ATmega ADC Registers

24.9.5 **DIDR0** – Digital Input Disable Register 0

<table>
<thead>
<tr>
<th>bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0x7B)</td>
<td>-</td>
<td>-</td>
<td>ADC5D</td>
<td>ADC4D</td>
<td>ADC3D</td>
<td>ADC2D</td>
<td>ADC1D</td>
<td>ADC0D</td>
</tr>
<tr>
<td>Read/Write:</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**bits 7-6**: Reserved
These bits are reserved for future use. To ensure compatibility with future devices, these bits must be written to zero when DIDR0 is written.

**bits 5-0**: ADC5...0 Digital Input Disable
When written to 1, these bits disable the associated digital input buffer for a pin being used as an analog input. This is not necessary, but it will **save power**. The corresponding PIN Register bit will always read as zero when this bit is set.

Note that ADC pins ADC7 and ADC6 do not have digital input buffers, and therefore do not require Digital Input Disable bits.
Figure 2-1. Block Diagram
1. Pin Configurations

Figure 1-1. Pinout ATmega48A/PA/88A/PA/168A/PA/328/P

32 TQFP Top View
As we turn the potentiometer, it moves the wiper. As the wiper moves up it decreases the resistance between the wiper and the upper lead and at the same time it increases the resistance between the wiper and the lower lead. Since there is a fixed current flowing through the potentiometer (5V / 10 MΩ). The voltage at the center lead (wiper) will increase as the wiper moves up, and conversely, decrease as the wiper moves down. Thus, measuring the voltage at the center lead with the A/D converter will tell us its position.
Analog conversion example code:

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

int main(void)
{
    uint16_t adcValue = 0;
    // Setup the Board LED (on digital pin 13, PB5) as Output
    DDRB = (1 << DDB5);

    // Turn on the ADC (ADEN), by default it is turned off
    // and make the prescaler 128 (7 in ADPS field)
    ADCSRA = (1 << ADEN) | (7 << ADPS0);

    // Connect the ADC to analog pin 0 (ADC0) with a 0 in the MUX field
    // and use the AVCC (5V) Reference with a 1 in the REFS field.
    ADMUX = (0 << MUX0) | (1 << REFS0);

    while (1)
    {
        // Manually start a conversion
        ADCSRA |= (1 << ADSC); // Also note that this RMW clears a set ADIF
        while (ADCSRA & (1 << ADSC)); // Wait for the conversion to complete
        adcValue = ADC; // Read the converted value (0 -- 1023)
        if (adcValue < 512)
        {
            PORTB &= ~(1 << DDB5); // Turn off the LED
        }
        else
        {
            PORTB |= (1 << DDB5); // else, turn on the LED
        }
    }
}
```
A Second ADC Example (main.c):

```c
#include <stdio.h>
#include <avr/io.h>
#include <util/delay.h>
#include "uartDriver.h"
#include "ADCDriver.h"

//Max setting on POT is 10000 Ohms
#define POTVALUE 10000.0

int main(void)
{
    uint16_t potInt, adcVal;

    //Setup Uart and connect it to stdin and stdout
    FILE uart_output = FDEV_SETUP_STREAM(uart_putchar, NULL, _FDEV_SETUP_WRITE);
    FILE uart_input = FDEV_SETUP_STREAM(NULL, uart_getchar, _FDEV_SETUP_READ);
    uart_init();
    stdout = &uart_output;
    stdin = &uart_input;

    initADC(); //Turn on ADC

    while (1)
    {
        adcVal = ReadADC(0); //Read ADC channel 0
        potInt = round(adcVal * POTVALUE / 1023); //Scale and round for output

        printf("Potentiometer value = %u Ohm. ADCval = %u \n", potInt, adcVal);
        _delay_ms(1000);
    }
}
```

Note: the basic compiler configuration has difficulty printing floats
A Second ADC Example (ADCdriver.c):

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

//ADCdriver.h contains function prototypes
#include "ADCDriver.h"

void initADC()
{
    //Select Vref=AVcc
    ADMUX = (1 << REFS0);
    //Set prescaler to 128 and enable ADC
    ADCSRA = (7 << ADPS0) | (1 << ADEN);
}

uint16_t ReadADC(uint8_t ADCchannel)
{
    //Select ADC channel with safety mask
    ADMUX = (ADMUX & 0xF0) | (ADCchannel & 0x0F);
    //Start a conversion
    ADCSRA |= (1 << ADSC); // Note: also clears ADIF
    //Wait until ADC conversion is complete
    while(ADCSRA & (1 << ADSC));
    return ADC;
}
```