

CPE 325: Embedded Systems Laboratory

Laboratory #9 Tutorial

Analog-to-Digital Converter

Aleksandar Milenković

Email: mlenka@uah.edu

Web: <http://www.ece.uah.edu/~mlenka>

Objective

This tutorial will introduce the configuration and operation of the MSP430 12-bit analog-to-digital converter (ADC12). Programs will demonstrate the use of ADC12 to interface an on-board temperature sensor as well as external analog inputs. Specifically, you will learn how to:

Configure the ADC12

Choose reference voltages to maximize signal resolution

Create waveform lookup table in MATLAB

Interface of an on-board temperature sensor

Interface external analog signal inputs

Notes

All previous tutorials are required for successful completion of this lab, especially the tutorials introducing the TI Experimenter's board, UART communication, and Timer_A.

Contents

1	Analog-to-Digital Converters	2
1.1	ADC Resolution, Reference Voltages, and Signal Resolution.....	2
1.2	On-Chip Temperature Sensor	2
1.3	Example: Analog Thumbstick Configuration	7
4	References	14

1 Analog-to-Digital Converters

The world around us is analog. Sensors or transducers convert physical quantities such as, temperature, force, light, sound, and others, into electrical signals, typically voltage signals that we can measure. Analog-to-digital converters allow us to interface these analog signals and convert them into digital values that can further be stored, analyzed, or communicated.

The MSP430 family of microcontrollers has a variety of analog-to-digital converters with varying features and conversion methods. In this laboratory we focus on the ADC12 converter used in the MSP430F5529. The ADC12 converter has 16 configurable input channels; 12 input channels are routed to corresponding analog input pins; remaining 4 input channels are routed to internal voltages and an on-chip temperature sensor.

1.1 ADC Resolution, Reference Voltages, and Signal Resolution

There are several key factors that should be regarded when configuring your ADC12 to most effectively read the analog signal. The first parameter you should understand is the device's voltage resolution, i.e., the smallest change of an input analog signal that causes a change in the digital output. We will be using the ADC12 peripheral that has a vertical resolution of 12 bits. That means that it can distinguish between 2^{12} (0 to 4095) input voltage levels. An A/D converter described as “n-bit” can distinguish between 0 and 2^n-1 voltage steps.

After acknowledging your ADC vertical resolution, the reference voltages need to be set. Setting the reference voltages dials in the minimum and maximum values read by the ADC. For instance, you could set your V_- to -5V and your V_+ to 10 V. With that setup on the ADC12, the numerical sampled value 0 would correspond to a signal input of -5 V, and a sampled value of 4095 would correspond to a 10 V input.

It is very important to characterize the input signal you are expecting before you set up your ADC. If you expect a signal input between 0 V and 3 V, you should set your reference voltages to 0 V and 3 V. If you set them to -5V and +5V, you would be wasting a large amount of your sample “bit depth,” and your overall sample resolution would suffer because your sample input values would stay between 2048 and 3275. There would only be $(3275-2048=1227)$ steps of resolution for your input signal rather than 4095 if you choose 0 V and 3 V as your reference voltages.

An ADC typically relies on a timer to periodically generate a trigger to start sampling of the incoming signals. You should choose a timer period that triggers sampling frequently enough to recreate the original input signals (the minimum sampling frequency should be at least two times the frequency of the signal's largest harmonic).

1.2 On-Chip Temperature Sensor

The MSP430's ADC12 has an internal temperature sensor that creates an analog voltage proportional to its temperature. A sample transfer characteristic of the temperature sensor in a different MSP430 (namely MSP430FG4618) is shown in Figure 1. The output of the temperature sensor is connected to the input multiplexor channel 10 (INCHx=1010 which is true for MSP430F5529 as well). When using the temperature sensor, the sample time (the time ADC12 is looking at the analog signal) must be greater than 30 μ s. From the transfer characteristic, we get

that the temperature in degrees Celsius can be expressed as $TEMP_C = \frac{V_{TEMP} - 986 \text{ mV}}{3.55 \text{ mV}}$, where V_{TEMP} is the voltage from the temperature sensor (in millivolts). The transfer characteristic mentioned in Figure 1 is expressed in Volts.

The ADC12 transfer characteristic gives the following equation: $ADCResult = 4095 \cdot \frac{V_{TEMP}}{V_{REF}}$, or $V_{TEMP} = V_{REF} \cdot \frac{ADCResult}{4095}$. This can easily be deduced using the relation $ADCResult = (2^n - 1) \cdot \frac{V_{TEMP} - V_{REF-}}{V_{REF+} - V_{REF-}}$.

By using the internal voltage generator $V_{REF+} = 1,500 \text{ mV}$ (1.5 V) and $V_{REF-} = 0\text{V}$, we can derive temperature as follows: $TEMP_C = \frac{(ADCResult - 2692) \cdot 423}{4095}$.

Make sure your calculations match the equation given. How would equation change if instead of using $V_{REF} = 1.5 \text{ V}$ we use $V_{REF} = 2.5 \text{ V}$?

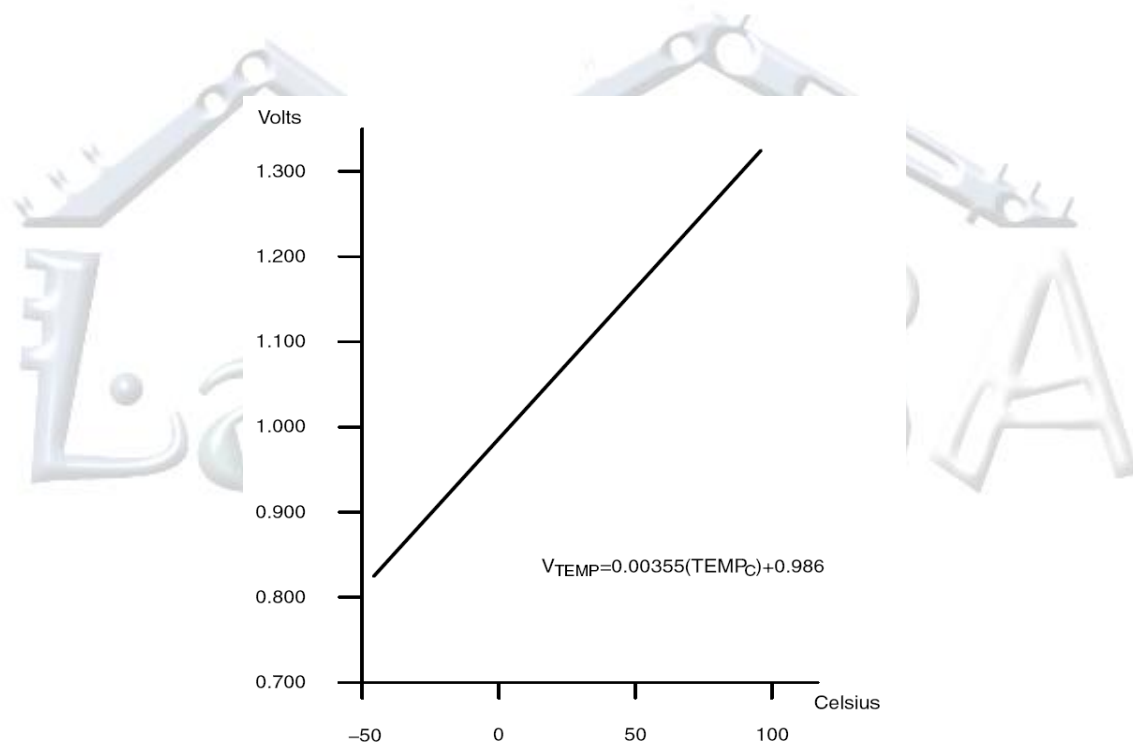


Figure 1. Internal Temperature Sensor Transfer Characteristic: $V=f(T)$ for MSP430F4618

For MSP430F5529, since the transfer characteristic given does not present the relation between the input temperature and output voltage generated by the sensor. Thus, we will be using a slightly different approach. In the sample code presented as Lab10_D1 in Figure 3, we use double intercept form of the given characteristic to determine the transfer function. Since the reference voltage is known to us, we can consult Page 106 of the datasheet of MSP430F5529 to refer to the values we need.

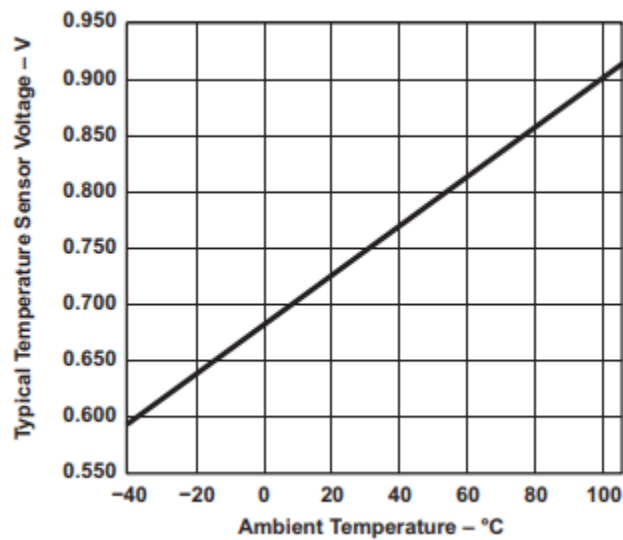


Figure 2. Temperature Sensor Transfer Function for MSP430F5529

In the C application shown in Figure 3 that samples the on-chip temperature sensor, converts the sampled voltage from the sensor to temperature in degrees Celsius and Fahrenheit, and sends the temperature information through a RS232 link to the Putty or MobaXterm application. Analyze the program and test it on the TI Experimenter's Board. Answer the following questions.

What does the program do?

What are configuration parameters of ADC12 (input channel, clock, reference voltage, sampling time, ...)?

What are configuration parameters of the USART0 module?

How does the temperature sensor work?

```

1  /*-----
2  * File:      Lab10_D1.c (CPE 325 Lab10 Demo code)
3  *
4  * Function:   Measuring the temperature (MPS430F5529)
5  *
6  * Description: This C program samples the on-chip temperature sensor and
7  *              converts the sampled voltage from the sensor to temperature in
8  *              degrees Celsius and Fahrenheit. The converted temperature is
9  *              sent to HyperTerminal over the UART by using serial UART.
10 *
11 * Clocks:     ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (~1MHz)
12 *              An external watch crystal between XIN & XOUT is required for ACLK
13 *
14 * Instructions: Set the following parameters in HyperTerminal
15 *               Port :      COM1
16 *               Baud rate :  115200
17 *               Data bits:   8
18 *               Parity:      None
19 *               Stop bits:   1
20 *               Flow Control: None

```

```

21  *
22  *
23  *
24  *
25  *
26  *
27  *
28  *
29  *
30  *
31  *
32  * Input:      Character Y or y or N or n
33  *
34  * Output:     Displays Temperature in Celsius and Fahrenheit in HyperTerminal
35  * Author:     Aleksandar Milenkovic, milenkovic@computer.org
36  *
37  * Prawar Poudel
38  *-----*/
39  #include <msp430.h>
40  #include <stdio.h>
41
42  #define CALADC12_15V_30C *((unsigned int *)0x1A1A) // Temperature Sensor
43  Calibration-30 C
44
45  //See device datasheet for TLV
46  table memory mapping
47  #define CALADC12_15V_85C *((unsigned int *)0x1A1C) // Temperature Sensor
48  Calibration-85 C
49
50  char ch; // Holds the received char from UART
51  unsigned char rx_flag; // Status flag to indicate new char is received
52
53  char gm1[] = "Hello! I am an MSP430. Would you like to know my temperature? (Y|N)";
54  char gm2[] = "Bye, bye!";
55  char gm3[] = "Type in Y or N!";
56
57  long int temp; // Holds the output of ADC
58  long int IntDegF; // Temperature in degrees Fahrenheit
59  long int IntDegC; // Temperature in degrees Celsius
60
61  char NewTem[25];
62
63  void UART_setup(void) {
64      P3SEL |= BIT3 + BIT4; // Set USCI_A0 RXD/TXD to receive/transmit data
65      UCA0CTL1 |= UCSWRST; // Set software reset during initialization
66      UCA0CTL0 = 0; // USCI_A0 control register
67      UCA0CTL1 |= UCSSEL_2; // Clock source SMCLK
68
69      UCA0BR0 = 0x09; // 1048576 Hz / 115200 lower byte
70      UCA0BR1 = 0x00; // upper byte
71      UCA0MCTL = 0x02; // Modulation (UCBRS0=0x01, UCOS16=0)
72
73      UCA0CTL1 &= ~UCSWRST; // Clear software reset to initialize USCI state machine
74      UCA0IE |= UCRXIE; // Enable USCI_A0 RX interrupt
75  }

```

```

76
77 void UART_putCharacter(char c) {
78     while (!(UCA0IFG&UCTXIFG)); // Wait for previous character to transmit
79     UCA0TXBUF = c; // Put character into tx buffer
80 }
81
82 void sendMessage(char* msg, int len) {
83     int i;
84     for(i = 0; i < len; i++) {
85         UART_putCharacter(msg[i]);
86     }
87     UART_putCharacter('\n'); // Newline
88     UART_putCharacter('\r'); // Carriage return
89 }
90
91 void ADC_setup(void) {
92     REFCTL0 &= ~REFMSTR; // Reset REFMSTR to hand over control
93 to
94     // ADC12_A ref control registers
95     ADC12CTL0 = ADC12SHT0_8 + ADC12REFON + ADC12ON;
96     // Internal ref = 1.5V
97     ADC12CTL1 = ADC12SHP; // enable sample timer
98     ADC12MCTL0 = ADC12SREF_1 + ADC12INCH_10; // ADC i/p ch A10 = temp sense i/p
99     ADC12IE = 0x001; // ADC_IFG upon conv result-ADCMEMO
100     delay_cycles(100); // delay to allow Ref to settle
101     ADC12CTL0 |= ADC12ENC;
102 }
103
104 void main(void) {
105     WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
106     UART_setup(); // Setup USCI_A0 module in UART mode
107     ADC_setup(); // Setup ADC12
108
109     rx_flag = 0; // RX default state "empty"
110     _EINT(); // Enable global interrupts
111     while(1) {
112         sendMessage(gm1, sizeof(gm1)); // Send a greetings message
113
114         while(!(rx_flag&0x01)); // Wait for input
115         rx_flag = 0; // Clear rx_flag
116         sendMessage(&ch, 1); // Send received char
117
118         // Character input validation
119         if ((ch == 'y') || (ch == 'Y')) {
120
121             ADC12CTL0 &= ~ADC12SC;
122             ADC12CTL0 |= ADC12SC; // Sampling and conversion start
123
124             _BIS_SR(CPUOFF + GIE); // LPM0 with interrupts enabled
125
126             //in the following equation,
127             // ..temp is digital value read
128             //..we are using double intercept equation to compute the
129             //... .. temperature given by temp value
130             //... .. using observations at 85 C and 30 C as reference

```



```

131     IntDegC = ((float)((((long)temp - CALADC12_15V_30C) * (85 - 30)) /
132                (CALADC12_15V_85C - CALADC12_15V_30C) + 30.0f;
133
134     IntDegF = IntDegC*(9/5.0) + 32.0;
135
136     // Printing the temperature on HyperTerminal/Putty
137     sprintf(NewTem, "T(F)=%ld\tT(C)=%ld\n", IntDegF, IntDegC);
138     sendMessage(NewTem, sizeof(NewTem));
139 }
140 else if ((ch == 'n') || (ch == 'N')) {
141     sendMessage(gm2, sizeof(gm2));
142     break; // Get out
143 }
144 else {
145     sendMessage(gm3, sizeof(gm3));
146 }
147 } // End of while
148 while(1); // Stay here forever
149 }
150
151 #pragma vector = USCI_A0_VECTOR
152 __interrupt void USCIA0RX_ISR (void) {
153     ch = UCA0RXBUF; // Copy the received char
154     rx_flag = 0x01; // Signal to main
155     LPM0_EXIT;
156 }
157
158 #pragma vector = ADC12_VECTOR
159 __interrupt void ADC12ISR (void) {
160     temp = ADC12MEM0; // Move results, IFG is cleared
161     _BIC_SR_IRQ(CPUOFF); // Clear CPUOFF bit from 0(SR)
162 }
163

```

Figure 3. C Program that Samples On-Chip Temperature Sensor

1.3 Example: Analog Thumbstick Configuration

The above program details configuration and use of the ADC12 for single channel use. However, many analog devices or systems would require multiple channel configurations. As an example, let us imagine an analog joystick as is used by controllers for most modern gaming consoles. So-called thumbsticks have X and Y axis voltage outputs depending on the vector of the push it receives as input. For this example, we will use a thumbstick that has 0 to 3V output in the X and Y axes. No push on either axis results in a 1.5V output for both axes. In Figure 4 below, note how a push at about 120° with around 80% power results in around 2.75V output for the Y axis and 0.8V output for the X axis.

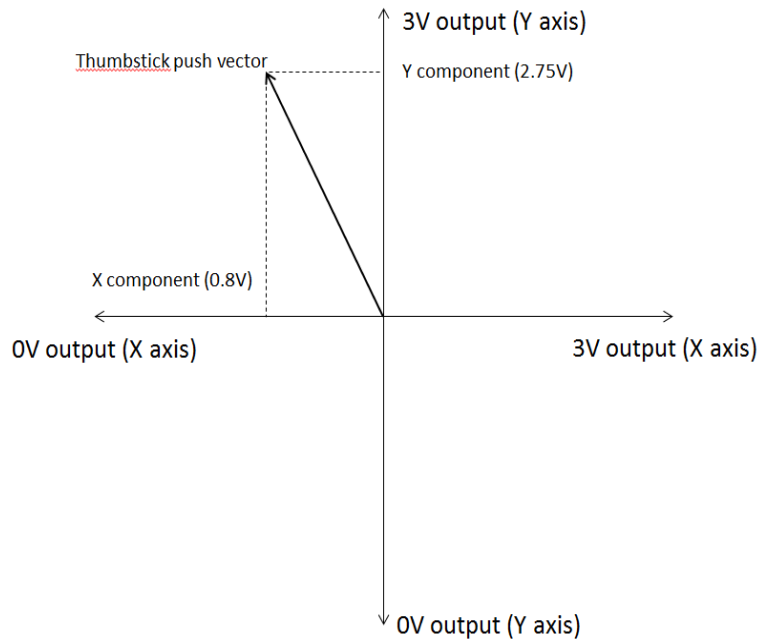


Figure 4. Performance data for hypothetical thumbstick

We want to test the thumbstick output using the UAH Serial App. To do this, we will first hook the thumbstick outputs to our device. Let's say we will use analog input A0 (P6.0) for the X axis and A1 (P6.1) for the Y axis.

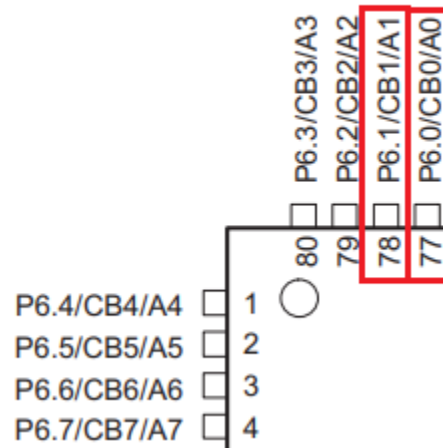


Figure 5. Pinouts and Header Connections for Analog Inputs

Because the outputs are from 0 to 3V, we need to set our reference voltages accordingly. We can use the board's ground and 3V supply as references.

We will want to have our output as the float datatypes because the output for each axis should be a percentage. In Figure 4, for example, the converted Y axis output would be 91.67% and the

X axis output would be 26.67%. Here is the formula you would use to convert the values (remember, the microcontroller is going to be receiving values from 0 to 4095 based on voltage values from 0V to 3V that we set as our references):

$$\text{Input ADC Value in steps} \times \frac{3V}{4095} \times \frac{100\%}{3V} = \%Power$$

We could send our information in a variety of ways including a vector format, signed percentage, or even just ADC “steps.” If we are using the percentage calculated as shown above, our packet to send to the UAH serial app would look like the one below (1 header byte, 2 single precision floating-point numbers). Figure 6 shows how to configure UAH Serial App to accept two channels including single-precision floating-point numbers. Figure 7 shows signals representing the percentage of HORZ and VERT direction of the thumbstick (read line, CH0, represents HORZ and blue line, CH1, represents VERT) when it is moved along HORZ and VERT axes. The value 100 (100%) of the red line indicates that thumbstick is moved fully in the horizontal direction.

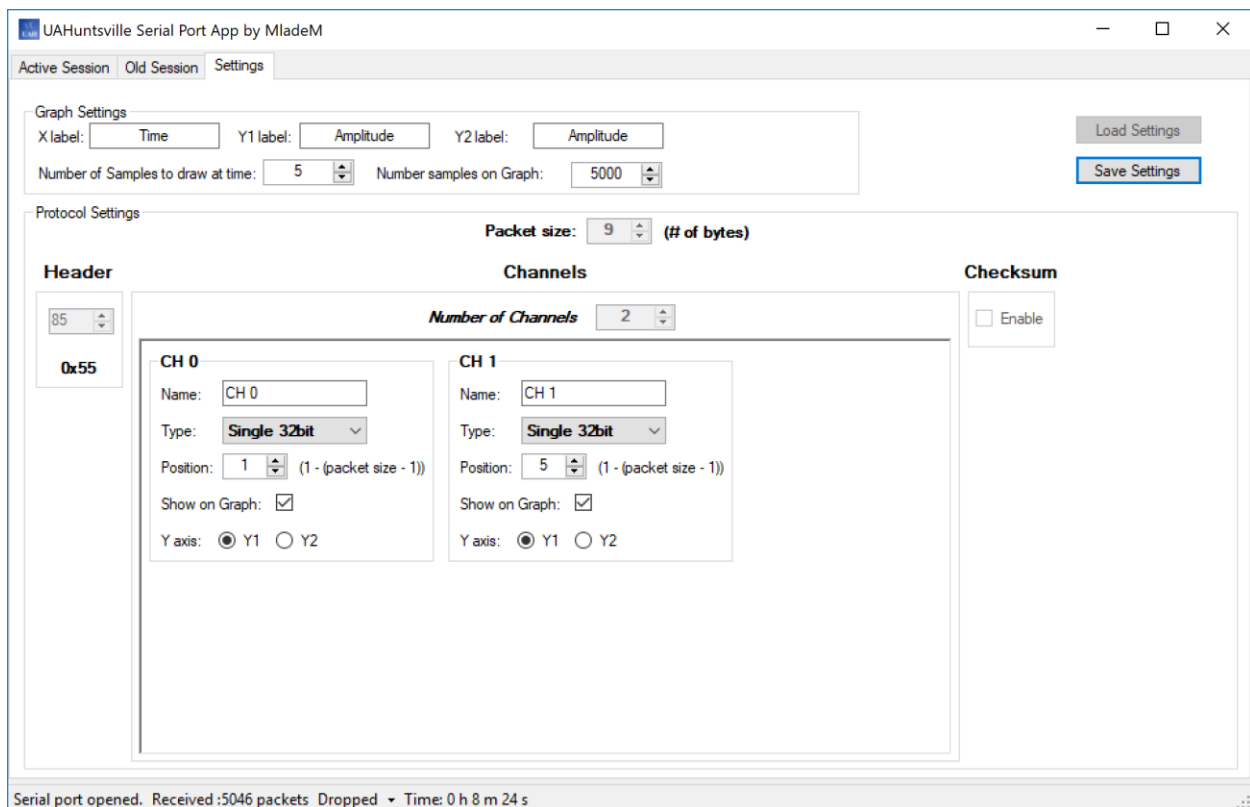
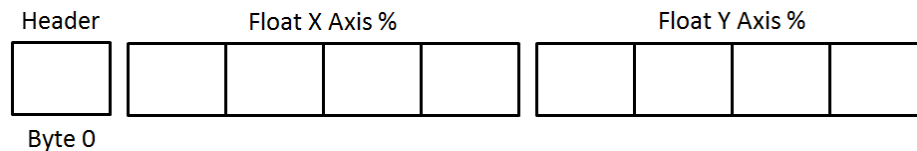


Figure 6. UAH Serial App Settings

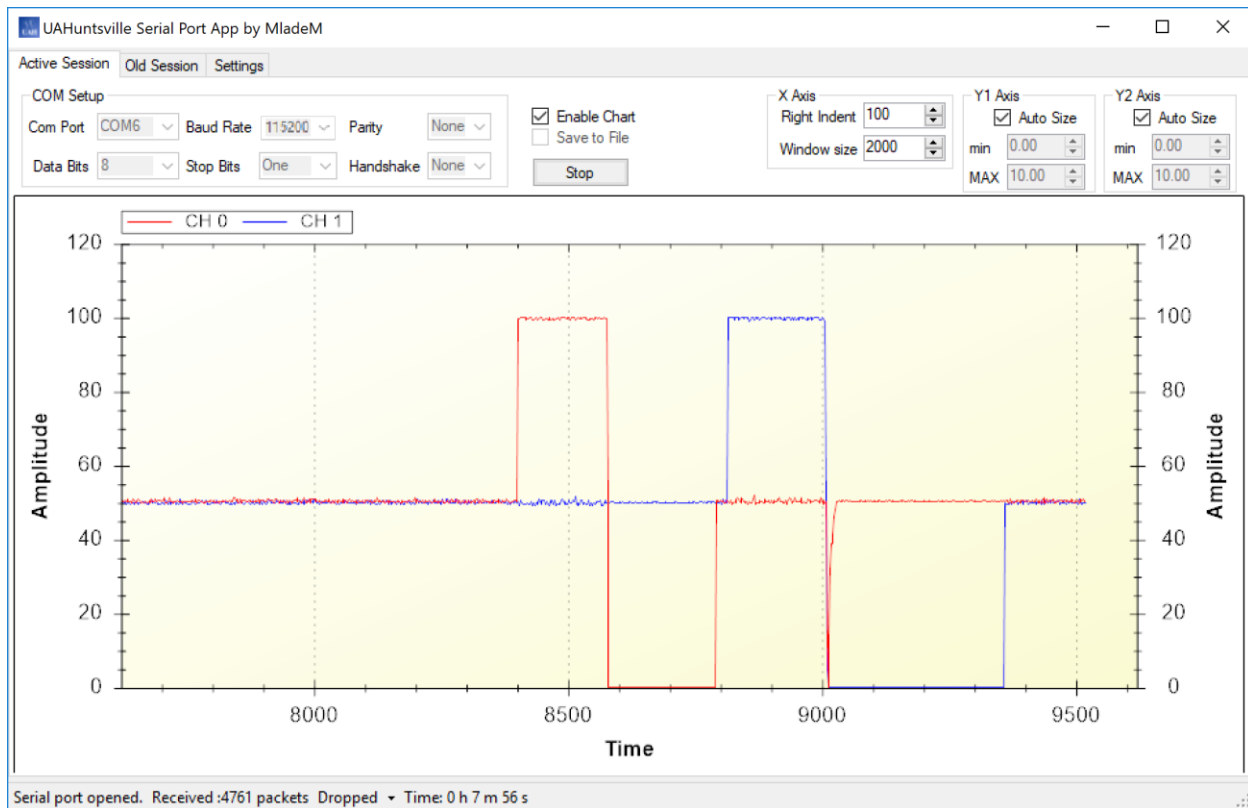


Figure 7. UAH Serial App Showing Percentage Signals from Thumbstick (CH0 – HORZ, CH1 – VERT)

Figure 8 shows demo code that could be used to set up the ADC12 and UART and send the thumbstick information to the UAH Serial App. Analyze the code and answer the following questions.

What does the program do?

What are configuration parameters of ADC12 (input channel, clock, reference voltage, sampling time, ...)?

How many samples per second is taken from ADC12?

How many samples per second per axis is sent to UAH Serial App?

You can connect your thumbstick to the pins shown in Figure 5 for use with launchpad kit. If you are using Grove Kit, note that the analog input A3 (port P6.3) corresponds to the pin Pin26 and analog input A7 (port P6.7) corresponds to the Pin 27 on the Grove Starter Kit. These pins can be accessible at J8 jumper when the Grove Kit is placed with its female connector attached to male connector of MSP-EXP529LP board. These pins are where we should connect horizontal HORZ and vertical VERT wires of the thumbstick when using the Grove Kit. Make sure to make appropriate changes in the source code presented in Figure 8 for use with Grove Kit.

```

1  /*-----
2  * File:      Lab09_D2.c
3  * Function:   Interfacing thumbstick

```

```

4  * Description: This C program interfaces with a thumbstick sensor that has
5  *              x (HORZ on Thumbstick connected to P6.0) and
6  *              y (VERT on Thumbstick connected to P6.1) axis
7  *              and outputs from 0 to 3V. A sample joystick can be found at
8  *
9  https://www.digikey.com/htmldatasheets/production/2262974/0/0/1/512.html
10 *
11 *              The value of x and y axis is sent as the percentage
12 *              of power to the UAH Serial App.
13 *
14 * Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (~1MHz)
15 *              An external watch crystal beten XIN & XOUT is required for ACLK
16 *              MSP-EXP430F5529LP
17 *
18 *              /|\|
19 *              | |
20 *              --RST
21 *
22 *              P3.3/UCA0TXD
23 *              P6.0(A0)-->|
24 *              P6.1(A1)-->|
25 *              P3.4/UCA0RXD
26 *
27 * Input:        Connect thumbstick to the board
28 * Output:       Displays % of power in UAH serial app
29 * Author(s):    Prawar Poudel, prawar.poudel@uah.edu
30 *              Micah Harvey
31 * Date:         August 8, 2020
32 *-----*/
33 #include <msp430.h>
34 volatile long int ADCXval, ADCYval;
35 volatile float Xper, Yper;
36
37 void TimerA_setup(void)
38 {
39     TA0CTL0 = CCIE;                // Enabled interrupt
40
41     TA0CCR0 = 3277;                // 3277 / 32768 Hz = 0.1s
42     TA0CTL = TASSEL_1 + MC_1;      // ACLK, up mode
43 }
44
45 void ADC_setup(void)
46 {
47     // configure ADC converter
48     P6SEL = 0x03;                  // Enable A/D channel inputs
49     ADC12CTL0 = ADC12ON+ADC12MSC+ADC12SHT0_8; // Turn on ADC12, extend sampling time
50                                         // to avoid overflow of results
51
52     ADC12CTL1 = ADC12SHP+ADC12CONSEQ_1; // Use sampling timer, repeated
53     sequence
54     ADC12MCTL0 = ADC12INCH_0;        // ref+AVcc, channel = A0
55     ADC12MCTL1 = ADC12INCH_1+ADC12EOS; // ref+AVcc, channel = A1, end seq.
56
57     ADC12IE = 0x02;                // Enable ADC12IFG.1

```

```

59     ADC12CTL0 |= ADC12ENC;                // Enable conversions
60 }
61
62
63 void UART_putCharacter(char c)
64 {
65     while (!(UCA0IFG&UCTXIFG));             // Wait for previous character to transmit
66     UCA0TXBUF = c;                          // Put character into tx buffer
67 }
68
69
70 void UART_setup(void)
71 {
72     P3SEL |= BIT3 + BIT4;                  // Set USCI_A0 RXD/TXD to receive/transmit data
73
74     UCA0CTL1 |= UCSWRST;                    // Set software reset during initialization
75     UCA0CTL0 = 0;                          // USCI_A0 control register
76     UCA0CTL1 |= UCSSEL_2;                  // Clock source SMCLK
77
78     UCA0BR0 = 0x09;                        // 1048576 Hz / 115200 lower byte
79     UCA0BR1 = 0x00;                        // upper byte
80     UCA0MCTL |= UCBRS0;                    // Modulation (UCBRS0=0x01, UCOS16=0)
81
82     UCA0CTL1 &= ~UCSWRST;                  // Clear software reset to initialize USCI state machine
83 }
84
85
86 void sendData(void)
87 {
88     int i;
89     Xper = (ADCXval*3.0/4095*100/3);        // Calculate percentage outputs
90     Yper = (ADCYval*3.0/4095*100/3);
91
92     // Use character pointers to send one byte at a time
93     char *xpointer=(char *)&Xper;
94     char *ypointer=(char *)&Yper;
95
96     UART_putCharacter(0x55);                // Send header
97     for(i = 0; i < 4; i++)
98     {
99         // Send x percentage - one byte at a time
100         UART_putCharacter(xpointer[i]);
101     }
102     for(i = 0; i < 4; i++)
103     {
104         // Send y percentage - one byte at a time
105         UART_putCharacter(ypointer[i]);
106     }
107 }
108
109 void main(void)
110 {
111     WDTCTL = WDTPW +WDTHOLD;                // Stop WDT
112
113     // Enable interrupts globally
114     __enable_interrupt();

```

```

114
115     ADC_setup();           // Setup ADC
116     UART_setup();
117     TimerA_setup();
118
119     while (1)
120     {
121         __bis_SR_register(LPM0_bits + GIE); // Enter LPM0
122         sendData();
123     }
124 }
125
126 #pragma vector = ADC12_VECTOR
127 __interrupt void ADC12ISR(void)
128 {
129     ADCXval = ADC12MEM0;           // Move results, IFG is cleared
130     ADCYval = ADC12MEM1;
131     __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
132 }
133
134 #pragma vector = TIMER0_A0_VECTOR
135 __interrupt void timerA_isr()
136 {
137     ADC12CTL0 |= ADC12SC;
138 }
139
140

```

Figure 8 C Program that takes the x- and y- axis Samples from a Thumbstick

3 References

To understand more about the ADC12 peripheral and its configuration, please refer the following materials:

- Davies Text, pages 407-438 and pages 485-492
- MSP430x5xx User's Guide, Chapter 28, pages 730-760 (ADC12)
- MSP430x5xx User's Guide, page 744 (Internal temperature sensor)

