Laboratory 8 *Analog-to-Digital Converter*

Instructor: Dr. Lihong Zhang

1 Objectives

With the completion of this lab, you should be able to:

- Design and construct a circuit to interface the ADC module of AVR microcontroller
- Configure the ADC module of ATmega32 microcontroller
- Operate on the EEPROM memory of ATmega32 microcontroller
- Write assembly language programs to capture samples from an analog signal and store them to EEPROM memory space

2 Introduction

Analog-to-Digital Converters (ADC) and Digital-to-Analog Converters (DAC) are extremely useful devices which enable us to interface the real world events (which are frequently analog) with microprocessors for monitoring or control. They are inexpensive but provide good precision and accuracy when compared to their pure analog counterparts.

There exist many types of ADCs and DACs differing in their internal construction, resolution, precision and accuracy, speed, compatibility with different microprocessors, and other capabilities like power supply requirements, unipolar/bipolar operation and coding. In this lab, you will work on the ADC module of ATmega32 on the STK600 board and test its function.

Caution!

The ADC module within AVR microcontroller that you will be using in this lab is sensitive to electrostatic fields and can be easily damaged. **Do not insert AVR microcontroller devices into powered sockets.** Remove power before insertion or removal. Do not touch any pins of these devices with your fingers. It is always a good practice to switch off the power supply before you make any change to the hardware.

The ADC devices are normally expensive – please be careful not to ruin any chips.

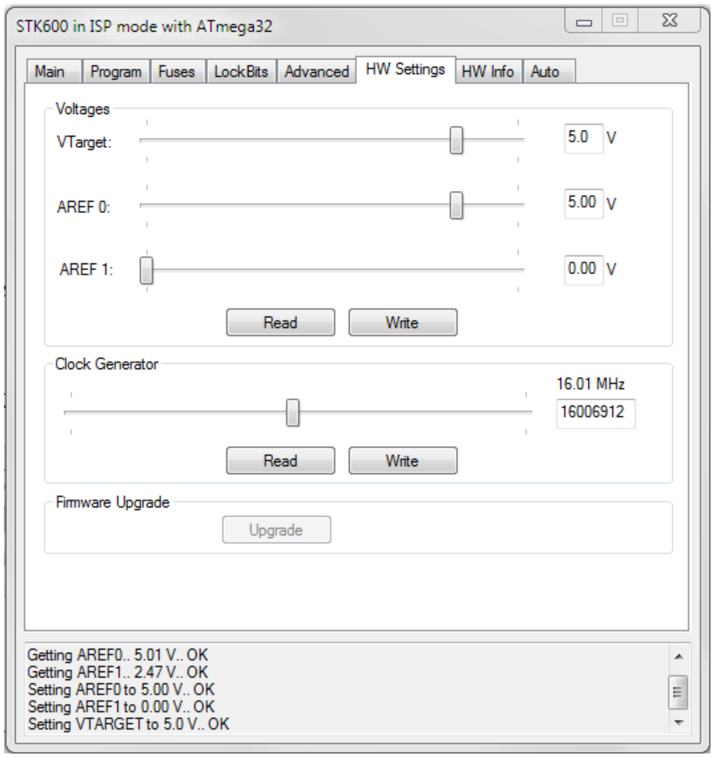
3 Analog-to-Digital Converter

Most of AVR microcontrollers have ADC modules integrated inside the chip. ATmega32 has one 10-bit ADC module and 8 analog input channels (sharing the pins with PORTA). To enable and configure the ADC module, you need to load proper control bytes to the related ADC registers.

3.1 ADC Configuration

To program the ADC module of the AVR, the following steps should be taken:

1. Make the pin for the selected ADC channel an input pin.
2. Turn on the ADC module of the AVR because it is disabled upon power-on reset to save power.
3. Select the conversion speed via ADPS2:0. It is recommended you put ADC Clock to CK/128 (i.e., ADPS2:0=111).
4. Select voltage reference V_{ref} (via REFS1 and REFS0 in the ADMUX register) and ADC input channels (via MUX4:0 in the ADMUX register). Note that you should select REFS1=0 and REFS0=0 (i.e., using AREF-Pin as V_{ref}). Other selections of REFS1:REFS0 may cause short circuit of the STK600. In this way, the voltage of V_{ref} can be tuned by AREF0 in the AVR Studio (as shown in the picture below). It is recommended that V_{ref} is set as the same as VTarget (e.g., 5V).



VTarget: 5.0 V
AREF 0: 5.00 V
AREF 1: 0.00 V
16.01 MHz
16006912

```
Getting AREF0.. 5.01 V.. OK
Getting AREF1.. 2.47 V.. OK
Setting AREF0 to 5.00 V.. OK
Setting AREF1 to 0.00 V.. OK
Setting VTARGET to 5.0 V.. OK
```

5. Activate the start conversion bit by writing a one to the ADSC bit of ADCSRA.
6. Wait for the conversion to be completed by polling the ADIF bit in the ADCSRA register.
7. After the ADIF bit has gone HIGH, read the ADCL and ADCH registers to get the digital data output. Notice that you have to read ADCL before ADCH.
8. If you want to read the selected channel again, go back to Step 5.

To simplify your operations, in this lab we only require 8-bit analog-to-digital conversion. That is to say, the two least significant bits of the conversion result can be ignored. Therefore, you can define the ADC data left-justified (i.e., ADLAR=1 in the ADMUX register) so that only ADCH is meaningful.

3.2 Circuit Wiring

Before circuit wiring, prepare an AVR assembly language program, which can keep converting one single-ended analog input channel (e.g., ADC0) to 8-bit digital data (neglecting the two least significant bits) and displaying such an 8-bit result to the built-in LEDs on the STK600. Note that the PORTD and LEDS heads should be connected together on the STK600.

Switch OFF all power supplies. Use a 2-wire cable (with a 2-pin connector on one end) or 10-wire cable (with a DIP connector on one end) to bring the PORTA signals of the STK600 to a breadboard located close-by. A single-ended analog input signal to the ADC module of ATmega32 should go to the corresponding pin on the breadboard instead of directly to the STK600 board. The single-ended analog input signal is obtained from a Signal Generator instrument, which is available on your work bench.

Note that the single-ended analog input signal MUST BE WITHIN 0V and V_{ref}. Otherwise, the target AVR or even the STK600 may be destroyed. It is strongly recommended you set V_{ref} as the same as VTarget (i.e., 5V) in this AVR Studio so that the single-ended analog input signal has a larger swing. In this situation, the single-ended analog input signal generated from the Signal Generator instrument should be always between 0V and 5V in this lab.

Do NOT connect the signal generator output to your breadboard yet. Make sure that all the ground points on the breadboard and STK600 board are connected together. Your partner should verify the connections.

Power on and set the signal generator to produce a +2.5V DC. Check the output of the signal generator with a digital multi-meter (DMM) or an oscilloscope. Then power off the signal generator and connect the signal generator output to the input pin on the breadboard. Remember to use a 1K ohm resistor in series with this input.

Turn ON the power supply of STK600. And then power on the signal generator (**You must follow the sequence above!**).

Question1: Compile your prepared program and download the hex file to the STK600. Check whether the ADC module operates properly. Reduce the input voltage (in steps) from 2.5V to 0V and find the converted binary value of the input signal in each case by reading the built-in LEDs on the STK600. Report your observation and comment on the results.

3.3 Signal Generation

Remove the connection between the output of the signal generator and the ADC module. Set the signal generator to produce a 500Hz sinusoidal waveform with a peak-to-peak voltage of 2.5V. Adjust the DC offset of the signal generator to ensure that its output never falls below 0V. This is required because you have configured the ADC for unipolar operation. Check your signal on an oscilloscope and then switch off the signal generator and STK600. Apply this signal to the corresponding pin of the single-ended ADC channel on the breadboard. Turn on the power supply of STK600. And then power on the signal generator (**You must follow the sequence above!**).

Try out to see the ADC output on the built-in LEDs of the STK600. You should obtain different numbers.

3.4 Software

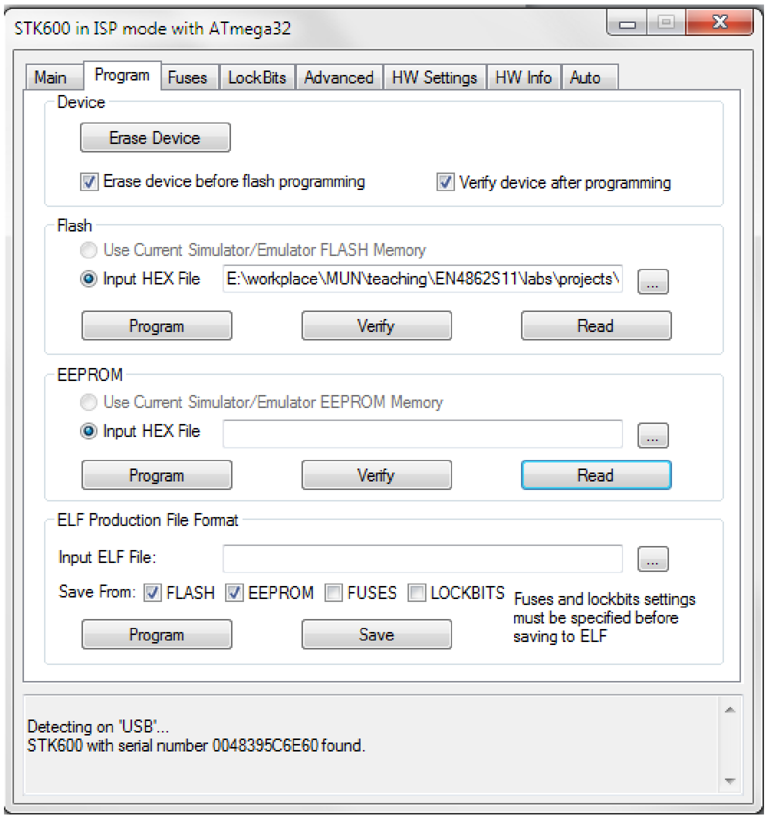
1. On the STK600 board, with two 10-wire cables make the connections between PORTD and LEDS headers, and between PORTB and SWITCHES headers. Modify the assembly language program you already had so that it can repeatedly read the single-ended input and output it to PORTD (i.e., LEDs of the STK600) until a push-button switch is pressed.

Question2: Does the display on the built-in LEDs of the STK600 make sense to you? Reduce the frequency of the signal generator and repeat the observation. To what frequency are you able to recognize the correct operation of the conversion?

2. Write an AVR assembly language program that reads the output of the ADC and stores this result in EEPROM memory. Make this operation repeat 256 times.

Check the lecture note or the textbook on how to operate on the EEPROM memory of ATmega32. To review the data on the EEPROM memory, follow the procedure below.

First you need to click Tools -> Program AVR -> Auto Connect to open the window of “STK600 in ISP mode with ATmega32” as below.



Device: [✓] Erase device before flash programming [✓] Verify device after programming

Flash: ○ Use Current Simulator/Emulator FLASH Memory ● Input HEX File E:\workplace\MUN\teaching\EN4862S11\labs\projects\

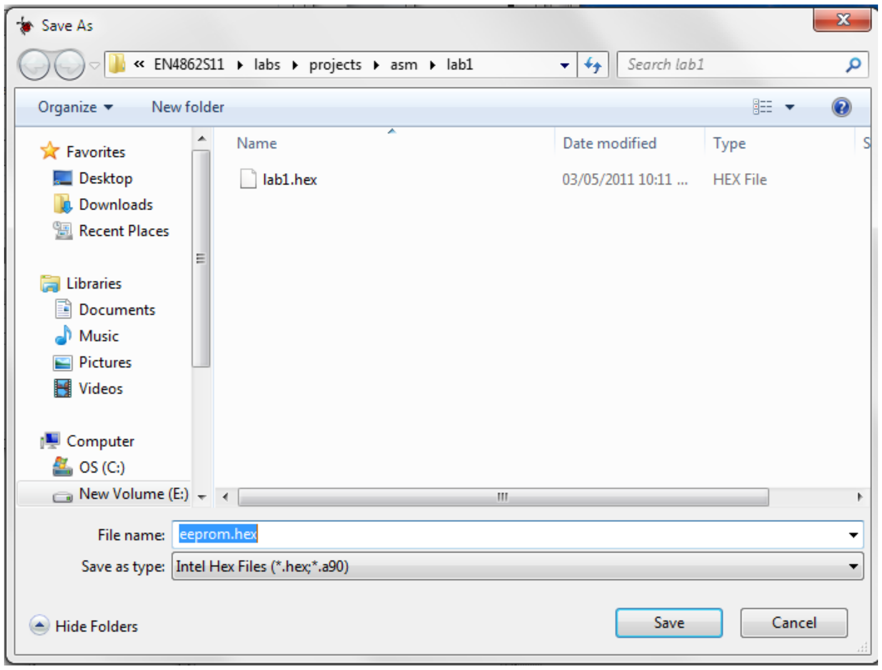
EEPROM: ○ Use Current Simulator/Emulator EEPROM Memory ● Input HEX File

Input ELF File:

Save From: [✓] FLASH [✓] EEPROM [ ] FUSES [ ] LOCKBITS Fuses and lockbits settings must be specified before saving to ELF

```
Detecting on 'USB'...
STK600 with serial number 0048395C6E60 found.
```

Under the panel of "EEPROM", click the button of "Read" and specify a file name in a pop-up "Save As" window (shown below).



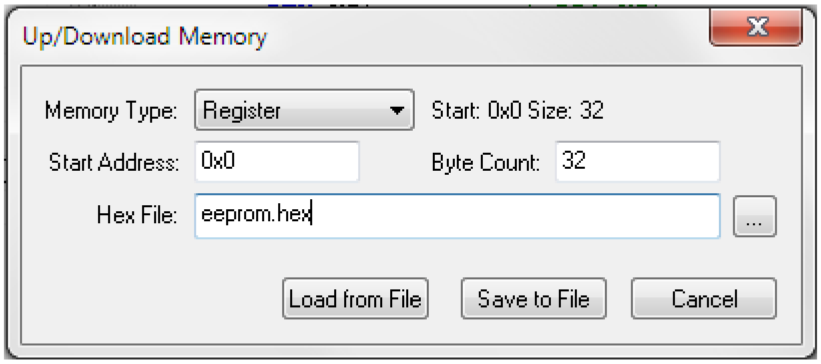
Name	Date modified	Type
lab1.hex	03/05/2011 10:11 ...	HEX File

File name: eeprom.hex

Save as type: Intel Hex Files (*.hex;*.a90)

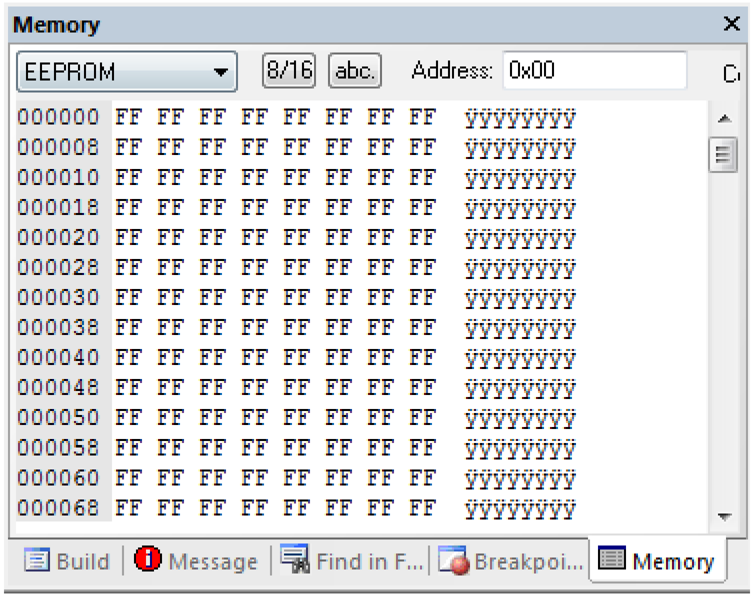
In this way, the current contents in the EEPROM memory will be saved to a file (say, eeprom.hex). To view the contents of this file, you can first click Build ->

Build and Run. Then click Debug -> Up/Download Memory, “Up/Download Memory” window should pops up as below. Provide your hex file name (e.g., eeprom.hex) under the title of “Hex File”. Click the button of “Load from File”.


Up/Download Memory

Memory Type: Register
Start: 0x0 Size: 32
Start Address: 0x0
Byte Count: 32
Hex File: eeprom.hex

Then you can review the EEPROM contents in the memory view windows as below.


Memory

EEPROM
Address: 0x00

```
000000 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000008 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000010 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000018 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000020 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000028 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000030 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000038 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000040 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000048 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000050 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000058 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000060 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
000068 FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿ
```

Question3: Reduce the frequency of the sinewave signal to around 5Hz. Run your program and observe the stored values. Report on your observations. ***Demonstrate correct operation of this program to a TA, who will sign your .ASM file.*** Following the default configuration (i.e., internal RC clock of the target microcontroller) of the STK600 board, the clock frequency of the ATmega32 in our labs is 1MHz. Based on the instruction execution time, determine what input sinusoid frequency is necessary so that exactly one period is captured and stored by your program. Set the signal generator appropriately, and report on your success when you run your program.

4 Submission

At the end of the lab, submit the print-outs of your ***.ASM files***. Include circuit diagrams, answers to all the questions in the lab and prelab documents, and test results.

Before leaving the lab, deconstruct your circuit and return your components to the provided kits.