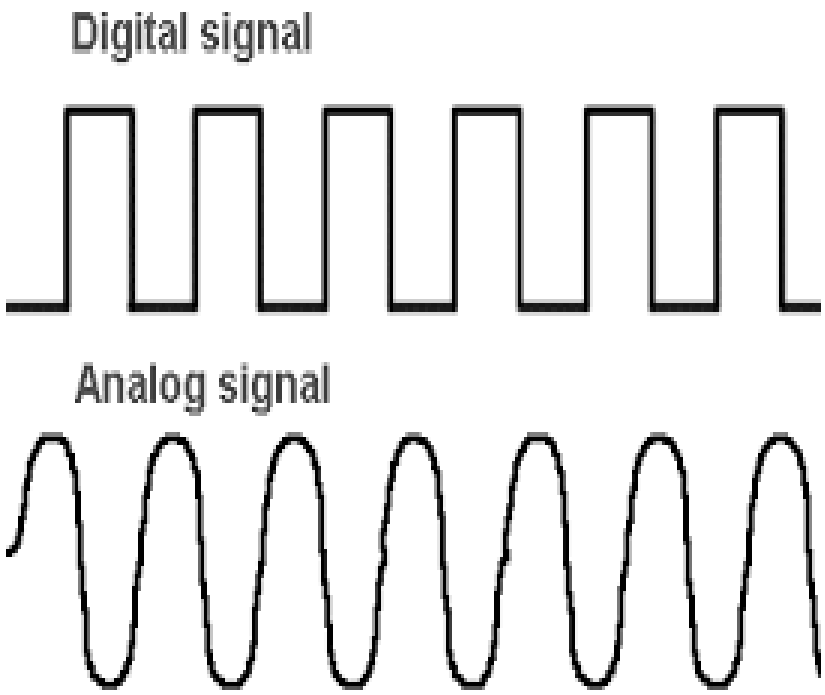
BASICS:

***What is a Signal?**

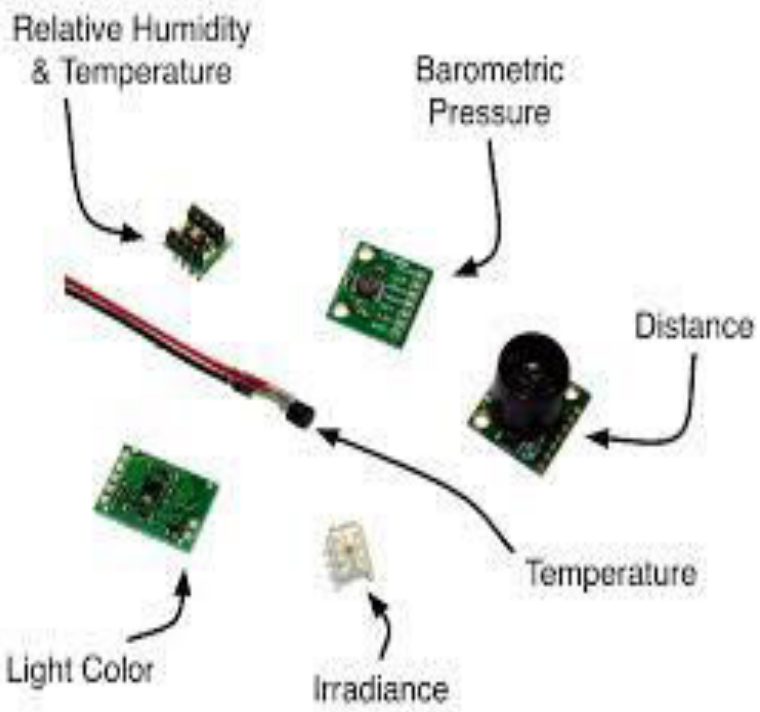
-Conveys information or instructions about behavior and attributes of a phenomenon

***Analog and digital signals?**



***Purpose of ADC?**

Almost all the real world signals are analog …. Output of the sensors we use are analog but microcontroller(digital IC) cannot understand these analog signals…So we have to convert these analog outputs to digital(0's and 1's).



PIN CONFIGURATION:

Generally most of the sensors have three basic terminals as shown in the figure…

***some have 4th terminal as external triggering.**

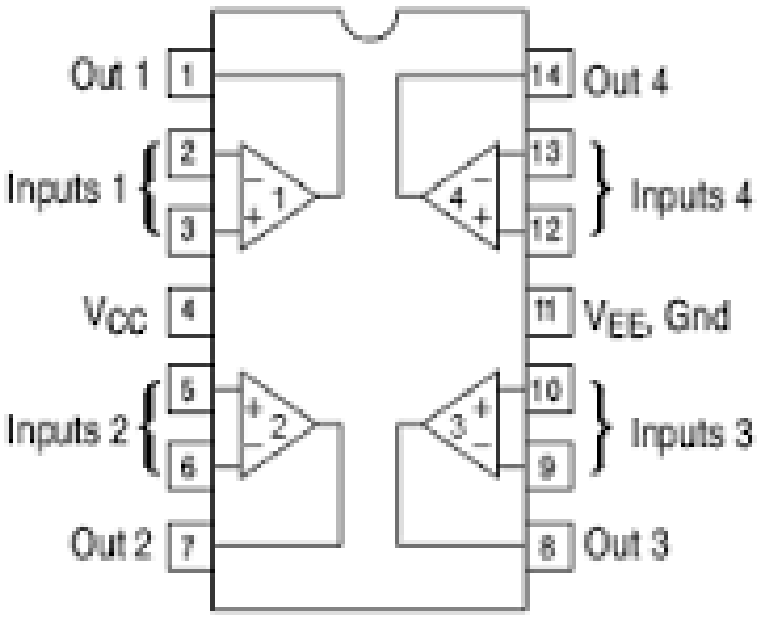
ADC(Analog to digital conversion):

1.Using comparator and trimpots(Hardware)

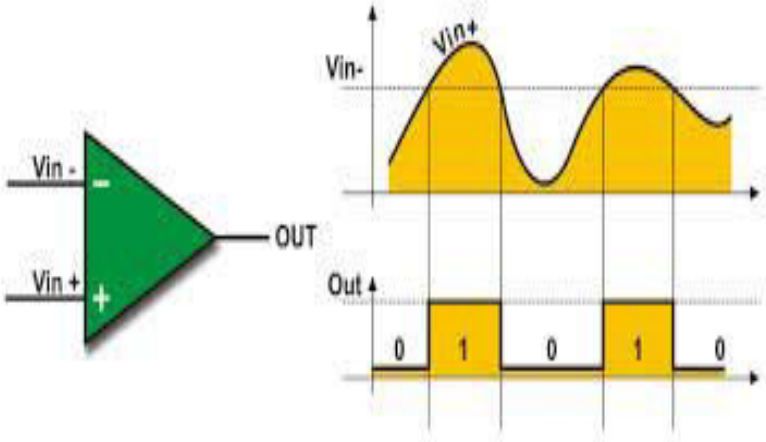
2.ADC programming (Software)

1.Using comparator and trimpots(Hardware)

***COMPARATOR:**



***OP-AMP:**



The amplifier's differential inputs consist of a non-inverting input (+) with voltage V_+ and an inverting input (–) with voltage V_-; ideally the op-amp amplifies only the difference in voltage between the two, which is called the *differential input voltage*. The output voltage of the op-amp V_{out} is given by the equation:

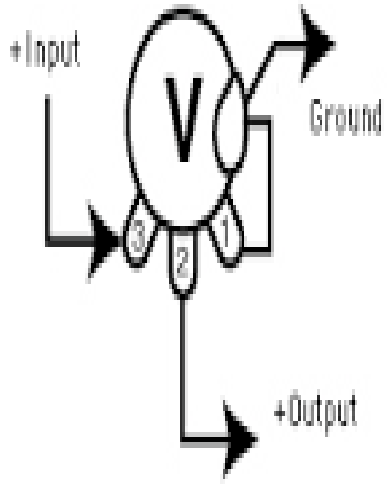
$$V_{\text{out}} = A_{\text{OL}} (V_+ - V_-)$$

where A_{OL} is the open loop gain of the amplifier (the term "open-loop" refers to the absence of a feedback loop from the output to the input).

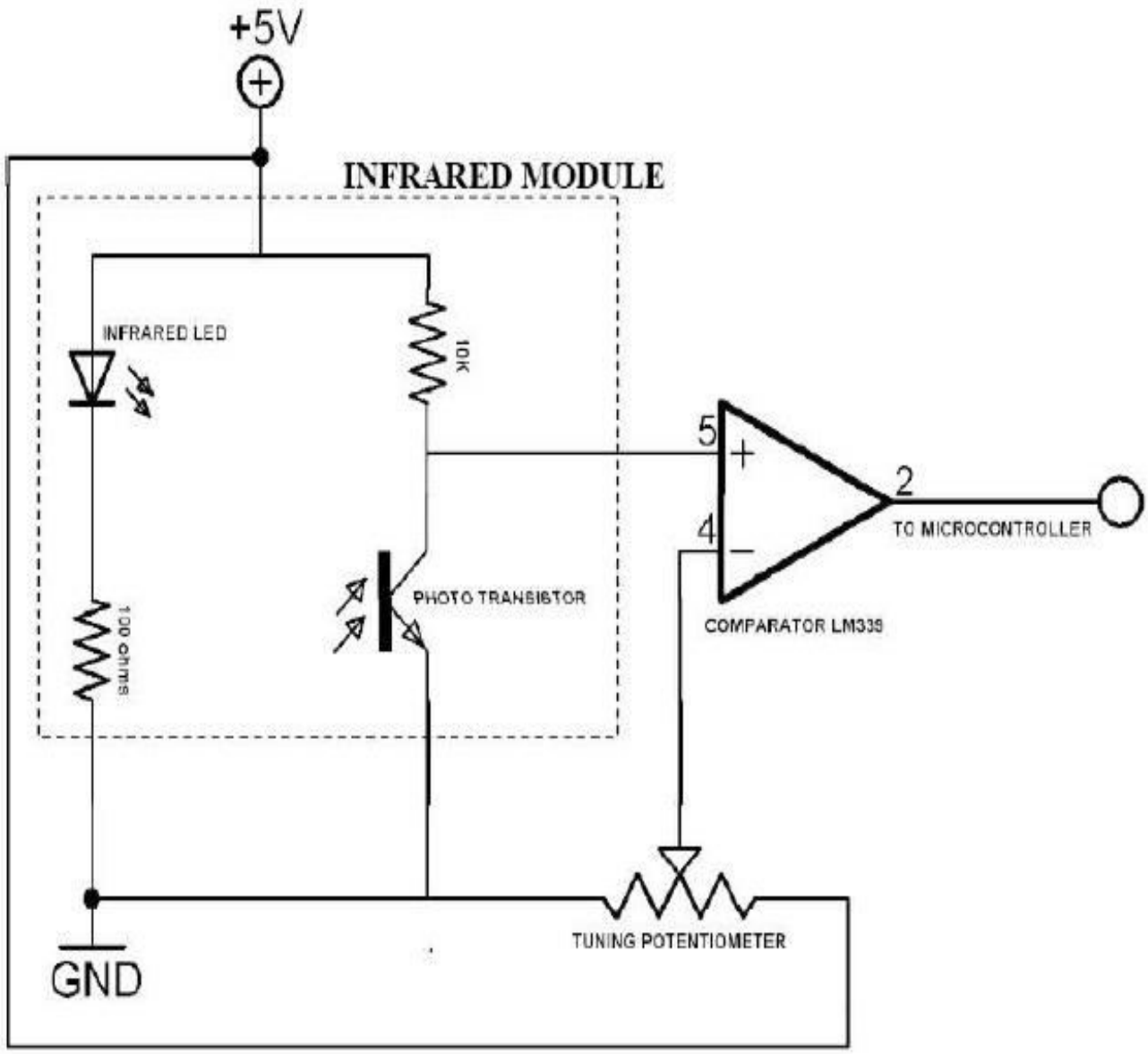
REFERENCE:

http://en.wikipedia.org/wiki/Operational_amplifier

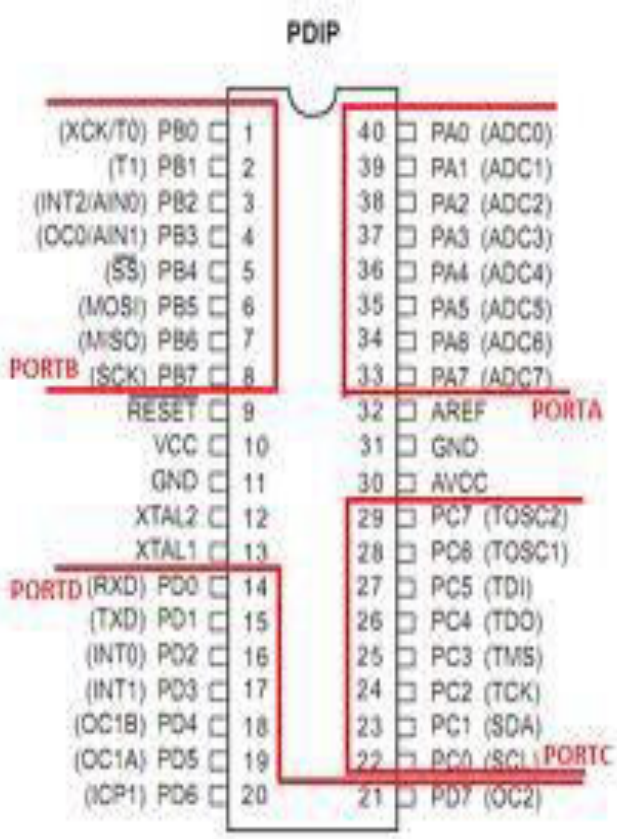
***TRIMPOT(Trimming Potentiometer):**



CIRCUIT FOR ADC:



ATMEGA16:


PDIP
(XCK/T0) PB0 1
(T1) PB1 2
(INT2/AIN0) PB2 3
(OC0/AIN1) PB3 4
(SS) PB4 5
(MOSI) PB5 6
(MISO) PB6 7
PORTB (SCK) PB7 8
RESET 9
VCC 10
GND 11
XTAL2 12
XTAL1 13
PORTD (RXD) PD0 14
(TXD) PD1 15
(INT0) PD2 16
(INT1) PD3 17
(OC1B) PD4 18
(OC1A) PD5 19
(ICP1) PD6 20
40 PA0 (ADC0)
39 PA1 (ADC1)
38 PA2 (ADC2)
37 PA3 (ADC3)
36 PA4 (ADC4)
35 PA5 (ADC5)
34 PA6 (ADC6)
33 PA7 (ADC7)
32 AREF
PORTA
31 GND
30 AVCC
29 PC7 (TOSC2)
28 PC6 (TOSC1)
27 PC5 (TDI)
26 PC4 (TDO)
25 PC3 (TMS)
24 PC2 (TCK)
23 PC1 (SDA)
22 PC0 (SCL)
PORTC
21 PD7 (OC2)

2.ADC (programming)

INTRODUCTION:

Analog to digital convertor is the one of the peripherals of the microcontroller. It converts voltages to digital numbers. As we use Atmega16 microcontroller it assigns the maximum voltage (5V) to number 1023 and 0v to the number 0. Because our microcontroller is of 10bit resolution. Mathematically 10 is the power of 2 which is equal to 1024. We are considering number 0 also, that's why 5v is assigned to 1023.

***RESOLUTION?**

Resolution is a property of a digital system and indicates its precision, defined as a percentage of the ratio of the smallest value to the full scale value it can handle. For example, the resolution of a system handling only variables as bytes (8bit long) will be 1 in 256(2^8).

Register:

Register is a memory element. Here each register will store 8bits of data.

Generally, if we look at the basic setting of bits using a DDR register, the functioning of all the 8bits in DDR register are similar.

But in ADC each register is different from others registers

and also each bit is different from the other bits. Each bit of a register is given a special name.

Registers in ADC:

We have three registers in ADC

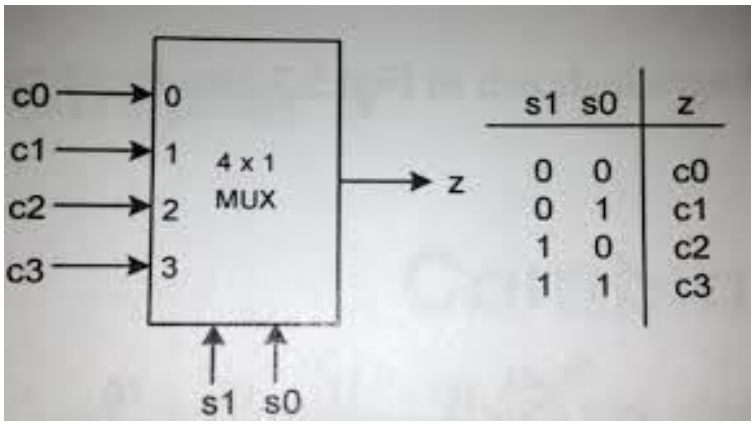
1. **ADMUX(ADC Multiplexer)**
2. **ADCSRA(ADC Control and Status Register)**
3. **ADC(ADC Data Register)**

ADC Registers Explanation:

1. ADMUX(ADC Multiplexer):

*In electronics, a **multiplexer** (or mux) is a device that selects one of several analog or digital input signals and forwards the selected input into a single line.

In the below figure Co , C1 , C2 , C3 denote the data inputs….S0 , S1 denote the control inputs….Z denotes the output….



As I have told you already each bit is given a special name. And here are the different names for bits of ADCMUX register and their initial settings.

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	

Let me explain each bit of the ADCMUX register.

- **MUX0 to MUX4:**

Bits MUX0 to MUX4 are similar. Remember you can only connect inputs to port A to use ADC peripheral. And each pin of the port A is called a channel. We have eight input channels(ADC0,ADC1,ADC2……..ADC7). But how do you tell your microcontroller that you have connected a analog sensor to the one of the pins of port A as a input?

See this chart

Input Channel	MUX2	MUX1	MUX0
ADC0	0	0	0
ADC1	0	0	1
ADC2	0	1	0
ADC3	0	1	1
ADC4	1	0	0
ADC5	1	0	1
ADC6	1	1	0
ADC7	1	1	1

Chat says if you set MUX0, MUX1 and MUX2 to zeros then the input channel will be ADC0 (PIN0). And remaining bits are similar to this. But what about MUX3 and MUX4?? I can say that those bits are reserved. Suppose In our microcontroller we have only eight input channels, but in some other Atmel microcontrollers there may be more than eight channels. In

that case we could use remaining two bits. For now they are kept zeros. How do you set these bits??

Setting type one: Let's us say you want to make ADC3 as inputs. To make it we have to set MUX0 and MUX1 to 1 as shown in the chart above. And we put remaining bits as zeros for now though they have specific purpose. Code is

ADMUX=0b00000011;

Setting type two: In "setting type one" to set MUX0 and MUX1 to 1,we have involved others bits also by setting them to 0.which is the drawback. But here the code is

ADMUX=(1<<MUX0)|(1<<MUX1);

In this I have only changed required bits without disturbing other bits. Here I have used left shift operators to make those two bits set.

"<<" is left shift operator.

"(1<<MUX1) "means left shift the value 1 to MUX1 steps. Then what is the value of MUX1?? Value is 1.

But how? If you look at the ADMUX register figure there the number (value) given to the MUX1 bit is 1. The value given to the MUX4 bit is 4. The value given to the ADLAR bit is 5.

“(1<<MUX1)” means left shift the value 1 to 1 step(s).

Now let’s set MUX4 bit..

ADMUX=(1<<MUX4);

We try to understand this step. It means left shift value 1 to 4 steps. Initially value one will be 0 th position and it moves 4 times left. As shown below.

ADMUX=0b00000001; (initially)

ADMUX=0b00000010; (step 1)

ADMUX=0b00000100; (step 2)

ADMUX=0b00001000; (step 3)

ADMUX=0b00010000; (step 4(final step))

If you do not understand “type two setting” just write the name of bit you want to set and left shift it to one.

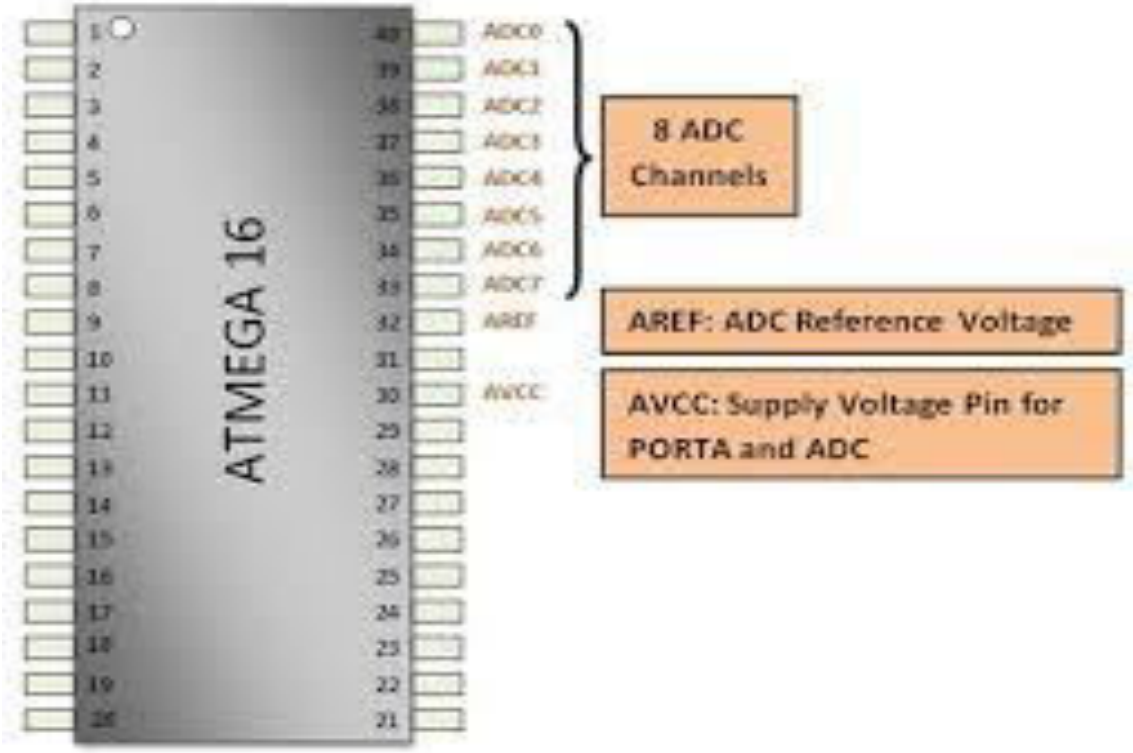
register=(1<<name of bit);

This is how we can easily set bits without disturbing the other bits.

- ## ADLAR (ADC Left Adjust Result):

I will discuss about this bit while discussing ADC register.

- **REFS1 and REFS0 (ADC reference selection bits):**



As I told you before, maximum voltage gets assigned to 1023. But we can also have a chance to assign different voltage levels to 1023. Assigning a voltage lesser than 5V to 1023 will give you more possibilities. Suppose if you want make 4V assigned to 1023 then you have to provide 4V to AREF pin of microcontroller. See the chart below

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

If you set REFS1 and REFS0 to 0, then AREF pin gets turned off .(default 5v reference)

If you set REFS1 to 0 and REFS0 to 1, then you can provide voltage reference to AREF pin and make it equals to 1023. And connect 0.01uF capacitor between AVCC and AREF for reducing noise.

If you set REFS1 to 1 and REFS0 to 0, nothing will happen as it is reserved for some other purpose by the manufacturer.

If you set BOTH to 1, 2.56 volts of internal reference will be activated and gets assigned to 1023. And connect 0.01uF capacitor between AVCC and AREF for reducing noise.

Setting:

ADMUX= (1<<REFS1)|(1<<REFS0);

BITWISE OPERATORS works on bits and performs bit by bit operation…

If A=1001

B=0110

Then A|B=1111

A&B=0000

“|” says “OR” ing like this

ADMUX=0b01000000; ADMUX = (1<<REFS1);

ADMUX=0b10000000; ADMUX = (1<<REFS0);

OR above two we get these

ADMUX=0b11000000; ADMUX= (1<<REFS1)|(1<<REFS0);

2. ADCSRA(ADC Control and Status Register)

These are the bits in ADCSRA register. See the figure below

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	

- **ADPS0 to ADPS2 (ADC Prescaler Selection bits):**

Generally ADC works in the frequency between 50KHZ to 150KHZ. To set the frequency ,we have to divide system clock

frequency with a integer so that obtained frequency will be in the range of 50KHZ to 150KHZ this is called PRESCALING

System clock frequency will be set by the user like below on the top of the total code like this. For 1MHZ.
#define F_CPU 1000000UL.
See the chart below

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

If you set system clock frequency for 1MHZ then it's better to divide it with integer 16 by setting only ADPS2 to 1.So that ADC frequency of 62.5KHZ which is in the range of 50KHZ to 150KHZ.

- **ADIE (ADC Interrupt Enable):**
  This bit is for enabling interrupt. It's not used generally. Used only if we arrange external interrupts.

Setting: **ADCSRA= (1<<ADIE);**

- **ADIF (ADC Interrupt Flag):**

  This bit set to one automatically when ADC conversion is finished.

  Setting: **ADCSRA= (1<<ADIF);**

- **ADATE (ADC Auto Trigger Enable):**

  When this bit is set to one triggering is enabled and free running mode is activated. if it is zero only once conversion gets completed.

  Setting: **ADCSRA= (1<<ADIE);**

- **ADSC (ADC Start Conversion bit):**

  When this bit is set to one ADC conversion gets started. We have to wait until ADIF bit becomes one as it is the bit which indicates completion of ADC conversion.

Setting: **ADCSRA= (1<<ADSC);**

Once you set ADSC bit, you have to wait until ADC conversion completes. This means you have to wait until ADIF bit becomes 1.

Waiting code: **while(! (ADCSRA & (1<<ADIF)));**

(Take pen and paper try to analyze how the above step is waiting for ADIF to become 1)

- **ADEN(ADC Enable):**

  This bit must be set to one to enable ADC (set initially).

  Setting: ADCSRA= (1<<ADSC);

3. ADC (ADC Data Register):

Once voltage levels get converted into numbers, those numbers will be stored in ADC register as ADC values. These values will be stored in two sub registers ADCH and ADCL. Like below

ADLAR = 0

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 = 1

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	

The storing of ADC values depends on the ADLAR (I left this bit earlier to explain now) bit in ADMUX register. It's is left adjustment register if this bit set to 0 then the ADC values will be right adjusted as first case in the above figure.

If the bit is set to 1 then ADC values will be left adjusted. If the ADC values are left adjusted we have to read ADC values from ADCH which is 8 bit data. Otherwise we have to read from ADCL and then from ADCH.

Reading ADC values:

Read into a variable as shown below

```
X=ADCH;
```

Now use the obtained ADC values for running your robot…..

ALGORITHM:

1. **Set initially clock, input, prescaler and enable bits**
2. **Start conversion setting ADSC conversion register.**
3. **Wait until the conversion is over(check ADIF)**
4. **Read ADC values from ADCH and ADCL**
5. **Use ADC values for your purpose.**

SAMPLE CODE:

```
#define F_CPU 1000000UL //Clock frequency
#include<avr/io.h>        // Header file
#include<util/delay.h>      //Header file
int main (void)
{
  DDRB = 0b11111111;      // port B output

  int x = 0,y=0;

  // Activate ADC with Prescaler 16 --> 1Mhz/16 = 62.5kHz
  ADCSRA = (1<<ADEN) | (1<<ADPS2);

  While(1) {
```

```
    // Select pin ADC2 using MUX
    ADMUX = (1<<MUX0)|(1<<MUX1);
//This  below line can be ignored as those bits are set to  “0” by default
   ADMUX = (0<<REFS0)|(0<<REFS1);



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

    // wait until conversion completed
    while (!(ADCSRA & (1<<ADIF)) )
{
}

    // get converted value
```

```
      x = ADCL;
      y = ADCH;

    // taking output from first pin
    PORTB=0b00000001;



  }
return 0;
    }
```

PROJECTS:

These are some of the basic projects involving ADC….

1. Automatic Railway Gate Opening System.

2. Security System using GSM module.

3. Smart Door Opening System.

4. Accelerometer robot.

5. Line follower.

6. Obstacle Avoider.

7. Security System using LDR , Laser.

8. Heart Beat Detector.

9. Self Balancing Robots.

10. Automatic Fruit Drinks Filler.

11. Smart Garden Watering System.

12. Autonomous Fire Extinguisher.

THANK YOU