

Designing a Humidity Sensing System With UART Communication

James Fashoda

11/13/15

This guide will teach the user how to properly design and program a system which takes the analog output of a humidity sensor, processes the signal through the analog-to-digital converter (ADC) of a microcontroller, and outputs it to a computer using UART communication.

Communication, Sensors, UART, ADC, Microcontrollers

Parts Necessary:

Microcontroller w/ UART & ADC	[MSP430G2553]
Sensor w/ Analog Output	[HIH-5030]
Transmitter/Receiver Pair (w/ Antennas)	[MX-FS-03V, MX-05V]
External Power Source	

Introduction:

In today's day and age, sensor systems have become a part of everyday life. This type of system can be found in many modern day appliances and this tutorial in specific walks through the implementation of a humidity sensor system which takes its readings from inside the drum of a dryer. This information is then transmitted to a control unit located outside of the drum. For the sake of this tutorial, we will emulate the RF communication outside of the drum to prevent any possible interference.

Creating the Circuit

Before we can start building a circuit, the components which we will be using must be analyzed so that we can properly interface them together. A key characteristic we must focus on early is power consumption. This information is most easily readable in a table format as shown below:

Component	Operating Voltage Range	Current
HIH-5030	2.7-5.5V	200uA-500uA
MSP430G2553	1.8-3.6V	Active: 230uA Standby: .5uA Off Mode: .1uA
MX-FS-03V	3.5-12V	
MX-05V	5V	

Since our component don't work together comfortably at a single voltage (possibly 3.55V, but let's keep it safe), we must use a voltage regulator to take the larger voltage necessary for the sensor and transmitter and step it down to a voltage which the MCU can operate in. However, since we will be programming/debugging the MCU using the USB interface of the experimental PCB, we can power the MCU with the USB and power the sensor and transmitter with a 4V source.

Now we must interface our components with the MCU. The humidity sensor will output an analog voltage which increases linearly with the humidity. In order to convert this analog voltage into a digital value which the MCU can use, we must interface this sensor with one of the 10 available ADC hosted on the MCU. There is an ADC located on Pin 1.1, so we will use that one. We will discuss programming the ADC later on in this guide. The MSP430G2553 has a UART pin located on Pin 1.2 which can be used specifically for a transmitter, so we can interface our transmitter with this pin. Once we have all of our components placed correctly, we can begin programming the MCU using the TI Code Composer Studio.

Programming the ADC

We will now discuss how to set up a basic ADC to convert the output of our humidity sensor and display the result using the TI Code Composer Studio. The next few steps will highlight how to create a new project.

Once we select a workspace to save all of our work in, we can begin our new project. Navigate to the window toolbar and select **File > New > CCS Project**. After selecting a proper project name, we must find our device in the section labeled **Device**. Under **Project templates and examples**, select **Empty Project** and then we can click **Finish**. At this point, we can create our source file which we will build our basic code under. Select **File > New > Source File**, properly name your source file, then click on finish.

Now we can finally begin writing our code. Below is a sample code:

```
#include <msp430g2553.h>
float voltageRaw;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD; // Watchdog Timer

    // Set Frequency
    DCOCTL = CALDCO_1MHZ;
    BCSCTL1 = CALBC1_1MHZ;

    // Initialize ADC
    ADC10CTL1 = INCH_1 + CONSEQ_2;
    ADC10AE0 = BIT1;
    ADC10CTL0 = SREF_1 + REF2_5V + REFON + ADC10ON + MSC;

    // Wait for Reference Voltage to settle
    _delay_cycles(5);

    while(1)
    {
        // Begin conversion
        ADC10CTL0 |= ENC + ADC10SC;
        voltageRaw = ADC10MEM;
    }
}
```

The first line of code is used to include the header file specific to the MSP430G2553. Next, we can initialize all of our variables necessary for our program. We then step into the main body of our program.

The first few lines of our main function simply initialize of the clocks and registers which we will be using. More detailed information on how to set up these registers can be found in the MSP430G2553 User's Guide^[2]. The first line is used to stop what is known as the Watchdog Timer, which is an interrupt that runs every time the MSP430 is started and can interrupt the while loop we are using. The next 2 lines simply set the frequency of the clock system to 1MHz.

Next, we start initializing our ADC. The next line initializes the ADC control register by selecting ADC channel A1 (INCH_1) and selecting repeat single channel mode (CONSEQ_2). Next, we initialize our input pin since all of the analog pins are initially turned off to conserve power. Finally, we configure the final control register by setting the reference voltage at 2.5V (which is the max output voltage of HIH-5030), turning on the ADC, and setting the ADC to continue taking samples instead of waiting to be told to take another. The delay cycle is used to allow the reference voltage to settle.

It is here that we can finally begin our conversion process inside of our while loop. The first line both enables conversion and starts the conversion process. The conversion result is then stored in the variable **ADC10MEM**. Since we have a 10 bit ADC, the digital value which will be saved is between 0 and $2^{10} - 1$, which is equal to 0 to 1023. This value will reflect what the sensor is currently outputting, with 0V corresponding to 0 and 2.5V corresponding to 1023. The result is saved into a temporary variable **voltageRaw** so that we may operate on it later if need be.

Once we have successfully designed our code, we can debug it using the debugging tool available in TI Code Composer Studio. Here, we can actively see what our sensor is currently outputting in real time. With our ADC completed, we can now shift our focus to sending the result of our ADC to a computer.

Initiating UART Transmission

To communicate with our computer, we must utilize the UART registers available on the MSP430G2553. Sending the data by itself is actually as simple as filling a register, but initializing the UART is rather complicated. In depth information on how to initialize the UART can also be found in the MSP4302553 User's Guide^[2].

There are many methods we can use to transmit data, but the most effective method involves the use of interrupts and interrupt service routines (ISR). These ISR are called when interrupts occur and they are used to run a routine in response to that interrupt. This is very useful for power consumption as we can have the system usually in a sleep state which only activates when there is

information to transfer. Here, we will use an interrupt to transmit data whenever the transmitter interrupt is called. The code which we used is below:

```
#include "msp430g2553.h"

char *string = "Hello, world.\r\n";    // a 15 character message
int tx_count = 0;

void main(void) {
    WDTCTL = WDTPW + WDTHOLD;          // Stop Watchdog timer

    // Set Frequency
    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;

    // Initialize UART registers
    UCA0CTL1 = UCSWRST;
    UCA0CTL1 |= UCSSEL_2;               // CLK = SMCLK

    UCA0BR0 = 104;                      // 1MHz/9600 = 104
    UCA0BR1 = 0x00;
    UCA0MCTL = 0;                       // Modulation = 0

    P1SEL = BIT2;                       // P1.2=TXD
    P1SEL2 = BIT2;                      // P1.2=TXD

    UCA0CTL1 &= ~UCSWRST;               // Enable UART

    // Sleep; ISRs operate from here
    while( 1 ) {
        __bis_SR_register( LPM3_bits + GIE );
        UC0IE |= UCA0TXIE;
    }
}

// Transmitter ISR
void USCI_A_tx (void) __attribute__((interrupt(USCIAB0TX_VECTOR)));
void USCI_A_tx (void) {
    UCA0TXBUF = string[ tx_count++ ];   // Load the next character

    // All characters finished
    if( tx_count >= 15 ) {
        IE2 &= ~UCA0TXIE;              // Disable this routine so it won't be called again
        tx_count = 0;
    }
}
```

First, we initialize a character string which we will use to store our information and pass it to the computer and we also initialize a counter. The first

few lines of our main function should look familiar since we set them the same way for our ADC. After this is set up, we can begin initializing our UART.

First, we reset the control register to clear any previous data. Since the MSP430G2553 hosts 3 different clocks we must choose the one best suited for this design, which will be the SMCLK. Next, we set the frequency to 1MHz and the baud rate to 9600 by initializing **UCA0BR0** & **UCA0BR1** with predetermined values located in the User's Guide. We don't want any modulation with our baud rate, so the next line sets our modulation to 0. The next 2 lines sets Pin 1.2 as an output since will be hosting our transmitter on the TXD pin located on Pin 1.2. We finally enable the UART communication and step into an infinite while loop. Here, we can set the MSP430 into low power and allow the interrupts to take over. The 2nd line is used to re-enable the interrupt after a transmission is finished.

The transmitter ISR is pretty straightforward. Once it is called, it sends the string through a transmitter buffer **UCA0TXBUF** once char at a time until the entire message is sent. Once it is finished, the interrupt is disabled so that it is not called again with nothing left to transmit. Since we have accomplished both ADC and UART communication, we can now implement them together to complete our system.

Final Code:

```
#include "msp430g2553.h"

char string[] = "                \r\n";
int tx_count = 0;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;                // Stop Watchdog timer

    // UART settings depend on an accurate 1 MHz clock.
    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;

    // Initialize the UART
    UCA0CTL1 = UCSWRST;
    UCA0CTL1 |= UCSSEL_2;                    // CLK = SMCLK
    UCA0BR0 = 104;                          // 1MHz/9600 =104
    UCA0BR1 = 0x00;
    UCA0MCTL = 0;                          // Modulation = 0
```

```

// Configure ports
P1SEL = BIT2;           // P1.2=TXD
P1SEL2= BIT2;           // P1.2=TXD

UCA0CTL1 &= ~UCSWRST;   // Enable UART

// Initialize the ADC10
ADC10CTL0 = SREF_1 + REF2_5V + REFON + ADC10ON + MSC;
ADC10CTL1 = INCH_1 + CONSEQ_2;
ADC10AE0 = BIT1;

// Begin Loop
while( 1 ) {
    __bis_SR_register( LPM3_bits + GIE );

    ADC10CTL0 |= ENC + ADC10SC;
    while( ADC10CTL1 & ADC10BUSY ) ;

    itoa( ADC10MEM, &hello, 10 );           // convert to a string
    IE2 |= UCA0TXIE;                         // enable interrupts
}

// Transmitter ISR
void USCI_A_tx (void) __attribute__((interrupt(USCIAB0TX_VECTOR)));
void USCI_A_tx (void) {
    UCA0TXBUF = hello[ tx_count++ ];        // Load the next character

    // Character have all sent
    if( tx_count >= sizeof string ) {
        IE2 &= ~UCA0TXIE;                  // disable this routine so it won't be called again
        tx_count = 0;                       // Reset Count
    }
}

```

Conclusion:

In conclusion, developing a system such as this takes a great deal of research into different aspects of circuit design and programming. Since I am currently designing a similar system, developing this process has been a great learning experience and should be very helpful as I progress through my own problems. This guide is not all encompassing, as there are many other methods which can be used to emulate this same process. Users who would like to develop this method further or simply start their own should refer to the User's Guide of the MSP430G2553 in order to fully optimize their system.

References:

[1] *TI MSP430G2x53/MSP430G2x13 Datasheet*; May 2013

<http://www.ti.com/lit/ds/symlink/msp430g2553.pdf>

[2] *TI MSP430x2xx Family User's Guide*; July 2013

<http://www.ti.com/lit/ug/slau144j/slau144j.pdf>

[3] *Honeywell HIH-5030/5031 Datasheet*

http://sensing.honeywell.com/index.php?ci_id=49692

[4] Enrico Garante (July 3, 2013) *MSP430 Launchpad Tutorial*

<http://www.embeddedrelated.com/showarticle/420.php>