**ANALOG TO DIGITAL CONVERTER**

**Aayahna M. Herbert**

**November 21, 2017**

**ECE 3720 Section 007**

**Microcontroller Interfacing Laboratory**

**Abstract:** A circuit was designed that receives an analog output from a FSR, converts it to a digital value, and show eight bits on LED lights. Within the microcontroller, an analog interrupt was set up so that whenever a certain amount of pressure is put on the FSR, the eight LED lights lit up; the more pressure put on the FSR led to higher binary values displayed on the LEDS.

**INTRODUCTION**

The purpose of this lab was to convert the output of an FSR circuit to a digital value and display eight bits on LEDs. Along with this the most significant eight bits should be shown, LED should be handled with the ADC interrupt, and sample and convert should be done automatically. As the FSR is pressed, the LEDs should start to light up, but none will be on when it is released.

**EXPERIMENTAL PROCEDURES**

The equipment used include one PIC32 MC, NI-ELVIS II board, and an FSR. The first thing to be done, to avoid forgetting it later, is to set pins B0 and C0-C7 as the input from the FSR to the microcontroller and the outputs from the microcontroller to LED0-7, respectively. In order to configure the ADC module, there are fifteen steps that must be done first:

1. Configure the analog port pins in ANSELx
2. Select the analog inputs to the ADC multiplexers in AD1CHS
3. Select the format of the ADC result using FORM
4. Select the sample clock source using SSRC
5. Select the voltage reference source using VCFG
6. Select the Scan Mode using CSCNA
7. Set the number of conversions per interrupt SMP if interrupts used
8. Set Buffer Fill mode using BUFM
9. Select the MUX to be connected to the ADC in ALTS AD1CON2
10. Select the ADC clock source using ADRC
11. Select the sample time using SAMC if auto-convert is to be used
12. Select the ADC clock pre-scaler using ADCS
13. Turn the ADC module on using AD1CON1
14. To configure ADC interrupt
    1. Clear the AD1IF bit (IFS)
    2. Select ADC interrupt priority AD1IP (IPC) and sub-priority AD1IS (IPC)
15. Start the conversion sequence by initiating sampling

Along with these ten steps, in order to access the analog interrupt, interrupts must also be globally enabled by adding INTEnableSystemMultiVectoredInt() and INTEnableInterrupts() to the main function. With the main function comes the ISR function for the specific register number in use, void \_\_ISR(23) ADInt(void). Inside of these functions goes the commands for what should happen to the LEDs when the force is applied to the FSR. For this function, we want to find the average time of all buffers in use (in this case, four buffers are used from ADC1BUF0 to ADC1BUF3) and shift through the bits; this value will be set equal to PORTC since that is the port holding the LED outputs. In terms of wiring, when setting the wiring for the FSR, ground should be connected to a resistor and the analog input on the microcontroller while its power should be connected to +3.3V (this can be obtained by connected it to the 3.3V on the microcontroller). In order to make the LED response easier, a resistor of 10kΩ is used since the force decreases as resistance increases.

**RESULTS**

After the circuit was hooked up and ready to be tested, the program detected no compiling errors when ran so the board could then be tested. Each time the FSR was pushed, LEDs lit up in a way the reflected a bigger value when more pressure was added.

**DISCUSSION**

Problems arose initially when the LED lights didn’t light up when the FSR was pressed. The code and wiring were second-checked by the TA and the wires all came from a new pack recently purchased to avoid the possibility of dead wires being used and affecting the lab results. The source of the error was due to selecting the incorrect analog input.

**CONCLUSIONS**

In general, the take-away of this lab was to learn how to use ADC and an FSR.

**FIGURES AND TABLES**

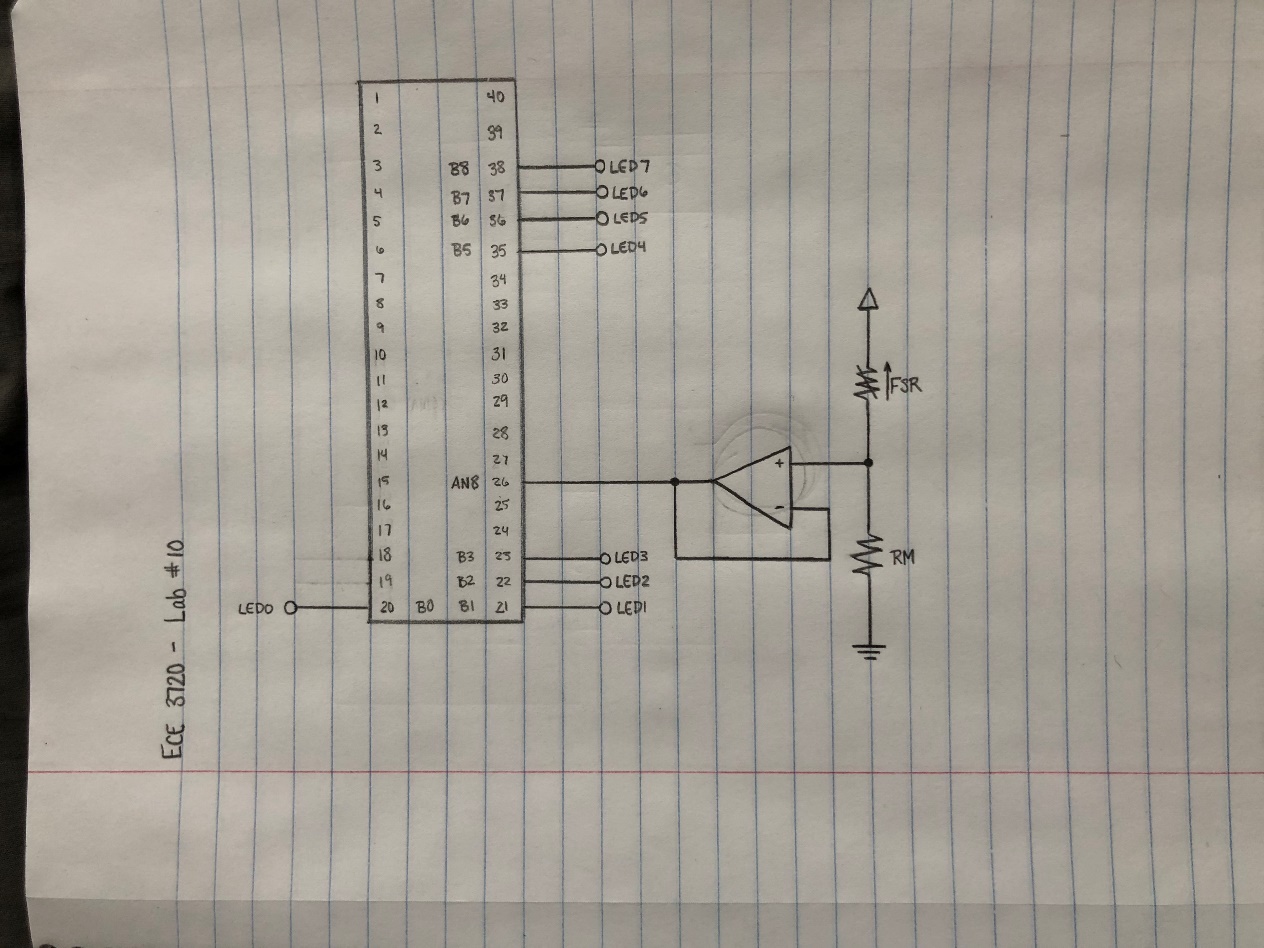


Figure 1: Circuit Schematic

**CODE**

#include<plib.h>

//Interrupt Function

void \_\_ISR(23) ADInt(void)

{

PORTC = ((ADC1BUF0 + ADC1BUF1 + ADC1BUF2 + ADC1BUF3)/4) >> 2;

IFS0bits.AD1IF = 0;

}

main()

{

INTEnableSystemMultiVectoredInt();

INTEnableInterrupts();

/\* Set up inputs and outputs \*/

TRISCbits.TRISC0 = 0;

TRISCbits.TRISC1 = 0;

TRISCbits.TRISC2 = 0;

TRISCbits.TRISC3 = 0;

TRISCbits.TRISC4 = 0;

TRISCbits.TRISC5 = 0;

TRISCbits.TRISC6 = 0;

TRISCbits.TRISC7 = 0;

TRISBbits.TRISB0 = 1;

/\* 15 STEPS: ADC MODULE CONFIGURATION \*/

/\* 1. Configure the analog port pins in ANSELx \*/

ANSELBbits.ANSB0 = 1;

/\* 2. Select the analog inputs to the ADC multiplexers in AD1CHS \*/

AD1CHSbits.CH0SA = 2;

/\* 3. Select the format of the ADC result using FORM \*/

AD1CON1bits.FORM = 0;

/\* 4. Select the sample clock source using SSRC \*/

AD1CON1bits.SSRC = 7;

/\* 5. Select the voltage reference source using VCFG \*/

AD1CON2bits.VCFG = 0;

/\* 6. Select the Scan Mode using CSCNA \*/

AD1CON2bits.CSCNA = 0;

/\* 7. Set the number of conversions per interrupt SMP if interrupts used \*/

AD1CON2bits.SMPI = 3;

/\* 8. Set the Buffer Fill mode using BUFM \*/

AD1CON2bits.BUFM = 0;

/\* 9. Select the MUX to be connected to the ADC in ALTS AD1CON2 \*/

AD1CON2bits.ALTS = 0;

/\* 10. Select the ADC clock source using ADRC \*/

AD1CON3bits.ADRC = 0

/\* 11. Select the sample time using SAMC if auto-convert is to be used \*/

AD1CON3bits.SAMC = 4;

/\* 12. Select the ADC clock pre-scaler using ADCS \*/

AD1CON3bits.ADCS = 3;

/\* 13. Turn the ADC module on using AD1CON1 \*/

AD1CON1bits.ON = 1;

/\* 14. To configure ADC interrupt \*/

IECObits.AD1IE = 1;

//(a) Clear the ADIF bits (IFS1)

IFS0bits.AD1IF = 0;

//(b) Select ADC interrupt priority AD1IP (IPC) and sub-priority AD1IS (IPC)

IPC5bits.AD1IP = 1;

/\* 15. Start the conversion sequence by initiating sampling \*/

AD1CON1bits.ASAM = 1;

AD1CON1bits.SAMP = 1;

while(1) //Run Continuously

{

}

}