

Revised by Dr. Haroon Ahmed Khan, June 25th, 2020

Lab # 08: Interfacing Analog Sensors (LM35) using Analog to Digital Converter

Objectives:

- Understand the function of ADC in microcontroller.
- Learn different mode of operation of ADC
- Integration of LM35 temperature sensor with Atmega16.
- Implementing DAC and Bus interface in Proteus

Tools:

Software Tools:

- Microchip Studio
- Proteus ISIS
- AVRDUDE

Hardware Tools:

Name	Value	Quantity
Arduino Nano	-	1
Breadboard	-	1
LM35	-	1

Table 8.1: List of Components

Pre-Lab

ADC is a system that converts an analog signal into a digital signal. In almost all digital systems there is a frequent need to convert analog signals generated by analog devices such as microphone, sensors and potentiometers into digital values that can be stored and processed by digital system.

Internal ADC of ATMEGA328P microcontroller

The ADC system of Atmega328P has following features

- 10 bit ADC
- ± 2 LSB absolute accuracy

- 13 ADC clock cycle conversion rate
- 8 multiplexed single-ended input channels
- Selectable right or left result justification
- 0 to V_{cc} ADC input voltage range

Operation

The ADC in ATMEGA 328P converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AVCC or an internal 1.1V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

The analog input channel is selected by writing to the MUX bits in ADMUX. Any of the ADC input pins, as well as GND and a fixed voltage reference, can be selected as single ended inputs to the ADC.

The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX. ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion. The ADC has its own interrupt which can be triggered when a conversion completes.

A single conversion is started by writing a '0' to the Power Reduction ADC bit in the Power Reduction Register (PRR.PRADC), and writing a '1' to the ADC Start Conversion bit in the ADC Control and Status Register A (ADCSRA.ADSC). ADSC will stay high as long as the conversion is in progress, and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

Alternatively, a conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit (ADCSRA.ADATE). The trigger source is selected by setting the ADC Trigger Select bits in the ADC Control and Status Register B (ADCSRB.ADTS).

By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100 kHz. The pre scaling is set by the ADPS bits in ADCSRA.

ADC Conversion Result

After the conversion is complete (ADCSRA.ADIF is set), the conversion result can be found in the ADC Result Registers (ADCL, ADCH). For single ended conversion, the result is:

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

Where VIN is the voltage on the selected input pin, and VREF the selected voltage reference

Registers involved in ADC

ADC Multiplexer Selection Register – ADMUX

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

Bit 7:6 – REFS1:0: Reference Selection Bits

These bits select the voltage reference for the ADC, as shown in Table

REFS[1:0]	Voltage Reference Selection
00	AREF, Internal V _{ref} turned off
01	AV _{CC} with external capacitor at AREF pin
10	Reserved
11	Internal 1.1V Voltage Reference with external capacitor at AREF pin

Table8.2: ADC Voltage reference selection

Bit 5 – ADLAR: ADC Left Adjust Result

The ADLAR bit affects the presentation of the ADC conversion result in the ADC Data Register. Write one to ADLAR to left adjust the result. Otherwise, the result is right adjusted.

Bits 4:0 – MUX3:0: Analog Channel Selection Bits

The value of these bits selects which analog input is connected to the ADC. The table below show these bit settings for selecting various channels:

MUX[3:0]	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	Temperature sensor
1001	Reserved
1010	Reserved
1011	Reserved
1100	Reserved
1101	Reserved
1110	1.1V (V_{BG})
1111	0V (GND)

Table 8.3: Mux ADC selection table

ADC Control and Status Register A – ADCSRA

7	6	5	4	3	2	1	0	
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA

Bit 7 – ADEN: ADC Enable

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off.

Bit 6 – ADSC: ADC Start Conversion

In Single Conversion mode, write this bit to one to start each conversion. ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

Bit 5 – ADATE: ADC Auto Trigger Enable

When this bit is written to one, Auto Triggering of the ADC is enabled (This mode is not covered in this lab).

Bit 4 – ADIF: ADC Interrupt Flag

This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag.

Bit 3 – ADIE: ADC Interrupt Enable

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

Bits 2:0 – ADPS2:0: ADC Prescaler Select Bits

These bits determine the division factor between the XTAL frequency and the input clock to the ADC.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Table 8.4: division factor calculation for frequency

The ADC Data Register –ADCL and ADCH

These two registers contain 10 bits of the conversion result. The result is stored in two different ways depending on the setting of ADLAR bit as shown below:

ADLAR=0:

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

ADLAR=1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	

Digital Input Disable Register 0-DIDR0

When the respective bits are written to logic one, the digital input buffer on the corresponding ADC pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the ADC7...0 pin and the digital input from this pin is

not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

Bit	7	6	5	4	3	2	1	0
	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Modes of operation:

The ADC has two fundamental operation modes:

1. Single conversion mode
2. Free running mode

Single Conversion Mode:

In Single Conversion mode, you have to initiate each conversion. When it is done, the result is placed in the ADC Data register pair and no new conversion is started.

Free Running Mode:

It start the conversion only once, and then, the ADC automatically will start the following conversion as soon as the previous one is finished. Its conversion is continuous. Once initialized it takes 13 ADC cycles for single conversion. In this mode ADC data register has to be read before new value is written.

LM35:

The LM35 is an integrated circuit sensor that can be used to measure temperature with an electrical output proportional to the temperature (in °C).

PIN Configuration:

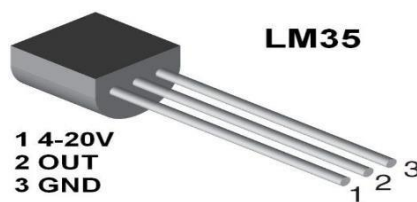


Figure 8.1 LM35 Pinout

Pre lab Task:

Explain the working of five analog sensors. Explain what physical quantity do they measure and provide the mapping function for the output voltage.



In Lab:

Task-1

Use LM35 to sense the room temperature. Convert this data into digital using atmega328P ADC and display temperature value on virtual terminal.
Complete the Code and simulate on Proteus.

Code:

```
//*****  
  
// Module description:  
  
/*  
In this lab the students will learn to use the Analog to Digital  
Converter present as a peripheral device in Atmega328p MCU. The ADC has  
a 10- bit resolution and offers 8 channels for single ended conversions.  
The students will explore two modes of operation of the ADC for single  
ended conversions:  
  
    1. Single Conversion Mode  
    2. Auto Trigger Mode (Free running sub-mode)  
  
Task1:  
Interface a temperature sensor (LM35) using the ADC of Atmega328P MCU  
  
Task2:  
Record an analog signal using the ADC and output to a DAC after  
processing  
*/
```

```
//*****

#include <inttypes.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#define F_CPU 8000000UL
#include <util/delay.h>
#include <string.h>
#include <math.h>

/*
This part allows you to choose which section of the code to compile
*/

//Comment out this line to activate task 2 compilation

#define TASK1      // Activate Task1 code compilation /* */

// If Task 1 not activated then activate Task 2 code compilation
#ifndef TASK1 /* */
    #define TASK2

#endif

#ifdef TASK1

    #define SERIAL_DEBUG // Using UART for Task 1

    #define ADC_MODE 1 // ADC will be initialized in Single
                      //Conversion Mode

#endif

#ifdef TASK2

    #define USE_DAC      // Using DAC for Task 2


    #define ADC_MODE 2 // ADC will be initialized in Free Running
                      //Mode

#endif

/***** Definitions for UART *****/
```



```


#ifdef SERIAL_DEBUG /**/

    #include "debug_prints.h"
    #include "debug_prints.c"
    #define BAUD0 9600          // Baud Rate for UART
    #define MYUBRR (F_CPU/8/BAUD0-1)

#endif


    // U2X = 1

/*****

#define ADC_CHANNEL0    0
#define ADC_CHANNEL1    1
#define ADC_CHANNEL2    2
#define ADC_VREF        5      // Ref voltage for ADC is 5 Volts
#define ADC_RES          10    // Resolution of ADC in bits
#define ADC_QLEVELS     1024   // Quantization levels for the ADC
/**/

unsigned char ADC_Initialize(unsigned char mode);
// Initialize ADC. Mode 1 = Single Conversion, Mode 2 = Free Running

unsigned int ADC_Read(unsigned char channel);
// Reads the result of a single conversion from the ADC

/**/
float ADC_Convert(unsigned int);
unsigned char VinToTemp(float Vin);
unsigned char read_temp_sensor(unsigned char ADC_channel);

#ifdef TASK2
    #define DAC_DDR    DDRD
    #define DAC_PORT    PORTD
    ISR(ADC_vect)
    {
        DAC_PORT = ADCH;    // output the digital value on DAC Port
    }
#endif
#define TEMP_SENSOR_CHANNEL ADC_CHANNEL0

```




```

/*****/
int main(void)
{
    ADC_Initialize(ADC_MODE);
    DIDR0=0xFF;
    /* */
#ifdef TASK1
    UART0_init(MYUBRR); //(F_CPU/8/BAUD0-1)
    printSerialStrln("Lab 8: ");
    unsigned char temperature;
    DDRD = 0xFF;
    while(1)
    {
        /* */
        printSerialStr("Temperature is: ");
        temperature = read_temp_sensor(TEMP_SENSOR_CHANNEL);
        printSerialInt(temperature);
        printSerialStr("\r \n");
        PORTD = temperature;
    }
#endif

    /* */
#ifdef TASK2
    DAC_DDR = 0xFF;           // Configure port for output
    ADMUX &= ~(0x07);         // clear previous selection of channel
    ADMUX |= ADC_CHANNEL1;    // Select the new channel
    //write command for this  // Enable interrupts globally
    //write command for this  // Trigger the free running ADC
    while(1);
#endif
}

```

```
/*
Function Initializes the ADC for 10-Bit Single Conversion
(mode == 1) Or Free Running (mode == 2) modes.

The function returns 0 if initialization successful.
If mode argument is other than 1 or 2,
the function returns an error code 0x02
*/
/**/
unsigned char ADC_Initialize(unsigned char mode)
{
    if(mode == 1) // Single Conversion Mode
    {
        /**/
        /** Write Code for this ***/
        /*
        Left adjust result. Vref = AVCC = 5V
        Select the pre-scaler for 16 MHz System Clock
        Pre-scaler = 128
        Clock for ADC = 125 KHz (should be between 50K to 200K)
        */
        return(0);
    }
    if(mode == 2) // Free Running Mode
    {
        /**/
        /** Write Code for this ***/
        /*
        Left adjust result. Vref = AVCC = 5V
        Select the pre-scaler for 16 MHz System Clock
        Pre-scaler = 128
        Clock for ADC = 125 KHz (should be between 50K to 200K)
        */
        return(0);
    }
    else
        return (0x02); // return error code: Wrong Mode Selection
}
}
```

```

/*
Function reads the result of a single conversion
from the ADC channel given as an argument
*/

unsigned int ADC_Read(unsigned char channel)
{
    /* */
    unsigned char ADC_lo;
    unsigned char ADC_hi;
    unsigned int result;
    ADMUX &= ~(0x07);    // clear previous selection of channel
    ADMUX |= channel;    // Select the new channel
    // Delay needed for the stabilization of the ADC input voltage
    _delay_us(10);
    //wait for ADC to finish any ongoing operation
    while((ADCSRA & (1<<ADSC)) != 0);
    ADCSRA |= (1 << ADSC); //start conversion
    /* */
    while((ADCSRA & (1<<ADIF)) == 0);
    ADCSRA |= (1<<ADIF); // clear the flag by writing 1 to it
    /* */
    //result = (ADCH<<8) | (ADCL & 0xC0);    // Left adjust result
    ADC_lo = ADCL;
    ADC_hi = ADCH;
    /*Compute the result below*/
    result = ?? // Right adjust result
    return result;
}

/*
This function takes an uint16 as input from the ADC.
This uint16 is an unsigned integer result of the ADC
encoded result. The function then converts this result
to floating point Voltage using the ADC_RES (resolution)
and ADC_REF (reference voltage) defined earlier
*/

float ADC_Convert(unsigned int ADC_value)
{
    float Vin;
    /* */
    Vin = ADC_VREF * ((float)ADC_value/ADC_QLEVELS);
    return Vin;
}

```

```
/*
This function takes the floating-point Voltage value
as input and converts it to corresponding Temperature
in Celsius for an LM35 in Basic Configuration. The
function returns an 8-Bit integer value of the
Temperature
*/

unsigned char VinToTemp(float Vin)
{
    unsigned char temperature=0;

    // 10 mv per centigrade
    float VoltsPerCentigrade = 0.01;
    /* **** */
    temperature = (unsigned char) floor(Vin/VoltsPerCentigrade);

    // Temperature value returned
    return temperature;
}

/*
This function reads the Value of Temperature Sensor
(LM35) from pre-defined ADC Channel and returns the
result in Degree Celsius as an 8-Bit unsigned int
*/
unsigned char read_temp_sensor(unsigned char ADC_channel)
{
    // Read the sensor Connected at ADC_channel
    unsigned int ADC_value = ADC_Read(ADC_channel);

    // Get the value in floating point
    float Vin = ADC_Convert(ADC_value);
    unsigned char temp_celsius = VinToTemp (Vin);

    // Convert to temprature and return
    return temp_celsius;
}
```

Proteus Simulation

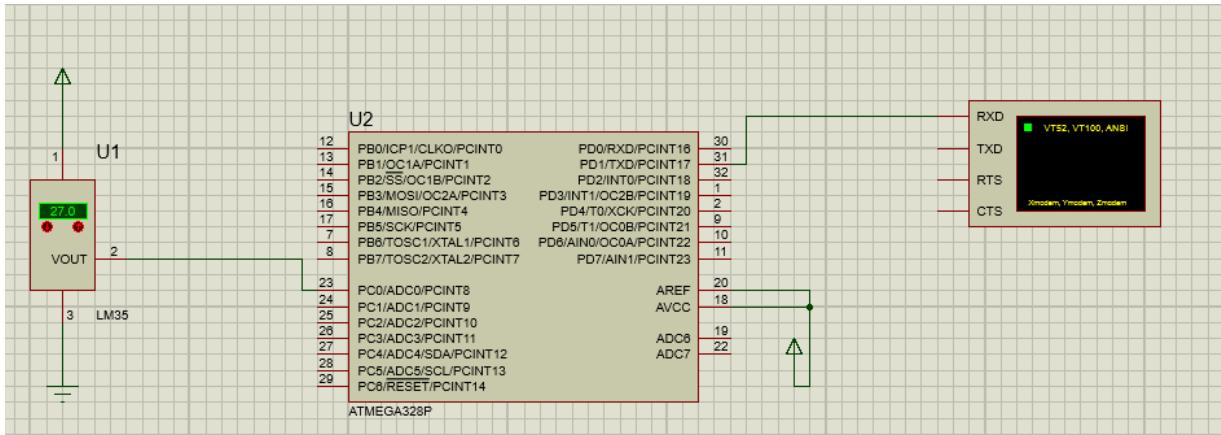


Figure 8.2: Task-1 Proteus diagram

Hardware:

Interface a temperature sensor with Arduino Nano and use serial monitor of Arduino IDE to display temperature.

Post Lab Task:

Complete the In-Lab code for task 2.

Record a signal using ADC. Use ADC in free running mode. Add 8-bit DAC at Port D and convert back the signal in analog and display result on oscilloscope for comparison.

Task description:

Implementation of Bus in Proteus.

- To implement 8bit DAC in Proteus use 10 DAC.
- Take “Bus” (Label) from “Terminals Mode” at side bar.
- Placed “Bus” at root sheet.
- Connect DAC Bus (Blue line) to “Bus”.
- Double click at “Bus” and change its name to “DAC[0..9]”.
- Connect PD0 to DAC Bus. Right click on wire and select “Place Wire Label”.
- From drop down string menu select “DAC2”.
- Repeat last 2 steps with rest of PD pins.
- Connect DAC0 and DAC1 to ground.

The diagram shows an ATmega328P microcontroller (U2) interfaced with a DAC0808 DAC and a 4-bit digital-to-analog converter. The microcontroller's PB0-PB7 pins are connected to the DAC's data bus. The DAC's VCC and GND pins are connected to the power supply. The DAC's output is connected to the DAC0808's input. The DAC0808's output is connected to the DAC's output. The DAC's output is also connected to the DAC's output. The DAC's output is also connected to the DAC's output.

Lab # 08 Interfacing Analog Sensors Using Analog to Digital Converter

Lab Assessment		
Pre Lab	/1	/10
In Lab	/5	
Post Lab	/5	
Instructor Signature and Comments		

