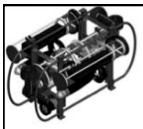
IMDL Software Series

Atmel AVR Xmega Code using GPIO from the Atmel ASF for the ADC on DAQ Boards

A. A. Arroyo



IMDL Software Series

ADC

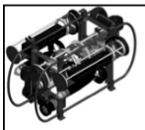
The Analog to Digital Converters (ADC) is used to sample and digitize analog signals. These devices are simple to communicate with and are quicker to set up than a USART.

Typical Applications:

- Audio Input
- Checking battery voltage
- Sensor - i.e., IR range finder

XMega A1 devices have two ADCs. These two modules can be operated simultaneously, individually or synchronized.

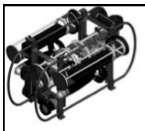
In the DAQ board they convert a 0 to 5v range into a 12-bit range (0 to 4095), about a ~.025% increment, e.g., 0 = 0v; 0xFFF=4095 ≈ 5v; 2047 ≈ 2.5v; 511 ≈ 0.625v, etc.



IMDL Software Series
ADC

28.1 Features

- 12-bit resolution
- Up to two million samples per second
  - Two inputs can be sampled simultaneously using ADC and 1x gain stage
  - Four inputs can be sampled within 1.5µs
  - Down to 2.5µs conversion time with 8-bit resolution
  - Down to 3.5µs conversion time with 12-bit resolution
- Differential and single-ended input
  - Up to 16 single-ended inputs
  - 16x4 differential inputs without gain
  - 8x4 differential input with gain
- Built-in differential gain stage
  - 1/2x, 1x, 2x, 4x, 8x, 16x, 32x, and 64x gain options
- Single, continuous and scan conversion options
- Four internal inputs
  - Internal temperature sensor
  - DAC output
  - V_{CC} voltage divided by 10
  - 1.1V bandgap voltage
- Four conversion channels with individual input control and result registers
  - Enable four parallel configurations and results
- Internal and external reference options
- Compare function for accurate monitoring of user defined thresholds
- Optional event triggered conversion for accurate timing
- Optional DMA transfer of conversion results
- Optional interrupt/event on compare result



IMDL Software Series
ADC

Figure 28-1. ADC overview.

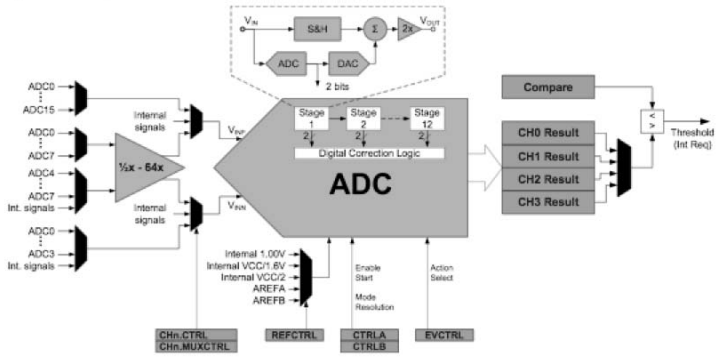


Figure 28-5. Single-ended measurement in unsigned mode.

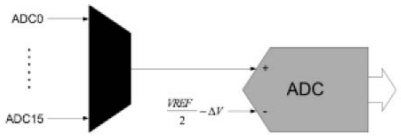
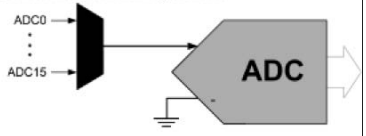
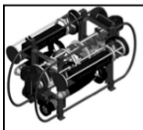


Figure 28-4. Single-ended measurement in signed mode.

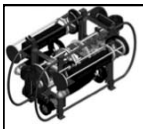




IMDL Software Series
ADC

Before a conversion is started, the desired input source must be selected for one or more ADC channels. An ADC conversion for a ADC channel can either be started by the application software writing to the start conversion bit for the ADC channel, or from any of the events in the Event System. It is possible to write the start conversion bit for several ADC channels at the same time, or to use one event to trigger conversions on several ADC channels at the same time. This makes it possible to scan several or all channels from one event. The scan will start from the lowest channel number.

.



IMDL Software Series
ADC

- The ADC is enabled by setting the ADC Enable bit0 in the CTRLA control Register.
  Example:

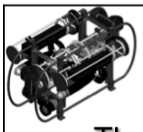
```
ADCA_CTRLA |= 0x01;  //Enable ADCA
ADCA_CTRLA |= 0x04; //Start Conversion on ADCA Chn 0
```

CTRLA – Control register A

Bit	7	6	5	4	3	2	1	0
+0x00	DMASEL[1:0]			CHSTART[3:0]			FLUSH	ENABLE
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 5:2 – CHSTART[3:0]: Channel Start Single Conversion**
  Setting any of these bits will start a conversion on the corresponding ADC channel. Setting several bits at the same time will start conversions on all selected ADC channels, starting with the channel with the lowest number. These bits are cleared by hardware when the conversion has started.

- **Bit 0 – ENABLE: Enable**
  Setting this bit enables the ADC.



IMDL Software Series
ADC

- The ADC 12-bit result is placed in the CHnRES Registers, CHnRESH and CHnRESL. By default, the result is presented right adjusted, but can optionally be presented left adjusted. Example:

```
ADCA_CTRLB = 0x00;        //12bit, right adjusted
```

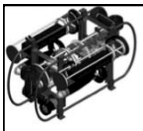
CTRLB – ADC Control register B

Bit	7	6	5	4	3	2	1	0
+0x01	IMPMODE	CURRLIMIT[1:0]		CONVMODE	FREERUN	RESOLUTION[1:0]		–
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R
Initial Value	0	0	0	0	0	0	0	0

- **Bit 3 – FREERUN: Free Running Mode**
  When the bit is set to one, the ADC is in free running mode and the ADC channels defined in the EVCTRL register are swept repeatedly.
- **Bit 2:1 – RESOLUTION[1:0]: Conversion Result Resolution**
  These bits define whether the ADC completes the conversion at 12- or 8-bit result resolution. They also define whether the 12-bit result is left or right adjusted within the 16-bit result registers. See Table 28-4 on page 352 for possible settings.

Table 28-4. ADC conversion result resolution.

RESOLUTION[1:0]	Group configuration	Description
00	12BIT	12-bit result, right adjusted
01		Reserved
10	8BIT	8-bit result, right adjusted
11	LEFT12BIT	12-bit result, left adjusted



IMDL Software Series
ADC

- The ADC reference is selected in the REFCTRL.
  Example: On my Epiphany DIY

```
ADCA_REFCTRL = 0x10;   //Set to Vref = 5v on ADCA
                       //Set to Vref = Vcc/1.6 ≈ 2.0v on ADCB
```

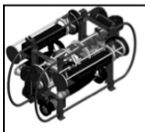
REFCTRL – Reference Control register

Bit	7	6	5	4	3	2	1	0
+0x02	–	REFSEL[2:0]			–	–	BANDGAP	TEMPREF
Read/Write	R	R/W	R/W	R/W	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7 – Reserved**
  This bit is unused and reserved for future use. For compatibility with future devices, always write this bit to zero when this register is written.
- **Bits 6:4 – REFSEL[2:0]: Reference Selection**
  These bits selects the reference for the ADC according to Table 28-5 on page 353.

Table 28-5. ADC reference selection.

REFSEL[2:0]	Group configuration	Description
000	INT1V	1/1.1 of bandgap (1.0V)
001	INTVCC	V_{CC}/1.6
010	AREFA	External reference from AREF pin on PORT A
011	AREFB	External reference from AREF pin on PORT B
100	INTVCC2	V_{CC}/2
101 - 111		Reserved



IMDL Software Series
ADC

- Set Pre-Scaler Configuration. Example:

  ADCA_PRESCALER = 0x03; // sets Prescaler to DIV32

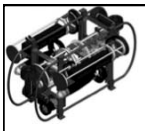
PRESCALER – Clock Prescaler register

Bit	7	6	5	4	3	2	1	0
+0x04	–	–	–	–	–	PRESCALER[2:0]		
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 2:0 – PRESCALER[2:0]: Prescaler Configuration**
  These bits define the ADC clock relative to the peripheral clock according to Table 28-9 on page 355.

Table 28-9. ADC prescaler settings.

PRESCALER[2:0]	Group configuration	Peripheral clock division factor
000	DIV4	4
001	DIV8	8
010	DIV16	16
011	DIV32	32
100	DIV64	64
101	DIV128	128
110	DIV256	256
111	DIV512	512



IMDL Software Series
ADC

- To start single-ended conversions we use the CTRL register. Example:

  ADCA_CH0_CTRL = 0x01; //set to single-ended

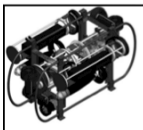
CTRL – Channel Control register

Bit	7	6	5	4	3	2	1	0
+0x00	START	–	–	GAIN[2:0]			INPUTMODE[1:0]	
Read/Write	R/W	R	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7 – START: START Conversion on Channel**
  Setting this bit will start a conversion on the channel. The bit is cleared by hardware when the conversion has started. Setting this bit when it already is set will have no effect. Writing or reading this bit is equivalent to writing the CH[3:0]START bits in "CTRLA – Control register A" on page 351.

Table 28-11. Channel input modes, CONVMODE=0 (unsigned mode).

INPUTMODE[1:0]	Group configuration	Description
00	INTERNAL	Internal positive input signal
01	SINGLEENDED	Single-ended positive input signal
10		Reserved
11		Reserved



IMDL Software Series
ADC

- The MUX register defines the input source for the channel. Example:

  ADCA_CH0_MUXCTRL = 0x08; //default to Channel 1

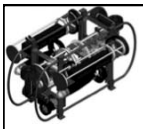
MUXCTRL – ADC Channel MUX Control registers

The MUXCTRL register defines the input source for the channel.

Bit	7	6	5	4	3	2	1	0
+0x01	–	MUXPOS[3:0]				MUXNEG[2:0]		
Read/Write	R	R/W	R/W	R/W	R/W	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Table 28-14. ADC MUXPOS configuration when INPUTMODE[1:0] = 01 (single-ended) or INPUTMODE[1:0] = 10 (differential) is used.

MUXPOS[3:0]	Group configuration	Description
0000	PIN0	ADC0 pin
0001	PIN1	ADC1 pin
0010	PIN2	ADC2 pin
0011	PIN3	ADC3 pin
0100	PIN4	ADC4 pin
0101	PIN5	ADC5 pin
0110	PIN6	ADC6 pin
0111	PIN7	ADC7 pin



IMDL Software Series
ADC

- Conversions are controlled with the INTCTRL. Example:

  ADCA_CH0_INTCTRL = 0x00; //set flag at conversion complete. Disable interrupt

INTCTRL – Channel Interrupt Control registers

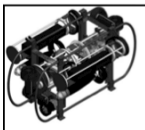
Bit	7	6	5	4	3	2	1	0
+0x02	–	–	–	–	INTMODE[1:0}		INTLVL[1:0]	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 3:2 – INTMODE: Interrupt Mode**
  These bits select the interrupt mode for the channel according to Table 28-18.

Table 28-18. ADC channel select.

INTMODE[1:0]	Group configuration	Interrupt mode
00	COMPLETE	Conversion complete
01	BELOW	Compare result below threshold
10		Reserved
11	ABOVE	Compare result above threshold

- **Bits 1:0 – INTLVL[1:0]: Interrupt Priority Level and Enable**
  These bits enable the ADC channel interrupt and select the interrupt level, as described in "Interrupts and Programmable Multilevel Interrupt Controller" on page 130. The enabled interrupt will be triggered for conditions when the IF bit in the INTFLAGS register is set.



IMDL Software Series
ADC

- Conversions flags are in the INTFLAGS.

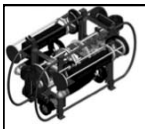
  Example:

```
// wait for conversion to complete
while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01);
```

INTFLAGS – ADC Channel Interrupt Flag registers

Bit	7	6	5	4	3	2	1	0
+0x03	–	–	–	–	–	–	–	IF
Read/Write	R	R	R	R	R	R	R	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7:1 – Reserved**
  These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.
- **Bit 0 – IF: Channel Interrupt Flag**
  The interrupt flag is set when the ADC conversion is complete. If the channel is configured for compare mode, the flag will be set if the compare condition is met. IF is automatically cleared when the ADC channel interrupt vector is executed. The bit can also be cleared by writing a one to the bit location.



IMDL Software Series
ADC - Epiphany

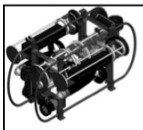
* AAA_Util.c - Notepad2 (Administrator)

File Edit View Settings ?

```
/*********
 * Xmega *
 *********/
void XmegaInit(void) {                  // XmegaInit: initialize Xmega CPU to 32MHz
    CCP          = 0xD8;                // Configuration Change Protection: IOREG (0xD8)
    CLK_PSCTRL   = 0x00;                // System Clock Prescaler Reg All prescalers No Division
    OSC_CTRL     = 0x02;                // Bit 1- RC32MEN: Setup 32 MHz Internal CPU RC Oscillator Enable
    while ((OSC_STATUS & 0x02) == 0);   // wait for Bit 1 - RC32MRDY: 32 MHz Internal RC Oscillator Ready
    CCP          = 0xD8;                // Configuration Change Protection: IOREG (0xD8)
    CLK_CTRL     = 0x01;                // Bit 2:0 - SCLKSEL[2:0]: System Clock Selection
}   // End XmegaInit                    // 001 : RC32MHZ   32MHz Internal RC Oscillator

/********
 * ADCA *
 ********/
void ADCAInit(void) {               // ADCAInit: Initialize ADC on PortA pins
    ADCA_CTRLB       = 0x00;        // 12bit, right adjusted
    ADCA_REFCTRL     = 0x10;        // set to Vref = Vcc/1.6 = 2.0v (approx) In Tim's Board 5v
    ADCA_CH0_CTRL    = 0x01;        // set to single-ended
    ADCA_CH0_INTCTRL = 0x00;        // set flag at conversion complete.  Disable interrupt
    ADCA_CH0_MUXCTRL = 0x08;        // default set to Channel ADC1 PortA Pin 1
    ADCA_PRESCALER   = 0x03;        // sets ADC prescaler to DIV32
    ADCA_CTRLA      |= 0x01;        // Enable ADCA
}   // End ADCAInit

int analog(int channel) {               // analog(channel{ADCA PortA<0:7>}) Read the PortA ADC channel
    ADCA_CH0_MUXCTRL = channel*8;       // Set to ADCn PortA Pin n where n=channel is <7-0>
    ADCA_CTRLA      |= 0x04;            // Start Conversion on ADCA Channel 0
    while ((ADCA_CH0_INTFLAGS & 0x01) != 0x01); // wait for conversion to complete
    _delay_ms(5);                       // wait for result to stablize {may not be needed}
    int value        = ADCA_CH0_RES;    // grab result
    return value;                       // return result
}   // End analog(channel)
```



IMDL Software Series
ADC - DAQ

IMDL Software Series
ADC

- The next example uses the ADC to sample a Cds cell and turn on/off the debug LED on PC1 in the main loop in Tim's DAQ Board.

  See project DAQ_GPIO_ADC

- If you wish to use Tim's USART functions, then import uart.c and uart.h from Tim's Software

  See project DAQ_GPIO_ADC
  Change the include in main.c to <uart.h>

IMDL Software Series
Arduino ADC

- The example uses the ADC functions in the Arduino MEGA2560 Board to blink internal/ external LEDs and uses the serial port to display data and control the LEDs after blinking the set a few times. Serial input *j* & *k* alternate the two external LEDs on pins 12 & 11 {on, off} and {off, on} respectively. Analog input A0 is connected to a Cds cell that controls the internal DBLED
  - See Arduino Sketch: IMDL_ADC_Serial_External_LED

IMDL Software Series

```
/*
  EEL-5666 Example 3: Use of Digital Output Pins
  Turn on/off External LEDs connected to Pins 11 & 12
  alternatively and the internal LED on Pin 13 on/off.
  Add a delay in each state of 1/2 sec. Use serial input
  to control the LEDs. {i,o} for internal LED {j,k} for
  external LEDs. Read Cds cell on Analog A0 and control
  the internal LED.
  Arroyo January 18, 2016

*/

int extLED1 = 11;   // External LED connected to digital pin 11
int extLED2 = 12;   // External LED connected to digital pin 12
int intLED  = 13;   // Internal LED connected to digital pin 13

void setup( ) {
  pinMode(extLED1, OUTPUT);    // sets digital pin 12 as output
  pinMode(extLED2, OUTPUT);    // sets digital pin 12 as output
  pinMode(intLED,  OUTPUT);    // sets digital pin 13 as output
  Serial.begin(57600);        // connect to the serial port
}

  int rcvData=0, count=0, ambient, a1;
void loop( ) {
  if (count < 1) {
    Serial.println("Arduino MEGA2560 Example");
    Serial.println("Cds Cell connected on Analog A0");
    Serial.println("Control the internal LED on pin 13");
    ambient = analogRead(A0);
    Serial.print("Ambient: ");
    Serial.println(ambient);

    for (int j=0; j<5; j++){
      Serial.print("j= ");
      Serial.println(j);
      digitalWrite(13, HIGH);   // set the internal LED on
```

Normal text file | length : 2739 lines : 74 | Ln : 28 Col : 55 Sel : 3 | 0

IMDL Software Series

```
      digitalWrite(12, LOW);    // set the external LED1 off
      digitalWrite(11, HIGH);   // set the external LED2 on
      delay(500);               // wait for a 1/2 second
      digitalWrite(13, LOW);    // set the internal LED off
      digitalWrite(12, HIGH);   // set the external LED on
      digitalWrite(11, LOW);    // set the external LED off
      delay(500);               // wait for a 1/2 second
    }
    digitalWrite(13, HIGH);   // set the internal LED  on
    digitalWrite(12, HIGH);   // set the external LED1 on
    digitalWrite(11, HIGH);   // set the external LED2 on
  }
  count=1;
  Serial.print("Ambient: ");
  Serial.print(ambient);
  Serial.print("   A1: ");
  a1 = analogRead(A0);
  if (a1 > 1.1*ambient ) digitalWrite(13, LOW); // turn off DBLED
  else digitalWrite(13, HIGH);    // else turn on DBLED
  Serial.println(a1);
  delay(100);

  // blink data only when you receive data:
  if (Serial.available() > 0) {
  // read the incoming byte:
     rcvData = Serial.read();
  // say what you got:
     Serial.print("I received: ");
     Serial.write(rcvData);
     Serial.println();
     if (rcvData == 'i') digitalWrite(13, HIGH);      // internal LED on
     else if (rcvData == 'o') digitalWrite(13, LOW);  // internal LED off
     else if (rcvData == 'j') {digitalWrite(12,LOW); digitalWrite(11,HIGH);}
     else if (rcvData == 'k') {digitalWrite(12,HIGH); digitalWrite(11,LOW);}
     }
}

Normal text file   length : 2739   lines : 74   Ln : 28   Col : 55   Sel : 3 | 0
```


IMDL Software Series

The End!