

Lab 8 Report

EEL4742C - 00446

Yousef Awad

October 2025

Contents

Introduction	2
8.1 Using the ADC SAR-Type	2
8.2 Reading the X- and Y- Coordinates of the Joystick	6
Student Q&A	9
1	9
2	9
Conclusion	9

Introduction

This experiment introduces the functionality of the Analog-to-Digital Converter (ADC). The primary objective is to learn how to configure and use the ADC module to interface with an analog peripheral, specifically the 2D joystick on the Educational BoosterPack. The lab begins with an overview of the Successive Approximation Register (SAR) ADC's operation, including the importance of the sample-and-hold time (SHT). This theory is applied by calculating the minimum SHT required for the joystick and configuring the ADC12_B module's registers accordingly. Practical work involves programming the MCU to read the joystick's horizontal (X-axis) coordinate , and then expanding the configuration to read both the X- and Y-coordinates using a "sequence-of-channels". Finally, this knowledge is applied to develop a "Platform Balancing Control" application, which uses the joystick to adjust the height of four platform corners while monitoring for unsafe height differences.

8.1 Using the ADC SAR-Type

So, first we must calculate the SHT (Sample-and-Hold Time) with a 12-bit resolution ADC in the msp430fr6989. Now to do that, I had to look into the datasheet and found the following:

- $C_{internal} = 15pF$
- $R_{internal} = 10K\Omega$

And then with the given external resistance and capacitance of $10k\Omega$ and $1pF$, respectively, all plugged into the following function

$$\begin{aligned} t &\geq (R_I + R_E) * (C_I + C_E) * \ln(2^{n+1}) \\ t &\geq (10 * 10^3 + 10 * 10^3) * (15 * 10^{-12} + 1 * 10^{-12}) * \ln(2^{12+1}) \\ t &\geq 2.8883 * 10^{-6} \end{aligned}$$

Therefore, we get, when rounded up, $3\mu s$ to be the minimum Sample-and-Hold Time. Now, to find the SHT duration we simply take the highest value of the frequency, 5.4MHz and find the time per cycle, or $\frac{1}{5.4MHz}$ and use it as the divisor to the previous $3\mu s$ from before. With that, I got, ≈ 11 SHT Cycles, which would round up to 16 due to it needing to be a power of 2. As well... Thankfully, the returned values are valid looking and make sense!

```
1 #include <msp430fr6989.h>
2 #include <stdint.h>
3
4 #define redLED BIT0
5 #define FLAGS UCA1IFG
6 #define RXFLAG UCRXIFG
7 #define TXFLAG UCTXIFG
8 #define TXBUFFER UCA1TXBUF
```

```

9 #define RXBUFFER UCA1RXBUF
10
11 // 9600 baud based on 1 MHz SMCLK w/16x oversampling.
12 // 8 bits, no parity, LSB first, 1 stop bit UART communication
13 void Initialize_UART(void)
14 {
15     // Configure pins to UART functionality
16     P3SEL1 &= ~(BIT4 | BIT5);
17     P3SEL0 |= (BIT4 | BIT5);
18     // Main configuration register
19     UCA1CTLW0 = UCSWRST;
20     // Engage reset; change all the fields to zero
21     // Most fields in this register, when set to zero, correspond to
22     // the
23     // popular configuration
24     UCA1CTLW0 |= UCSSEL_2; // Set clock to SMCLK
25     // Configure the clock dividers and modulators (and enable
26     // oversampling)
27     UCA1BRW = 6;
28     // divider
29     // Modulators: UCBRF = 8 = 1000--> UCBRF3 (bit #3)
30     // UCBR5 = 0x20 = 0010 0000 = UCBR5 (bit #5)
31     UCA1MCTLW = UCBRF3 | UCBR5 | UCOS16;
32     // Exit the reset state
33     UCA1CTLW0 &= ~UCSWRST;
34 }
35
36 void Initialize_ADC(void)
37 {
38     // Configure the pins to analog functionality
39     // X-axis: A10/P9.2, for A10 (P9DIR=x, P9SEL1=1, P9SEL0=1)
40     P9SEL1 |= BIT2;
41     P9SEL0 |= BIT2;
42     // Turn on the ADC module
43     ADC12CTL0 |= ADC12ON;
44     // Turn off ENC (Enable Conversion) bit while modifying the
45     // configuration
46     ADC12CTL0 &= ~ADC12ENC;
47     //***** ADC12CTL0 *****
48     // ADC12SHT0x sets SHT cycles for results 0-7, 24-31
49     // ADC12MSC sets multiple analog inputs
50     ADC12CTL0 |=
51         ADC12SHT0_2; // Sets SHT of 16 cycles (found in doc. slau367o
52         // table 34.4)
53     //***** ADC12CTL1 *****
54     // ADC12SHS sets read trigger
55     // ADC12SHP sets SAMPCON use
56     // ADC12DIV sets clock divider
57     // ADC12SSEL sets clock base
58     // ADC12CONSEQx sets conversion sequence mode
59     ADC12CTL1 |= ADC12SHS_0; // 0 = ADC12SC bit
60     ADC12CTL1 |= ADC12SHP; // 1 = SAMPCON sourced from clock
61     ADC12CTL1 |= ADC12DIV_0; // 0 = /1
62     ADC12CTL1 |= ADC12SSEL_0; // 0 = MODOSC
63     // ADC12CTL1 |= ADC12CONSEQ_1;
64     // values in doc. slau367o table 34.5
65     //***** ADC12CTL2 *****

```

```

62 // ADC12RES sets bit resolution
63 // ADC12DF sets data format
64 ADC12CTL2 |= ADC12RES_2; // 2 = 12-bit
65 ADC12CTL2 &= ~ADC12DF; // 0 = unsigned binary
66 //***** ADC12MCTL0 *****
67 // ADC12VRSELx sets VR+ and VR- sources as well as buffering
68 // ADC12INCHx sets analog input
69 ADC12MCTL0 |= ADC12VRSEL_0; // 0 -> VR+ = AVCC and VR- = AVSS
70 ADC12MCTL0 |= ADC12INCH_10; // 10 = A10 input
71 //***** ADC12MCTL1 *****
72 // set ENC bit at end of config
73 ADC12CTL0 |= ADC12ENC;
74 }

75
76 void uart_write_char(unsigned char ch)
77 {
78     while ((FLAGS & TXFLAG) == 0)
79     {
80         // Wait for any ongoing transmission to complete
81     }
82     // Copy the byte to the transmit buffer
83     TXBUFFER = ch; // Tx flag goes to 0 and Tx begins!
84     return;
85 }

86
87 void uart_write_12bit(uint16_t n)
88 {
89     const char hex_digits[] = "0123456789ABCDEF"; // Digits used in
90     // hexadecimal
91     uint8_t digit; // one hex digit =
92     // 4 bits
93     int i;
94     // print the 0x part of hex format
95     uart_write_char('0');
96     uart_write_char('x');
97     // Extract and print hex digits from input
98     // i = 8 because bits 12-15 will always be 0000
99     for (i = 8; i >= 0; i = i - 4)
100    {
101        digit = (n >> i) & 0xF;
102        uart_write_char(hex_digits[digit]);
103    }
104
105 void main(void)
106 {
107     WDTCTL = WDTPW | WDTHOLD;
108     PM5CTL0 &= ~LOCKLPM5;
109     P1DIR |= redLED;
110     P1OUT |= redLED;
111
112     Initialize_UART();
113     Initialize_ADC();
114
115     for (;;) {
116

```

```
117     ADC12CTL0 |= ADC12SC; // Triggers ADC12BUSY while reading input
118     while ((ADC12CTL1 & ADC12BUSY) != 0)
119     {
120         // Wait for flag to drop
121     }
122     ADC12CTL0 &= ~ADC12SC;
123     uint16_t x_coord = ADC12MEM0; // ADC12MEM0 linked to A10, x-
124     input
125     uart_write_12bit(x_coord);    // Print x-coordinate to console
126     uart_write_char('\n');        // newline
127     P1OUT ^= redLED;            // toggle red LED
128     _delay_cycles(500000);       // 0.5 second delay
129 }
```

8.2 Reading the X- and Y- Coordinates of the Joystick

```
1 #include <msp430fr6989.h>
2 #include <stdint.h>
3
4 #define redLED BIT0
5 #define FLAGS UCA1IFG
6 #define RXFLAG UCRXIFG
7 #define TXFLAG UCTXIFG
8 #define TXBUFFER UCA1TXBUF
9 #define RXBUFFER UCA1RXBUF
10
11 // 9600 baud based on 1 MHz SMCLK w/16x oversampling.
12 // 8 bits, no parity, LSB first, 1 stop bit UART communication
13 void Initialize_UART(void)
14 {
15     // Configure pins to UART functionality
16     P3SEL1 &= ~(BIT4 | BIT5);
17     P3SEL0 |= (BIT4 | BIT5);
18     // Main configuration register
19     UCA1CTLW0 = UCSWRST;
20     // Engage reset; change all the fields to zero
21     // Most fields in this register, when set to zero, correspond to
22     // the
23     // popular configuration
24     UCA1CTLW0 |= UCSSEL_2; // Set clock to SMCLK
25     // Configure the clock dividers and modulators (and enable
26     // oversampling)
27     UCA1BRW = 6;
28     // divider
29     // Modulators: UCBRF = 8 = 1000--> UCBRF3 (bit #3)
30     // UCBRS = 0x20 = 0010 0000 = UCBRSS5 (bit #5)
31     UCA1MCTLW = UCBRF3 | UCBRSS5 | UCOS16;
32     // Exit the reset state
33     UCA1CTLW0 &= ~UCSWRST;
34 }
35
36 void Initialize_ADC(void)
37 {
38     // Configure the pins to analog functionality
39     // X-axis: A10/P9.2, for A10 (P9DIR=x, P9SEL1=1, P9SEL0=1)
40     P9SEL1 |= BIT2;
41     P9SEL0 |= BIT2;
42     // Y-axis: A4/P8.7 (P8DIR=x, P8SEL1=1, P8SEL0=0)
43     P8SEL1 |= BIT7;
44     P8SEL0 |= BIT7;
45     // Turn on the ADC module
46     ADC12CTL0 |= ADC12ON;
47     // Turn off ENC (Enable Conversion) bit while modifying the
48     // configuration
49     ADC12CTL0 &= ~ADC12ENC;
50     //***** ADC12CTL0 *****
51     // ADC12SHT0x sets SHT cycles for results 0-7, 24-31
52     // ADC12MSC sets multiple analog inputs
53     ADC12CTL0 |= ADC12SHT0_2; // Sets SHT of 16 cycles (found in doc.
```

```

    slau367o table 34.4)
51 ADC12CTL0 |= ADC12MSC; // 1= multiple inputs
52 //***** ADC12CTL1 *****
53 // ADC12SHS sets read trigger
54 // ADC12SHP sets SAMPCON use
55 // ADC12DIV sets clock divider
56 // ADC12SSEL sets clock base
57 // ADC12CONSEQx sets conversion sequence mode
58 ADC12CTL1 |= ADC12SHS_0; // 0 = ADC12SC bit
59 ADC12CTL1 |= ADC12SHP; // 1 = SAMPCON sourced from clock
60 ADC12CTL1 |= ADC12DIV_0; // 0 = /1
61 ADC12CTL1 |= ADC12SSEL_0; // 0 = MODOSC
62 ADC12CTL1 |= ADC12CONSEQ_1;
63 // values in doc. slau367o table 34.5
64 //***** ADC12CTL2 *****
65 // ADC12RES sets bit resolution
66 // ADC12DF sets data format
67 ADC12CTL2 |= ADC12RES_2; // 2 = 12-bit
68 ADC12CTL2 &= ~ADC12DF; // 0 = unsigned binary
69 //***** ADC12CTL3 *****
70 // ADC12CSTARTADDx sets first ADC12MEM register in conversion
sequence
71 ADC12CTL3 |= ADC12CSTARTADD_0;
72 //***** ADC12MCTL0 *****
73 // ADC12VRSELx sets VR+ and VR- sources as well as buffering
74 // ADC12INCHx sets analog input
75 ADC12MCTL0 |= ADC12VRSEL_0; // 0 -> VR+ = AVCC and VR- = AVSS
76 ADC12MCTL0 |= ADC12INCH_10; // 10 = A10 input
77 //***** ADC12MCTL1 *****
78 // ADC12ENC sets final conversion channel
79 ADC12MCTL1 |= ADC12VRSEL_0;
80 ADC12MCTL1 |= ADC12INCH_4;
81 ADC12MCTL1 |= ADC12EOS; // 1 = last converted input
82 // set ENC bit at end of config
83 ADC12CTL0 |= ADC12ENC;
84 }
85
86 void uart_write_char(unsigned char ch)
87 {
88     while ((FLAGS & TXFLAG) == 0)
89     {
90         // Wait for any ongoing transmission to complete
91     }
92     // Copy the byte to the transmit buffer
93     TXBUFFER = ch; // Tx flag goes to 0 and Tx begins!
94     return;
95 }
96
97 void uart_write_12bit(uint16_t n)
98 {
99     const char hex_digits[] = "0123456789ABCDEF"; // Digits used in
hexadecimal
100    uint8_t digit; // one hex digit =
        4 bits
101    int i;
102    // print the 0x part of hex format
103    uart_write_char('0');

```

```

104     uart_write_char('x');
105     // Extract and print hex digits from input
106     // i = 8 because bits 12-15 will always be 0000
107     for (i = 8; i >= 0; i = i - 4)
108     {
109         digit = (n >> i) & 0xF;
110         uart_write_char(hex_digits[digit]);
111     }
112 }
113
114 void main(void)
115 {
116     WDTCTL = WDTPW | WDTHOLD;
117     PM5CTL0 &= ~LOCKLPM5;
118
119     P1DIR |= redLED;
120     P1OUT |= redLED;
121
122     Initialize_UART();
123     Initialize_ADC();
124
125     for (;;)
126     {
127         ADC12CTL0 |= ADC12SC; // Triggers ADC12BUSY while reading input
128         while ((ADC12CTL1 & ADC12BUSY) != 0)
129         {
130             // Wait for flag to drop
131         }
132         ADC12CTL0 &= ~ADC12SC;
133         uint16_t x_coord = ADC12MEM0; // ADC12MEM0 linked to A10, x-
134         input
135         uint16_t y_coord = ADC12MEM1; // ADC12MEM1 linked to A4, y-
136         input
137         uart_write_12bit(x_coord);    // Print x-coordinate to console
138         uart_write_char(' ');        // Readability space
139         uart_write_12bit(y_coord);    // Print y-coordinate to console
140         uart_write_char('\n');        // newline
141         P1OUT ^= redLED;            // toggle red LED
142         _delay_cycles(500000);       // 0.5 second delay
143     }
144 }
```

Student Q&A

1

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

- It would take 14 clock cycles.

2

Given: In this experiment, we set our reference voltages $VR+ = AVCC$ (Analog Vcc) and $VR- = AVSS$ (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 $Vss=0$. It is the following:

- $AVCC = V_{cc} = 3.3V$
- $AVSS = V_{ss} = 0.0V$

Conclusion

This lab successfully demonstrated the configuration and application of the ADC12_B module for interfacing with the analog joystick. A foundational part of the lab involved calculating the minimum sample-and-hold time (SHT) based on SAR ADC theory and configuring the necessary control registers, such as ‘ADC12CTL0’ and ‘ADC12CTL1’, for the correct clock, resolution, and SHT cycles. The experiment progressed from reading a single analog channel for the X-axis to implementing the ADC’s “sequence-of-channels” mode to capture both X and Y coordinates from one trigger. This cumulative knowledge was then applied to build a ”Platform Balancing Control” application, which used the joystick input to manage a system and enforce safety constraints, providing comprehensive practical experience in managing and utilizing analog-to-digital conversions.