configure pins to UART functionality
    P3SEL1 &= ~(BIT4 | BIT5);
    P3SEL0 |= (BIT4 | BIT5);
    // Main configuration register
    UCA1CTLW0 = UCSWRST; // Engage reset; change all the fields to zero
    // Most fields in this register, when set to zero, correspond to the
    // popular configuration
    UCA1CTLW0 |= UCSSEL__SMCLK; // Set clock to SMCLK
    // Configure the clock dividers and modulators (and enable oversampling)
    UCA1BRW = 6; // divider
    // Modulators: UCBRF = 8 = 1000 --> UCBRF3 (bit #3)
    // UCBRS = 0x20 = 0010 0000 = UCBRS5 (bit #5)
    UCA1MCTLW = UCBRF3 | UCBRS5 | UCOS16;
    // Exit the reset state
    UCA1CTLW0 &= ~UCSWRST;
}

// Configures ACLK to 32 KHz crystal
void config_ACLK_to_32KHz_crystal()
{
    // By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz
    // Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;
    // Wait until the oscillator fault flags remain cleared
    CSCTL0 = CSKEY; // Unlock CS registers
    do
    {
        CSCTL5 &= ~LFXTOFFG; // Local fault flag
        SFRIFG1 &= ~OFIFG;   // Global fault flag
    } while ((CSCTL5 & LFXTOFFG) != 0);
    CSCTL0_H = 0; // Lock CS registers
    return;
}

void uart_write_uint16(unsigned int number)
{
    // Max number of digits in a uint16_t is 5
    int divisor = 10000;

    while (divisor > 0)
    {
        int digit = number / divisor;
        if (digit > 0 || divisor == 1)
        {
            uart_write_char('0' + digit);
        }
        number %= divisor;
        divisor /= 10;
    }
}

void uart_write_string(const char *str)
{
    uint16_t length = strlen(str);
    uint16_t i;
    for (i = 0; i < length; i++)
    {
        uart_write_char(str[i]);
    }
}

void uart_write_nstring(void *pStr, uint16_t size)
```



```c
{
    uint8_t *str = (uint8_t *)pStr;
    uint16_t i = 0;
    for (i = 0; i < size; i++)
    {
        uart_write_char(str[i]);
    }
}

void uart_write_char(unsigned char ch)
{
    // Wait for any ongoing transmission to complete
    while ((FLAGS & TXFLAG) == 0)
    {
    }
    // Copy the byte to the transmit buffer
    TXBUFFER = ch; // Tx flag goes to 0 and Tx begins!
    return;
}

// The function returns the byte; if none received, returns null character
uint8_t uart_read_char(void)
{
    uint8_t temp;
    // Return null character (ASCII=0) if no byte was received
    if ((FLAGS & RXFLAG) == 0)
        return 0;
    // Otherwise, copy the received byte (this clears the flag) and return it
    temp = RXBUFFER;
    return temp;
}
#endif

void Initialize_ADC()
{
    // Divert the pins to analog functionality
    // X-axis: A10/P9.2, for A10 (P9DIR=x, P9SEL1=1, P9SEL0=1)
    P9SEL1 |= BIT2;
    P9SEL0 |= BIT2;
    // Turn on the ADC module
    ADC12CTL0 |= ADC12ON;
    // Turn off ENC (Enable Conversion) bit while modifying the configuration
    ADC12CTL0 &= ~ADC12ENC;
    //*************** ADC12CTL0 ***************
    // Set ADC12SHT0 (select the number of cycles that you determined)

    // RI = 10 kOhm
    // CI = 15 pF
    // RE = 10 kOhm
    // CE = 1 pF
    // t >= (20 kOhm) * (16 pF) * ln(2^13)
    // t >= (3.2 10^-8) * ln(2^13)
    // t >= 0.2883 * 10^-6
    // t >= 0.2883 us
    // t >= (approx) .32 us
    // MODCLK: [4 4.8 5.4] MHz
    // clock cycles: (12 + 1 for bits) + (? conversion time)
    ADC12CTL0 |= ADC12SHT0_2;

    //*************** ADC12CTL1 ***************
    // Set ADC12SHS (select ADC12SC bit as the trigger)
    // Set ADC12SHP bit
    // Set ADC12DIV (select the divider you determined)
    // Set ADC12SSEL (select MODOSC)
    ADC12CTL1 |= ADC12SC | ADC12SHP | ADC12DIV_0 | ADC12SSEL_0;

    //*************** ADC12CTL2 ***************
    // Set ADC12RES (select 12-bit resolution)
    // Set ADC12DF (select unsigned binary format)
    ADC12CTL2 |= ADC12RES__12BIT;
    //*************** ADC12CTL3 ***************
    // Leave all fields at default values
    //*************** ADC12MCTL0 ***************
    // Set ADC12VRSEL (select VR+=AVCC, VR-=AVSS)
    // Set ADC12INCH (select channel A10)
    // Turn on ENC (Enable Conversion) bit at the end of the configuration
    ADC12CTL3 |= ADC12INCH_10 | ADC12VRSEL_0;

    ADC12CTL0 |= ADC12ENC;
    return;
}

void Initialize_ADC2()
{
    // Divert the pins to analog functionality
    // X-axis: A10/P9.2, for A10 (P9DIR=x, P9SEL1=1, P9SEL0=1)
    P9SEL1 |= BIT2;
    P9SEL0 |= BIT2;
    // Y axis A4/P8.7
    P8SEL0 |= BIT7;
    P9SEL1 |= BIT7;

    // Turn on the ADC module
    ADC12CTL0 |= ADC12ON;
    // Turn off ENC (Enable Conversion) bit while modifying the configuration
    ADC12CTL0 &= ~ADC12ENC;
    //*************** ADC12CTL0 ***************
    // Set ADC12SHT0 (select the number of cycles that you determined)
    ADC12CTL0 |= ADC12SHT0_3 | ADC12SHT1_3 | ADC12MSC;
    //*************** ADC12CTL1 ***************
    // Set ADC12SHS (select ADC12SC bit as the trigger)
    // Set ADC12SHP bit
    // Set ADC12DIV (select the divider you determined)
    // Set ADC12SSEL (select MODOSC)
    ADC12CTL1 |= ADC12SHP | ADC12CONSEQ_1;

    //*************** ADC12CTL2 ***************
    // Set ADC12RES (select 12-bit resolution)
    // Set ADC12DF (select unsigned binary format)
    ADC12CTL2 |= ADC12RES_2;

    //*************** ADC12CTL3 ***************
    //
    ADC12CTL3 &= ~ADC12CSTARTADD_31;

    //*************** ADC12MCTL1 ***************
    // Set ADC12VRSEL (select VR+=AVCC, VR-=AVSS)
    // Set ADC12INCH (select channel A10)
    // EOS stuff
    ADC12MCTL1 |= ADC12INCH_4 | ADC12EOS;

    //*************** ADC12MCTL0 ***************
    // Set ADC12VRSEL (select VR+=AVCC, VR-=AVSS)
    // Set ADC12INCH (select channel A10)
    ADC12MCTL0 |= ADC12INCH_10;

    // Turn on ENC (Enable Conversion) bit at the end of the configuration
    ADC12CTL0 |= ADC12ENC;
    return;
}

int main()
{
    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
    PM5CTL0 &= ~LOCKLPM5;     // Enable the GPIO pins

    Initialize_UART();
    Initialize_ADC2();

    uint16_t lastX = 0;
    uint16_t lastY = 0;

    for (;;)
    {

        ADC12CTL0 |= ADC12SC;
        while ((ADC12CTL1 & ADC12BUSY) == ADC12BUSY)
        {
        }
        uint16_t x = ADC12MEM0;
        uint16_t y = ADC12MEM1;
        if (lastX == 0 || lastY == 0)
        {
            lastX = x;
            lastY = y;
            continue;
        }
        if (MAX(lastX, x) - MIN(lastX, x) > 100 ||
            MAX(lastY, y) - MIN(lastY, y) > 100)
        {
            uart_write_string("<");
            uart_write_uint16(x);
            uart_write_string(", ");
            uart_write_uint16(y);
            uart_write_string(">\n\r");
            _delay_cycles(5e3);
            lastX = x;
            lastY = y;
        }
    }

    return 0;
}
```

3.0 Student Q&A

What are the values of the ADC's RI and CI? If these values have a range show the range. Did you use the lower or upper range of these values? Justify your choice.

$$R_I = 10\ k\Omega$$

$$C_I = 15\ pF$$

I used the upper bound of these values because Vcc was higher than 2-Volts and for C_I also because a higher capacitor and/or resistance require longer time to charge and therefore these values would satisfy the worst case.

What is the minimum sample-and-hold time? Show how you computed this duration

```
    // RI = 10 kOhm
    // CI = 15 pF
    // RE = 10 kOhm
    // CE = 1 pF
    // t >= (20 kOhm) * (16 pF) * ln(2^13)
    // t >= (3.2 10^-8) * ln(2^13)
    // t >= 0.2883 * 10^-6
    // t >= 0.2883 us
    // t >= (approx) .32 us
    // MODCLK: [4 4.8 5.4] MHz
```

1. How many cycles does it take the ADC to convert a 12-bit result? (look in the configuration register that contains ADC12RES).

14 clock cycles.

2. In this experiment, we set our reference voltages VR+ = AV CC (Analog Vcc) and VR− = AV SS (Analog Vss). What voltage values do these signals have? Look in the MCU data sheet (slas789c) in Table 5.3. Assume that Vcc=3.3V and Vss=0

$$V_R^+ = 3.3\ V$$

$$V_R^- = 0\ V$$

4.0 Conclusion

The experiment focused on utilizing the MSP430 microcontroller's Analog-to-Digital Converter (ADC) for interfacing with a two-dimensional joystick. The provided code initialized the ADC module for X and Y-axis readings, configured settings, and continuously transmitted data over UART based on threshold comparisons. The report also covered key aspects such as determining ADC values and understanding conversion cycles. Overall, the experiment demonstrated effective ADC configuration and practical application in embedded systems.