

230+ Circuits to Learn and Build electronics - [Browse Circuits Library](#)

» Gadgetronicx > Microcontroller > ATtiny85 > ATtiny85 tutorials > ATtiny85 ADC tutorial with interrupts

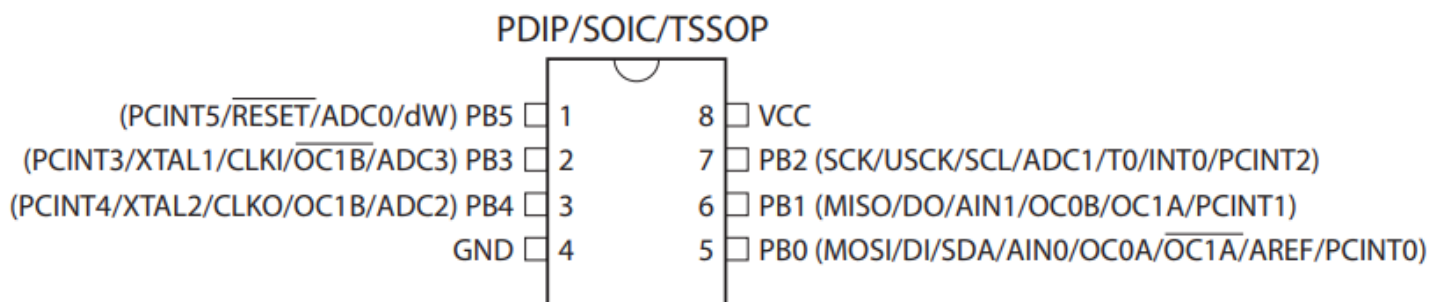
ATtiny85 ADC tutorial with interrupts

I Gadgetronicx Team | December 3, 2020 | v 0 Comments

m ATtiny85, ATtiny85 tutorials, Microcontroller

Following the series of [ATtiny85 tutorials](#) this article explains how to configure ADC in an ATtiny85 microcontroller. Analog to digital converter aka ADC is quite an important feature in microcontrollers where it will translate the Analog signals to digital data. This is most commonly used with sensors to measure parameters from operating environment and enable microcontroller to interpret the environmental conditions.

Analog to Digital Converter (ADC) in ATtiny85:



There are total of 4 ADC channels present in ATtiny85. These channels are interface to pins PB2, PB3, PB4, PB5 in port B. Also ATtiny85 ADC is of 10 bit resolution. Meaning the input Analog signal can be translated to 10 bit data. The maximum analog input voltage to this microcontroller can range from 0V to 5V.

Let's look into the configuration of this ADC in ATtiny85 controller.

ADMUX – ADC Multiplexer Selection Register:

Bit	7	6	5	4	3	2	1	0	
0x07	REFS1	REFS0	ADLAR	REFS2	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 should be used to configure ADC peripheral in ATtiny85. REFS0, REFS1, REFS2 bits are used to select reference voltage used with ADC. We are using Vcc as reference voltage hence used the values 0,0,0.

REFS2	REFS1	REFS0	Voltage Reference (V_{REF}) Selection
X	0	0	V_{CC} used as Voltage Reference, disconnected from PB0 (AREF).
X	0	1	External Voltage Reference at PB0 (AREF) pin, Internal Voltage Reference turned off.
0	1	0	Internal 1.1V Voltage Reference.
0	1	1	Reserved
1	1	0	Internal 2.56V Voltage Reference without external bypass capacitor, disconnected from PB0 (AREF) ⁽¹⁾ .
1	1	1	Internal 2.56V Voltage Reference with external bypass capacitor at PB0 (AREF) pin ⁽¹⁾ .

MUX0, MUX1, MUX2, MUX3 bits are used to select the ADC channel which the user intended to use. We have selected ADC1 to use for our tutorial purposes hence we write 0001 to MUX[3:0].

Circuits Library - 220+ practical circuits

MUX[3:0]	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
0000	ADC0 (PB5)	N/A		
0001	ADC1 (PB2)			
0010	ADC2 (PB4)			
0011	ADC3 (PB3)			
0100	N/A	ADC2 (PB4)	ADC2 (PB4)	1x
0101 ⁽¹⁾		ADC2 (PB4)	ADC2 (PB4)	20x
0110		ADC2 (PB4)	ADC3 (PB3)	1x
0111		ADC2 (PB4)	ADC3 (PB3)	20x
1000		ADC0 (PB5)	ADC0 (PB5)	1x
1001		ADC0 (PB5)	ADC0 (PB5)	20x
1010		ADC0 (PB5)	ADC1 (PB2)	1x
1011		ADC0 (PB5)	ADC1 (PB2)	20x
1100 ⁽²⁾	V_{BG}	N/A		
1101	GND			
1110	N/A			
1111 ⁽³⁾	ADC4			

Finally the ADLAR is used to define how the resultant translated digital data should be stored in the register. There are two data registers associated with ADC, these are ADCL and ADCH. The final result after ADC conversion is stored in these two data registers. Each register has a maximum of 8 bits. Since our ADC resolution is 10 bits, the data will be split into two and stored into ADCL and ADCH register.

If ADLAR is set to 0, 8 bits of converted data will be stored in ADCL and 2 bits in ADCH. This is known as right shifted result.

ADLAR = 0

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	

If ADLAR is set to 1, 8 bits of converted data will be stored in ADCH and 2 bits in ADCL. This is known as left shifted result.

ADLAR = 1

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	

ADCSRA – ADC Control and Status Register

Bit	7	6	5	4	3	2	1	0	
0x06	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	

ADEN is ADC enable bit which has to be set to 1 to enable ADC feature.

ADSC: ADC Start Conversion bit starts the conversion when set to 1. After ADC conversion is performed this bit will be set back to zero by hardware.

ADATE: Enabling this bit enables the auto triggering of ADC conversion. This conversion occurs every positive edge of trigger (discussed below) and continues to repeat after each conversion as long as the trigger signal is intact. The converted values from ADCH and ADCL has to be read before the next conversion replaces the current result.

ADIF: ADC Interrupt Flag, this will be set to 1 by hardware when ADC conversion completes. The ADC interrupt will be executed when ADIF goes high if you choose to enable the ADC interrupt using ADIE bit. ADIF bit will be cleared by the hardware when that particular service routine is executing.

ADIE: ADC Interrupt Enable – When this bit is set 1 and global interrupt enable bit I in SREG register is set then ADC conversion triggers interrupt.

Bits ADPS2, ADPS1 and ADPS0 are the prescaler selection bits to divide the clock frequency at which ADC operates. The below table shows the values in these bits and their respective divisible factor. Setting these bits to 111 will choose the divisible factor of 128 which sets the ADC to operate at lowest operating frequency.

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

ADCSRB register:

Bit	7	6	5	4	3	2	1	0	
0x03	BIN	ACME	IPR	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R/W	R/W	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADTS0, ADTS1 and ADTS2 bits in ADCSRB register decides when ADC conversion will be triggered. Below table shows the various trigger sources for the ADC conversion. The ADSC in ADCSRA bit has to high in order for these trigger sources to take effect.

Table 17-6. 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 A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter0 Compare Match B
1	1	0	Pin Change Interrupt Request

When you select the free running mode. Conversion repeats over and over again until it is stopped. In this free running mode the interrupt will not be triggered even when the ADC interrupt flag ADIF flag is set.

Other trigger sources doesn't fall under above category. For example when these bits are written 100, then ADC will be triggered when [Timer/ Counter0 overflows](#). This will set the ADIF flag and ISR will be executed if ADC interrupt is configured.

The ADC conversion result for the single ended conversion can be calculated with the formula

$$ADC = (V_{in} \times 1024) / V_{ref}$$

So, for this tutorial, we have chose V_{ref} is 5V. Let's say a sensor that we use with outputs a voltage of 2.46V, then ADC converted value will be

$$ADC = V_{in} \times 1024 / V_{ref}$$

$$= 2.46 \times 1024 / 5 = 2519.04 / 5$$

$$= 503.80$$

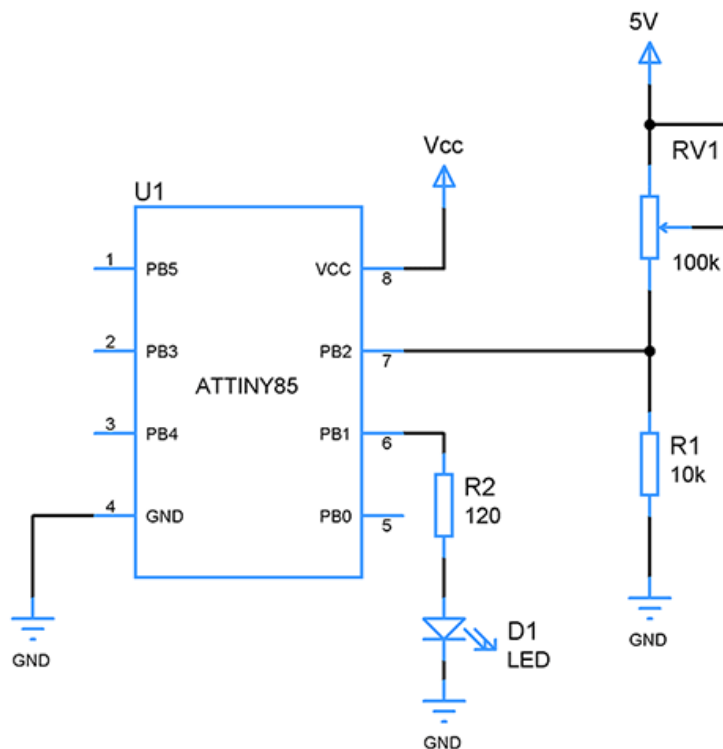
This value will be stored in binary form in both ADCH and ADCL register when 10 bit resolution is selected.

SREG- AVR Status Register

Bit	7	6	5	4	3	2	1	0	
0x3F	I	T	H	S	V	N	Z	C	SREG
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	

As stated earlier interrupts can be used with ADC. This will be triggered at the end of each ADC conversion. In order to use this conversion, Bit I of SREG register which is global interrupt enable bit should be written as 1. This should be followed by enabling ADIE bit ADCSRA register.

Circuit diagram:



Sample Code for ADC in free running mode :

Code to read Analog signal from potentiometer connected to PB2 and light up or toggle LED connected to PB1 based on the analog voltage.

```
#include<avr/io.h>
#define F_CPU 16500000UL

void adc_setup()
{
    DDRB|=(1<<PB1);    //PB1 as output to activate LED
    ADCSRA|=(1<<ADEN);  //Enable ADC module
    ADMUX=0x01; // configuring PB2 to take input
    ADCSRB=0x00;        //Configuring free running mode
    ADCSRA|=(1<<ADSC)|(1<<ADATE); //Start ADC conversion and enabling Auto trigger
}

int main()
{
    adc_setup();
    while(1)
    {
        int adc_l=ADCL; //value of Input Voltage in lower register
        int adc_val=(ADCH<<8)|adc_l; //Reading ADCH and combining the data
        if (adc_val>=510&&adc_val<=520) //Lights up the LED under certain voltage 2.25 to 2.3 level
        {
            PORTB|=(1<<PB1); //LED remain ON
        }

        else    //blink condition
        {
            PORTB &=~(1<<PB1); //LED toggle
        }
        ADCSRA|=(1<<ADIF);
    }
}
```

Sample Code for ADC with Timer0 overflow trigger:

Code to read Analog signal from potentiometer connected to PB2 and light up LED in PB1 by means of Timer0 value overflow as trigger.

```
#include<avr/io.h>
```

```

void adc_setup()
{
  DDRB|=(1<<PB1);
  TCCR0A=0x00;          //Timer0 normal mode
  TCCR0B=0x00;
  TCCR0B |= (1<<CS00)|(1<<CS02); //prescaling with 1024
  TCNT0=0;
  ADCSRA|=(1<<ADEN);    //Enable ADC module
  ADMUX=0x01; // configuring PB2 to take input
  ADCSRB|=1<<ADTS2; //Timer / Counter 0 overflow triggers the ADC to perform conversion
  ADCSRA|=(1<<ADSC)|(1<<ADATE); //Enabling start of conversion and Auto trigger
}

int main()
{
  adc_setup();
  while(1)
  {
    int adc_l=ADCL; //value of Input Voltage
    int adc_val=(ADCH<<8)|adc_l; //Storing entire ADC value in a variable
    if (adc_val>=510&&adc_val<=520) //Trigger the ADC when the voltage value falls within 2.25 to 2.3v
    {
      PORTB|=(1<<PB1);
    }

    else //blink condition
    {
      PORTB &=~(1<<PB1); //Toggle LED state
    }
    TIFR|=(1<<TOV0); //Clearing overflow flag
  }
}

```

Try this out:


- Configure two ADC channels ADC1 & ADC2 to run in free running mode, read the ADC values and update it in a variable every 10 seconds.

Other ATtiny85 tutorials:

- [ATtiny85 Analog comparator tutorial](#)


- [ATtiny85 PWM: Fast and Phase correct modes](#)

Related Posts:




Learn
ATtiny85

Learn ATtiny85
microcontroller in 10 days



ATtiny85 Tutorial
Analog comparator


ATtiny85 Analog
comparator tutorial



Introduction to
ATtiny85 Microcontroller


Zoom session
For 100 participants
16th Jan, Saturday - 4:30 PM IST / 11 AM GMT

Upcoming live sessions




ATtiny85 Tutorial
External and Pin change interrupt

ATtiny85 external and pin
change interrupt tutorial



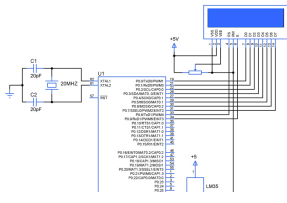
ATtiny85 Tutorial
GPIO as output

ATtiny85 Microcontroller
tutorial: GPIO as output

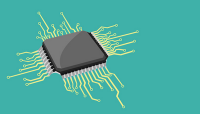


ATtiny85 Tutorial
Low power Sleep modes

ATtiny85 sleep modes
tutorial



ADC Programming in ARM
Microcontrollers



Most popular
microcontrollers among
engineers

[← Previous post](#)
[Next post →](#)