Christopher Brant
C19816588
Due on 11/30/2017

ECE 3270 Microcontroller Interfacing Lab
Lab 10: Analog to Digital Conversion

Abstract

The purpose of this lab was to learn about the use of the Analog to Digital Conversion, or ADC, module on the PIC32MX150F128D microcontroller. This was done by utilizing a pressure sensitive variable resistor that allowed variable amounts of current through, and its output signal was then converted from an analog signal to a digital signal. That digital output signal was then displayed on the series of 8 LEDs in binary.

Introduction

In this experiment, we learned how to utilize the PIC32MX150F128D's ADC module and we utilized an FSR to create an analog signal to send to the MCU as the FSR is pressed with variable amounts of pressure. When the FSR is pressed, the digital value that the MCU is supposed to output to the series of LEDs will change, increasing as pressure increases, and decreases as pressure decreases.

Experimental Procedures

- The first thing that must be done is to wire the overall circuit to be just as it is shown in Figure 1.
- The next thing that is to be done is to write the code for the interrupt to write the output values of the ADC module to the LEDs.
- Then, in your code you must write the set up of the MCU's pins, the ADC module, including the basic steps that are outlined in Section 17.4, as well as setting up the interrupt enable and flags for the ADC module as well.
- Then the rest of the code should be written just as it is seen in Figure 2, as the set up is the majority of this lab, since the ADC module takes care of the majority of the necessary steps for shifting data as long as the module has its setup bits written correctly.

Results

There were no tabulated values and no values necessary for the record. Observations of this lab show that the difficulty in ensuring your wires are in the correct rows they should be in on the breadboard becomes more and more difficult for each successive lab, up until and including this final lab. Overall, this lab was quite straightforward and made sense as long as all of the setup bits were set correctly in your code. Otherwise, there is not much to worry about other than wiring.

Christopher Brant
C19816588
Due on 11/30/2017

Discussion

Overall, the final conclusions of this experiment include the statements that the microcontroller can be programmed to take in an input value of a continuous and analog signal, which then can be input into an internal module which will convert the signal to a digital value and then output that signal where it is told to write to. Another conclusion would be that the importance of having every setup bit set correctly cannot be understated, as anything set incorrectly will cause the ADC module to function incorrectly or not at all, and this is especially important as there are so many setup bits to set, including remembering to setup the interrupt itself that needs to have a function and be enabled.

Conclusions

The conclusions of this experiment include that the MCU can be programmed to convert a series of analog signal values into a set binary width digital value, which is what the ADC module is designed to do of course, as well as another conclusion is that the importance of ensuring your setup bits are correct cannot be understated. Lastly, this experiment propelled us to learn the setup for the configuration necessary to utilize the ADC module in conjunction with an FSR to observe the operation of the ADC module on our MCU.

Figures and Tables

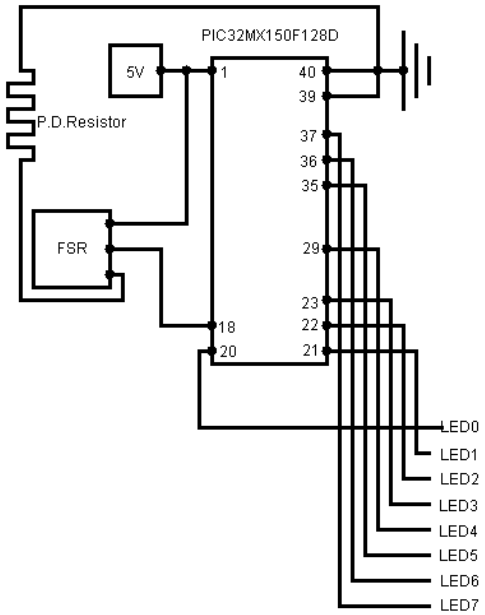

Figure 1: Wiring Diagram of the circuit for Lab 10: Analog to Digital Conversion

Christopher Brant
C19816588
Due on 11/30/2017

```c
/* Christopher Brant
 * C19816588
 * Lab 10: Analog to Digital Conversion
 * 11/16/2017
 */

#include <plib.h>

int ADCsum, LEDvalue;

void __ISR(23) adcDone(void)
{
    ADCsum = 0;

    // Sum the values of the 8 most significant bits in each buffer
    ADCsum += (ADC1BUF0 >> 2);
    ADCsum += (ADC1BUF1 >> 2);
    ADCsum += (ADC1BUF2 >> 2);
    ADCsum += (ADC1BUF3 >> 2);
    ADCsum += (ADC1BUF4 >> 2);
    ADCsum += (ADC1BUF5 >> 2);
    ADCsum += (ADC1BUF6 >> 2);
    ADCsum += (ADC1BUF7 >> 2);
    ADCsum += (ADC1BUF8 >> 2);
    ADCsum += (ADC1BUF9 >> 2);
    ADCsum += (ADC1BUFA >> 2);
    ADCsum += (ADC1BUFB >> 2);
    ADCsum += (ADC1BUFC >> 2);
    ADCsum += (ADC1BUFD >> 2);
    ADCsum += (ADC1BUFE >> 2);

    // Find the mean of those values
    LEDvalue = ADCsum / 15;

    // Write that value to the LEDs
    LATB = LEDvalue;
}
int main(void)
{
    // Setting tristate registers
    TRISA = 0x01;
    TRISB = 0x00;

    // Setting analog inputs
    ANSELA = 0x01;

    // Setting up ADC interrupts for use
    INTEnableSystemMultiVectoredInt();
    IFS0bits.AD1IF = 0;     // Clears ADC interrupt flag
    IPC5bits.AD1IP = 1;     // Sets ADC interrupt priority to 1
    IEC0bits.AD1IE = 1;     // Enables ADC interrupt

    // Setting up the data output format
    AD1CON1bits.FORM = 0;   // Sets to 16-bit integer
    // Setting up the sampling and conversion triggers
    AD1CON1bits.SSRC = 7;   // Sets auto-conversion on
    AD1CON1bits.ASAM = 1;   // Sets auto-sampling on
    // Setting voltage references
    AD1CON2bits.VCFG = 0;   // Sets the voltage references to auto-values
    // Setting channel scan and samples/converts per interrupt
    AD1CON2bits.CSCNA = 0;  // Turns scanning off
    AD1CON2bits.SMPI = 14;  // Interrupts after the 15th sample/convert
    // Setting buffer mode select and input sample select
    AD1CON2bits.BUFM = 0;   // Sets buffer to a single 16-bit word buffer
    AD1CON2bits.ALTS = 0;   // Sets MUX A as input always
    // Setting clock reference and auto-sample time bits
    AD1CON3bits.ADRC = 0;   // Sets clock to the PBCLK
    AD1CON3bits.SAMC = 12;  // Sets auto-sample time to 12 TAD
    // Setting the prescaler for the ADC conversion clock
    AD1CON3bits.ADCS = 0;   // Sets the prescaler to 2*TPB
    // Setting analog channel for reading
    AD1CHSbits.CH0SA = 0;   // Sets AN0 as the input select bits

    // Enabling the ADC module last
    AD1CON1bits.ON = 1;     // Turn on the ADC module

    while(1);

    return 0;
}
```

Figure 2: Code for Lab 10: Analog to Digital Conversion