Intro to Embedded Systems
MEMS Project

Aakash Tripathi, Alexander Yu, John Newman
Section 1
December 12, 2020

Introduction

In this lab, our group was required to write code for the MSP430F5529 that took data from the on-board ADC12 and transmit it over simulated UART. It also asked us to use MATLAB or LabView to show the data being output by the COM port in a graphical format. To do this, we must first connect the two outputs of the MEMS microphone to two input pins on the microcontroller, and then transmit it over UART to the computer.

Objectives

1. To understand the operation of the MSP430F5529's ADC12 and UART
2. To become familiar with the concepts of UART and analog to digital converters in general
3. Gain experience with real-life applications of embedded systems

Methodology

The MSP430 part of our design uses a fairly linear flow of data. First, the AC and DC outputs of MEMS microphone are connected to pins 6.0 and 6.1, respectively. These pins are connected to several of the peripherals on the board, namely the GPIO and analog-to-digital converter with a reference voltage of 2.5 volts. The ADC, which has a resolution of 12 bits, takes the analog voltage reading from the pin and converts it to a 12-bit digital number out of 4095, and stores it on a control register. The ADC then sends the data from these registers to the microcontroller which converts these numbers into ASCII. The microcontroller then sends the data to pin 4.4. This pin is connected to the USCI simulated UART output. This is connected to the onboard USB protocol handler, which is connected via USB cable to the COM port on a computer which reads in the ASCII numbers.

The MATLAB side of the project first reads in the data in the form of ASCII numbers and takes them as data points. Next, the program converts the ASCII into double data type. Next, the data is broken into its respective AC and DC parts. Which is finally used for plotting both the AC and the DC data simultaneously. This graph will run indefinitely until the user terminates the program. Because of this, we set up the graph to scroll as the data comes in, only showing the data from the last second or so.

Discussion

The general linear data flow outlined in the methodology section can be seen better in the high-level view of the hardware layout , shown in Figure 1. More specifically in figure 2 the data flow block diagram highlights the ADC 12 communication between the MEMs mic and the MSP 430. The communication between the MEMs mic and the MSP 430 microcontroller is done by the ADC 12. The code flow chart for the ADC 12 control register can be seen outlined in figure 3.

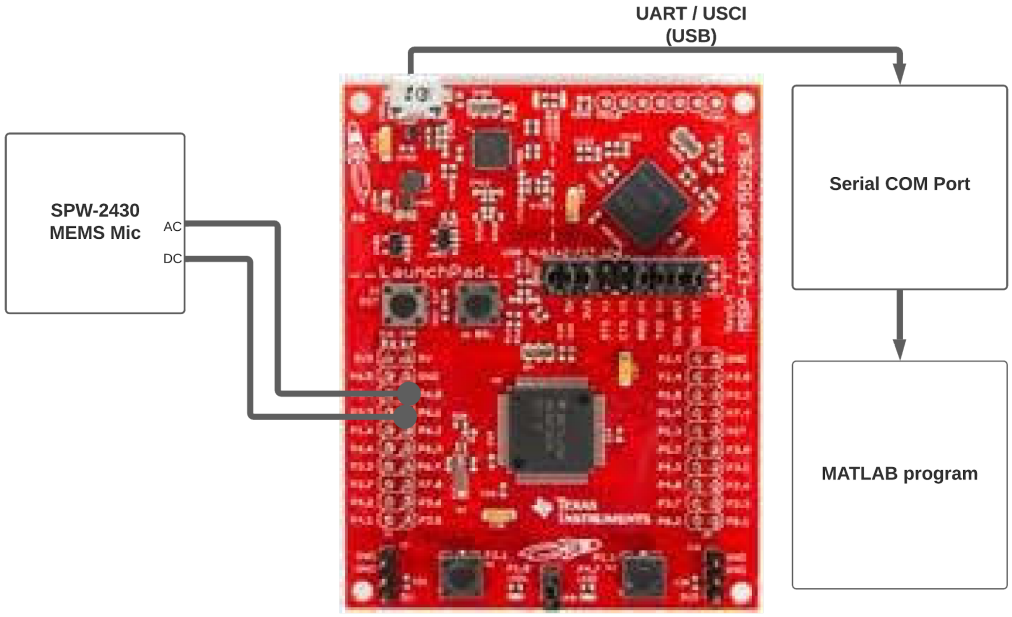

Figure 1: High level view of the hardware layout

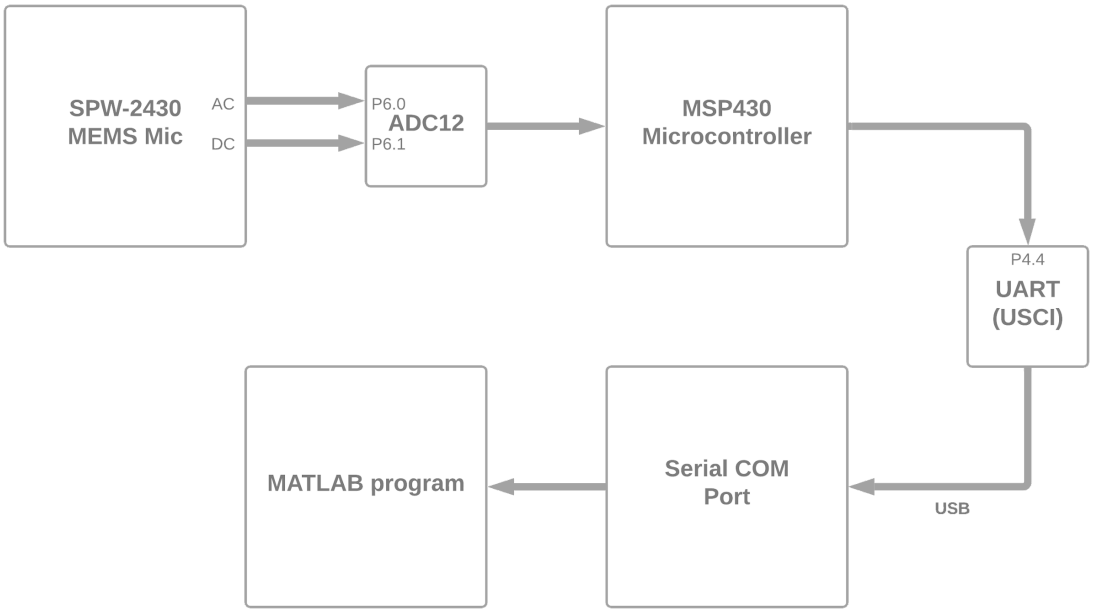

Figure 2: Data flow block diagram

The ADC 12 has four corresponding control registers used for converting analog to digital signal. The four controlled registers are: ADC 12 memory control 0, ADC 12 memory control 1, ADC 12+control 0, and ADC 12+ control 1. The ADC-12 memory control registers, zero and one, are responsible for creating two input channels. These two input channels allow for both the

AC and the DC inputs of the mems mic to be registered, and then be transmitted from ADC-12 conversion memory one and memory 2 using TX.

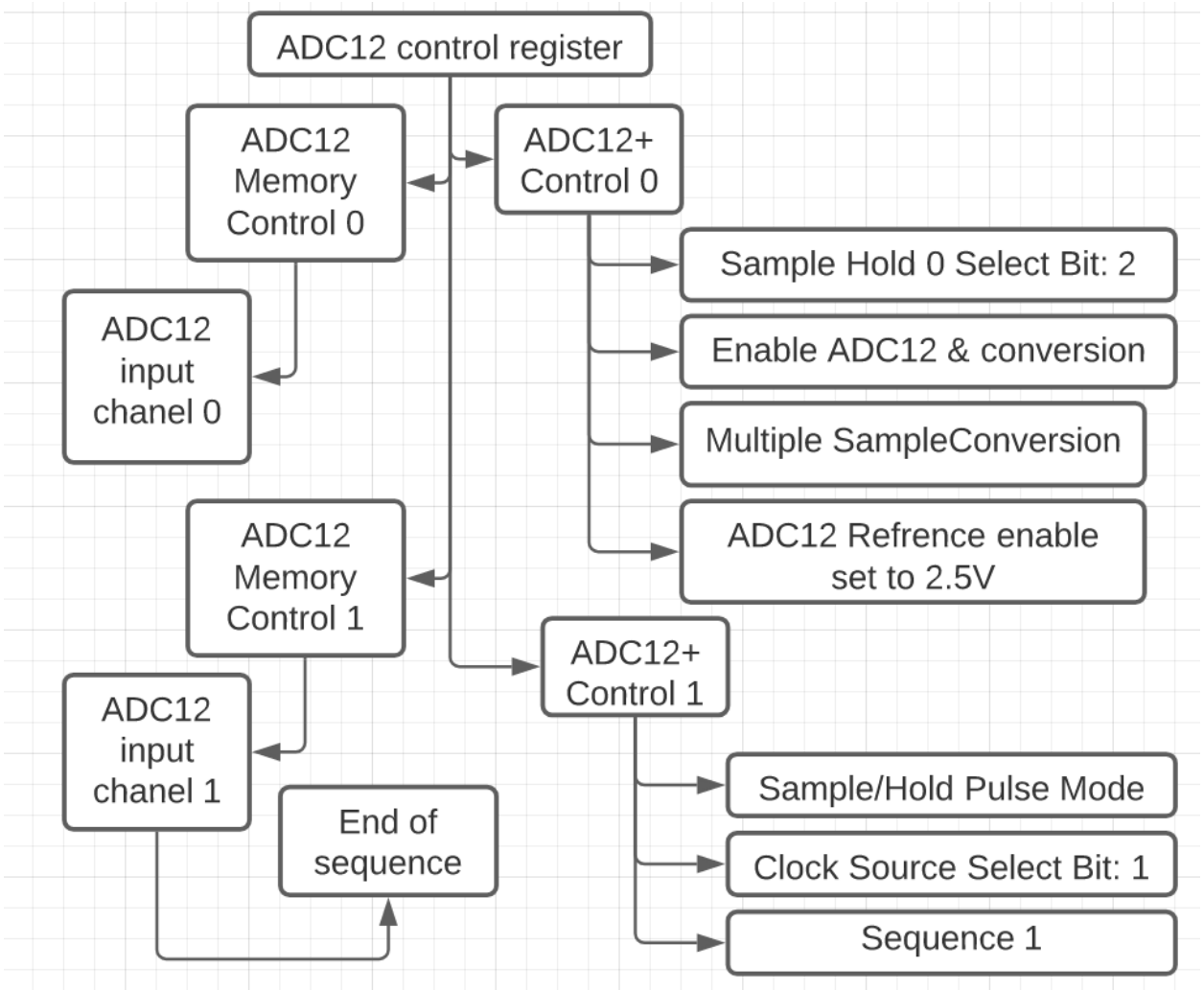

Figure 3: ADC 12 control register flowchart

The output of the system and the performance of the code can be manipulated in various ways. Most significantly, the quality of readings that you get from the ADC depend greatly on the resolution of the ADC. The ADC used in this project has a resolution of 12 bits. That means that the ADC has 2^{12} - 1, or 4095, distinct steps corresponding to voltage inputs. The more steps that the ADC can read, the more accurate the readings are. Another way to change the output of the code is to change the reference voltage that the ADC uses. We use a reference voltage of 2.5V, but the MSP430F5529 can use a 1.5V reference as well. Using a higher reference voltage gives you a wider range of voltages to work with, but the distance between each step is wider, while using a lower reference has the opposite effect. The performance of the code, namely the power consumption, can be affected by the use of interrupts and low-power-modes of the MSP430, though we did not use them in our design.

Conclusion

Over the course of the lab, the group has picked up many skills related to embedded systems. As we were told, we wrote code to take data from the ADC12 and transmit it over to UART. The group took a lot of time figuring out how to set up the MSP code to read serial data. The input data that needed to be read came from the MEMS microphone. This allowed us to learn and understand how to read and convert the AC and DC data that the mic was producing. After that was done, the group then fed the data into MATLAB. Through MATLAB, we were able to plot and analyze the data that was collected. Overall, the experience taught us many concepts that can be taken through our college careers.

Appendix 1: MSP430F5529 Code

```
#include <msp430.h>
int main(void)
{
   WDTCTL = WDTPW + WDTHOLD;   // Stop WDT
   P4SEL |= BIT4 + BIT5;       // P4.4 & P4.5 = USCI_A1 TXD / RXD
   UCA1CTL1 |= UCSWRST;        // Reset state machine
   UCA1CTL1 |= UCSSEL_2;       // SMCLK
   UCA1BR0 = 109;              // Baud Rate = 9600
   UCA1BR1 = 0;                // USCI A1 Baud Rate 1
   UCA1MCTL |= UCBRS_2 + UCBRF_0; // Modulation UCBRSx=2, UCBRFx=0
   UCA1CTL1 &= ~UCSWRST;       // Initialize USCI state machine

   P6SEL |= BIT0; // P6.0 Select
   P6SEL |= BIT1; // P6.1 Select

   REFCTL0 &= ~REFMSTR;
   /*
    *REF Shared Reference control register 0 &= ~REF Master Control
    *Changes internal reference voltage to ADC12 control registers
    */

   ADC12CTL0 = ADC12SHT02 | ADC12ON | ADC12MSC| ADC12REFON  |
ADC12REF2_5V;
   /* ADC12CTL0    =    ADC12+ Control 0
    * ADC12SHT02   =    ADC12 Sample Hold 0 Select  Bit: 2
    * ADC12ON      =    ADC12 On/enable
    * ADC12MSC     =    ADC12 Multiple SampleConversion
    * ADC12REFON   =    ADC12 Reference on
    * ADC12REF2_5V =    ADC12 Ref 0:1.5V / 1:2.5V
    */


   ADC12CTL1 = ADC12SHP | ADC12SSEL1 | ADC12CONSEQ_1;
   // ADC12 Sample/Hold Pulse Mode, ADC12 Clock Source  Select Bit: 1, sequence 1

   ADC12MCTL0 |= ADC12INCH_0;
   // ADC12 Memory Control 1 |= ADC12 Input Channel  0
```

```
  ADC12MCTL1 |= ADC12INCH_1 | ADC12EOS;
  // ADC12 Memory Control 1 =  ADC12 Input Channel  1 | End of sequence

  ADC12CTL0 |= ADC12ENC;
  // ADC12 Enable Conversion

  while(1)
  {
    ADC12CTL0 |= ADC12SC;
    while (ADC12CTL1 & ADC12BUSY);
    int thou =  ((int)ADC12MEM0)/1000;
    int hund =  ((int)ADC12MEM0)%1000/100;
    int ten  =  ((int)ADC12MEM0)%1000%100/10;
    int one  =  ((int)ADC12MEM0)%1000%100%10;
    while (!(UCA1IFG&UCTXIFG)); // USCI_A1 TX buffer ready?
    UCA1TXBUF= thou + 0x30;
    while (!(UCA1IFG&UCTXIFG)); // USCI_A1 TX buffer ready?
    UCA1TXBUF= hund + 0x30;
    while (!(UCA1IFG&UCTXIFG)); // USCI_A1 TX buffer ready?
    UCA1TXBUF= ten + 0x30;
    while (!(UCA1IFG&UCTXIFG)); // USCI_A1 TX buffer ready?
    UCA1TXBUF= one + 0x30;
    thou =  ((int)ADC12MEM1)/1000;
    hund =  ((int)ADC12MEM1)%1000/100;
    ten  =  ((int)ADC12MEM1)%1000%100/10;
    one  =  ((int)ADC12MEM1)%1000%100%10;
    while (!(UCA1IFG&UCTXIFG)); // USCI_A1 TX buffer ready?
    UCA1TXBUF= thou + 0x30;
    while (!(UCA1IFG&UCTXIFG)); // USCI_A1 TX buffer ready?
    UCA1TXBUF= hund + 0x30;
    while (!(UCA1IFG&UCTXIFG)); // USCI_A1 TX buffer ready?
    UCA1TXBUF= ten + 0x30;
    while (!(UCA1IFG&UCTXIFG)); // USCI_A1 TX buffer ready?
    UCA1TXBUF= one + 0x30;
    while (!(UCA1IFG&UCTXIFG)); // USCI_A1 TX buffer ready?
    UCA1TXBUF= (0x0A);  //Enter
    while (!(UCA1IFG&UCTXIFG)); // USCI_A1 TX buffer ready?
    UCA1TXBUF= (0x0D);  //Carriage return
  }
}
```

Appendix 2: MATLAB Code

```
close all; clear all; clf
s = serial('COM3', 'BaudRate', 9600, 'FlowControl', 'hardware');
s.InputBufferSize = 512;
s.Terminator = 'CR';
fopen(s);
time = 0;
ACvoltage = 0;
DCvoltage = 0;
count = 0;
tic

for j = 0:10
   out = fscanf(s);
end

while(1)
   out = fscanf(s);
   ac = str2double(out(1:4))*2.5/4095;
   dc = str2double(out(5:8))*2.5/4095;
   count = count + 1;
   time(count) = toc;
   ACvoltage(count) = ac;
   DCvoltage(count) = dc;

   plot(time,ACvoltage,'r',time,DCvoltage,'b')
   xlim([time(count)-1 time(count)]);
   ylim ([0 2.5])

%    title('MEMS Mic AC & DC Data')
%    xlabel('time(t)')
%    ylabel('Voltage(V)')
%    legend({'AC','DC'})
   drawnow;
   pause(1e-10);
end
```