

Analog-to-Digital Conversion with the ATmega16

BJ Furman
03NOV2009

Features of the ADC system for the ATmega16

- 8 or 10-bit resolution
 - 8 bit $\Rightarrow 2^8 = 256$ output states, so resolution of $V_{FSR}/2^8 = 1$ part in 256 of V_{FSR} (V_{REF})
 - 10 bit $\Rightarrow 2^{10} = 1024$ output states, so resolution of $V_{FSR}/2^{10} = 1$ part in 1024 of (V_{REF})
- 8 channel MUX \Rightarrow 8 single-ended (i.e. referenced to GND) voltage inputs on PORTA
- 16 combinations of differential inputs
 - Two (ADC1, ADC0 and ADC3, ADC2) have a programmable gain stage with 1X, 10X, or 200X gain selectable
 - 1X or 10X can expect 8 bit resolution
 - 200X 7-bit resolution
 - Seven differential channels share ADC1 as the common negative terminal (ADC0-ADC1)
- Input voltage range is $0\text{ V} - V_{CC}$
- V_{REF} can be internal (either 2.56 V or AVCC) or externally supplied (but must be less than V_{CC})
- Auto-trigger or single conversion modes
 - It takes 12 clock cycles to initialize the ADC circuitry on the first conversion after ADC is enabled. Thereafter, it takes 13 clock cycles to complete a conversion.
 - ADC circuitry needs 50 kHz to 200 kHz clock signal. So if you are using an 8 MHz system clock, then you need a pre-scaler of at least $8/0.2 = 40$. The higher the frequency, the faster the conversion, but also the less accurate.
 - $8\text{E}6/64=125\text{ kHz}/13=9.6\text{ kHz} \Rightarrow 4.8\text{ kHz}$ to avoid aliasing
- Interrupt on ADC conversion complete
- The results appear in ADCL and ADCH. Need to read ADCL ***first*** to prevent ADCH from being overwritten with new data!

Steps for Using the ADC

Set up ADCSRA and ADMUX according to your needs:

1. Configure ADCSRA
 - a. Choose the input channel, mode (single-ended or differential), and gain (if applicable) using ADMUX bits 3-0
 - b. Choose V_{REF} (AREF, AVCC, or internal $2.56 V_{ref}$) using ADMUX bits 7-6
 - c. Choose left or right alignment of conversion result in ADC Data Register using ADMUX bit 5
 - d. Choose to enable or disable ADC Auto-trigger using ADCSRA bit 5. If enabled, you then need to set bits 7-5 in SFOR to select the auto-trigger source
 - e. Choose to enable or disable ADC Interrupt using ADCSRA bit 3. If enabled, you will also need to enable Global Interrupts in order for the ADC Interrupt to be acted upon ($SREG \mid= (1 \ll SREG_I);$). You will also need an Interrupt Service Routine (ISR) to service the interrupt

- f. Choose the clock prescaler using ADCSRA bits 2-0
 - g. Turn on the ADC using ADCSRA bit 7: 1 to turn on; 0 to turn off
2. Start a Conversion
 - a. Write a 1 to bit 6 of ADCSRA to start a conversion
 3. Get the Converted Result
 - a. Either by ADC Interrupt or by checking bit 6 of ADCSRA (conversion is complete when this bit is zero) or by checking bit 4 of ADCSRA (conversion is complete when this bit is one)

Notes

1. To use an interrupt, you will need an interrupt service routine (ISR)
 - This is a bit of code that the program will jump to if the ADC Interrupt Enable bit is set in ADCSRA and if the Global Interrupt bit has been set. Here is an example:

```
ISR(ADC_vect)
{
    /* do stuff here */
}
```

- The interrupt handling routine call will, by hardware, clear the ADIF flag

- Make sure to #include <avr/interrupt.h>

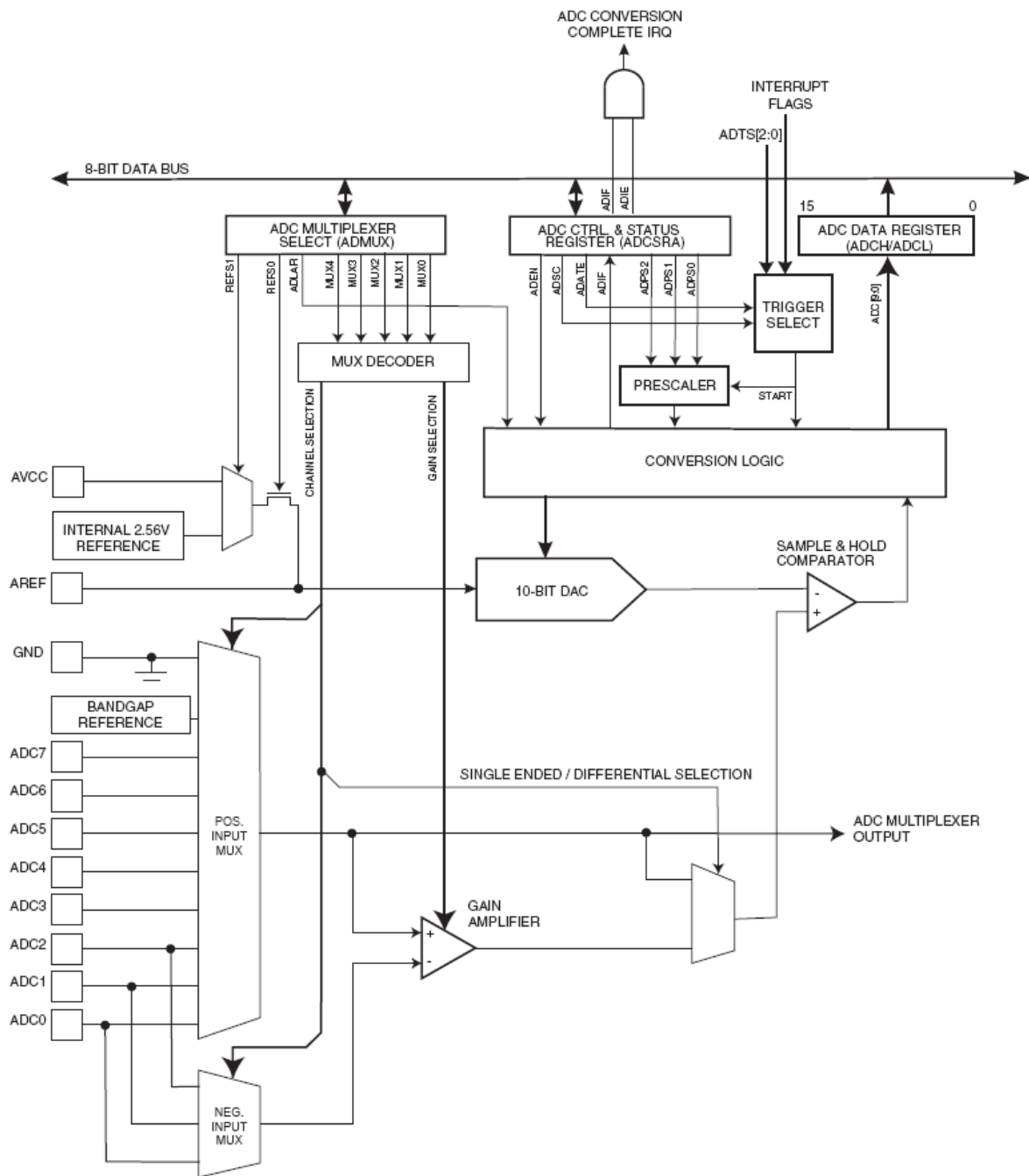
2. Read data from ADC Data register: ADCL first, then ADCH (if 10 bit desired)
 - a. Note that access to the ADC data register is blocked until both ADCL and ADCH are read. Once ADCH is read, the ADC data register can be updated.
 - b. ADLAR=0 (right shifted)

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

- c. ADLAR=1 (left-shifted)

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

Figure 98. Analog to Digital Converter Block Schematic



(Source: ATmega16 data sheet, p. 205, rev. 2466S-AVR-05/09)

ADC Registers (source: ATmega16 datasheet)

ADCSRA (ADC Control and Status Register A)

Bit	Description
7	ADC Enable: 1 to enable; 0 to disable
6	ADC Start: write 1 to start; will clear (0) when conversion is ready
5	ADC Auto Trigger Enable: 1 to enable (see ADTS bits in SFIOR); 0 to disable
4	ADC Interrupt Flag: 1 is set when conversion completes; 0 otherwise
3	ADC Interrupt Enable: 1 to enable; 0 to disable
2	ADC Pre-scaler Bits: (see table in data sheet)
1	
0	

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

Bit	7	6	5	4	3	2	1	0	SFIOR
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 85. ADC Prescaler Selections

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 86. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

ADMUX (ADC Multiplexer Register)

Bit	Description
7	Voltage Reference Selection Bits (see table)
6	
5	ADC Left Adjust Result: 1 to left adjust; 0 to right adjust
4	Analog Channel and Gain (see table)
3	
2	
1	
0	

REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
7	6	5	4	3	2	1	0

Table 83. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Table 84. Input Channel and Gain Selections

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	-	-	-	-	-	-	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	-	-	-	-	-	-	ADCL
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

```

#include <avr/io.h>
#include "uart.h"

void init();
//Initializes UART, ADC.

uint16_t do_adc();
//Starts an ADC conversion on ADC0 (PA0), waits for
//conversion to finish, and returns 10-bit adc value.

int main(void)
{
    uint16_t adc_value; //Unsigned integer to hold the ADC value.

    init();

    while(1)
    {
        adc_value = do_adc();

        //Add your code here.
    }

    return 0;
}

void init()
{
    //Init ADC. See the ATmega 16 datasheet for more info.
    ADMUX = _BV(REFS0); //Set the ADC voltage reference to AVCC (Analog
    voltage in)
    ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1) | _BV(ADPS0); //Enable the
    ADC and set PS.

    //Init UART
    uart_init();
}

uint16_t do_adc()
{
    //Start an ADC conversion by setting the ADSC bit.
    ADCSRA |= _BV(ADSC);

    //Wait for the conversion to finish. The ADC signals that it's finished
    // by clearing the ADSC bit that we set above.
    while( ADCSRA & _BV(ADSC) )
        ; //Do nothing.

    //Return the ADC result from the ADC register.
    return ADC;
}

```