
AVR1505: XMEGA training - ADC

Prerequisites

- Required knowledge
 - Atmel® AVR® XMEGA™ - Basics Hands-On
 - Atmel XMEGA - EventSystem Hands-On (recommended)
- Software prerequisites
 - Atmel AVR Studio® 4.18 SP2 or later
 - WinAVR/GCC 20100110 or later
- Hardware prerequisites
 - Flat-headed isolated screwdriver
 - Xplain evaluation board
 - JTAGICE mkII
- Estimated completion time:
 - 2 hours



8-bit **AVR**®
Microcontrollers

Application Note

Rev. 8320A-AVR-08/10



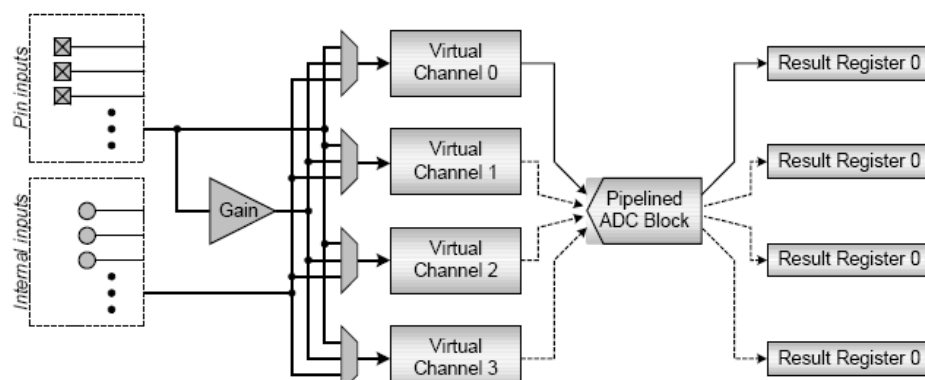
1 Introduction

The ADC in the Atmel XMEGA A series has 12-bit resolution. It is capable of converting up to 2 million samples per second (MSPS), and has a conversion time of 7 ADC clock cycles (add 1 cycle if gain is enabled) for 12-bit results. The input selection is flexible, and both single-ended and differential measurements can be done. The ADC can provide both signed and unsigned results, and an optional gain stage is available to increase the dynamic range.

This training is based on the Atmel AVR Xplain evaluation kit which features the ATxmega128A1 microcontroller. Different from most Atmel AVR ADCs, the ADC in the XMEGA A has a pipelined architecture. This means that a new analog voltage can be sampled each ADC clock cycle. Thus, a new ADC measurement can be started while other ADC measurements are ongoing. The result is that the conversion rate is much higher than the conversion time.

ADC measurements can either be done continuously, or be started by application software or events from other peripherals in the device. Up to four different virtual channels are provided. These have individual result and input selection (MUX) registers to make it easier for the application to keep track of the data. It is also possible to use DMA to move ADC results directly to memory or peripherals.

Figure 1-1. ADC Overview



2 Overview

Here is a short overview of the tasks in this training:

Task 1: Single Ended Conversion

The most common use of the ADC feature is for single-ended conversions. Here you will learn how this can be done with the Atmel XMEGA.

Task 2: Compare Function

The ADC has a compare function; this task shows how it works and why it is useful.

Task 3: Virtual Channels

This task shows a basic example on how virtual channels can simplify software and ease the implementation when several ADC inputs are used in one application.

Task 4: Event Triggered Conversion

Events can be used to trigger one or several ADC conversions. It can also be used to trigger two ADC's at the same time for parts with dual ADC.

Good Luck!

3 Task 1: Single Ended Conversion

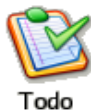
Single ended conversions are the most basic type of conversions. A single voltage, either from an external pin or from an internal reference, is used as input to the ADC.

In this task we will use the Virtual Channel 0 (CH0) (virtual channels will be covered more in detail in task3). We will configure Virtual Channel 0 to measure the potentiometer voltage on the Xplain board.

The goal for this task is that you understand how to:

- Set up and enable the ADC using the `adc_driver.c`
- Use the Xplain potentiometer signal as input source
- Output the signal to the LEDs

The idea is to convert the potentiometer voltage and output it to the LEDs port.



1. Locate the XMEGA-ADC folder and open the `SingleEnded.aps` project file in Atmel AVR Studio. The required `task1.c` and `adc_driver.c` with all the code are already included
2. Spend some time to understand the code, how it works, and what is required to get the ADC up and running in Single Conversion Mode
3. Build the project, ensure that there are no errors and start a debug session in the AVR Studio
4. Run the code and you will see that the LEDs are showing the result of the ADC conversion of the potentiometer voltage
5. Locate the potentiometer near the USB port and try turning it with a screwdriver. A flat headed with isolated handle (due to skin voltage potential) is recommended

When turning, you should see that the value is changing on the LEDs immediately.



What is the reference signal used in this task? (Hint: look in the C-code)

- a) What is the upper limit of the input signal when using the reference in a)?
- b) What is the difference between signed mode and unsigned mode? What mode were we using in this task?

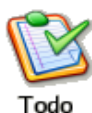
4 Task 2: Compare Function

In addition to normal single-ended conversions the ADC also has a compare function. When this function is used in combination with free-running mode (continuously sampling), the ADC will signal a result (that is, set the channel's interrupt flag) when the result is above or below a selectable threshold value.

This feature is very useful for applications where you only wish to check that the input signal is not higher or lower than a specific voltage. Without the compare function, it would be natural to check the result in software for every conversion. With the compare function in the Atmel XMEGA ADC, the ADC will signal the application via an interrupt or event when the threshold is reached. This reduces the CPU load, and keeps the resource use to a minimum.

The goal for this task is that you:

- Understand the compare function
- Are able to set up different compare interrupt modes and compare values



1. Locate the `CompareFunction.aps` project and open the `task2.c`, most of the code is already included. Notice that we will use interrupts for this task. A simple interrupt handler for the CH0 interrupt is already prepared for you

2. The code for the input selection we did in the previous task is already done, but some code must be added to

- a. Enable Medium Level Interrupt in the PMIC
- b. Set the correct interrupt mode

Locate the part of the code where this is supposed to be set up, and add the required code.

3. Build the project, ensure that there are no errors and start a debug session in Atmel AVR Studio

4. Run the code. If you try to keep the input signal low (close to 0 V), nothing should happen to the LEDs. However, when increasing the input voltage above the compare value, the LEDs will start to change values

5. In the AVR Studio editor, change the interrupt mode for interrupts when the result is lower than the threshold (hint: `ADC_CH_INTMODE_BELOW_gc`)

6. Recompile and run the code again. You should now see that you get interrupts when the result is lower than the threshold



5 Task 3: Virtual Channels

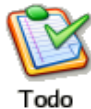
The Atmel XMEGA ADC has up to four (two for the Atmel XMEGA D series) sets of input MUX and result registers, and these sets are referred to as virtual channels (CH0 to CH3). Each virtual channel is fully independent of the others. They all have separate:

- Input selection through dedicated `CHn.MUXCTRL` registers
- Result registers, named `CHn.RES`
- Interrupt control and vectors
- Conversions start bits

In task1 and task2 we used Virtual Channel 0 with `CH0.MUXCTRL`, `CH0.RES` and so on. With four such channels we have the possibility to have four separate settings, and to start conversions on the different channels without changing any configuration.

The goal for this task is that you:

- Understand the concept of virtual channels and how they work
- Know how to set up and start conversion sweeps on virtual channels with free running mode

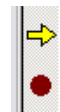


1. Locate and open the `VirtualChannels.aps` project in Atmel AVR Studio. Open the `task3.c` file and familiarize yourself with the code

The main difference from task1 and task2 is that both CH0 and CH1 are set up, and with different input selection settings. CH0 uses pin input on pin1 as before, while CH1 reads an internal channel; the built-in temperature sensor.

2. At the top of `main.c`, locate the place where the code to enable sweep mode must be added, and add the required code. Refer to the header file `adc_driver.h`
3. As you see, CH0 is set up in Single-Ended Mode. CH1 is set up to use an internal source as input (not an external pin). Add the missing code to use the internal temperature signal
4. Finally, add code to enable the Free Running Mode. Refer to the `adc_driver.h`
5. Build the project and ensure that there are no errors and start a debug session in AVR Studio

6. Place a breakpoint in the code as shown below:



```
do { } while( (ADCA.INTFLAGS & ADC_CH1IF_bm) != ADC_CH1IF_bm );  
ADCA.INTFLAGS |= ADC_CH1IF_bm; //Clear CH1IF
```

7. Run the code and ensure that Atmel AVR Studio breaks where the breakpoint is set

8. Single step a few times and see how the results of the two virtual channels are available for read-out in separate registers and that no re-configuration is needed even if we measure two different inputs (almost) at the same time



Do you understand how virtual channels work, and what their benefits are?

6 Task 4: Event Triggered Conversion

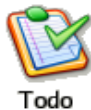
Up to now we have started ADC conversions from software by setting the `CHnSTART` bit for each conversion, or by setting the ADC in free running mode.

In this task we will use events to trigger conversions. The ADC can be set up to accept events to start a single conversion or a conversion sweep on two or more virtual channels.

In addition, the ADC can be set up to do a synchronized sampling, that is, flush the ADC and start a new conversion next peripheral clock cycle. This is very useful when you need accurate timing of the measurements.

The goal for this task is that you:

- Are able to set up event-triggered conversions for the ADC
- Can synchronize sampling on two ADCs
- Know that the ADC can use both PORTA and PORTB as input
- Can do basic debugging of the code



1. Locate and open the `EventTriggered.aps` project in Atmel AVR Studio. Take a look at the `task4.c` file and familiarize yourself with the code. You will also see that some C-code is missing

Eventually, when the missing code has been filled in, you will be able to use one of the ADC modules (ADCB can measure from the onboard peripherals while the ADCA inputs are available on the 10 pins port A header) for event triggered and synchronized conversions. The conversions will be synchronized by using the event system to produce events each time the Timer Counter C0 is overflowing.

2. First, you need to set up a pointer to the ADC module. The pointer should be of type `ADC_t *`, have the name `ADC_Pointer` and contain the address of ADCB (`&ADCB`). Look for the “ADD CODE” text in the start of the `main()` function and fill in
3. Open the Atmel XMEGA A Manual and look in the header file (`iox128a1.h`) to find the macro that sets up the event system Channel 0 MUX (`EVSYS.CH0MUX`)
4. In the `InitADC()` function, you need to set up the sweep and synchronization of it. Add the missing code. If you are unfamiliar with this, please refer to the XMEGA A Manual

The code for setting up the ADC should now be complete.

5. Build the project; ensure that there are no errors and start the debug mode in AVR Studio

6. Run the code. When you change the voltage by turning the potentiometer (as in task1), the LEDs should update if the code is correct
7. If you need to correct something in your code, build the project again and test. Use the IO-view to look at the registers and set breakpoints to do some basic debugging with the JTAGICE mkII
8. Change the code to use ADCA on PORTA instead of ADCB on PORTB. You only need to change one line of the code. Re-build and run the program. Note that an analog source must be connected to the PORTA header on the Xplain to get a signal for the measurements
9. If you have time, change the code to output a conversion result from both a PORTA pin and a PORTB pin on the LEDs. You may split the result in two parts, thus outputting four bits of each of the results to the eight LEDs. The purpose of this last exercise is to do two synchronized conversions at a time with one event



7 Summary

Here are some of the highlights of the ADC training:

- Single Conversion Mode
- Free Running Mode
- Compare Function
- Virtual Channels
- Use of the Event System to trigger and synchronize conversions

8 Resources

- Atmel XMEGA Manual and Datasheets
 - <http://www.atmel.com/xmega>
- Atmel AVR Studio with help files
 - <http://www.atmel.com/products/AVR/>
- WINAVR GCC compiler
 - <http://winavr.sourceforge.net/>
- Atmel IAR Embedded Workbench® compiler
 - <http://www.iar.com/>

9 Atmel Technical Support Center

Atmel has several support channels available:

- | | | |
|---------------|---|----------------------------|
| • Web portal: | http://support.atmel.no/ | All Atmel microcontrollers |
| • Email: | avr@atmel.com | All Atmel AVR products |
| • Email: | avr32@atmel.com | All 32-bit AVR products |

Please register on the web portal to gain access to the following services:

- Access to a rich FAQ database
- Easy submission of technical support requests
- History of all your past support requests
- Register to receive Atmel microcontrollers' newsletters
- Get information about available trainings and training material



Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2010 Atmel Corporation. All rights reserved. Atmel®, Atmel logo and combinations thereof, AVR®, AVR® logo, AVR Studio® and others, are the registered trademarks, XMEGA™ and others are trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.