EEL 4742C: Embedded Systems

Name: Hamzah Ullah

Lab 10: Analog to Digital Converter(ADC)

Introduction:

In this lab we will be learn about the Analog to Digital Convertor or the ADC for short. There are many different types, but we will be using the successive approximation register (SAR). We will use ADC to read joystick inputs and display them via UART on the CPU screen.

************PART 2 STARTS ON PAGE 6*****************

Part 1: Using ADC SAR-Type

In this lab we are using an ADC to connect an analog device, in this case the joystick. In order to determine the number of bits that result from the ADC, we need to look at the amount of voltage passing through based on the capacitance and voltage limits of the MCU. We also want to determine at what speed we want our input sample hold time, in this lab we determined 3µs. We than need to find the correct ratio of capacitance to resistance to get as close to 3µs as possible. We than determine the amount of cycles that will occur in that time period and pick an interval for the clock so that the SHT can be performed during that time. In our lab, the number of cycles ended up being ~16 cycles, so the next interval is needed to get a clear and efficient reading (32 cycles).

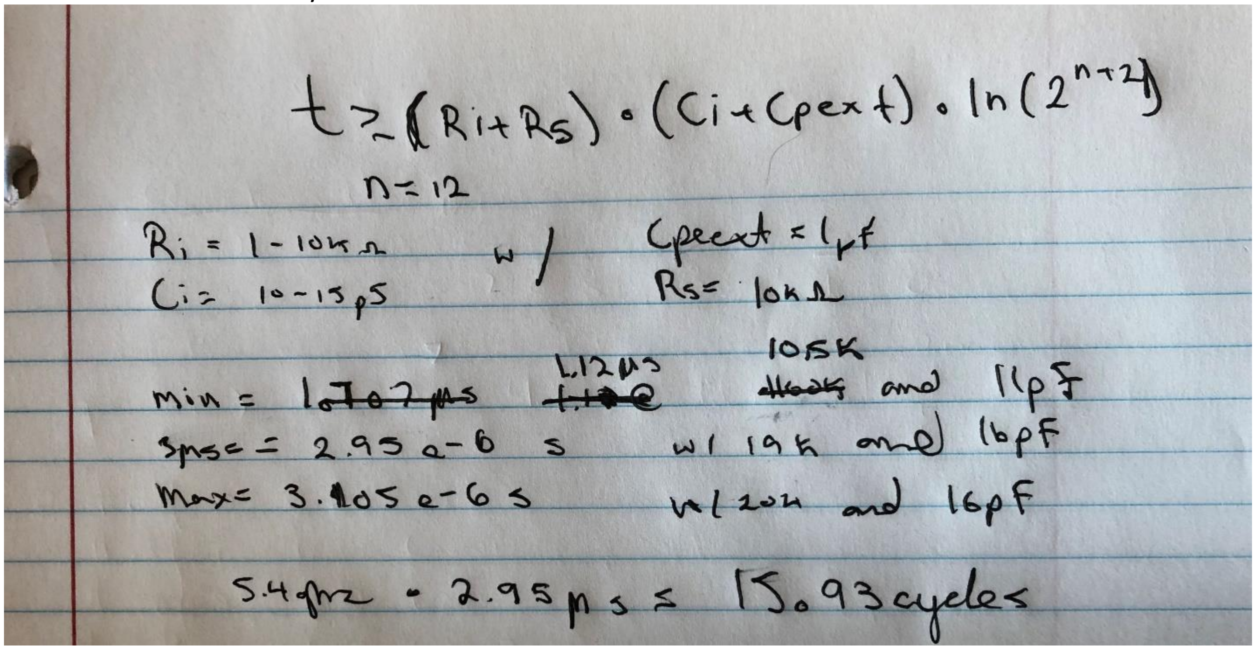
Now we need to set up the joystick by porting the correct values and setting up the required registers for the ADC to function to allow the analog joystick to function stick to function. We can do it while the ADC is not enabled. Below is the code that demonstrates the x axis movement.

Questions

10k ohms and 15pF for R_I and C_I. The ranges were 1k-10k ohms and 10p-15pF I used the upper range for the capacitor lower range for the resistor to try and get as close to the value of 3µs. I used a very high capacitor and resistor to try and bring the time down to as low as possible to read the time threshold required.

Upper range because we require 3 µs sample hold time and to do that it requires higher resistance and capacitance with 12 bits.

Below is the calculation for minimum sample hold time with 500k ohm resistance and R_I and 11pF C_I. No divider was used for MOSOSC. 32 cycles were used because it was very close to 16 cycle interval so I chose the one above 16 cycles.



$t \gtrsim (R_i + R_s) \cdot (C_i + C_{pext}) \cdot \ln(2^{n+2})$

$n = 12$

$R_i = 1 - 10k\Omega$ w/ $C_{pext} \approx 1pF$

$C_i = 10 - 15pF$ $R_s = 10k\Omega$

min = ~~1.707 µs~~ 1.12µs ~~[illegible]~~ ~~[illegible]~~ 105K and 11pF

Spnsc = 2.95 e-6 s w/ 19k and 16pF

Max = 3.105 e-6 s w/ 20k and 16pF

5.4 MHz · 2.95 µs ≤ 15.93 cycles

I was getting the correct values for the joystick value of the X coordinate.

Below is the code for reading the value on the X coordinate joystick through UART from ADC12MEM0.

```
#include <msp430.h>
#define FLAGS UCA1IFG //Contains the transmit and receive flags
#define RXFLAG UCRXIFG //receive flag
#define TXFLAG UCTXIFG //Transmit flag
#define TXBUFFER UCA1TXBUF //Transmit buffer
#define RXBUFFER UCA1RXBUF //Receive Buffer
/**
 * main.c
 */
int main(void)
{
        WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
          PM5CTL0 &= ~LOCKLPM5;   //enable GPIO pins
      // X-axis: A10/P9.2, for A10 (P9DIR=x, P9SEL1=1, P9SEL0=1)
      P9SEL1 |= BIT2;
      P9SEL0 |= BIT2;

      TA0CCR0 = 16384-1;
      TA0CCTL0 |= CCIFG;
      TA0CCTL0 &= ~CCIFG;

      config_ACLK_to_32KHz_crystal();
      TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;
          TA0CTL &= ~TAIFG;

      Initialize_UART();
      Initialize_ADC();
      int i=0;
      for(;;)
      {
          ADC12CTL0 |= ADC12SC;
          while(ADC12BUSY != 0){
              if((TA0CCTL0 & CCIFG) != 0){
              unsigned char * ptr[] = &ADC12MEM0;
              unsigned int data=0;
              data = *ptr[i];
              uart_write_uint16(data);
              uart_write_char('\n');
              uart_write_char('\r');
              //if(i>31)
                //  i=0;
              TA0CTL &= ~TAIFG;
              }
          }
      }
      return 0;
}
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)
ADC12CTL0 |= (ADC12SHT00|ADC12SHT01); //32 cycles ADC12 CLK
ADC12CTL0 &= ~ADC12CONSEQ1;
//*************** ADC12CTL1 ***************
// Set ADC12SHS (select ADC12SC bit as the trigger)
// Set ADC12SHP bit
// Set ADC12DIV (select the divider you determined)
// Set ADC12SSEL (select MODOSC)
ADC12CTL1 |= ADC12SHS_0 | ADC12SHP | ADC12DIV_0 | ADC12SSEL_0;
//*************** ADC12CTL2 ***************
// Set ADC12RES (select 12-bit resolution)
// Set ADC12DF (select unsigned binary format)
ADC12CTL2 |= ADC12RES_2;
ADC12CTL2 &= ~ADC12DF;
//*************** ADC12CTL3 ***************
// Leave all fields at default values
//*************** ADC12MCTL0 ***************
// Set ADC12VRSEL (select VR+=AVCC, VR-=AVSS)
// Set ADC12INCH (select channel A10)
ADC12MCTL0 |= ADC12VRSEL0 | ADC12INCH_10;
// Turn on ENC (Enable Conversion) bit at the end of the configuration
    ADC12CTL0 |= ADC12ENC;
return;
}
void Initialize_UART(void)
{
    //Divert Pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);

    //Use smclk clock; leave the other settings default
    UCA1CTLW0 |= UCSSEL_2;

    //Configure clock dividers and modulators
    //UCBR = 6, UCBRF = 13, UCBRS = 0x22, UCOS16=1 (oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS5 | UCBRS1| UCBRF3| UCBRF2| UCBRF0| UCOS16;

    //exit the reset state(so transmission/reception cna begin)
    UCA1CTLW0 &= ~UCSWRST;
}
void uart_write_char(unsigned char ch)
{
    //wait for any ongoing transmission to complete
    while ((FLAGS & TXFLAG) == 0)
    {
    }

    //Write the byte to the transmit buffer
    TXBUFFER = ch;
}
void uart_write_string(char * str)
```

```
{
    int i;
    for(i=0; i < strlen(str); i++){
        uart_write_char(str[i]);
    }
}

void uart_write_uint16(unsigned int n)
{
    int temp;
    if(n>=10000)
    {
        temp = n/10000;
        uart_write_char(temp+48);
    }
    if(n>=1000)
    {
        temp = n/1000 % 10;
        uart_write_char(temp+48);
    }
    if(n>=100)
    {
        temp = n/100 % 10;
        uart_write_char(temp+48);
    }
    if(n>=10)
    {
        temp = (n/10) % 10;
        uart_write_char(temp+48);
    }
     n = n % 10;
     uart_write_char(n+48);

    //Write the byte to the transmit buffer
    //TXBUFFER = n;
}
void config_ACLK_to_32KHz_crystal()
{
    //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

    //Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;

    //Wait until the oscillator fault flags remain cleared
    CSCTL0 = CSKEY;
    do
    {
        CSCTL5 &= ~LFXTOFFG;    //local fault flag
        SFRIFG1 &= ~OFIFG;      //Global fault flag
    }
    while ((CSCTL5 & LFXTOFFG) != 0);

    CSCTL0_H = 0; //lock CS registers
    return;
}
```
Part 2: Reading the X- and Y- Coordinates of the Joystick

This part of the lab is similar to the first except we must initialize the second joystick direction, Y by porting the values. We than must chance how the ADC12 reads by changing the ADC12CONSEQ register to sequence of channels. When we read the data, each piece of data is saved in its own memory, meaning x is in mem0 and y is in mem1.

Below is the code showing the X and Y output reading every second. The output came out as expected.

Code for reading the X and y Coordinates on the joystick and outputting it through UART every second.

```
#include <msp430.h>
#define FLAGS UCA1IFG //Contains the transmit and receive flags
#define RXFLAG UCRXIFG //receive flag
#define TXFLAG UCTXIFG //Transmit flag
#define TXBUFFER UCA1TXBUF //Transmit buffer
#define RXBUFFER UCA1RXBUF //Receive Buffer
#define greenLED BIT7

/**
 * main.c
 */
int main(void)
{
      WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;    //enable GPIO pins

    // X-axis: A10/P9.2, for A10 (P9DIR=x, P9SEL1=1, P9SEL0=1)
    P9SEL1 |= BIT2;
    P9SEL0 |= BIT2;

    config_ACLK_to_32KHz_crystal();
    TA0CTL = TASSEL_1 | ID_0 | MC_2 | TACLR;
        TA0CTL &= ~TAIFG;

     P9OUT |= greenLED;
     P9DIR &= ~greenLED;

    Initialize_UART();
    Initialize_ADC();
    int i=0;
    int dataX = 0;
    int dataY = 0;
    char * headerX[] = 'X Coordinate';
    char * headerY[] = 'Y Coordinate';
    for(;;)
    {
        ADC12CTL0 |= ADC12SC;
        while((ADC12CTL1 & ADC12BUSY) != 0){}
        if((TA0CTL & TAIFG) != 0){
            dataX = ADC12MEM0;
            uart_write_char('X');
            uart_write_uint16(dataX);
            uart_write_char('\n');
            uart_write_char('\r');
            dataY = ADC12MEM1;
            uart_write_char('Y');
            uart_write_uint16(dataY);
            uart_write_char('\n');
            uart_write_char('\r');
            TA0CTL &= ~TAIFG;
        }

      }
    }
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;
    //Y AXIS
    P8SEL1 |=BIT7;
    P8SEL0 |=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_2 | ADC12MSC; //for multisample conversions
//*************** ADC12CTL1 ***************
// Set ADC12SHS (select ADC12SC bit as the trigger)
// Set ADC12SHP bit
// Set ADC12DIV (select the divider you determined)
// Set ADC12SSEL (select MODOSC)
ADC12CTL1 |= ADC12SHS_0 | ADC12SHP | ADC12DIV_0 | ADC12SSEL_0 | ADC12CONSEQ_1;


//*************** ADC12CTL2 ***************
// Set ADC12RES (select 12-bit resolution)
// Set ADC12DF (select unsigned binary format)
ADC12CTL2 |= ADC12RES_2;
ADC12CTL2 &= ~ADC12DF;
//*************** ADC12CTL3 ***************
// Leave all fields at default values
ADC12CTL3 |= ADC12CSTARTADD_0; //STARTADD to 0
//*************** ADC12MCTL0 ***************
// Set ADC12VRSEL (select VR+=AVCC, VR-=AVSS)
// Set ADC12INCH (select channel A10)
ADC12MCTL0 |= ADC12VRSEL_0 | ADC12INCH_10;

//*************** ADC12MCTL1 ***************
// Set ADC12VRSEL (select VR+=AVCC, VR-=AVSS)
// Set ADC12INCH (select the analog channel that you found)
// Set ADC12EOS (last conversion in ADC12MEM1)
ADC12MCTL1 |= ADC12VRSEL_0 | ADC12INCH_4 | ADC12EOS;
// Turn on ENC (Enable Conversion) bit at the end of the configuration
ADC12CTL0 |= ADC12ENC;
return;
```

```
}
void Initialize_UART(void)
{
    //Divert Pins to UART functionality
    P3SEL1 &= ~(BIT4|BIT5);
    P3SEL0 |= (BIT4|BIT5);

    //Use smclk clock; leave the other settings default
    UCA1CTLW0 |= UCSSEL_2;

    //Configure clock dividers and modulators
    //UCBR = 6, UCBRF = 13, UCBRS = 0x22, UCOS16=1 (oversampling)
    UCA1BRW = 6;
    UCA1MCTLW = UCBRS5 | UCBRS1| UCBRF3| UCBRF2| UCBRF0| UCOS16;

    //exit the reset state(so transmission/reception cna begin)
    UCA1CTLW0 &= ~UCSWRST;
}
void uart_write_char(unsigned char ch)
{
    //wait for any ongoing transmission to complete
    while ((FLAGS & TXFLAG) == 0)
    {
    }

    //Write the byte to the transmit buffer
    TXBUFFER = ch;
}
void uart_write_string(char * str)
{
    int i;
    for(i=0; i < strlen(str); i++){
        uart_write_char(str[i]);
    }
}

void uart_write_uint16(unsigned int n)
{
    int temp;
    if(n>=10000)
    {
        temp = n/10000;
        uart_write_char(temp+48);
    }
    if(n>=1000)
    {
        temp = n/1000 % 10;
        uart_write_char(temp+48);
    }
    if(n>=100)
    {
        temp = n/100 % 10;
        uart_write_char(temp+48);
    }
    if(n>=10)
    {

        temp = (n/10) % 10;
        uart_write_char(temp+48);
    }
     n = n % 10;
     uart_write_char(n+48);

    //Write the byte to the transmit buffer
    //TXBUFFER = n;
}
void config_ACLK_to_32KHz_crystal()
{
    //By default, ACLK runs on LFMODCLK at 5MHz/128 = 39kHz

    //Reroute pins to LFXIN/LFXOUT functionality
    PJSEL1 &= ~BIT4;
    PJSEL0 |= BIT4;

    //Wait until the oscillator fault flags remain cleared
    CSCTL0 = CSKEY;
    do
    {
        CSCTL5 &= ~LFXTOFFG;    //local fault flag
        SFRIFG1 &= ~OFIFG;      //Global fault flag
    }
    while ((CSCTL5 & LFXTOFFG) != 0);

    CSCTL0_H = 0; //lock CS registers
    return;
}
```

QUESTIONS:

1. **How many cycles does it take the ADC to convert a 12-bit result?**
   13 Cycles.
2. **The conversion time you found in the previous question does not include SHT(Sample hold time) Find the total conversion time of your setup.**
   32+13= 45 cycles. 45/5.4mhz = 8.33µs
   It is higher because even though I calculated a 3µs value, it was too close to the cycle intervals to select (15.93 cycles) so I picked one cycle interval higher.
3. **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.**
   VR_- = 1.8V, VR_+ = 3.6
4. **Its possible for the ADC12_B module to use reference voltages that are generated by the module REF_A. What voltage levels does REF_A provide?**
   1.2V, 2.0V, 2.5V.