CPE 325: Embedded Systems Laboratory
Laboratory #10 Tutorial
Analog-to-Digital Converter

Aleksandar Milenković
Email: milenka@uah.edu
Web: http://www.ece.uah.edu/~milenka

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

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 input channels are routed to internal voltages 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 µs. From the transfer characteristic, we get that the temperature in degrees Celsius can be expressed as $degC = \frac{V_{sensor} - 986\ mV}{3.55\ mV}$, where V_{sensor} is the voltage from the temperature sensor. The ADC12 transfer characteristic gives the following equation: $ADCResult = 4095 \cdot \frac{V_{sensor}}{V_{REF}}$, or $V_{sensor} = V_{REF} \cdot \frac{ADCResult}{4095}$. By using the internal voltage generator V_{REF}=1,500 mV (1.5 V), we can derive temperature as follows: $degC = \frac{(ADCResult - 2692) \cdot 423}{4095}$. Make sure your calculations match the equation given. How would equation change if instead of using V_{REF}=1.5 V we use V_{REF}=2.5 V?

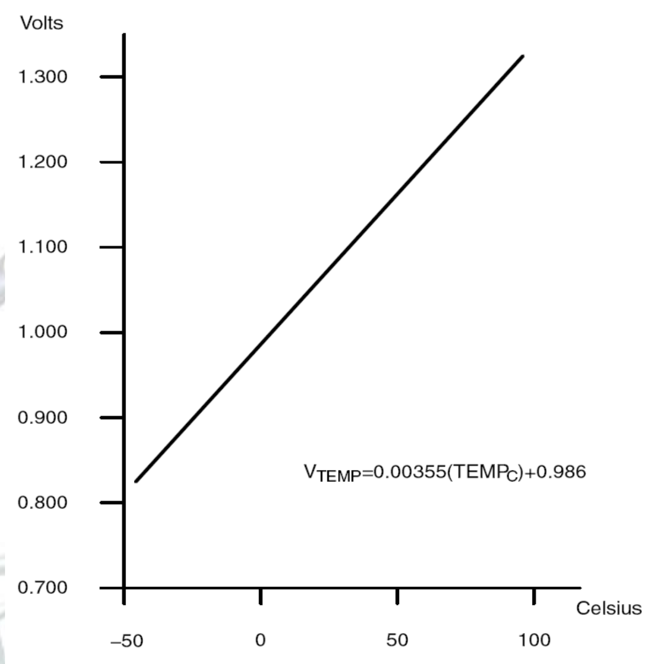


Figure 1. Internal Temperature Sensor Transfer Characteristic: V=f(T)

For MSP430F5529, since the transfer characteristic is not available, we will be using a slightly different approach. In the sample code presented as Lab10_D1 in Figure 2, we use double intercept form of the transfer 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.

In the C application shown in Figure 2 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?

```c
/*------------------------------------------------------------------------------
 * File:        Lab10_D1.c (CPE 325 Lab10 Demo code)
 *
 * Function:    Measuring the temperature (MPS430F5529)
 *
 * Description: This C program samples the on-chip temperature sensor and
 *              converts the sampled voltage from the sensor to temperature in
 *              degrees Celsius and Fahrenheit. The converted temperature is
 *              sent to HyperTerminal over the UART by using serial UART.
 *
 * Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (~1MHz)
 *              An external watch crystal between XIN & XOUT is required for ACLK
 *
 * Instructions:Set the following parameters in HyperTerminal
 *                  Port :        COM1
 *                  Baud rate :   115200
 *                  Data bits:    8
 *                  Parity:       None
 *                  Stop bits:    1
 *                  Flow Control: None
 *
 *                         MSP430F5529
 *                      -------------------
 *                   /|\|              XIN|-
 *                    | |                 | 32kHz
 *                    --|RST          XOUT|-
 *                      |                 |
 *                      |     P3.3/UCA0TXD|------------>
 *                      |                 | 115200 - 8N1
 *                      |     P3.4/UCA0RXD|<------------
 *                      |                 |
 * Input:       Character Y or y or N or n
 *
 * Output:      Displays Temperature in Celsius and Fahrenheit in HyperTerminal
 * Author:      Aleksandar Milenkovic, milenkovic@computer.org
 *              Prawar Poudel
 *------------------------------------------------------------------------------*/

#include  <msp430.h>
#include  <stdio.h>

#define CALADC12_15V_30C  *((unsigned int *)0x1A1A)   // Temperature Sensor
Calibration-30 C
                                                      //See device datasheet for TLV
table memory mapping
#define CALADC12_15V_85C  *((unsigned int *)0x1A1C)   // Temperature Sensor
Calibration-85 C

char ch;                    // Holds the received char from UART
unsigned char rx_flag;      // Status flag to indicate new char is received

char gm1[] = "Hello! I am an MSP430. Would you like to know my temperature? (Y|N)";
```

```
char gm2[] = "Bye, bye!";
char gm3[] = "Type in Y or N!";

long int temp;                      // Holds the output of ADC
long int IntDegF;                   // Temperature in degrees Fahrenheit
long int IntDegC;                   // Temperature in degrees Celsius

char NewTem[25];

void UART_setup(void) {

    P3SEL |= BIT3 + BIT4;   // Set USCI_A0 RXD/TXD to receive/transmit data
    UCA0CTL1 |= UCSWRST;    // Set software reset during initialization
    UCA0CTL0 = 0;           // USCI_A0 control register
    UCA0CTL1 |= UCSSEL_2;   // Clock source SMCLK

    UCA0BR0 = 0x09;         // 1048576 Hz  / 115200 lower byte
    UCA0BR1 = 0x00;         // upper byte
    UCA0MCTL = 0x02;        // Modulation (UCBRS0=0x01, UCOS16=0)

    UCA0CTL1 &= ~UCSWRST;   // Clear software reset to initialize USCI state machine
    UCA0IE |= UCRXIE;                        // Enable USCI_A0 RX interrupt
}

void UART_putCharacter(char c) {
    while (!(UCA0IFG&UCTXIFG));    // Wait for previous character to transmit
    UCA0TXBUF = c;                  // Put character into tx buffer
}

void sendMessage(char* msg, int len) {
    int i;
    for(i = 0; i < len; i++) {
        UART_putCharacter(msg[i]);
    }
    UART_putCharacter('\n');        // Newline
    UART_putCharacter('\r');        // Carriage return
}

void ADC_setup(void) {
    REFCTL0 &= ~REFMSTR;                     // Reset REFMSTR to hand over control
to
                                             // ADC12_A ref control registers
    ADC12CTL0 = ADC12SHT0_8 + ADC12REFON + ADC12ON;
                                             // Internal ref = 1.5V
    ADC12CTL1 = ADC12SHP;                    // enable sample timer
    ADC12MCTL0 = ADC12SREF_1 + ADC12INCH_10; // ADC i/p ch A10 = temp sense i/p
    ADC12IE = 0x001;                         // ADC_IFG upon conv result-ADCMEMO
    __delay_cycles(100);                      // delay to allow Ref to settle
    ADC12CTL0 |= ADC12ENC;
}

void main(void) {
    WDTCTL = WDTPW | WDTHOLD;         // Stop watchdog timer
    UART_setup();                     // Setup USCI_A0 module in UART mode
    ADC_setup();                      // Setup ADC12
```

```

    rx_flag = 0;                      // RX default state "empty"
    _EINT();                          // Enable global interrupts
    while(1) {
        sendMessage(gm1, sizeof(gm1));// Send a greetings message

        while(!(rx_flag&0x01));       // Wait for input
        rx_flag = 0;                  // Clear rx_flag
        sendMessage(&ch, 1);          // Send received char

        // Character input validation
        if ((ch == 'y') || (ch == 'Y')) {

            ADC12CTL0 &= ~ADC12SC;
            ADC12CTL0 |= ADC12SC;                   // Sampling and conversion start

            _BIS_SR(CPUOFF + GIE);      // LPM0 with interrupts enabled

            //in the following equation,
            // ..temp is digital value read
            //..we are using double intercept equation to compute the
            //.. .. temperature given by temp value
            //.. .. using observations at 85 C and 30 C as reference
            IntDegC = (float)(((long)temp - CALADC12_15V_30C) * (85 - 30)) /
                    (CALADC12_15V_85C - CALADC12_15V_30C) + 30.0f;

            IntDegF = IntDegC*(9/5.0) + 32.0;

            // Printing the temperature on HyperTerminal/Putty
            sprintf(NewTem, "T(F)=%ld\tT(C)=%ld\n", IntDegF, IntDegC);
            sendMessage(NewTem, sizeof(NewTem));
        }
        else if ((ch == 'n') || (ch == 'N')) {
            sendMessage(gm2, sizeof(gm2));
            break;                      // Get out
        }
        else {
            sendMessage(gm3, sizeof(gm3));
        }
    }                                   // End of while
    while(1);                           // Stay here forever
}

#pragma vector = USCI_A0_VECTOR
__interrupt void USCIA0RX_ISR (void) {
    ch = UCA0RXBUF;                 // Copy the received char
    rx_flag = 0x01;                 // Signal to main
    LPM0_EXIT;
}

#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR (void) {
    temp = ADC12MEM0;               // Move results, IFG is cleared
    _BIC_SR_IRQ(CPUOFF);            // Clear CPUOFF bit from 0(SR)
}
```

Figure 2. 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 3 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.

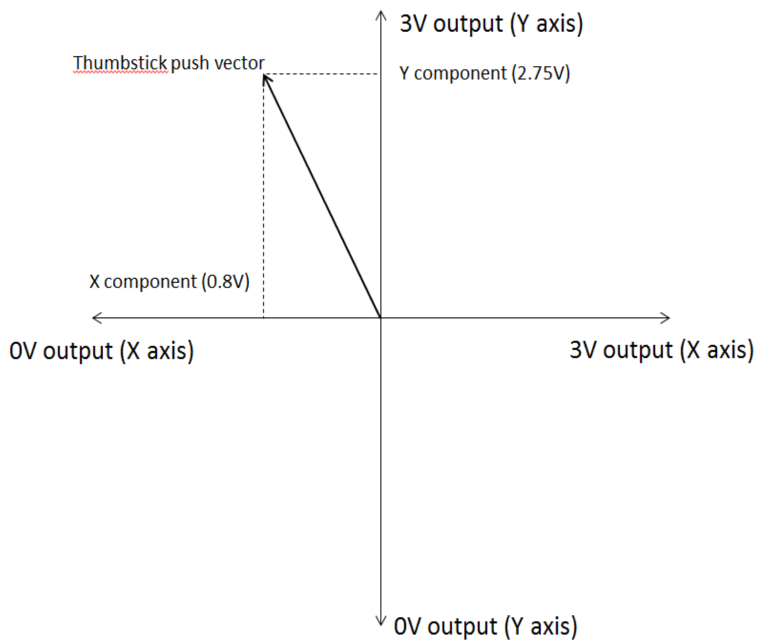


Figure 3. Performance data for hypothetical thumbstick

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 A3 for the X axis and A4 for the Y axis.

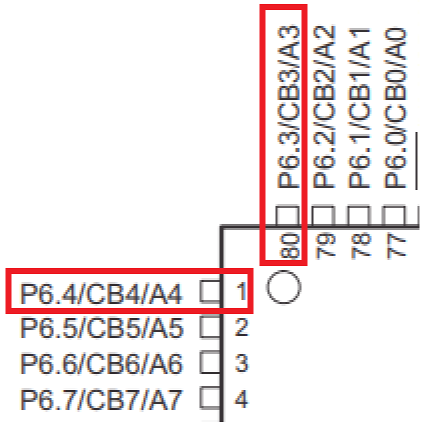


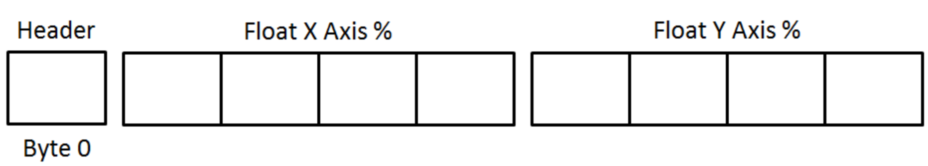
Figure 4. Pinouts and Header Connections for Analog Inputs

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-EXPF5529LP board. These pins are where we will connect horizontal HORZ and vertical VERT wires of the thumbstick. 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. The output for each axis should be a percentage. In Figure 3, 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):

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

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 5 shows how to configure UAH Serial App to accept two channels including single-precision floating-point numbers. Figure 6 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.



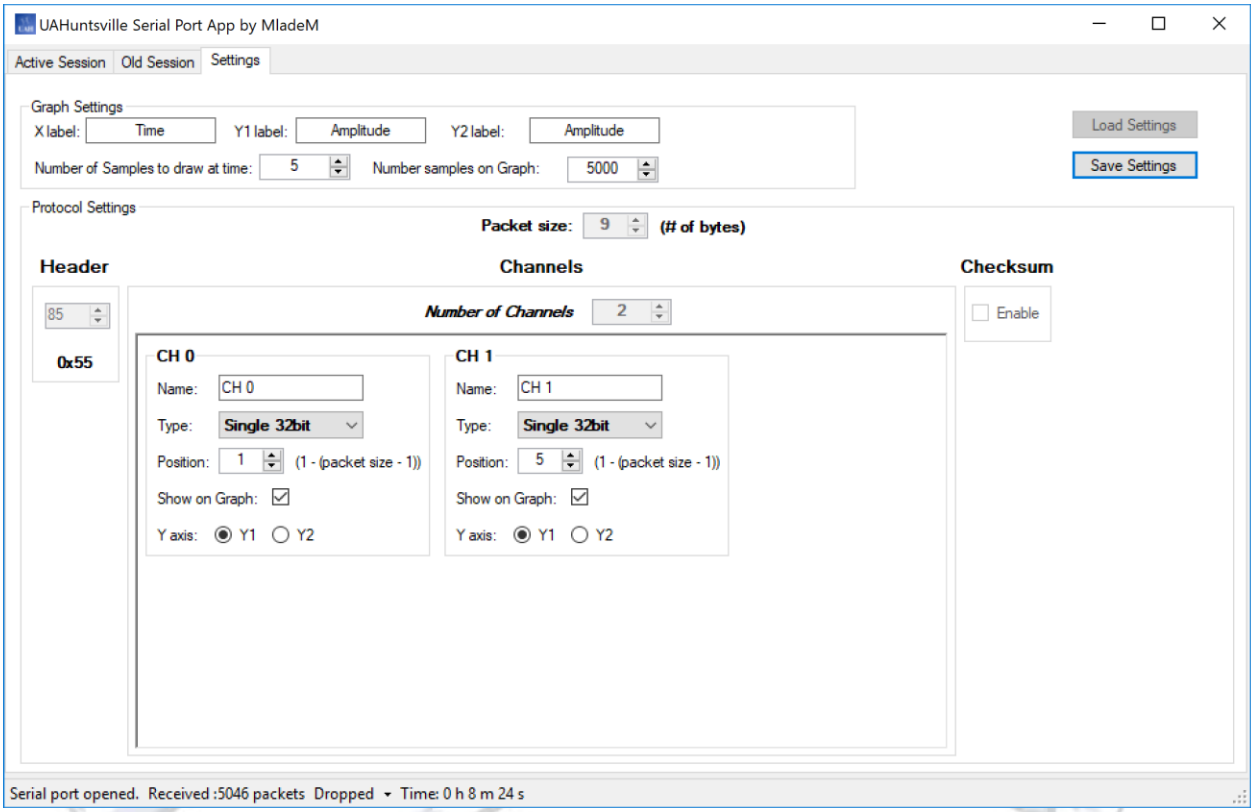


Figure 5. UAH Serial App Settings

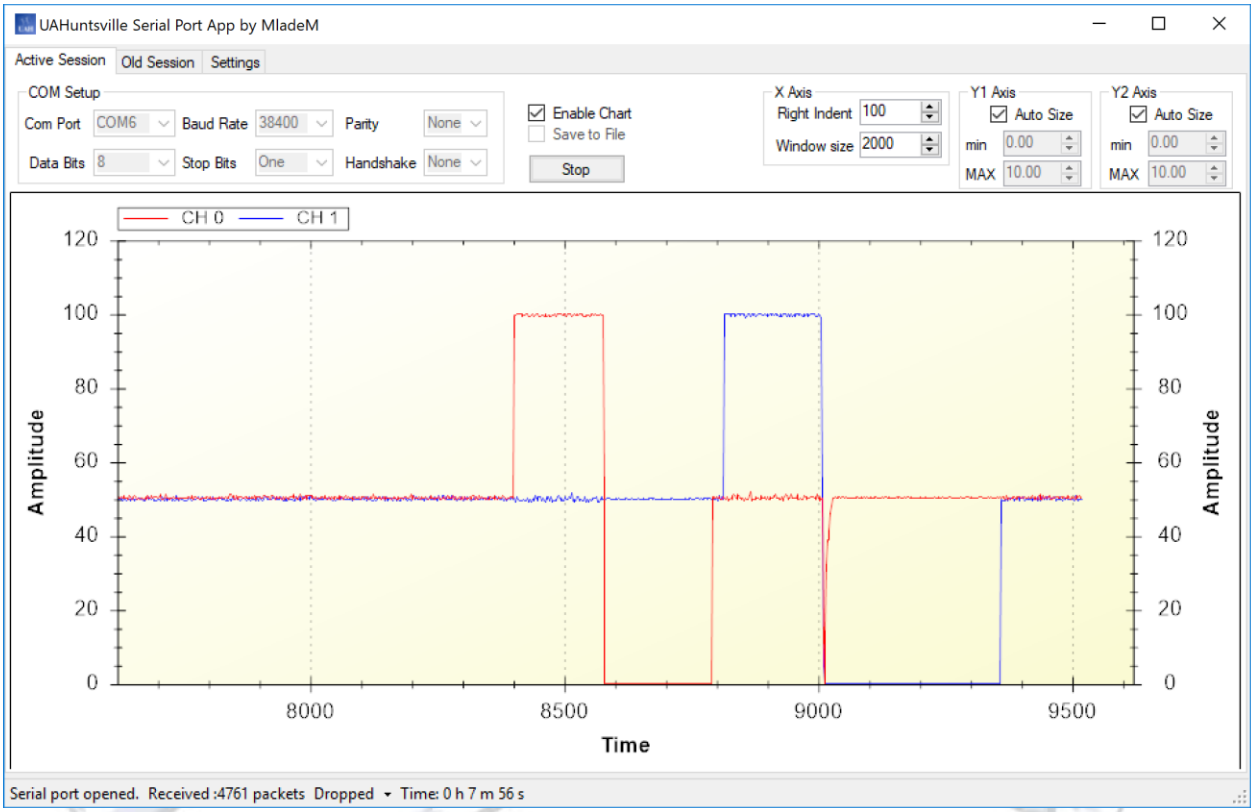


Figure 6. UAH Serial App Showing Percentage Signals from Thumbstick (CH0 – HORZ, CH1 – VERT)

Figure 7 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?

```
/*------------------------------------------------------------------------------
 * File:        Lab10_D2.c (CPE 325 Lab10 Demo code)
 * Function:    Interfacing thumbstick (MPS430F5529)
 * Description: This C program interfaces with a thumbstick sensor that has
 *              x (HORZ) and y (VERT) axis and outputs from 0 to 3V.
 *              The value of x and y axis
 *              is sent as the percentage of power to the UAH Serial App.
 *
 * Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (~1MHz)
 *              An external watch crystal between XIN & XOUT is required for ACLK
 *
```

```
 *                         MSP430F5529
 *                      -------------------
 *                   /|\|               XIN|-
 *                    | |                  | 32kHz
 *                    --|RST           XOUT|-
 *                      |                  |
 *                      |     P3.3/UCA0TXD|------------>
 *                      |                  | 38400 - 8N1
 *                      |     P3.4/UCA0RXD|<------------
 *                      |                  |
 * Input:       Connect thumbstick to the board
 * Output:      Displays % of power in UAH serial app
 * Author:      Micah Harvey
 *              Prawar Poudel
 *------------------------------------------------------------------------------*/

#include <msp430.h>

volatile long int ADCXval, ADCYval;
volatile float Xper, Yper;

void TimerA_setup(void) {
    TA0CCR0 = 3277;                         // 3277 / 32768 Hz = 0.1s
    TA0CTL = TASSEL_1 + MC_1;               // ACLK, up mode
    TA0CCTL0 = CCIE;                        // Enabled interrupt
}

void ADC_setup(void) {
    int i =0;

    P6DIR &= ~BIT3 + ~BIT4;                // Configure P6.4 and P6.4 as input pins
    P6SEL |= BIT3 + BIT4;                  // Configure P6.3 and P6.4 as analog pins
    // configure ADC converter
    ADC12CTL0 = ADC12ON + ADC12SHT0_6 + ADC12MSC_L;
    ADC12CTL1 = ADC12SHP + ADC12CONSEQ1;            // Use sample timer, single sequence
    ADC12MCTL0 = ADC12INCH_3;                   // ADC A3 pin - Stick X-axis
    ADC12MCTL1 = ADC12INCH_4 + ADC12EOS;            // ADC A4 pin - Stick Y-axis
                                           // EOS - End of Sequence for Conversions
    ADC12IE |= 0x02;                       // Enable ADC12IFG.1
    for (i = 0; i < 0x3600; i++);          // Delay for reference start-up
    ADC12CTL0 |= ADC12ENC;                      // Enable conversions
}

void UART_putCharacter(char c) {
    while (!(UCA0IFG&UCTXIFG));     // Wait for previous character to transmit
    UCA0TXBUF = c;                   // Put character into tx buffer
}

void UART_setup(void) {
    P3SEL |= BIT3 + BIT4;                  // Set up Rx and Tx bits
    UCA0CTL0 = 0;                          // Set up default RS-232 protocol
    UCA0CTL1 |= BIT0 + UCSSEL_2;           // Disable device, set clock
    UCA0BR0 = 27;                          // 1048576 Hz / 38400
    UCA0BR1 = 0;
    UCA0MCTL = 0x94;
```

```c
    UCA0CTL1 &= ~BIT0;                    // Start UART device
}

void sendData(void) {
    int i;
    Xper = (ADCXval*3.0/4095*100/3);      // Calculate percentage outputs
    Yper = (ADCYval*3.0/4095*100/3);
    // Use character pointers to send one byte at a time
    char *xpointer=(char *)&Xper;
    char *ypointer=(char *)&Yper;

    UART_putCharacter(0x55);              // Send header
    for(i = 0; i < 4; i++) {              // Send x percentage - one byte at a time
        UART_putCharacter(xpointer[i]);
    }
    for(i = 0; i < 4; i++) {              // Send y percentage - one byte at a time
        UART_putCharacter(ypointer[i]);
    }
}

void main(void) {
    WDTCTL = WDTPW +WDTHOLD;              // Stop WDT
    TimerA_setup();                       // Setup timer to send ADC data
    ADC_setup();                          // Setup ADC
    UART_setup();                         // Setup UART for RS-232
    _EINT();

    while (1){
        ADC12CTL0 |= ADC12SC;                 // Start conversions
        __bis_SR_register(LPM0_bits + GIE); // Enter LPM0
    }
}

#pragma vector = ADC12_VECTOR
__interrupt void ADC12ISR(void) {
    ADCXval = ADC12MEM0;                    // Move results, IFG is cleared
    ADCYval = ADC12MEM1;
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void timerA_isr() {
    sendData();                             // Send data to serial app
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}
```

Figure 7 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)