EE443 - Embedded Systems

Experiment 5

Analog Input Output

Purpose: Sample an analog signal using the analog-to-digital converter (ADC) available on the MCU. Display the ADC results on a LCD. Use an external digital-to-analog converter (DAC) to obtain analog output.

Programming Hints

ATmega328 Analog-to-Digital Converter (ADC)

An ADC generates a digital output corresponding to the signal level at its analog input. The 10-bit ADC in ATmega328 generates numbers between **0** and **1023** for the input voltage range between **0 V** and the selected reference voltage. The most significant 8 bits of the ADC output can be read from a single register when 8-bit resolution is sufficient. It is possible to connect one of several analog input pins (ADC0 ... ADC5) to the ADC input through a multiplexer. Following SFRs are related to the ADC operations:

Register	Description	bit-7	bit-6	bit-5	bit-4	bit-3	bit-2	bit-1	bit-0
PRR	Power Reduction Register	PRTWI	PRTIM2	PRTIM0	---	PRTIM1	PRSPI	PRUSART0	PRADC
ADMUX	ADC Multiplexer Selection Register	REFS[1:0]		ADLAR	---	MUX[3:0]			
ADCSRA	ADC Control and Status Register A	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS[2:0]		
ADCSRB	ADC Control and Status Register B	---	ACME	---	---	---	ADTS[2:0]		
DDRC	Port C Data Direction Register	---	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
DIDR0	Digital Input Disable Register 0	---	---	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
ADCH	ADC Data Register High and Low when ADLAR=0	---	---	---	---	---	---	ADC[9:8]	
ADCL		ADC[7:0]							
ADCH	ADC Data Register High and Low when ADLAR=1	ADC[9:2]							
ADCL		ADC[1:0]		---	---	---	---	---	---

PRR – Power Reduction Register

Setting a bit in this register turns off power to the corresponding peripheral module. Therefore, **PRADC** bit must be cleared to turn on ADC. Initially all PRR bits are **0** after power up.

ADMUX– ADC Multiplexer Register

REFS[1:0] : Select ADC reference voltage; **00**=> external (AREF pin), **01**=> analog supply ($\mathbf{AV_{cc}}$), **10**=> reserved, **11**=> internal 1.1 V reference.

ADLAR : Sets ADC output data bit positions (see **ADCH** and **ADCL** in the table).

MUX[3:0] : Selects ADC input channel; **0000...1000**=> **ADC0...ADC8**.

ADCSRA– ADC Control and Status Register A

ADEN : Set to **1** to enable ADC.

ADSC : Set to **1** to start conversion. **ADSC** returns back to **0** when the conversion is complete.

ADATE : Enables ADC auto trigger mode (set to **0**).

ADIF : ADC interrupt flag.

ADIE : Enables ADC interrupt after conversion (set to **0**).

ADPS[2:0] : ADC prescaler bits that determine the clock frequency required for the ADC operation. The maximum ADC clock frequency is **200 KHz**. The prescaler bits select the divider to determine the frequency ratio,

`system clock` / `ADC clock` = power of 2 given by value of **ADPS[2:0]**.

For example, if **1 MHz** system clock is used then set **ADPS[2:0]** bits to "**011"** to obtain **125 kHz** ADC clock with a frequency ratio of **8 = 2^3**.

ADCSRB– ADC Control and Status Register B

ACME : Enables analog comparator multiplexer (set to **0**).

ADTS[2:0] : Select ADC auto trigger source. Ignored when **ADATE = 0**.

DDRC– Port C Data Direction Register

Port-C pins that are used as analog inputs should be used as input pins by setting the corresponding bit to **0** in DDRC register.

DIDR0– Digital Input Disable Register

Writing 1 to a register bit disables the corresponding digital input buffer to reduce power.

ADCH and ADCL– ADC Data Registers

10-bit ADC conversion result can be read from these registers. Position of bits depends on **ADLAR** setting in **ADMUX** register (see **ADCH** and **ADCL** in the table).

Preliminary Work

1. Read the Section 24 on the Analog-to-Digital Converter in the ATmage328 datasheet. Determine the SFR settings that initializes ADC configuration according to the following requirements.

- Two analog inputs at ADC0 and ADC1. ADC0 will be used in the first part of the experiment.
- External reference input to set the analog input range between **0 V** and **+5 V**.
- Better than **20 mV** ADC resolution.
- Minimum conversion time that can be obtained with **8 MHz** CPU clock.
- ADC conversions are started by software commands (no auto trigger).

2. Write a main program to perform the following tasks.

- Initialize LCD interface through port-B.
- Initialize ADC for conversion of input from **ADC0** (**PC0**) using the SFR settings determined above.
- In the main loop:
  1. Start an ADC conversion by setting the **ADSC** start bit.
  2. Wait until the end of conversion (while **ADSC** bit is set).
  3. Read the 8 MSBs of the ADC result from ADCH register.
  4. Display the 8-bit ADC result as a 3-digit decimal number on the LCD.
  5. Set a delay time to obtain **~1 kS/s** conversion rate as the main loop is repeated. Calculate the CPU time spent during ADC conversion and LCD update to find the remaining delay time to obtain **~1 ms** loop execution time.

Remember that **`LCD_Init(.)`** and **`LCD_Clear(.)`** should be called **only once** in initialization section of the main program. Calling these functions in the main loop will cause long time delays (~4 ms).

Use the **`sprintf(...)`** C library function (include **`<stdio.h>`**) to obtain the character string that will be sent to the LCD module:

```
// Declare the character array for one LCD line:
char  LCDtext[16];
---
---
// Write 3-digit decimal number into LCDtext:
sprintf(LCDtext, "%3d", ADCout);
// Place LCD cursor at column-1 of line-1:
LCD_MoveCursor(1, 1);
// Send LCDtext to the LCD module:
LCD_WriteString(LCDtext);
```

Note: WinAVR compiler may give an error message related to long integer multiply operation when **`sprintf(...)`** function is used. In that case, replace **`sprintf(...)`** with the **`PrintByte(...)`** function provided in **PrintByte.c**.

Procedure

1. Create a new project directory and a new AVR project for Experiment 5. Copy **LCDmodule.c** and **LCDmodule.h** used in the previous experiment into the new AVR project directory, and add these files to the AVR project in Code::Blocks. Compile the main program you wrote in the preliminary work and debug if necessary.

2. Copy the provided Proteus design file, **Exp5_ADC.dsn**, into the project directory. Open the **Edit Properties** window for **ATMEGA328P** microcontroller on the ISIS schematic and select the **.hex** file generated by the compiler for Experiment-5. Apply a saw tooth waveform to the **ADC0** input using a pulse generator with the following settings:

Analog Type: Pulse,
Low Voltage: **0 V,** High Voltage: **5 V,**
Start Time: **0**, Rise Time: **9999 ms**, Fall Time: **1 ms**,
Pulse width: **0**, Period: **10 seconds**

Test your program looking at the animated simulation results, and debug it, if necessary.

3. Modify the main program to sample the **ADC1** input (check the SFRs, **`ADMUX`**, **`DDRC`**, and **`DIDR0`**) and to send the ADC samples directly to the DAC instead of displaying them on LCD module. LCD module may remain connected to Port-B, but it will not display any meanigful text with the data sent to the DAC.

DAC analog reference inputs, **VREF-** and **VREF+** should be connected to **0 V** and **5 V**, respectively. DAC accepts the data at digital inputs immediately when its **LE** (Latch enable) input is pulled high.

Apply a **10 Hz** sinusoidal waveform varying between **0 V** and **5 V** to the **ADC1** input.

Place an oscilloscope (select the **Virtual Instruments** list on the main toolbar) and probe **ADC1** input of ATmega328 and the **DAC** output.

Test your program looking at the simulation results, and debug it, if necessary.

Note: If you close the oscilloscope window, you can activate it again by using the **Reset Popup Windows** option under the **Debug** menu of ISIS.

4. In the main program, add an attenuation control variable (i.e. **Atten**) that changes the relative DAC output amplitude as follows:

Atten = 1=>1/1, **2**=>1/2, **3**=>1/4, **4**=>1/8, **5**=>1/16, **6**=>1/32

Atten setting will be controlled by the two push-button switches as in the previous experiment:

- Decrement **Atten** (if Atten > 1) every time **SW0** is pressed.
- Increment **Atten** (if Atten < 6) every time **SW1** is pressed.

Calculate the attenuated DAC settings **by using a shift operation only** (without any multiplication, division, or **`switch`** statements). Test your program monitoring the input/output waveforms on the oscilloscope window, and debug if necessary.

Question 1: Explain the changes in the DAC output waveform that occur at high attenuation factors. How can you implement an attenuator that does not cause distortion at high attenuation factors by using two DACs?

Question 2: As a home exercise, modify the program to obtain the attenuation steps given below **by using shift and addition operations only**.

Atten = **1**=>1/1, **2**=>3/4, **3**=>1/2, **4**=>3/8, **5**=>1/4, **6**=>3/16,
7=>1/8, **8**=>3/32, **9**=>1/16, **10**=>3/64, **11**=>1/32

Hint: Calculation of DAC output can be done in two lines of C code. First, calculate the DAC output for 1/1, 1/2, 1/4, ... 1/32 attenuation. Implement 3/4, 3/8, 3/16, 3/32, and 3/64 attenuation steps when **Atten** is an even number.

Question 3: Suggest a method to send data to two external devices by using two more MCU pins. Port-B data should be directed to the target device without causing any unwanted data transfers (i.e. DAC will not receive LCD data, LCD will not receive DAC data).