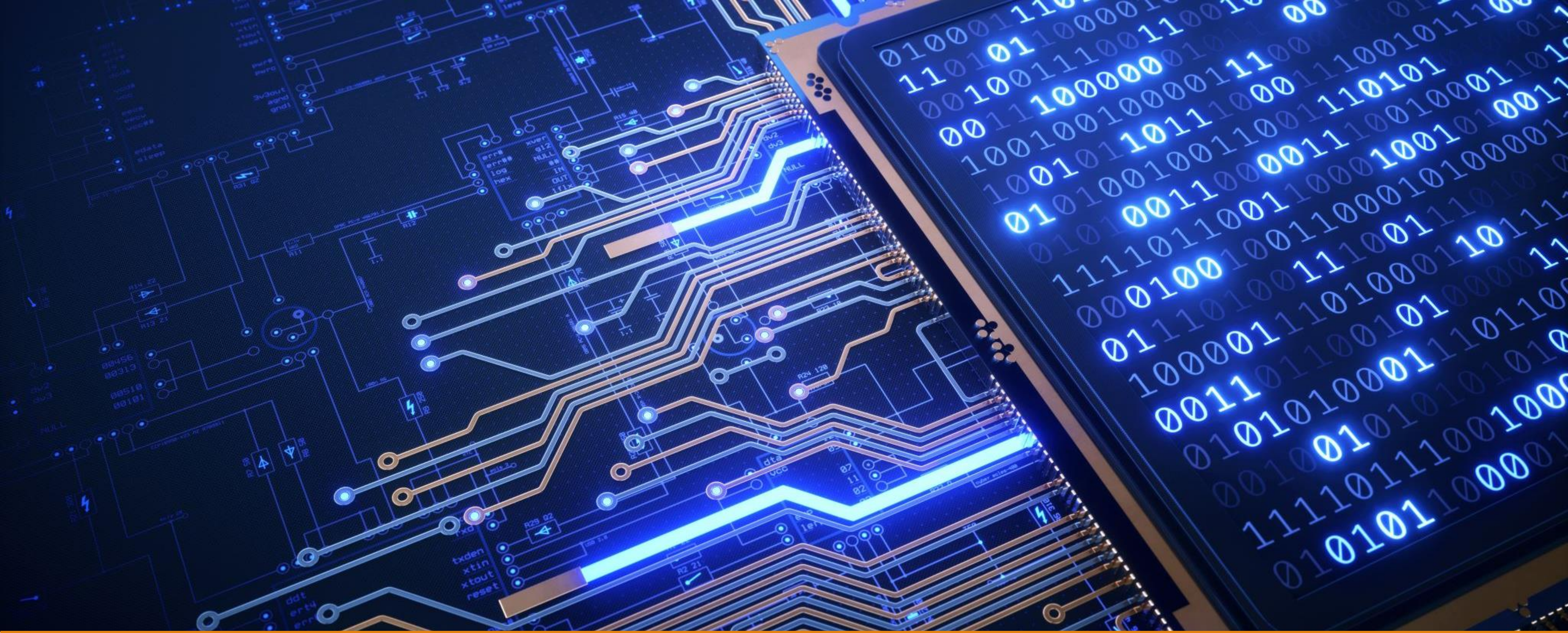


Embedded Systems

Dr. Abdelhay Ali

Lecture 6

EEPROM in AVR ATmega16

Outlines

Introduction

- **EEPROM** is Electrically Erasable Programmable Read-Only Memory.
- It is non-volatile type of memory as it holds the data even when power is off.
- The main advantage of this memory is that controller can read, modify/write this memory in runtime application.
- So EEPROM can be used for storing sensor values, important parameters etc. with no fear of loss even in case of power failure.

Introduction

- ❑AVR ATmega16 contain 512 bytes of data EEPROM memory.
- ❑It is organized as separate data space in which single byte can be read and written.
- ❑Normally EEPROM has limited life span.
- ❑ AVR ATmega16 EEPROM has endurance of 100,000 write/erase cycle.
- ❑So, we need to take care about while writing EEPROM that not to put EEPROM write operation in continuous loop. Note that it has only limit for write/erase cycle, there is no limit for read operation.

Introduction

- 512 byte having an endurance of at least 100,000 write/erase cycles.
- Write access time ≈ 8.5ms.
- While read CPU halted for 4 clock cycles
- While write CPU halted for 2 clock cycles

Outlines

EEPROM Registers

Access over EEPROM memory is made through three registers.

- **EEAR** (EEPROM Address register).
- **EEDR** (EEPROM Data register).
- **EECR** (EEPROM Control register).

EEPROM Registers

EEPROM Address Register (EEAR)

Atmega16 has 16 bit EEAR register which is used to address the location of EEPROM memory.

In EEAR, lower 9-bits are used to specify the address and remaining are reserved and will always read as zero.

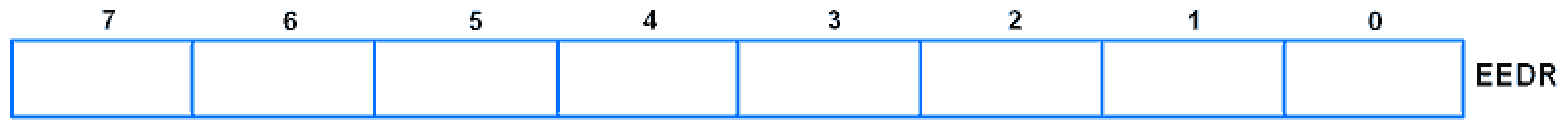
It has two 8-bit registers EEARH and EEARL. EEARL contain first 8-bit of address and EEARH contain last 9th bit of address

D15							D8	
---	---	---	---	---	---	---	EEAR8	EEARH

D7							D0	
EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL

EEPROM Registers

EEPROM Data Register (EEDR)



EEDR contains data to be written/read at the location addressed by EEAR in write/read operation.

EEPROM Registers

EEPROM Control Register (EECR)

EECR contains control bits to get access over EEPROM.

7	6	5	4	3	2	1	0	
---	---	---	---	EERIE	EEMWE	EEWE	EERE	EECR

Bit 0 – **EERE:** EEPROM Read Enable

1 = Read enable.

0 = Read disable.

Setting EERE while EEAR contains proper address enables read operation.
Let's see the write/read sequence for EEPROM,

Bit 1 – **EEWE**: EEPROM Write Enable

1 = Write enable.

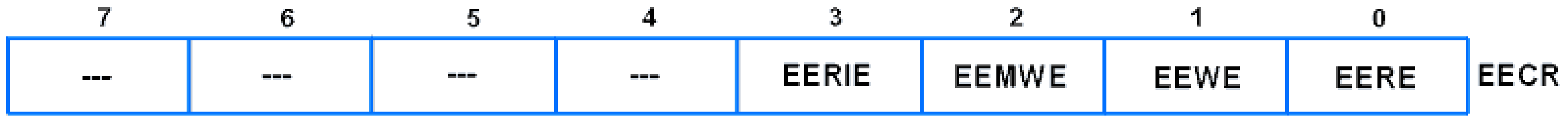
0 = Write disable.

Setting EEWE while EEAR and EEDR contain proper address and data respectively and EEMWE bit is set, perform write operation.

EEPROM Registers

EEPROM Control Register (EECR)

EECR contains control bits to get access over EEPROM.



7	6	5	4	3	2	1	0	
---	---	---	---	EERIE	EEMWE	EEWE	EERE	EECR

Bit 2 – **EEMWE:** EEPROM Master Write Enable

This bit is master enable for write operation.

1 = Setting EEWE within four clock cycles will write data to the EEPROM

0 = Setting EEWE will have no effect.

When EEMWE has been written to one by software, hardware clears the bit to zero after four clock cycles. So it is a must to set EEWE bit within four clock cycles after EEMWE is set to do write operation successfully.

EEPROM Registers

EEPROM Control Register (EECR)

EECR contains control bits to get access over EEPROM.

7	6	5	4	3	2	1	0	
---	---	---	---	EERIE	EEMWE	EEWE	EERE	EECR

Bit 3 – **EERIE:** EEPROM Ready Interrupt Enable.

1 = writing one enables EERIE if the I-bit in SREG is set.

0 = writing zero disables EERIE

It generates a constant interrupt when EEWE is cleared

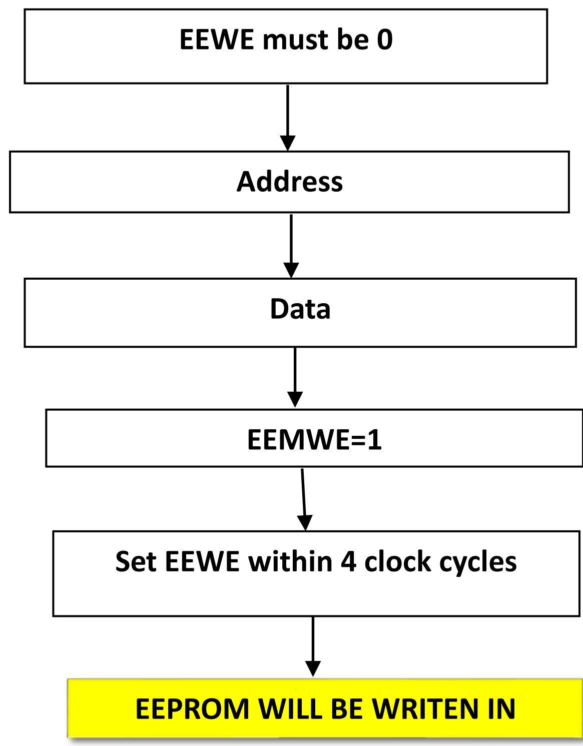
Bit 7:4:

These bits are reserved and always read as zero.

Outlines

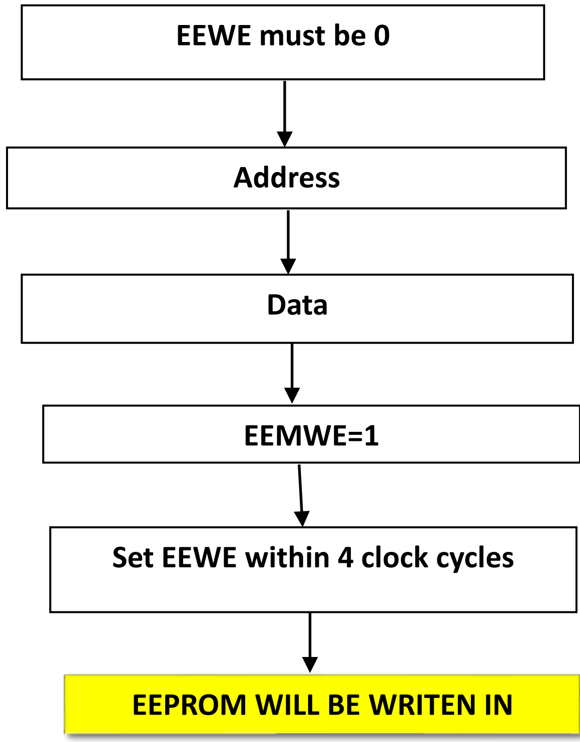
EEPROM Write sequence

1. Wait until EEWE becomes zero.
2. Wait until SPMEN (Store Program Memory Enable) in SPMCR becomes zero.
3. Write EEPROM address to EEAR.
4. Write EEPROM data to EEDR.
5. Write a logical one to the EEMWE bit while writing a zero to EEWE in EECR.
6. Within four clock cycles after setting EEMWE, write a logical one to EEWE.



EEPROM Write sequence

```
void eeprom_write(char data, char address)
{
while(EECR & (1<<EEWE));
 EEAR=address;
 EEDR= data;
 EECR|=(1<<EEMWE);
 EECR|=(1<<EEWE);
 }
```



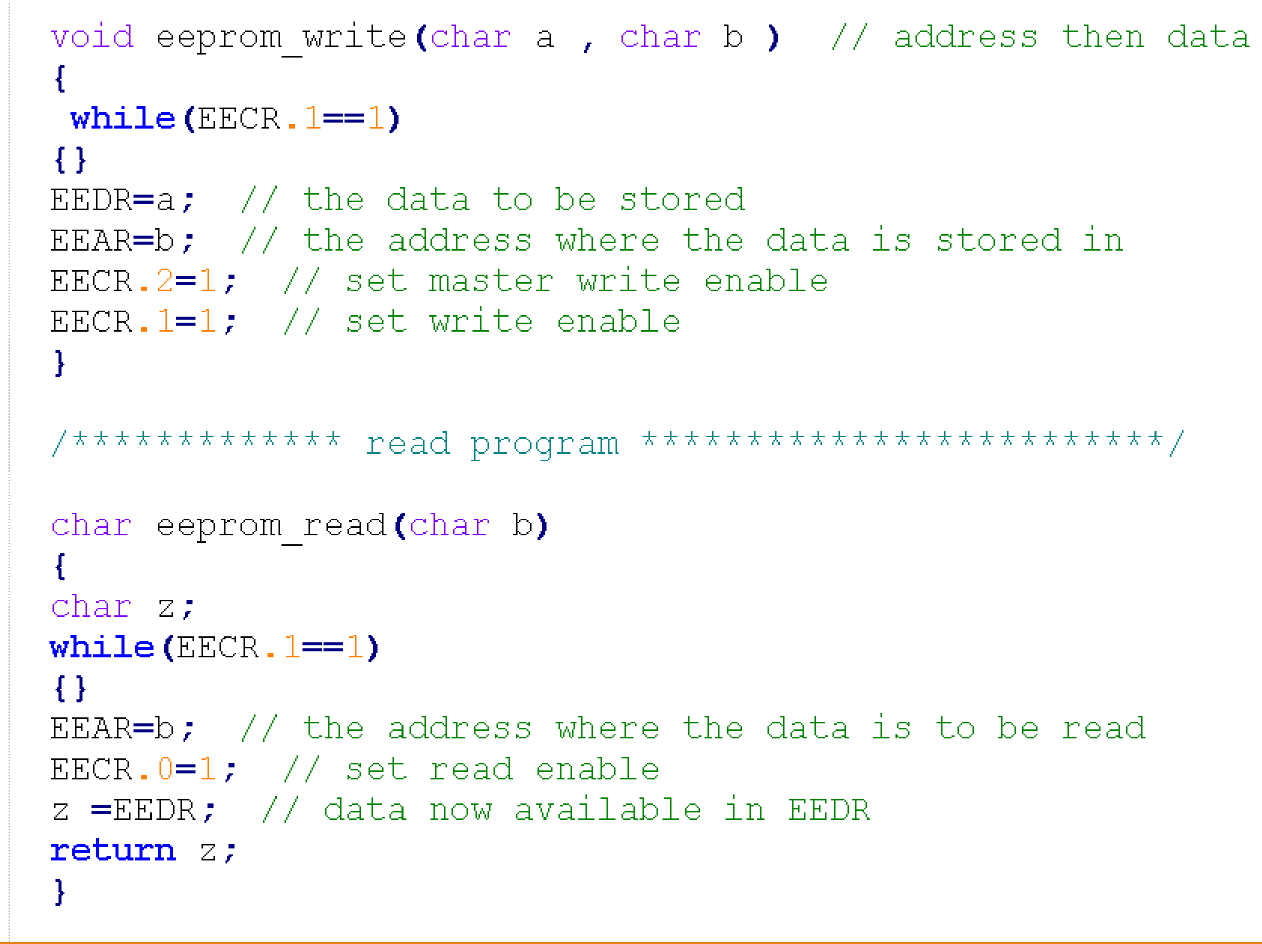
Outlines

EEPROM Read sequence

1. Wait until EEWE becomes zero.
2. Write EEPROM address to EEAR.
3. Write one to EERE to enable read operation from a specified address.
4. Read the EEDR register.

```
char eeprom_read(char address)
{
 char data ;
while(EECR & (1<<EEWE));
 EEAR=address;
EECR|=(1<<EERE);
data =EEDR;
return data;
 }
```

EEPROM Read and write sequence



```
void eeprom_write(char a , char b )  // address then data
{
 while(EECR.1==1)
{}
EEDR=a;  // the data to be stored
EEAR=b;  // the address where the data is stored in
EECR.2=1;  // set master write enable
EECR.1=1;  // set write enable
}

/************* read program ************************/

char eeprom_read(char b)
{
char z;
while(EECR.1==1)
{}
EEAR=b;  // the address where the data is to be read
EECR.0=1;  // set read enable
z =EEDR;  // data now available in EEDR
return z;
}
```

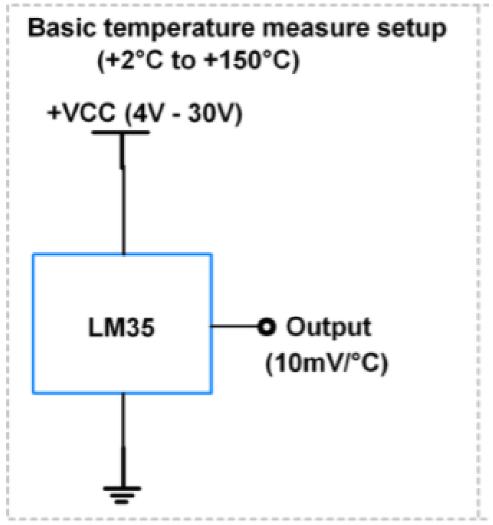
Outlines

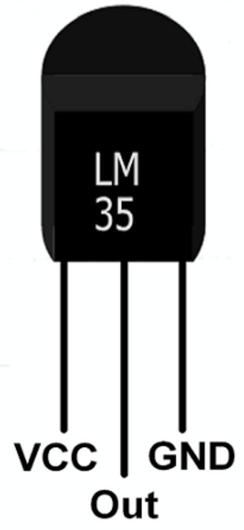
LM35 Temperature Sensor

LM35 is a temperature sensor that can measure temperature in the range of -55°C to 150°C.

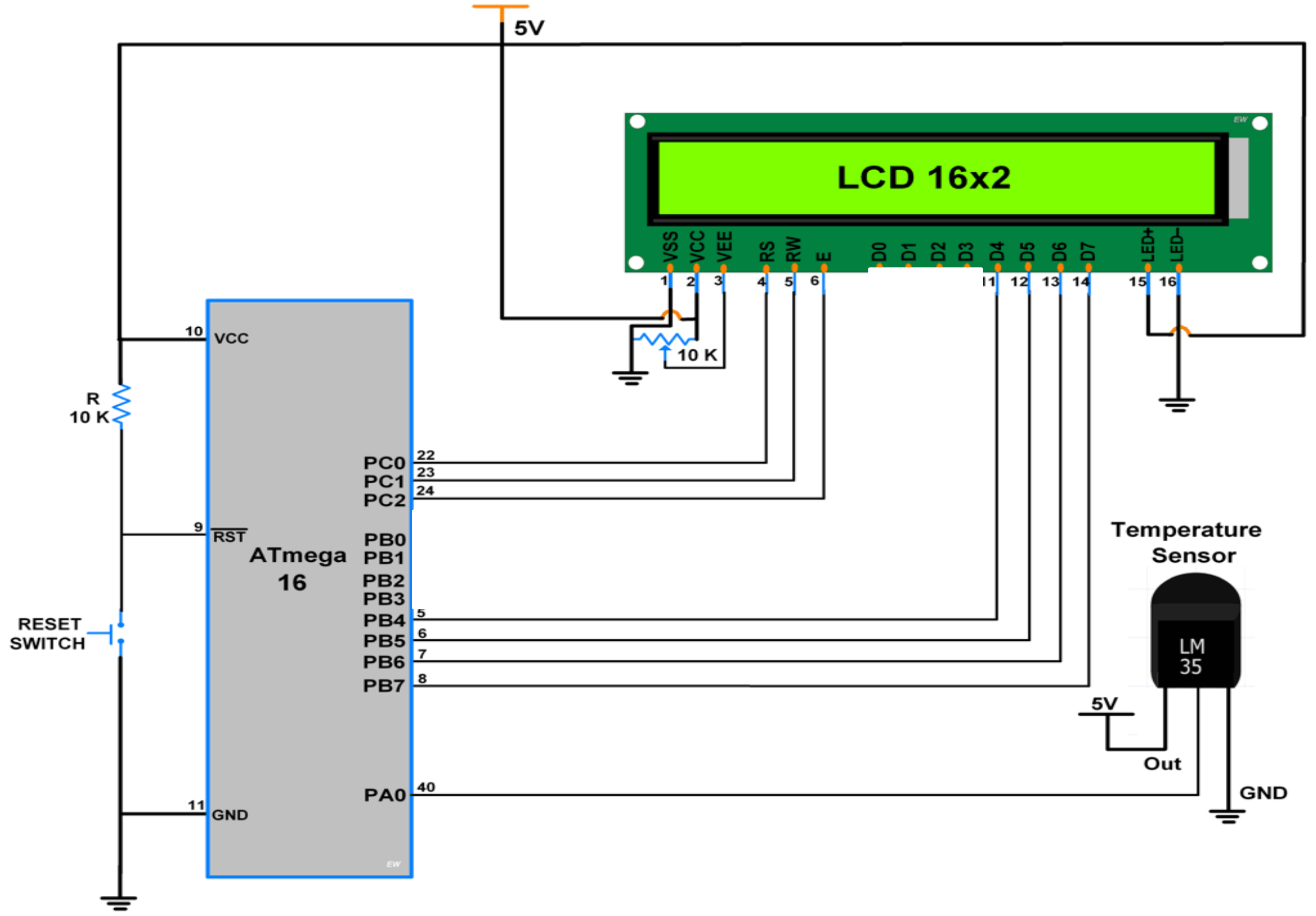
It is a 3-terminal device that provides an analog voltage proportional to the temperature. The higher the temperature, the higher is the output voltage.

The output analog voltage can be converted to digital form using ADC so tha microcontroller can process it.





LM35 Temperature Sensor



LM35 Temperature Sensor

```
#include <mega16.h>
 #include <delay.h>
#include <alcd.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>

int read_adc(char channel);
void eeprom_write(char data, char address);
char eeprom_read(char address);

void main(void)
{
int value;
float celsius;
char String[5];

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 */
        eeprom_write(value, 1);

        celsius = (read_adc(0)*4.88);
        celsius = (celsius/10.00);
        ftoa(celsius,String,2); /* float to string conversion */

         lcd_putsf(“Temp=");
         lcd_puts(String);
         delay_ms(500);



        }
}
```

Thanks