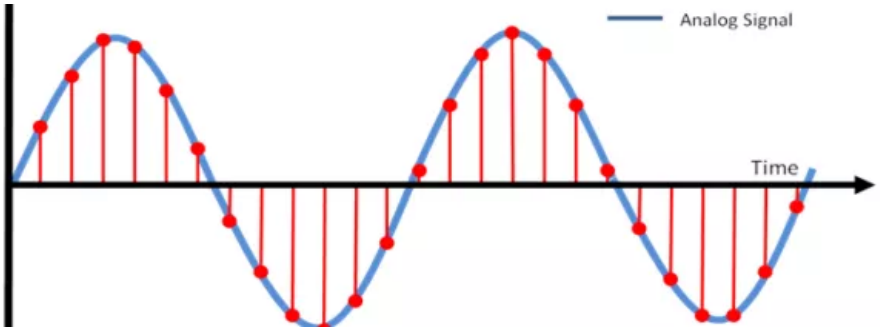
Posted by Max on Jun 20, 2011 in Atmel AVR, Microcontrollers | 279 comments



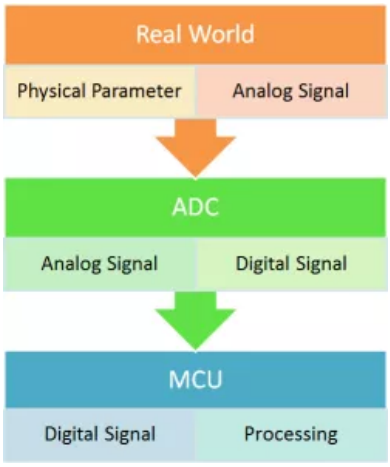
The ADC of the AVR

Analog to Digital Conversion

Most real world data is analog. Whether it be temperature, pressure, voltage, etc, their variation is always analog in nature. For example, the temperature inside a boiler is around 800°C. During its light-up, the temperature never approaches directly to 800°C. If the ambient temperature is 400°C, it will start increasing gradually to 450°C, 500°C and thus reaches 800°C over a period of time. This is an analog data.

Now, we must process the data that we have received. But analog signal

processing is quite inefficient in terms of accuracy, speed and desired output. Hence, we convert them to digital form using an Analog to Digital Converter (ADC).



Signal Acquisition Process

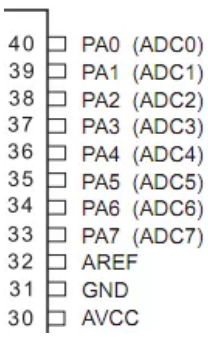
Signal Acquisition Process

In general, the signal (or data) acquisition process has 3 steps.

- In the **Real World**, a sensor senses any physical parameter and converts into an equivalent analog electrical signal.
- For efficient and ease of signal processing, this analog signal is converted into a digital signal using an **Analog to Digital Converter (ADC)**.
- This digital signal is then fed to the **Microcontroller (MCU)** and is processed accordingly.

Interfacing Sensors

In general, sensors provide with analog output, but a MCU is a digital one. Hence we need to use ADC. For simple circuits, comparator op-amps can be used. But even this won't be required if we use a MCU. We can straightaway use the inbuilt ADC of the MCU. In ATMEGA16/32, PORTA contains the ADC pins.



ADC Pins – ATMEGA16/32

The ADC of the AVR

The AVR features inbuilt ADC in almost all its MCU. In ATMEGA16/32, PORTA contains the ADC pins. Some other features of the ADC are as follows:

- **10-bit Resolution**
- **0.5 LSB Integral Non-linearity**
- **±2 LSB Absolute Accuracy**
- **13 - 260 µs Conversion Time**
- **Up to 15 kSPS at Maximum Resolution**
- **8 Multiplexed Single Ended Input Channels**
- **7 Differential Input Channels**
- **2 Differential Input Channels with Optional Gain of 10x and 200x**
- **Optional Left adjustment for ADC Result Readout**
- **0 - V_{CC} ADC Input Voltage Range**
- **Selectable 2.56V ADC Reference Voltage**
- **Free Running or Single Conversion Mode**
- **ADC Start Conversion by Auto Triggering on Interrupt Sources**
- **Interrupt on ADC Conversion Complete**
- **Sleep Mode Noise Canceler**

ADC Features – ATMEGA16/32

Right now, we are concerned about the **8 channel 10 bit resolution** feature.

TUE 06 without Microcontrollers
Posted by Max in Electronics

THU 16 LCD Interfacing with AVR
Posted by Max in Atmel AVR, Microcontrollers

FRI 10 I/O Port Operations in AVR
Posted by Max in Atmel AVR, Microcontrollers

Browse maxE by Categories

Browse maxE by Categories
Select Category

Subscribe to maxEmbedded

Email Address

Subscribe

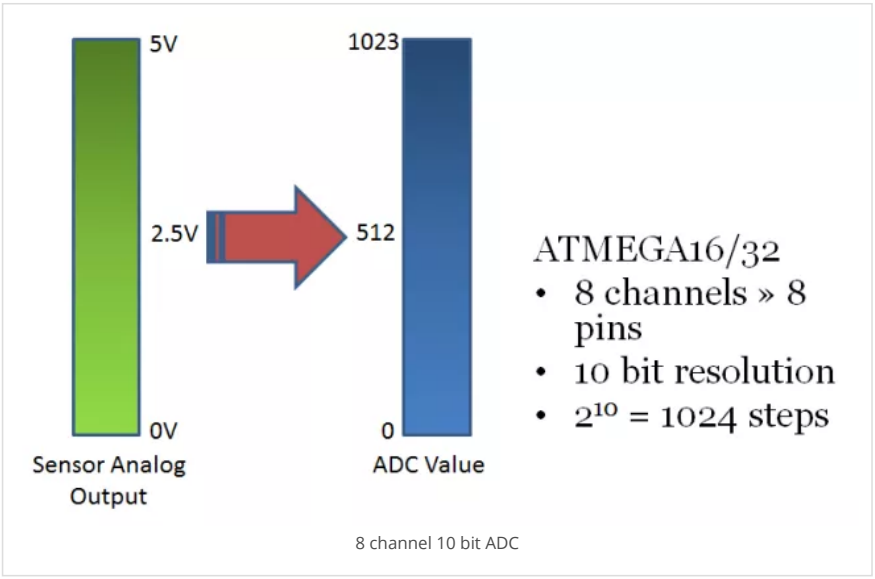
Like maxE on Facebook

MaxEmbedded
3,980 likes

Like Page Share

Sponsored Links

- **8 channel** implies that there are 8 ADC pins are multiplexed together. You can easily see that these pins are located across PORTA (PA0...PA7).
- **10 bit resolution** implies that there are 2^10 = 1024 steps (as described below).



8 channel 10 bit ADC

Suppose we use a 5V reference. In this case, any analog value in between 0 and 5V is converted into its equivalent ADC value as shown above. The 0-5V range is divided into 2^10 = 1024 steps. Thus, a 0V input will give an ADC output of 0, 5V input will give an ADC output of 1023, whereas a 2.5V input will give an ADC output of around 512. This is the basic concept of ADC.

To those whom it might concern, the type of ADC implemented inside the AVR MCU is of Successive Approximation type.

Apart from this, the other things that we need to know about the AVR ADC are:

- ADC Prescaler
- ADC Registers – ADMUX, ADCSRA, ADCH, ADCL and SFIOR

ADC Prescaler

The ADC of the AVR converts analog signal into digital signal at some regular interval. This interval is determined by the clock frequency. In general, the ADC operates within a frequency range of 50kHz to 200kHz. But the CPU clock frequency is much higher (in the order of MHz). So to achieve it, frequency division must take place. The prescaler acts as this division factor. It produces desired frequency from the external higher frequency. There are some predefined division factors – 2, 4, 8, 16, 32, 64, and 128. For example, a prescaler of 64 implies F_ADC = F_CPU/64. For F_CPU = 16MHz, F_ADC = 16M/64 = 250kHz.

Now, the major question is... which frequency to select? Out of the 50kHz-200kHz range of frequencies, which one do we need? Well, the answer lies in your need. **There is a trade-off between frequency and accuracy**. Greater the frequency, lesser the accuracy and vice-versa. So, if your application is not sophisticated and doesn't require much accuracy, you could go for higher frequencies.

ADC Registers

We will discuss the registers one by one.

ADMUX – ADC Multiplexer Selection Register

The ADMUX register is as follows.

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

ADMUX Register

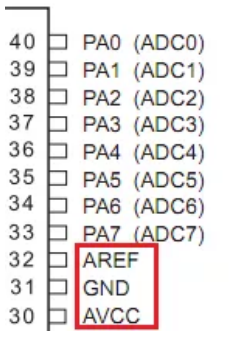
The bits that are highlighted are of interest to us. In any case, we will discuss all the bits one by one.

- **Bits 7:6 – REFS1:0 – Reference Selection Bits** – These bits are used to choose the reference voltage. The following combinations are used.

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

Reference Voltage Selection

The ADC needs a reference voltage to work upon. For this we have a three pins AREF, AVCC and GND. We can supply our own reference voltage across AREF and GND. For this, **choose the first option**. Apart from this case, you can either connect a capacitor across AREF pin and ground it to prevent from noise, or you may choose to leave it unconnected. If you want to use the VCC (+5V), **choose the second option**. Or else, **choose the last option** for internal Vref.



ADC Voltage Reference Pins

Let's choose the second option for Vcc = 5V.

- **Bit 5 – ADLAR – ADC Left Adjust Result** – Make it '1' to Left Adjust the ADC Result. We will discuss about this a bit later.

- **Bits 4:0 – MUX4:0 – Analog Channel and Gain Selection Bits** – There are 8 ADC channels (PA0...PA7). Which one do we choose? Choose any one! It these bits. ditions as follows. However, we are concerned only with the first 8 ions. Initially, all the bits are set to zero.

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0			
00001	ADC1			
00010	ADC2			
00011	ADC3	N/A		
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000		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	N/A	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
11101		ADC5	ADC2	1x
11110	1.22 V (V_{BG})	N/A		
11111	0 V (GND)			

Input Channel and Gain Selections

Thus, to initialize ADMUX, we write

```
ADMUX = (1<<REFS0);
```

ADCSRA – ADC Control and Status Register A

The ADCSRA register is as follows.

Bit	7	6	5	4	3	2	1	0	
	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	

ADCSRA Register

The bits that are highlighted are of interest to us. In any case, we will discuss all the bits one by one.

- **Bit 7 – ADEN – ADC Enable** – As the name says, it enables the ADC feature. Unless this is enabled, ADC operations cannot take place across

PORTA i.e. PORTA will behave as GPIO pins.

- **Bit 6 – ADSC – ADC Start Conversion** – Write this to '1' before starting any conversion. This 1 is written as long as the conversion is in progress, after which it returns to zero. Normally it takes 13 ADC clock pulses for this operation. But when you call it for the first time, it takes 25 as it performs the initialization together with it.

- **Bit 5 – ADATE – ADC Auto Trigger Enable** – Setting it to '1' enables auto-triggering of ADC. ADC is triggered automatically at every rising edge of clock pulse. View the SFIOR register for more details.

- **Bit 4 – ADIF – ADC Interrupt Flag** – Whenever a conversion is finished and the registers are updated, this bit is set to '1' automatically. Thus, this is used to check whether the conversion is complete or not.

- **Bit 3 – ADIE – ADC Interrupt Enable** – When this bit is set to '1', the ADC interrupt is enabled. This is used in the case of interrupt-driven ADC.

- **Bits 2:0 – ADPS2:0 – ADC Prescaler Select Bits** – The prescaler (division factor between XTAL frequency and the ADC clock frequency) is determined by selecting the proper combination from the following.

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

ADC Prescaler Selections

Assuming XTAL frequency of 16MHz and the frequency range of 50kHz-200kHz, we choose a prescaler of 128.

Thus, F_ADC = 16M/128 = 125kHz.

Thus, we initialize ADCSRA as follows.

```
ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
// prescaler = 128
```

ADCL and ADCH – ADC Data Registers

The result of the ADC conversion is stored here. Since the ADC has a resolution of 10 bits, it requires 10 bits to store the result. Hence one single 8 bit register is not sufficient. We need two registers – ADCL and ADCH (ADC Low byte and ADC High byte) as follows. The two can be called together as ADC.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	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 = 0

ADC Data Registers (ADLAR = 0)

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	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

ADC Data Registers (ADLAR = 1)

You can very well see the the effect of ADLAR bit (in ADMUX register). Upon setting ADLAR = 1, the conversion result is left adjusted.

SFIOR – Special Function I/O Register

In normal operation, we do not use this register. This register comes into play whenever ADATE (in ADCSRA) is set to '1'. The register goes like this.

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
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	

SFIOR Register

The bits highlighted in yellow will be discussed as they are related to ADATE. Other bits are reserved bits.

- **Bits 7:5 – ADC Auto Trigger Source** – Whenever ADATE is set to '1', these bits determine the trigger source for ADC conversion. There are 8 possible trigger sources.

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/Counter Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

ADC Auto Triggering Source Selections

> These options are will be discussed in the posts related to timers. Those who have prior knowledge of timers can use it. The rest can leave it for now, we won't be using this anyway.

ADC Initialization

The following code segment initializes the ADC.

```
void adc_init()
{
    // AREF = AVcc
    ADMUX = (1<<REFS0);

    // ADC Enable and prescaler of 128
    // 16000000/128 = 125000
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
}
```

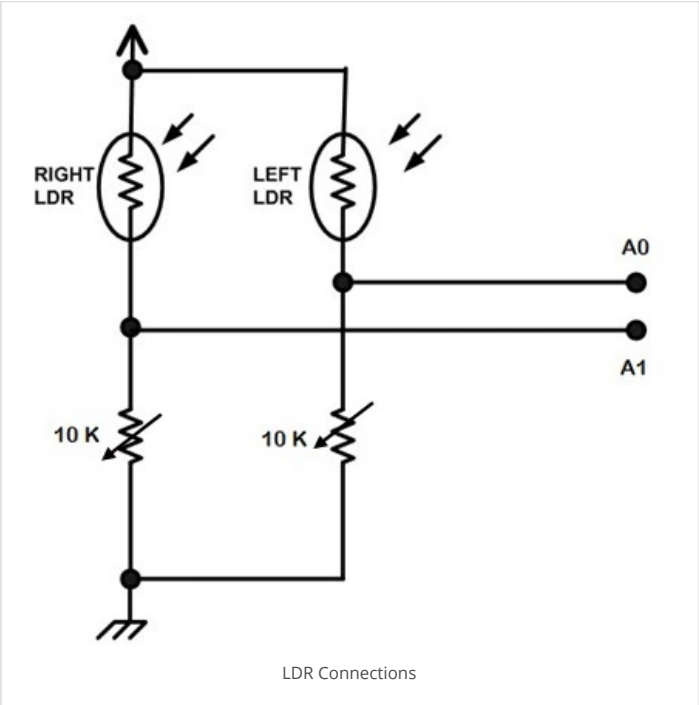
Reading ADC Value

The following code segment reads the value of the ADC. Always refer to the register description above for every line of code.

```
uint16_t adc_read(uint8_t ch)
{
  // select the corresponding channel 0~7
  // ANDing with ’7’ will always keep the value
  // of ‘ch’ between 0 and 7
  ch &= 0b00000111;  // AND operation with 7
  ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits b

  // start single convertion
  // write ’1’ to ADSC
  ADCSRA |= (1<<ADSC);

  // wait for conversion to complete
  // ADSC becomes ’0’ again
  // till then, run loop continuously
  while(ADCSRA & (1<<ADSC));

  return (ADC);
}
```

Physical Connections

Let’s connect two LDRs (Light Dependent Resistors) to pins PA0 and PA1 respectively. The connection is as follows. The function of potentiometers is explained in a later section, **Sensor Calibration**. You can scroll down to it. ;)



LDR Connections

Now suppose we want to display the corresponding ADC values in an LCD. So, we also need to connect an LCD to our MCU. Read this post to know about LCD interfacing.

Since it is an LDR, it senses the intensity of light and accordingly change its resistance. The resistance decreases exponentially as the light intensity increases. Suppose we also want to light up an LED whenever the light level decreases. So, we can connect the LED to any one of the GPIO pins, say PC0.

Note that since the ADC returns values in between 0 and 1023, for dark conditions, the value should be low (below 100 or 150) whereas for bright conditions, the value should be quite high (above 900).

Now let's write the complete code.

Example Code

To learn about LCD interfacing, view this post. You can type, compile and build it in AVR Studio 5. View this page to know how. To know about the I/O port operations in AVR, view this page.

```
+ expand source
```

Sensor Calibration

Calibration means linking your real world data with the virtual data. In the problem statement given earlier, I have mentioned that the LED should glow if the light intensity reduces. But *when* should it start to glow? The MCU/code doesn't know by itself. You get the readings from the sensor continuously in between 0 and 1023. So, the question is *how do we know that below 'such and such' level the LED should glow?*

This is achieved by calibration. You need to physically set this value. What you do is that you run the sensor for all the lighting conditions. You have the ADC values for all these levels. Now, you need to physically see and check the conditions yourself and then apply a threshold. Below this threshold,

the light intensity goes sufficiently down enough for the LED to glow.

The potentiometer connected in the circuit is also for the same reason. Now, by the basic knowledge of electronics, you could easily say that upon changing the pot value the ADC value changes. Thus, for various reasons (like poor lighting conditions, you are unable to distinguish between bright and dark conditions, etc), you can vary the pot to get desired results.

This is why I have given the two thresholds (RTHRES anf LTHRES) in the beginning of the code.

So, this is all with the ADC. I hope you enjoyed reading this. **Please post the comments below for any suggestion, doubt, clarification, etc.**

Max

Max is the founder and admin of maxEmbedded. He describes himself as an 'embedded electronics freak' and an Apple/Linux fan. He likes to experiment, learn and share new things in this field. Furthermore, he likes to write stuffs on his website for techie newbies and loves to teach others. In his spare time, you will find him with volunteering somewhere!

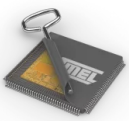
More Posts - Website

Follow Me:

Related

Sensor Fundamentals

In "Atmel AVR"



AVR Basics

In "Atmel AVR"

Using an Analog Accelerometer

In "Atmel AVR"

279 Comments

← Older Comments

zml October 21, 2014

Nice tutorial! Can you please share your documentation resources? When searching online, I keep finding tutorials referring to ADCW registers and so on but I was not able to find a good reference guide with all the functions/macros available for AVR C programming. Thank you and good work!

REPLY

Max November 28, 2014

Hello zml, most of my information comes from the AVR datasheet. Please read though it for detailed information. Thanks!

REPLY

Sarang S November 15, 2014

Hi,
I am using the ATmega164A. In that I am using PINA for my ADC input, So my question is can I use the other pins for GPIO?

REPLY

Sarang S November 15, 2014

PINA1*. So I want to use other pins of PORTA as GPIO. (Sorry was not clear in my post)

REPLY

Max November 28, 2014

Yes you can! Go for it!

REPLY

ayyappan November 19, 2014

ur website was very much useful for me and it is in simple language...
thanks for giving this website

REPLY

Vishwam January 2, 2015

Thanks for the support!

REPLY

saish December 18, 2014

sir
can we take input from adc 0 of atmega32 if we write ADMUX = 0X20;
PLZZ REPLY ASAP.

REPLY

Max December 19, 2014

I don't understand.. is this a question?! Read through the description of the ADMUX register. Then convert 0x20 to binary and notice the registers that are set. You'll find the answer on your own.

REPLY

Ridhi December 29, 2014

Sir,
do we need to reset ADIF before every coversion?
or do we require to manually set the ADIF after coversion but during data storage in registers ..like mentioning it after while command in adc read program written above?
plz tell its significance

REPLY

Max December 29, 2014

Hello Ridhi,
So there are two ways to read the ADC value.

METHOD ONE
1. Set ADSC.
2. Keep monitoring ADIF.
3. If ADIF is set, then read ADC value.
4. Set ADIF to one to reset the flag.

METHOD TWO
1. Set ADSC.
2. Keep monitoring ADSC.
3. If ADSC is reset, read the ADC value.

I have used the second method in my examples. In the second method, you don't give a shit about ADIF. Choose any method, both will work.

REPLY

Srikar December 29, 2014

If I want to use more than one ADC pin at the same time, how do I write 'ch' as in the adc_read?

REPLY

Srikar December 30, 2014

In the above program, you have written ch as ADC 7.
then you read the values from PA0 and PA1. I did not get this, ADC 7 means PA7 right?

REPLY

Max December 31, 2014

`ch = 0x00000111` is not the same as `ch &= 0x00000111`
Read this for more information.

REPLY

Max December 31, 2014

You'll have to do it one by one. You can't do them both together.

REPLY

srikar January 13, 2015

Okay.
But in the above full code, PA0 and PA1 are read. How can we read both in this case, when we can initialize only any one channel at a time?

REPLY

Max January 16, 2015

Try single stepping through the code, and you'll understand. Initialize to PA0, read from PA0, then initialize to PA1, read from PA1. Repeat if necessary.

REPLY

srikar January 18, 2015

Thank you. :)

Ridma January 6, 2015

I admire your effort and thanx a lot .its extreamly helpful for us newbieeeess..... keep it up..

REPLY

Prashant Agarwal January 7, 2015

Thanks & Keep reading ;)

REPLY

Eshwar February 15, 2015

sir, i have one doubt my sensor output value is 1.2v means what is the equivalent value for micro controller value(digital value).. what is the conversion steps please explain briefly sir....

REPLY

Max March 20, 2015

Good question. Let's assume that your range is 0-5v. So a value of 1.2v corresponds to 1.2/5*1024 = 245 (decimal) = 0011110101 (binary) assuming 10 bit ADC. This binary result is stored in the ADC register. Makes sense?

REPLY

Pier February 16, 2015

```
// now display on lcd
itoa(adc_result0, int_buffer, 10);
lcd_gotoxy(12,0);
lcd_puts(int_buffer);

itoa(adc_result0, int_buffer, 10);
lcd_gotoxy(12,1);
lcd_puts(int_buffer);
_delay_ms(50);
```

...too much adc_result0 in this section..! :-))
Regards
Pier

REPLY

Max March 20, 2015

Haha, thanks for that Pier! Now edited. :)

REPLY

Sb February 17, 2015

Confused..
ADMUX=(ADMUX&0xF8)|ch

REPLY

Vishwam March 6, 2015

Hi Sb

It is bit masking statement in which we are 'bit-wise ANDing' the contents of ADMUX register with 0xF8 and result is being 'Bit-Wise ORed' with the contents of ch.

Hope this helps!

REPLY

chet March 12, 2015

hai,
I am connecting 2 ldrs to two adc pins,and want to compare the output power of two ldrs.so how should i go with it?
as far as i know the ports read voltage/current of the ldrs.so how do I make it read the resistance and estimate their power?

REPLY

Max March 20, 2015

Check this out: http://www.atmel.com/images/doc42039.pdf

REPLY

nankuniyil March 21, 2015

I'm trying to get successive samples of an analog voltage signal. I'm using atmega16 with 4MHz crystal oscillator.
1) Do I need to use an additional adc for my purpose?
2) does the delay between two successive sample depend upon any other factors other than adc conversion time in atmega?
NB: Actually this is for a power system over voltage relay

Thanks in advance

REPLY

Vishwam May 1, 2015

Hi nankuniyil!

1) No you don't. The ATmega16 internal ADC Unit is enough.
2) No, it depends on how much delay you introduce, between sampling of two successive inputs, or the sampling frequency in free running mode.

Hope this helps!

REPLY

Robin March 30, 2015

Very helpful demonstration .. also i have one additional question how i can map a higher resolution(e.g 10 bit) adc value to lower resolution(e.g 6 bit)adc value

REPLY

Max April 3, 2015

You basically truncate the lower significant bits.

REPLY

m. hussain April 25, 2015

sir i have to make multimeter assigned by teacher that would measure current and voltage only using atmega16,i have no circuit diagram and code also,,can you provide me circuit diagram and programming code???i relly need this... i'ii be grateful to you..

REPLY

Vishwam May 1, 2015

Sorry, but we do not do people's assignments for them!

REPLY

alaa April 24, 2015

plz sir , i wanna monitor 4 analog signal and sending them to lcd i found 3 channels is the like the four channel (the last one).when i was trying to monitor one and deactivate the others each on was working.
i hope i hear from you soon.

REPLY

alaa April 24, 2015

thanks sir ,
i checked the code and i found i didn't clear the bits
ADMUX|=ADCport;
instead of
ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing

everything is working perfectly .

REPLY

Vishwam May 1, 2015

Great you could make it work!

REPLY

Vishwam May 1, 2015

Hi Alaa

We couldn't really understand your problem. Could you please elaborate? Thanks!

REPLY

vstrulev April 29, 2015

Thank you very much for your clear explanations of the avr adc. Very helpful and useful. I've spent lots of time reading datasheet and lots of other sources in order to get good understanding of the topic, but your explanation is just really nice, short and easy to understand. I really appreciate your work and time spent for educating us

REPLY

Vishwam May 1, 2015

Hi vstrulev

It's great to hear that we could be of help to you! Keep reading and sharing the knowledge! Cheers!

REPLY

Ahmad August 22, 2015

Hi MAX
Very useful
Thanks.

REPLY

← Older Comments

Trackbacks/Pingbacks

AVR Tutorials :: [TUT] Using AVR ADC | LED World - [...] Here is the link to the tutorial. http://maxembedded.wordpress.com/2011/06/20/the-adc-of-the-avr/ [...]
Sensor Interfacing and ADC | roboVITics - [...] Resources: http://maxembedded.wordpress.com/2011/06/18/sensor-fundamentals/ http://maxembedded.wordpress.com/2011/06/20/the-adc-of-the-avr/ [...]
ADC and Sensor Interfacing | roboVITics - [...] http://maxembedded.com/2011/06/20/the-adc-of-the-avr/ [...]
ADC Pada AVR ATMEGA « Nico's Blog - [...] Sumber : http://maxembedded.com/2011/06/20/the-adc-of-the-avr/ [...]
Using an Analog Accelerometer | maxEmbedded - [...] So, if you are not clear with the concepts of ADC, take a detour and go through Mayank's awesome ...
How to build an IR Sensor | maxEmbedded - [...] To learn how to use AVR microcontollers to convert and process analog data, please refer to this post on ...
The USART of the AVR | maxEmbedded - [...] of two 8-bit registers – UBRRH (high) and UBRRL (low). This is similar to the 16-bit ADC register (ADCH ...
How to build an IR Sensor | Clubotics | DAVIET Robotics Club - [...] To learn how to use AVR microcontollers to convert and process analog data, please refer to this postThe ADC of ...
[AVR] Simple C code for led blinking problem - [...] http://www.avrbeginners.net/architec...c_example.html http://maxembedded.com/2011/06/20/the-adc-of-the-avr/ ...

Leave a Reply

Enter your comment here...

Copyright	Tags	Top Posts & Pages
	accelerometer adapter adc assembly avr	The ADC of the AVR

adc avr basics avr eeprom avr interrupts avr serial avr timers basics bootloader code gallery code optimization color spaces counters cv basics filter circuit i2c ir sensor matlab msp430 basics opencv pcb design power supply printed circuit boards pwm raspberry pi rf communication rf module robotics robot locomotion rs232 sbc basics sensors Serial spi ssd timers toolchain transformer uart usart voltage regulator wireless

AVR Timers - TIMER0

The USART of the AVR

How to build your own Power Supply

AVR Timers - CTC Mode

How to build an IR Sensor

I/O Port Operations in AVR

maxEmbedded is a free and open source platform to share knowledge and learn about the concepts of robotics, embedded systems and computer vision. People from around the world who are enthusiastic about these topics and willing to support the open source community are invited to share their information, knowledge and expertise by means of written tutorials and videos on the website.