ADC with the Atmega128

BJ Furman
10APR2007

- (preliminaries: the concept of A/D conversion, resolution, quantization, etc. See p 321-334 in text)
- Describe successive approximation A/D

Features of the AD system for the 128

- 8 or 10-bit resolution
  8 bit => 2^8 = 256 output states, so resolution of $V_{FSR}/2^8$ = 1 part in 256 of V_{FSR} (V_{REF})
  10 bit => 2^{10} = 1024 output states, so resolution of $V_{FSR}/2^{10}$ = 1 part in 1024 of (V_{REF})
- 8 channel MUX => 8 single-ended (i.e. referenced to GND) voltage inputs on PORTF
- 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 V – V_{CC}
- V_{REF} can be internal (either 2.56 V or AVCC) or externally supplied (but must be less than V_{CC}
- Free running 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 prescaler of at least 8/0.2 = 40. The higher the frequency, the faster the conversion, but also the less accurate.
      - 8E6/64=125 kHz/13=9.6 kHz => 4.8 kHz to avoid aliasing
- Interrupt on ADC conversion complete

Two registers control the A/D converter:

ADCSRA (A/D Control and Status Register) See data sheet or p. 138 in Barnett book.

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

ADMUX (A/D Multiplexer Select Register)

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

The results appear in ADCL and ADCH. Need to read ADCL ***first*** to prevent ADCH from being overwritten with new data!

Procedure to Initialize ADC:

Set up ADCSRA and ADMUX:

1. Turn on the ADC (ADEN=1)
2. Choose single-conversion or free-run (ADFR=0 means single conversion)
3. Clock prescaler (selects system clock divider. Smaller divider => faster but less accurate conversion
4. Choose the voltage reference by selecting proper bits, in location 7 and 6 of ADMUX
5. Choose left or right adjustment of result (in ADMUX register, ADLAR=0 for right adjust)
6. Choose the AD channel to convert (in ADMUX, MUX bits)

Procedure to Do a Conversion:

1. Start a conversion by writing a 1 to ADC Status and Control Register, bit 6 (ADSC)
2. Wait until conversion is complete
   a. Can monitor bit 6 (ADSC). It will stay as 1 until conversion completes, or
   b. Generate an interrupt
      - Bit 4 (ADIF) of ADSC will be set when the conversion is complete
      - To use interrupts must:
        - Set bit 3 (ADIE) of ADCSRA and
        - Enable global interrupts: sei(); (which sets the I-bit in the Status Register SREG

          AND
        - Define an interrupt handling routine. Ex:
          ```
          SIGNAL(SIG_ADC)
          {
              /* do stuff here */
          }
          ```
          - The interrupt handling routine call will, by hardware, clear the ADIF flag
        - Make sure to #include <avr/interrupt.h>
3. 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

d. Example

```
// ADC Conversion Example
// BJ Furman 18APR05
//
//----- Include Files ---------------------------------------------------------
#include <avr/io.h> // include I/O definitions (port names, pin names, etc)
#include "global.h" // include our global settings
//
//----- Defines -------------------------------------------------------------
#define BV(bit)   (1<<(bit))    // Byte Value => converts bit into a byte value. One at bit location.
#define cbi(reg, bit)   reg &= ~(BV(bit))    // Clears the corresponding bit in register reg
#define sbi(reg, bit)   reg |= (BV(bit))       // Sets the corresponding bit in register reg
//
//----- Function Prototypes -------------------------------------------------------------
void adc_init(void);         // Will set up the registers for A/D conversion
//----- Begin Code -------------------------------------------------------------
int main(void)
{
    unsigned short adc_result;   // Just a variable to hold the result
    adc_init();    // Call the init function
    DDRF = 0x00; // configure a2d port (PORTF) as input so we can receive analog signals
    PORTF = 0x00; // make sure pull-up resistors are turned off (else we'll just read 0xCFF)
    while(1)
    {
        sbi(ADCSRA,ADSC);     // start a conversion by writing a one to the ADSC bit (bit 6)
        while(ADCSRA & 0b01000000); // wait for conversion to complete (bit 6 will change to 0)
        adc_result = ((ADCL) | ((ADCH)<<8)); // 10-bit conversion for channel 0 (PF0)
    }
    return 0;
} // end main()
//
void adc_init(void)
{
    sbi(ADCSRA,ADEN);   // enables ADC by setting bit 7 (ADEN) in the ADCSRA
    cbi(ADCSRA,ADFR);   // single sample conversion by clearing bit 5 (ADFR) in the ADCSRA
    ADCSRA = ((ADCSRA & 0b11111000) | 0b00000110);   // selects div by 64 clock prescaler
    ADMUX = ((ADMUX & 0b00111111) | 0b01000000);     // selects AVCC as Vref
    cbi(ADMUX,ADLAR);           // selects right adjust of ADC result
    ADMUX &= 0b11100000;     // selects single-ended conversion on PF0
}
```

Single-Ended vs. Differential Measurements

	Single-ended	Differential
ADC Value =	V_{in}*1024/V_{ref}	(V_{pos} – V_{neg})*Gain*512/V_{ref}
Voltage meas. =	(ADC value)*V_{ref}/1024	(V_{pos} – V_{neg})=(ADC value)*V_{ref}/(Gain*512)

For differential measurements, if you simply want to determine the polarity of the result, check the MSB in of the converted result (i.e., ADC9, bit 9 for right-adjusted result)

```
if(ADCH & 0x02)    // If true, then Vneg > Vneg
{…..}
```

```
// ADC Conversion Example
// BJ Furman 18APR05
//----- Include Files ---------------------------------------------------------
#include <avr/io.h> // include I/O definitions (port names, pin names, etc)
#include "global.h" // include our global settings
#include <avr/interrupt.h> // include interrupt support
#include "global.h" // include our global settings
#include "a2d.h" // include A/D converter function library
//----- Defines ---------------------------------------------------------------
#define   BV(bit)   (1<<(bit))
#define   cbi(reg, bit)   reg &= ~(BV(bit))
#define   sbi(reg, bit)   reg |= (BV(bit))
//----- Begin Code ------------------------------------------------------------
int main(void)
{
u08 ADres; // unsigned 8-bit integer
// Setup A/D converter
a2dInit(); // turn on and initialize A/D converter
DDRF = 0x00; // configure a2d port (PORTF) as input so we can receive analog signals
PORTF = 0x00; // make sure pull-up resistors are turned off
/* set the a2d prescaler (clock division ratio) a lower prescale setting will make the a2d converter
        go faster, and a higher setting will make it go slower, but the measurements will be more
        accurate - other allowed prescale values can be found in a2d.h*/
a2dSetPrescaler(ADC_PRESCALE_DIV32);
        /* set the a2d reference - the reference is the voltage against which a2d measurements
        are made - other allowed reference values can be found in a2d.h*/
a2dSetReference(ADC_REFERENCE_AVCC);
while(1)
{
ADres = a2dConvert8bit(0); // 8-bit conversion for channel 0 (PF0)
}
return 0;
} // end main()
```