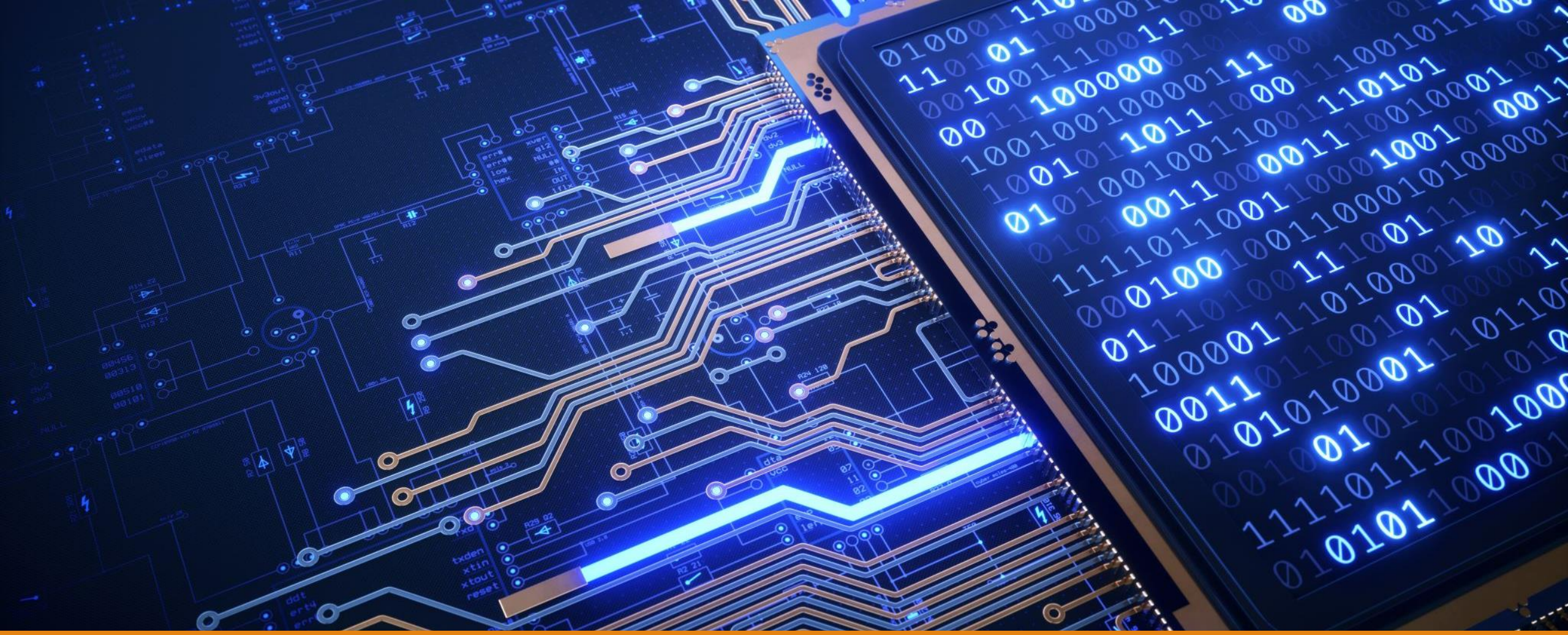


Embedded Systems

Dr. Abdelhay Ali

Lecture 3

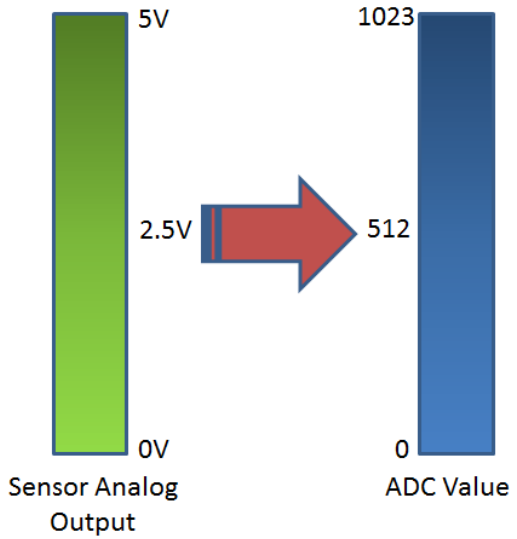
Analog to Digital Converters

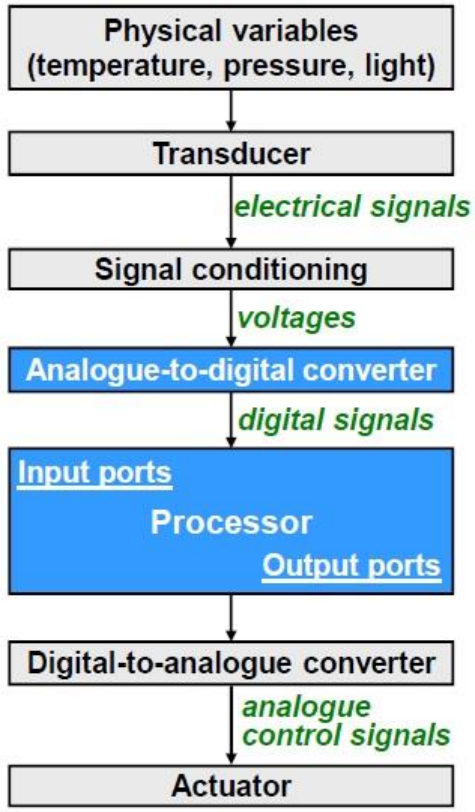
Outlines

1. Introduction
2. ATmega16 ADC
3. ADC Register
4. ADC Example
5. ADC Arduino

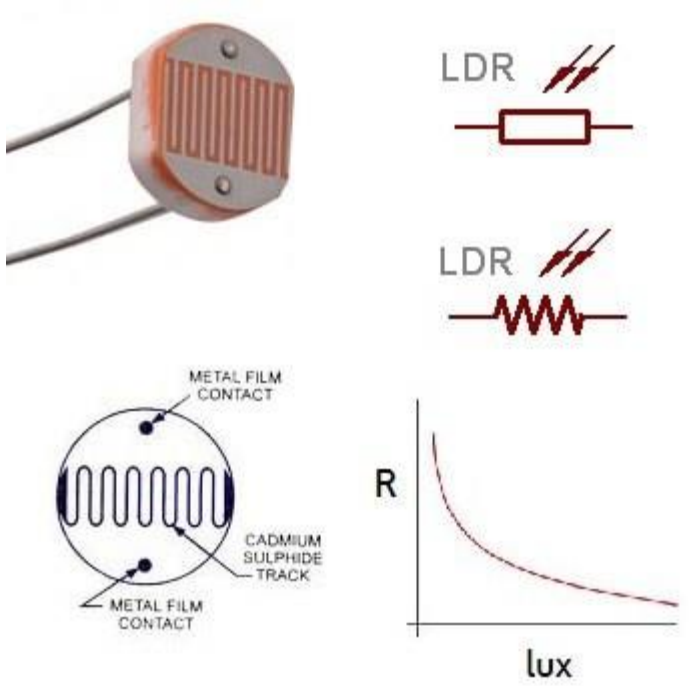
Introduction

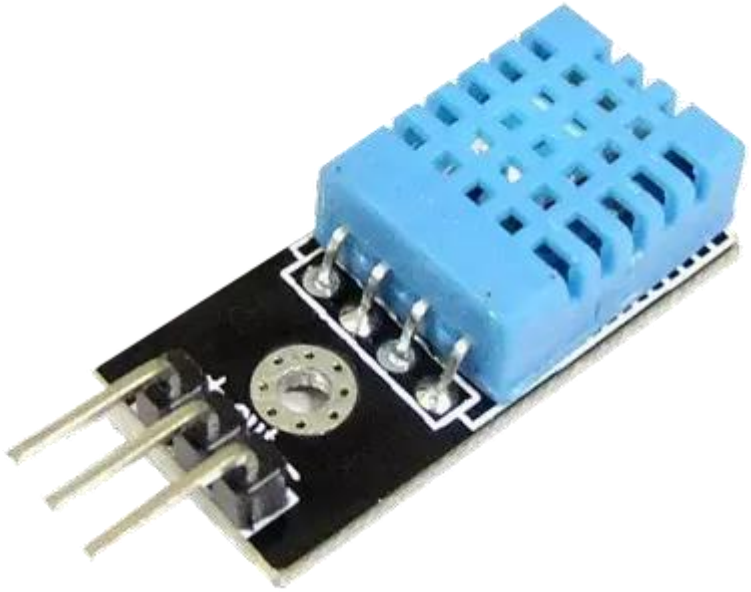
ADC (Analog to Digital converter) is the most widely used device in embedded systems which is designed especially for data acquisition. In the AVR ATmega series normally 10-bit ADC is inbuilt in the controller.





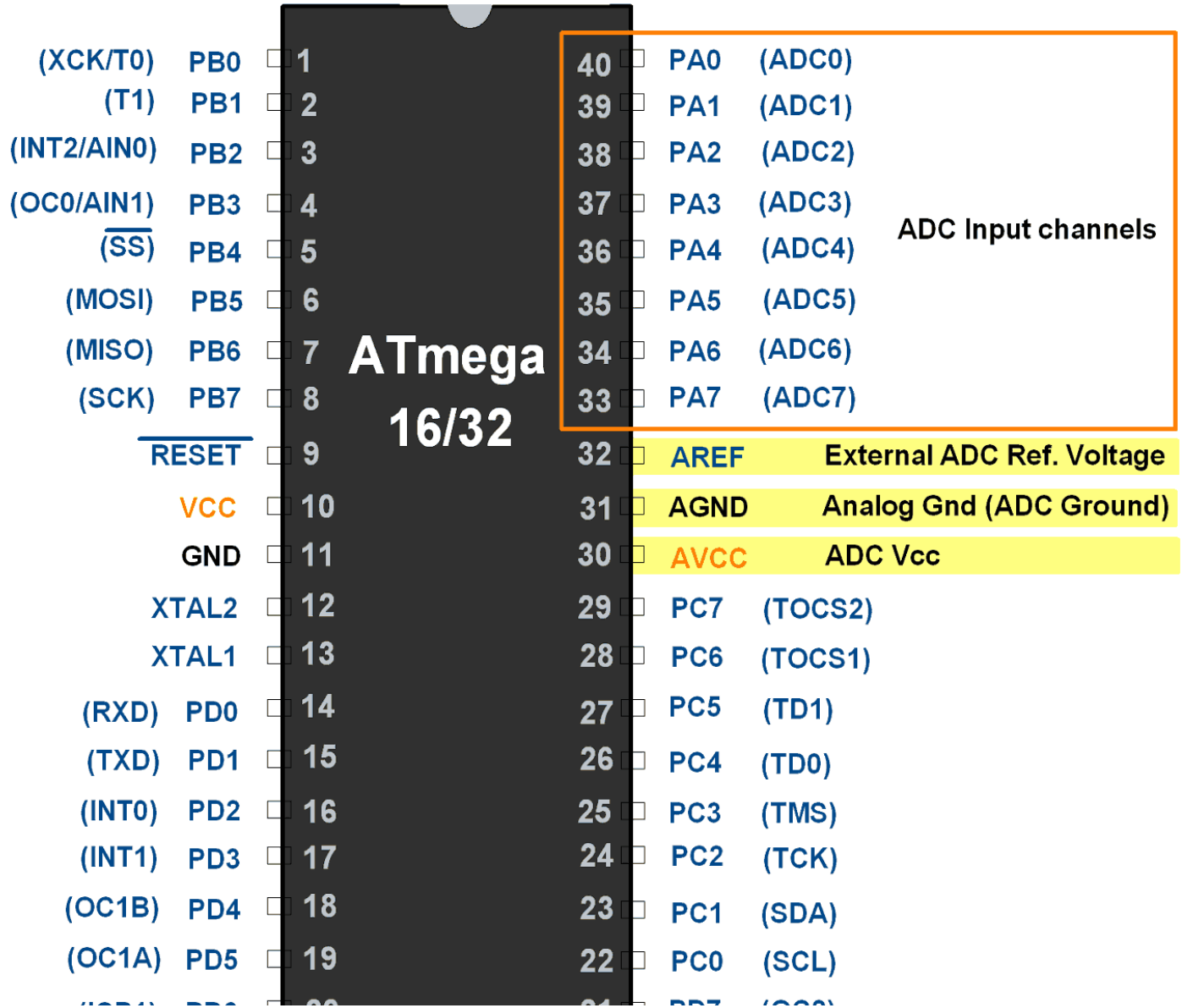
Introduction





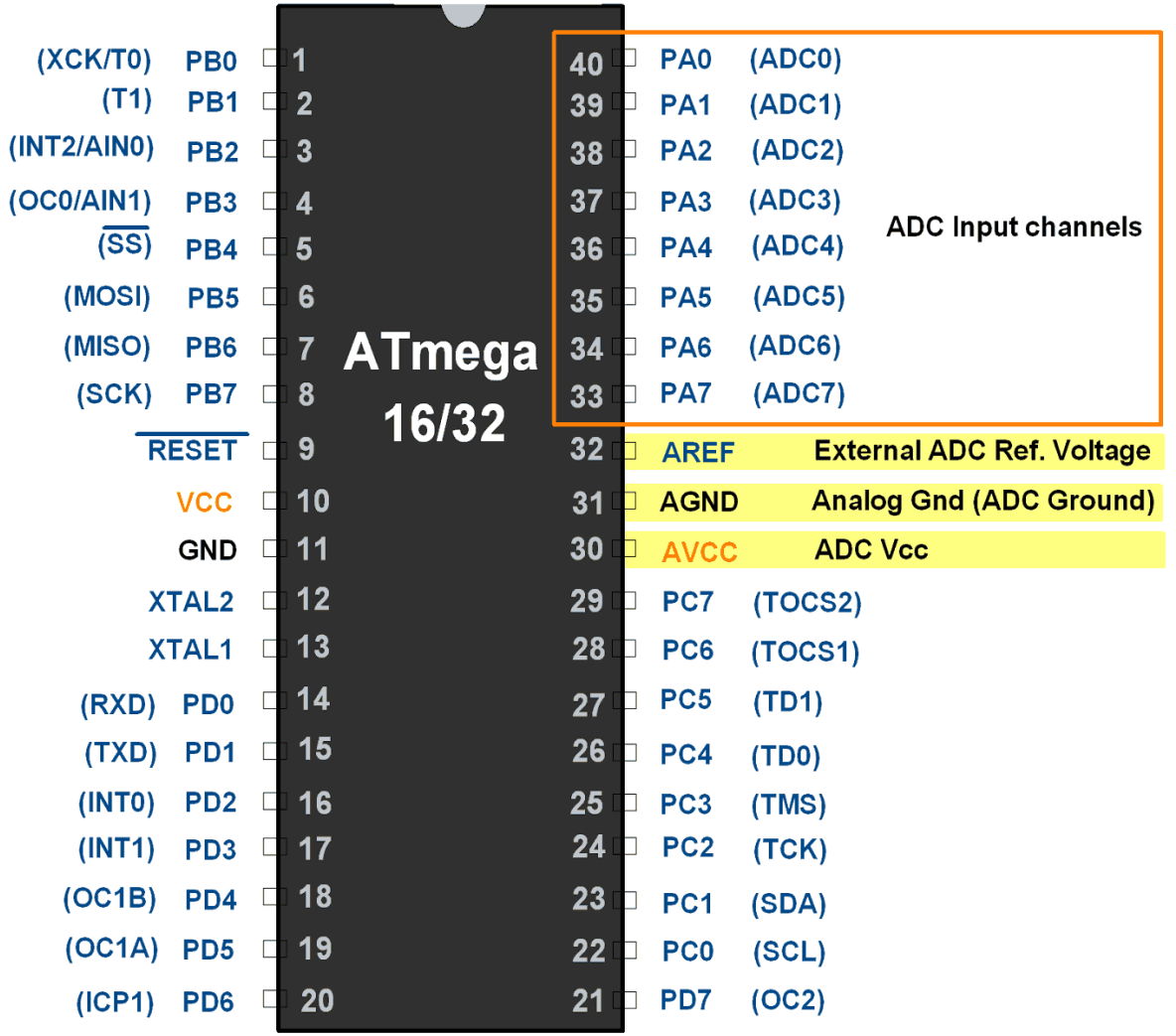
ATmega16 ADC

- ATmega16 supports eight ADC channels, which means we can connect eight analog inputs at a time.
- ADC channel 0 to channel 7 are present on PORTA. i.e. Pin no.33 to 40.



ATmega16 ADC

- The controller has 10 bit ADC, which means we will get digital output 0 to 1023.
  - i.e. When the input is 0V, the digital output will be 0V & when input is 5V (and Vref=5V), we will get the highest digital output corresponding to 1023 steps, which is 5V.
- So controller ADC has 1023 steps and
  - Step size with Vref=5V : 5/1023 = 4.88 mV.
  - Step size with Vref=2.56 : 2.56/1023 = 2.5 mV.
- So Digital data output will be D*out* = Vin / step size.



ATmega16 ADC

- It is 10-bit ADC
- Converted output binary data is held in two special functions 8-bit register ADCL (result Low) and ADCH (result in High).
- ADC gives 10-bit output, so (ADCH: ADCL) only 10-bits are useful out of 16-bits.
- We have options to use this 10-bits as upper bits or lower bits.
- We also have three options for Vref.
  1. AVcc (analog Vcc), 2. Internal 2.56 v, 3. External *Aref*. Pin.
- The total conversion time depends on crystal frequency and ADPS0: 2 (frequency devisor)
- If you decided to use AVcc or Vref pin as ADC voltage reference, you can make it more stable and increase the precision of ADC **by connecting a capacitor between that pin and GND**.

ADC Register

In AVR ADC, we need to understand four main register -

1. **ADCH:** Holds digital converted data higher byte
2. **ADCL:** Holds digital converted data lower byte
3. **ADMUX:** ADC Multiplexer selection register
4. **ADCSRA:** ADC Control and status register

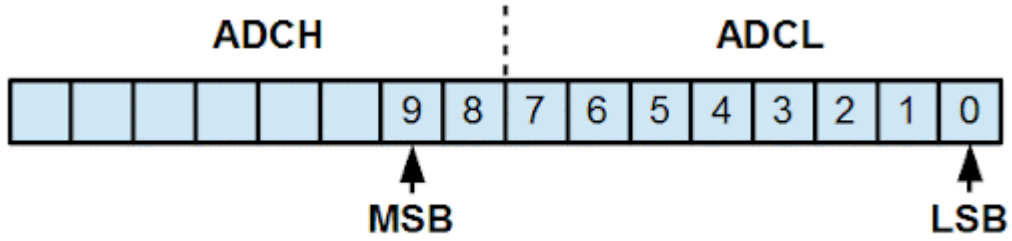
ADC Register

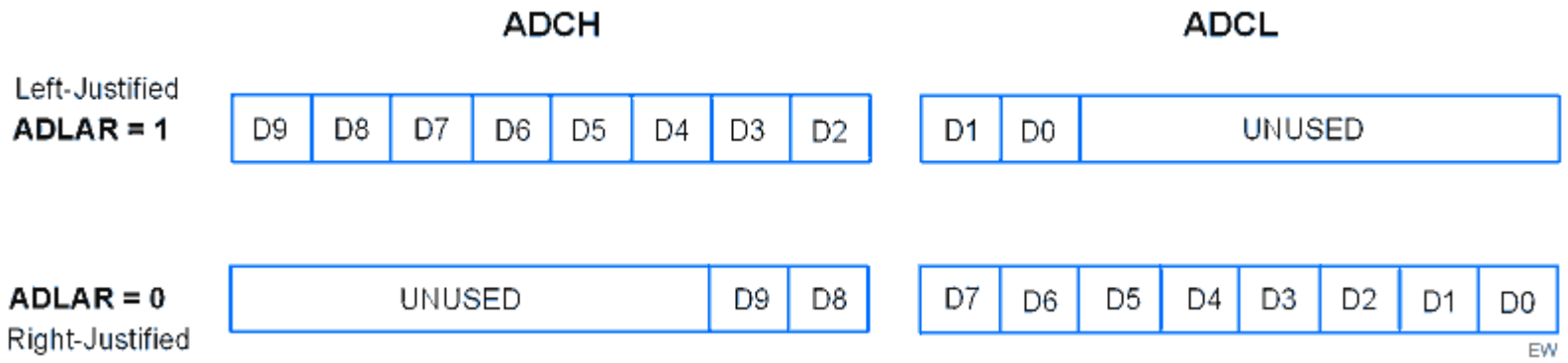
ADCH: ADCL register

First, two-register holds the digital converted data, which is 10-bit.

ADCH: Holds digital converted data higher byte

ADCL: Holds digital converted data lower byte





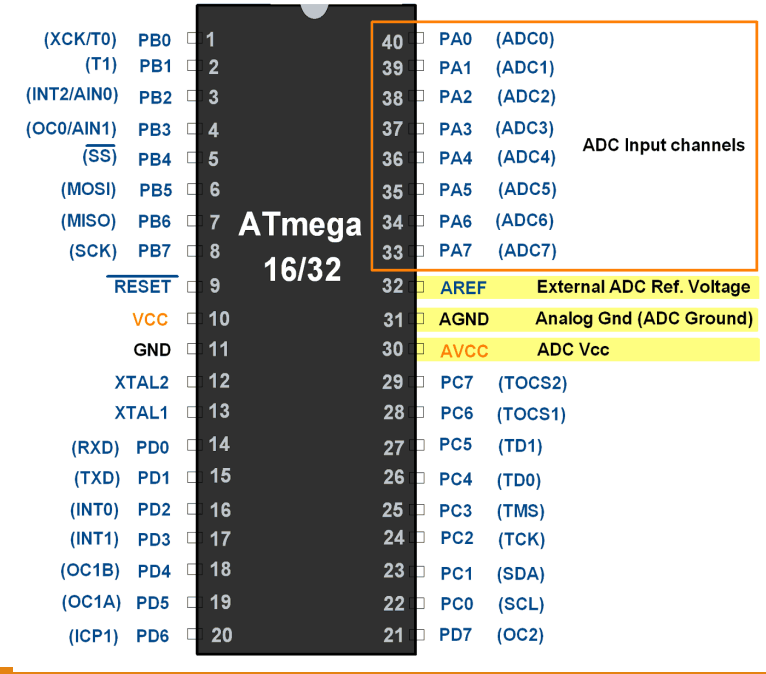
ADC Register

ADMUX Register

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

Bit 7: 6 – REFS1 : 0: Reference Selection Bits

REFS1	REFS0	Vref to ADC
0	0	AREF pin
0	1	AVCC pin i.e. Vcc 5 V
1	0	Reserved
1	1	Internal 2



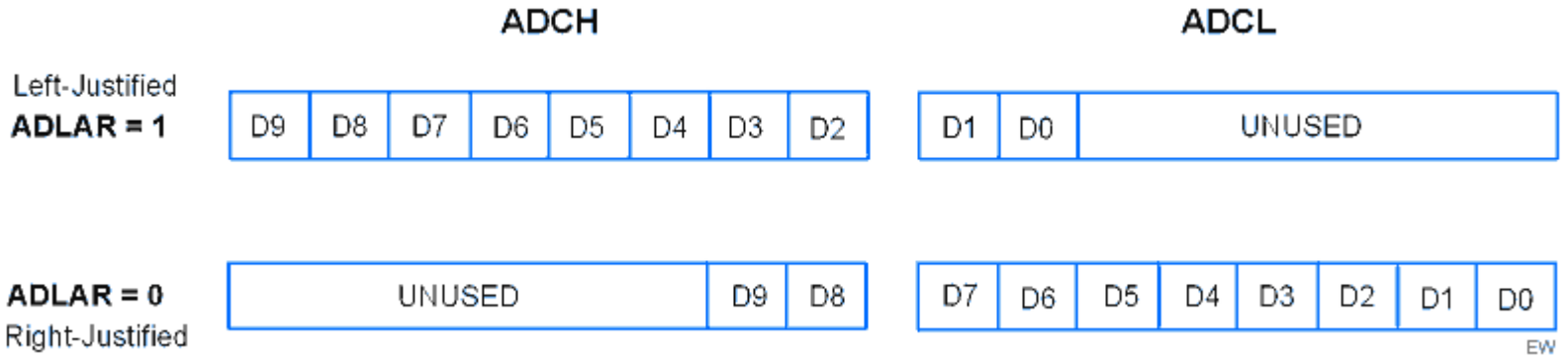
ADC Register

ADMUX Register

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

Bit 5 – ADLAR: ADC Left Adjust Result

Use 10-bits output as upper bits or lower bits in ADCH & ADCL.



ADC Register

ADMUX Register

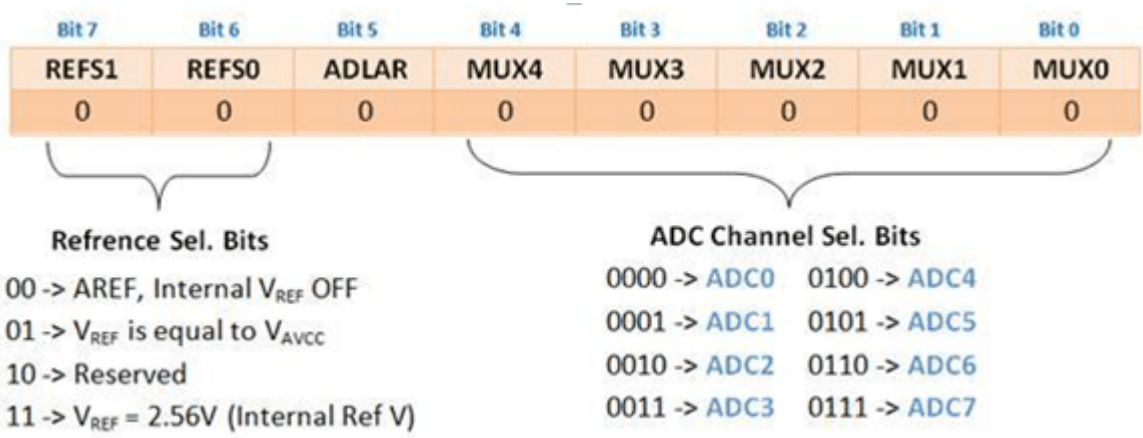
7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Bits 4 : 0 – MUX4 : 0: Analog Channel and Gain Selection Bits
We can select input channel ADC0 to ADC7 by using these bits.

Selecting a channel is very easy, just put the channel number in MUX4 : 0.

Suppose you are connecting the input to ADC channel 2 then put 00010 in MUX4 : 0.

Suppose you are connecting the input to ADC channel 5 then put 00101 in MUX4 : 0.



ADC Register

ADMUX Register

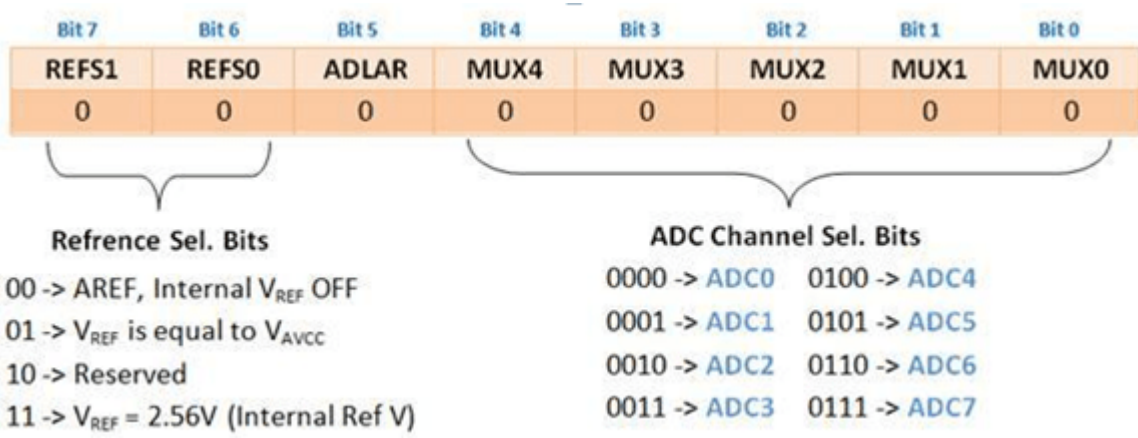
7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Bits 4 : 0 – MUX4 : 0: Analog Channel and Gain Selection Bits
We can select input channel ADC0 to ADC7 by using these bits.

Selecting a channel is very easy, just put the channel number in MUX4 : 0.

Suppose you are connecting the input to ADC channel 2 then put 00010 in MUX4 : 0.

Suppose you are connecting the input to ADC channel 5 then put 00101 in MUX4 : 0.



ADC Register

Single Conversion or Auto triggering !!

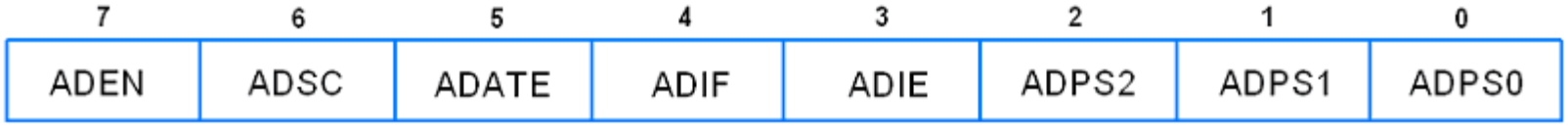
- In a single conversion mode you manually trigger the ADC process each time you want.

Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit.

- Provides a method of starting conversions at fixed intervals.
- Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. (Free running mode).
- At the end of each conversion the ADIF should be cleared by writing one !! So automatically the next conversion starts

ADC Register

ADCSRA Register:



- **Bit 7 – ADEN: ADC Enable**

Writing one to this bit enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

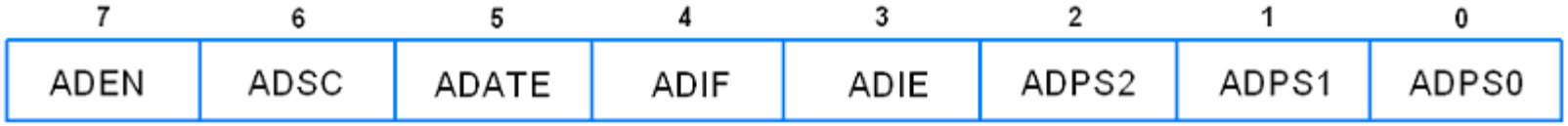
Writing one to this bit starts the conversion.

- **Bit 5 – ADATE: ADC Auto Trigger Enable**

Writing one to this bit, results in Auto Triggering of the ADC is enabled.

ADC Register

ADCSRA Register:



- **Bit 4 – ADIF: ADC Interrupt Flag**

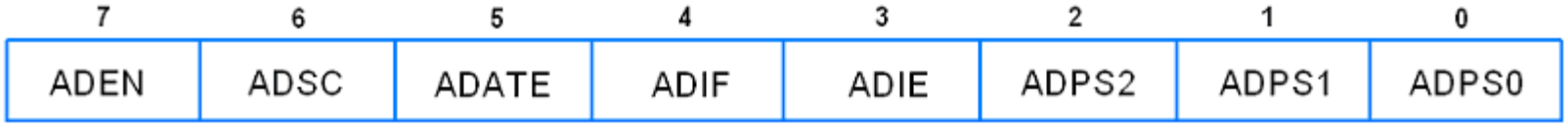
This bit is set when an ADC conversion completes and the Data Registers are updated.

- **Bit 3 – ADIE: ADC Interrupt Enable**

Writing one to this bit, the ADC Conversion Complete Interrupt is activated.

ADC Register

ADCSRA Register:



ADC clock frequency = XTAL frequency / Prescaler

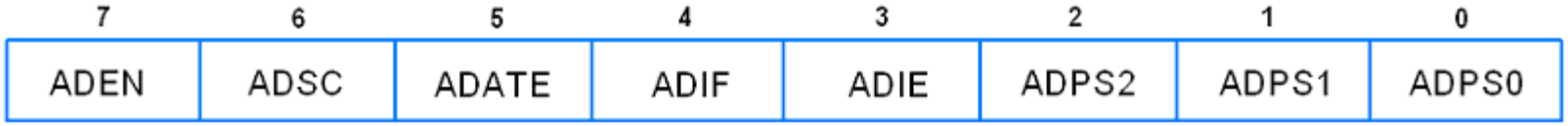
•Bits 2 : 0 – ADPS2 : 0: ADC Prescaler Select Bits

These bits determine the division factor between the XTAL frequency and the input clock to the ADC

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 Register

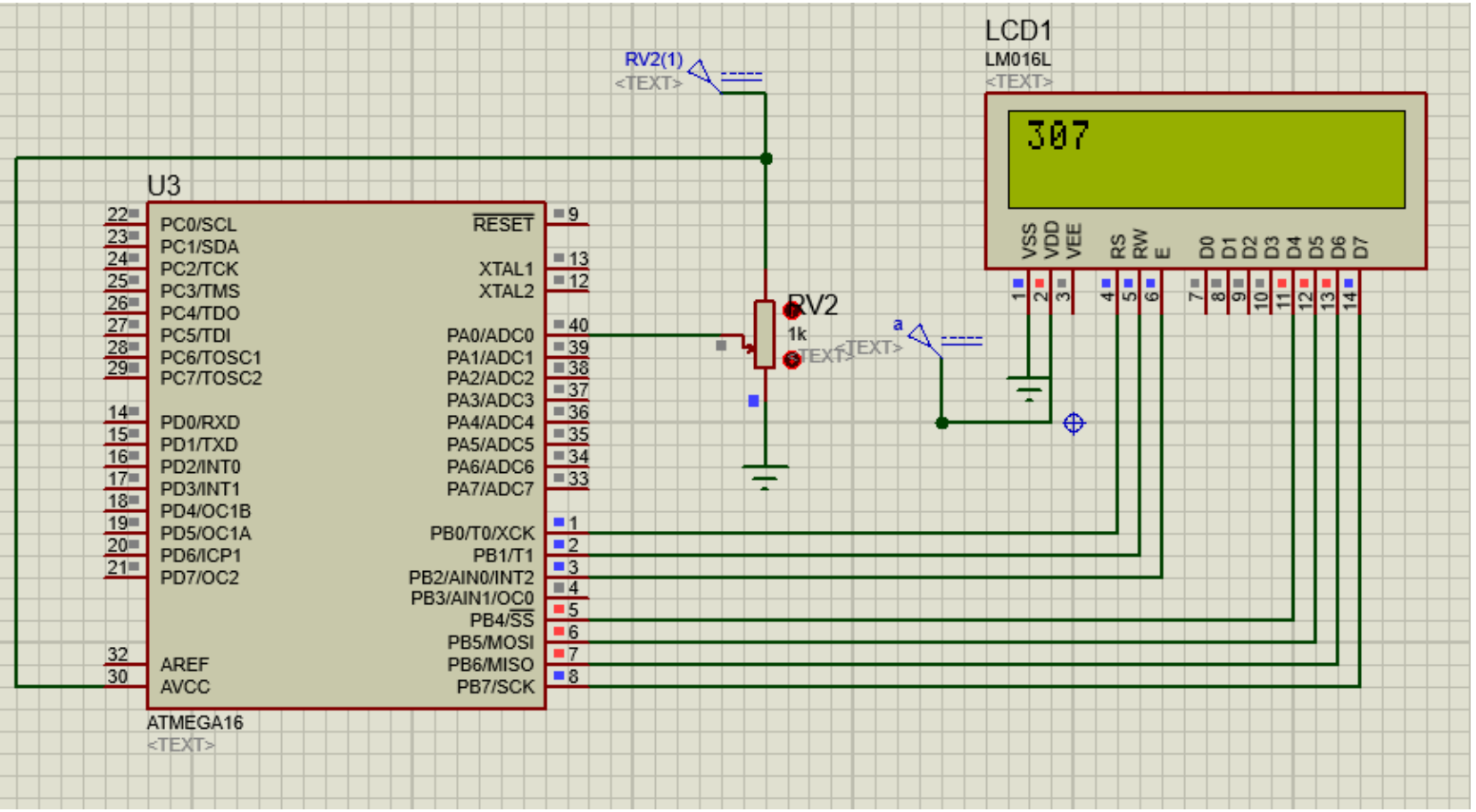
ADCSRA Register:



•Bits 2 : 0 – ADPS2 : 0: ADC Prescaler Select Bits
These bits determine the division factor between the XTAL frequency and the input clock to the ADC

ADC clock frequency = XTAL frequency / Prescaler
The ADC clock frequency must lie somewhere between *50 kHz to 200 kHz* to get maximum resolution.

Suppose your clock frequency of AVR is 8MHz, then we must have to use devisor 64 or 128. Because it gives 8MHz/64 = 125KHz, which is lesser than 200KHz.



ADC Example

LCD16x2 is used to show digital converted value from channel 0.

ADC Example

Steps to Program ADC

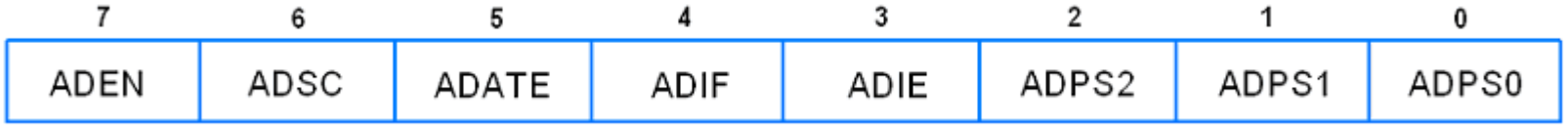
1. Make the ADC channel pin as an input.

```
DDRA=0x0; /* Make ADC port as input */
```

ADC Example

Steps to Program ADC

2. Set ADC enable bit in ADCSRA, select the conversion speed using ADPS2 : 0. For example, we will select devisor 128.



1 0 0 0 0 1 1 1 = 0x87

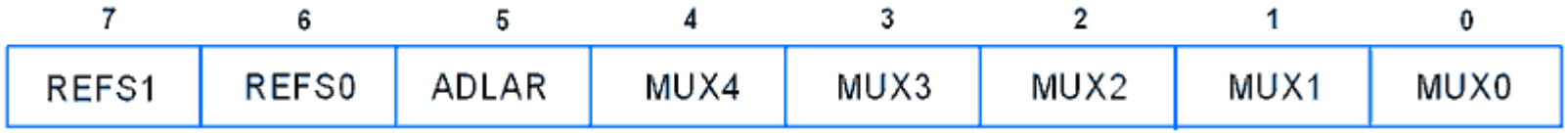
```
ADCSRA = 0x87; /* Enable ADC, fr/128 */
```

ADC Example

Steps to Program ADC

3. Select ADC reference voltage using REFS1: REFS0 in ADMUX register, for example, we will use AVcc as a reference voltage.

4. Select the ADC input channel using MUX4 : 0 in ADMUX, for example, we will use channel 0.




```
ADMUX = 0x40; /* Vref: Avcc, ADC channel: 0*/
```

ADC Example

Steps to Program ADC

5. Start conversion by setting bit ADSC in ADCSRA. E.g. ADCSRA |= (1<<ADSC);

```
ADCSRA |= (1<<ADSC); /* Start conversion */
```

6. Wait for conversion to complete by polling ADIF bit in ADCSRA register.

```
while((ADCSRA&(1<<ADIF))==0);  ///* Monitor end of conversion interrupt */
```

7. After the ADIF bit gone high, read ADCL and ADCH register to get digital output.

```
AinLow = (int)ADCL;     /* Read lower byte*/
Ain = (int)ADCH*256; /* Read higher 2 bits and Multiply with weight */
Ain = Ain + AinLow;
```

Notice that read ADCL before ADCH; otherwise result will not be valid

ADC Example

Steps to Program ADC

1. Make the ADC channel pin as an input.
2. Set ADC enable bit in ADCSRA, select the conversion speed using ADPS2 : 0. For example, we will select devisor 128.
3. Select ADC reference voltage using REFS1: REFS0 in ADMUX register, for example, we will use AVcc as a reference voltage.
4. Select the ADC input channel using MUX4 : 0 in ADMUX, for example, we will use channel 0. So our value in register ADCSRA = 0x87 and ADMUX = 0x40.
5. Start conversion by setting bit ADSC in ADCSRA. E.g. ADCSRA |= (1<<ADSC);
6. Wait for conversion to complete by polling ADIF bit in ADCSRA register.
7. After the ADIF bit gone high, read ADCL and ADCH register to get digital output.

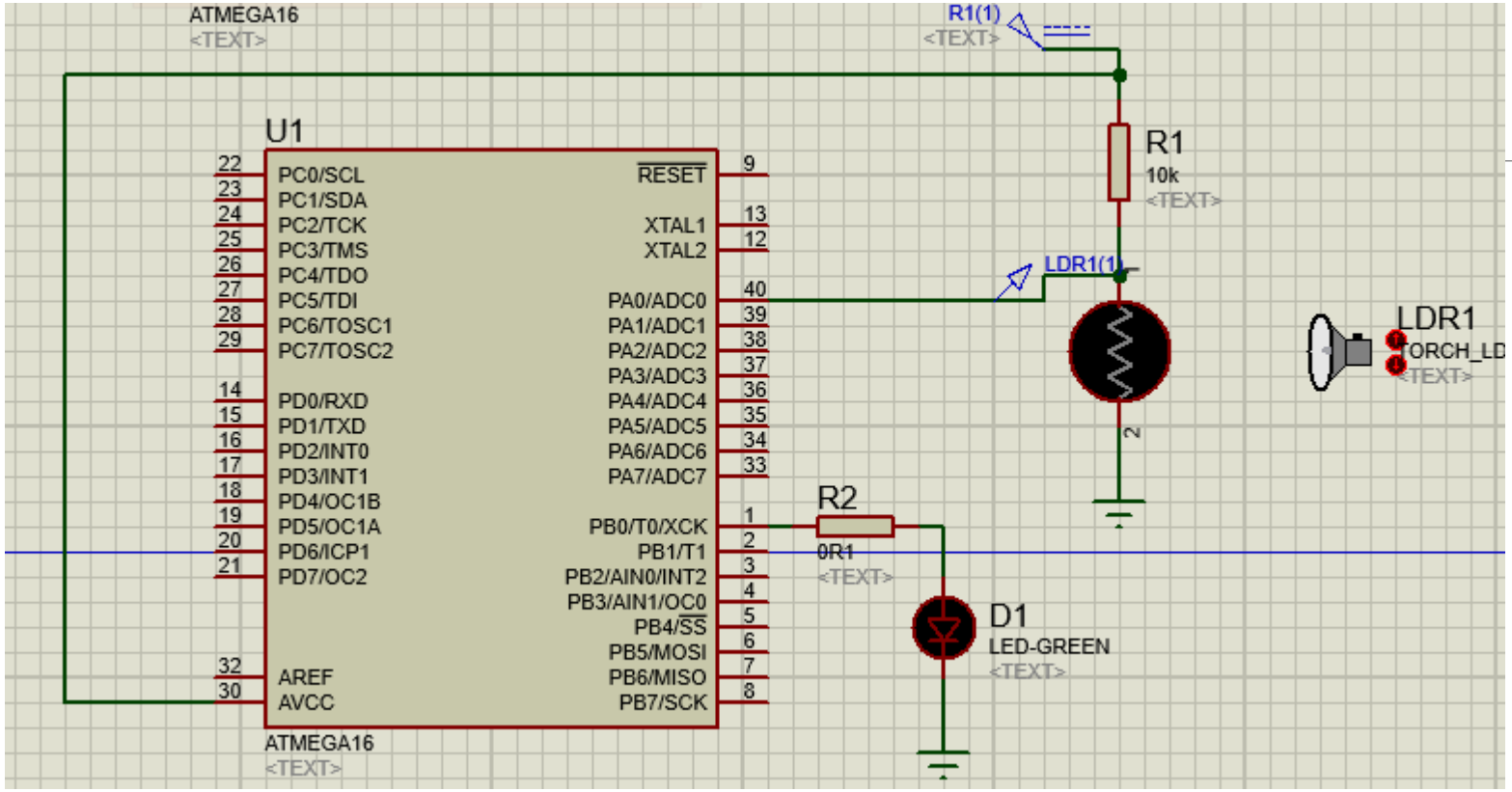
   Notice that read ADCL before ADCH; otherwise result will not be valid

ADC Example

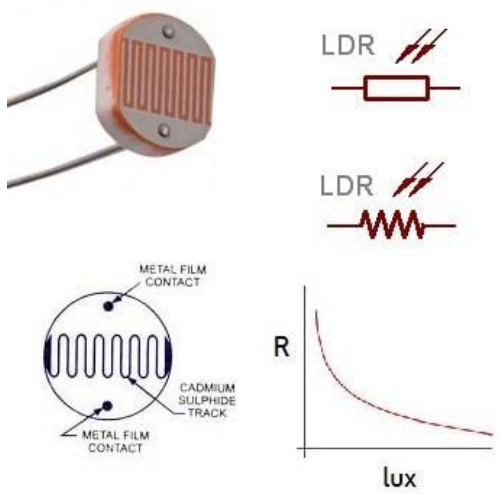
```
#include <mega16.h>
 #include <delay.h>
#include <stdlib.h>
#include <alcd.h>
int read_adc(char channel);

void main(void)
{
int value;
char String[5];
DDRA=0x0; /* Make ADC port as input */
lcd_init(16);
while (1)
     {
       value= read_adc(0); /* Read ADC channel 0 */
       itoa(value,String); /* Integer to string conversion */
        lcd_puts(String);
        delay_ms(500);
       lcd_clear();
     }
}
```

```
int read_adc(char channel)
{
int  AinLow, Ain;
ADCSRA = 0x87; /* Enable ADC, fr/128 */
ADMUX = 0x40; /* Vref: Avcc, ADC channel: 0*/
ADMUX=ADMUX|(channel & 0x0f); // Set input channel to read
delay_ms(10);

ADCSRA |= (1<<ADSC); /* Start conversion */

/* Monitor end of conversion interrupt */
 while((ADCSRA&(1<<ADIF))==0);
delay_ms(10);

AinLow = (int)ADCL; /* Read lower byte*/

 /* Read higher 2 bits and Multiply with weight */
Ain = (int)ADCH*256;

Ain = Ain + AinLow;
 return(Ain); /* Return digital value*/
}
```

ADC Example 2



Automatic Street Light Controller using Atmega and LDR as Light sensor



ADC Example 2

```
#include <mega16.h>
 #include <delay.h>
int read_adc(char channel);

void main(void)
{
int value;
DDRA=0x0; /* Make ADC port as input */
PORTB=0x00;
DDRB=0x01;      //LED pin as a output

while (1)
     {
       value= read_adc(0); /* Read ADC channel 0 */
        if(value> 512)
                PORTB=0x01;
       else
               PORTB=0x00;
      }
}
```

```
int read_adc(char channel)
{
int  AinLow, Ain;
ADCSRA = 0x87; /* Enable ADC, fr/128 */
ADMUX = 0x40; /* Vref: Avcc, ADC channel: 0*/
ADMUX=ADMUX|(channel & 0x0f); // Set input channel to read
delay_ms(10);

ADCSRA |= (1<<ADSC); /* Start conversion */

/* Monitor end of conversion interrupt */
 while((ADCSRA&(1<<ADIF))==0);
delay_ms(10);

AinLow = (int)ADCL; /* Read lower byte*/

 /* Read higher 2 bits and Multiply with weight */
Ain = (int)ADCH*256;

Ain = Ain + AinLow;
 return(Ain); /* Return digital value*/
}
```

Arduino ADC

Arduino boards contain a multichannel, 10-bit analog to digital converter.

This means that it will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values between 0 and 1023.

On an Arduino UNO, for example, this yields a resolution between readings of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit

Syntax

```
analogRead(pin)
```

Arduino ADC

Arduino boards contain a multichannel, 10-bit analog to digital converter.

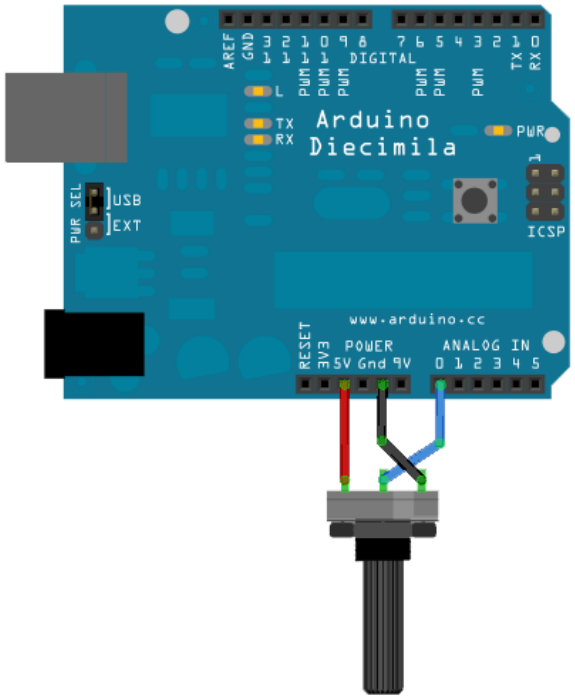
This means that it will map input voltages between 0 and the operating voltage(5V or 3.3V) into integer values between 0 and 1023.

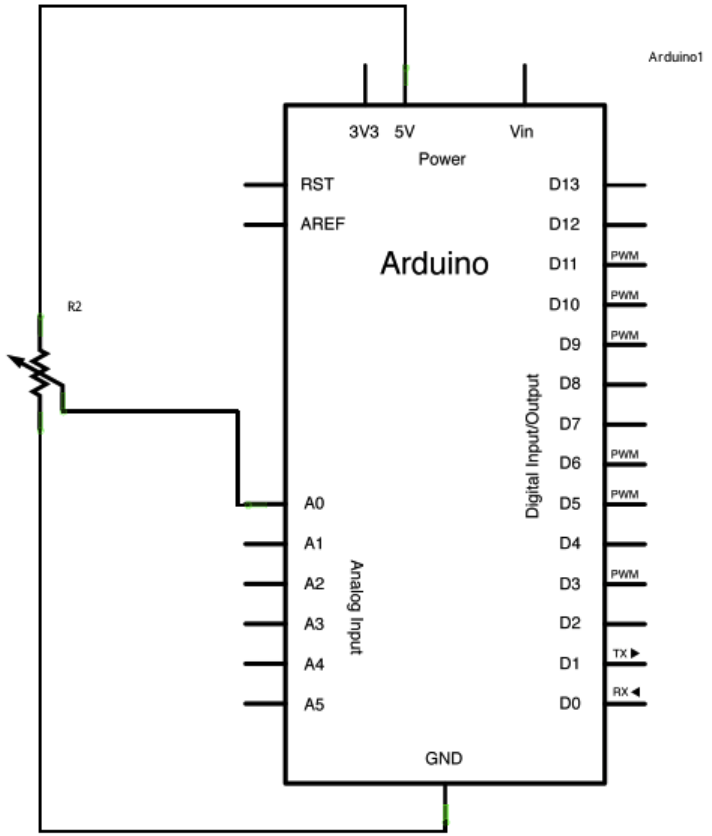
On an Arduino UNO, for example, this yields a resolution between readings of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit

Syntax

```
analogRead(pin)
```

Arduino ADC


AREF GND 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DIGITAL
PWM PWM PWM PWM PWM PWM
TX RX
L
TX
RX
Arduino
Diecimila
PWR
PWR SEL
USB
EXT
ICSP
www.arduino.cc
RESET 3V3 5V Gnd Gnd 9V
POWER
ANALOG IN
0 1 2 3 4 5


Arduino1
3V3 5V
Power
Vin
RST
AREF
Arduino
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
PWM
TX
RX
R2
A0
A1
A2
A3
A4
A5
Analog Input
Digital Input/Output
GND

Arduino ADC

```
int sensorPin = A0;    // select the input pin for the potentiometer
int ledPin = 13;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

Thanks