Blog Entry

Analog to Digital Converter AVR C Programming

November 19, 2008 by rwb, under Microcontroller.

One of the important features in today's modern microcontroller is the capability of converting the analog signal to the digital signal. This feature allows us to process the analog world easily such as temperature, humidity, light intensity, distance, etc; which usually captured by electronics sensor and represent it on the change of voltage level.

Some sensors that required the ADC circuit

In this tutorial we will show you how to program the Atmel AVR microcontroller for reading the analog signal. We will use the AVRJazz Mega168 board as our learning platform and let's start the fun by pasting this code to your AVR Studio 4 editor:

```
//*****************************************************************************
//  File Name    : ADC.c
//  Version      : 1.0
//  Description  : Using AVR ADC Peripheral
//  Author       : RWB
//  Target       : AVRJazz Mega168 Learning Board
//  Compiler     : AVR-GCC 4.3.0; avr-libc 1.6.2 (WinAVR 20080610)
//  IDE          : Atmel AVR Studio 4.14
//  Programmer   : AVRJazz Mega168 STK500 v2.0 Bootloader
//               : AVR Visual Studio 4.14, STK500 programmer
//  Last Updated : 21 March 2008
//*****************************************************************************
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
   unsigned char chSign,chEye;
   unsigned int iDelay;

   DDRD = 0xFF;                 // Set PORTD as Output
   chEye=0x01;                  // Initial Eye Variables with 0000 0001
   chSign=0;

   // Set ADCSRA Register in ATMega168
   ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1);

   // Set ADMUX Register in ATMega168
   ADMUX=0;
```

Search This Site

Search

Google Custom Search

Future Post

Controlling the Motor is one of interesting topics in the embedded world especially for the robotics enthusiasts, on the next post we will learn the basic of motor electronic circuit as well as how to control it with microcontroller.

Therefore don't miss it, stay tune on this blog !

Categories

Categories
Select Category

Blogroll

- ermicro shop
- ermicroblog Amazon Store
- ermicroblog on YouTube
- ermicroblog video on Metacafe

Archives

Archives Select Month

```
    for(;;) {                    // Loop Forever
      // Start conversion by setting ADSC in ADCSRA Register
      ADCSRA |= (1<<ADSC);

      // wait until conversion complete ADSC=0 -> Complete
      while (ADCSRA & (1<<ADSC));

      // Get ADC the Result
      iDelay = ADCW;

      if (iDelay < 1) iDelay=1;

      // Display the LED
      if (chSign == 0) {
        PORTD=chEye;
        _delay_ms(iDelay);          // Call Delay function
        chEye=chEye << 1;
        if (chEye >= 0x80) chSign=1;
      } else {
        PORTD=chEye;
        _delay_ms(iDelay);          // Call Delay function
        chEye=chEye >> 1;
        if (chEye <= 0x01) chSign=0;
      }
    }
    return 0;                        // Standard Return Code
}

// EOF: ADC.c
```

Before building the program first configure the frequency to **11059200** Hz from menu **Project** -> **Configuration Option**. Rebuild and down load the program to the AVRJazz Mega168 board by first putting the board in programming mode and select STK500 or AVRISP programmer from AVR Studio 4. For detail explanation of using this board feature you could go to AVRJazz Mega168 Learning and Development Board or Starting Atmel AVR C Programming Tutorial.

While running the program, the 8 blue LEDs start to shifting back and forth to the left and right; try to adjust the user's trimport by turning it to the left or right and you could see the speed of LED shifting will vary.

The user's trimport basically work as a voltage divider circuit and provide voltage input level to the microcontroller analog port (**RC0**) by changing the trimmer; it will change the voltage input level to the analog port.

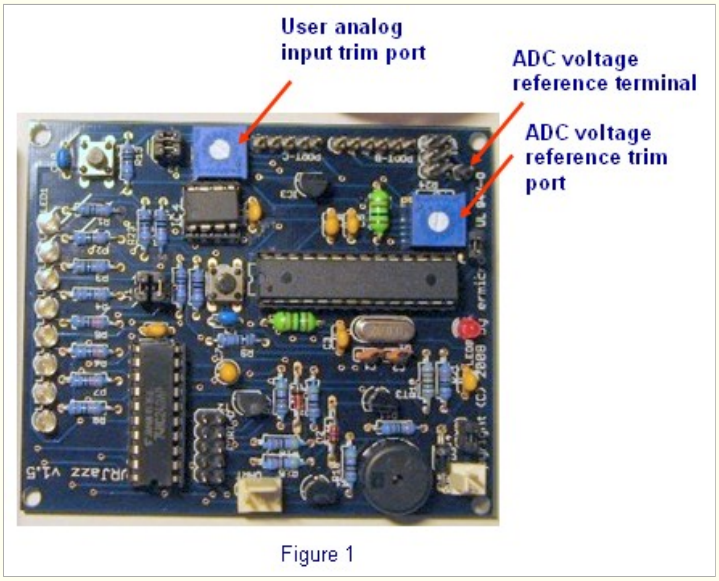


Figure 1

For the ADC peripheral programming on the Atmel AVR Mega168 microcontroller we will focus on these 4 important registers (special memory location on the Atmel AVR microcontroller families):

1. ADCSRA – Control Status and Status Register A

The function of this register is to control the microcontroller ADC operation such as enabling the ADC feature, start converting, prescaler selection and interrupt control.

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

On this tutorial we will focus on **ADEN, ADSC, ADPS2, ADPS1** and **ADPS0** bits. When setting the **ADEN** to logical "**1**" it's mean we enable the microcontroller ADC peripheral function; by setting the **ADSC** to logical "**1**" it's mean we instruct the microcontroller to start the conversion.

The **ADPS2, ADPS1** and **ADPS0** bits are used to set the circuit clock prescaler. In order for ADC successive approximation circuitry inside the microcontroller work to it's maximum resolution, it need to be supplied with **50 kHz to 200 kHz** frequency clock; this could be supplied from the system clock. Because the system clock is far away higher, therefore we need to prescale it by 64. it's mean we will supply this circuit with **11059200 Hz / 64 = 172800 Hz = 172.8 kHz**. The table bellow is the complete prescaler setting for these 3 bits:

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

The code for setting all these bits in **ADCSRA** control register:

```
// Set ADCSRA Register in ATMega168
ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1);
```

This code will set the **ADEN, ADPS2** and **ADPS1** bits; therefore the **ADCSRA** register will have this following value:

ADEN=1, ADSC=0, ADATE=0, ADIF=0, ADIE=0, ADPS2=1, ADPS1=1, ADPS0=0

2. ADMUX – ADC Multiplexer Selection Register

This register is used to select the analog port channel to be used, voltage reference for the ADC operation and the data presentation of the ADC result in the **ADCL** and **ADCH** data registers after conversion.

We just leave the **REFS1** and **REFS0** to it's default value (logical "**0**") which mean we use the external voltage reference. The **ADLAR** bit is set to logical "**0**"; means we instruct the microcontroller to place the highest 2 bits data conversion in **ADLR** and the lowest 8 bits data conversion in **ADLC** register.

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
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	

On the AVRJazz Mega168 board this voltage can be adjusted by using the voltage reference trim port and measure the reference voltage on the voltage reference terminal, the default value is set to 1 volt for 4.5 volt power supply.

The **MUX3, MUX2, MUX1** and **MUX0** bits is the analog input port selector. The AVR Mega168 (PDIP package) has 6 analog input port. The table bellow is the complete setting for these 4 bits:

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8[1]
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V_{BG})
1111	0V (GND)

The user analog input in the AVRJazz Mega168 board is attached to the **ADC0** (**RC0**), therefore the code for setting all these 8 bits in **ADMUX** multiplexer selection register is

```
// Set ADMUX Register in ATMega168
ADMUX=0;
```

The next two code inside the for loops statement; is to start the ADC conversion and wait for the microcontroller to finish the conversion by monitoring the **ADSCRA** control register

```
// Start conversion by setting ADSC in ADCSRA Register
ADCSRA |= (1<<ADSC);

// wait until conversion completed ADSC=0 -> Complete
while (ADCSRA & (1<<ADSC));
```

First we instruct the microcontroller to start the ADC conversion by setting the **ADSC** bit in **ADSCRA** control register to logical "**1**"; then we just wait for this bit to turn off (the C while statement). The microcontroller will automatically turn this bit off after ending the ADC conversion.

3. ADCL and ADCH – The ADC Data Register

The ADC conversion result will be placed in these two 8 bit ADC data register. It's need two 8 bits data register to hold the result; because the Atmel AVR ADC circuit used 10 bit resolution. Which's mean the first lowest 8 bits will be placed in the **ADCL** register and the remaining 2 bits will be placed in the **ADCH** register (assuming we use the default **ADLAR** value of **0** in the **ADMUX** register).

The result will be assigned to **iDelay** variable using the special GCC-AVR variable **ADCW** (remember this is not the AVR register) that actually hold the value of **ADCH + ADCL** register when the program compiled:

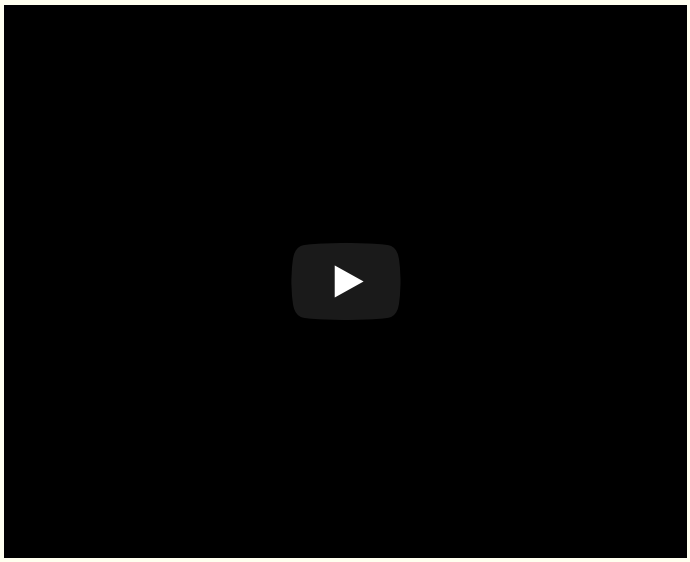
```
// Get ADC the Result
iDelay = ADCW;
```

or we could just create our own replacement for the **ADCW** variable as follow:

```
// Get ADC the Result
iDelay = ADCL;
iDelay += (ADCH << 8);
```

At last the remaining program; we just call the **_delay_ms()** function with the argument of **iDelay** variable before displaying data on PORTD (the LED), therefore the LED delay is depend on the ADC value. For further information about the ADC peripheral on Atmel AVR microcontroller please read the Atmel ATMega168 datasheet.

This is the running ADC program video in AVRJazz Mega168 board, you could see the speed of running LED is changing by changing the ADC input voltage level using the user trimport:



Bookmarks and Share

Related Posts

- How to use I2C-bus on the Atmel AVR Microcontroller
- AVR Twinkle Twinkle Using PWM Project
- AVR LCD Thermometer Using ADC and PWM Project
- Transforming your AVR Microcontroller to the I2C or TWI Slave I/O Expander Project
- Build Your Own Microcontroller Based PID Control Line Follower Robot (LFR) – Second Part

3 Responses to "Analog to Digital Converter AVR C Programming"

29.12.08 **#1**

Comment by **sunnymag**.

wow. thanks for the incredible tutorial. i really found it interesting and useful.

08.10.13 **#2**

Comment by **FreshMan**.

in what IDE do you write code ?

08.10.13 **#3**

Comment by **rwb**.

The Atmel AVR Studio 4.14 IDE when this project C code is written (see the program source header information)