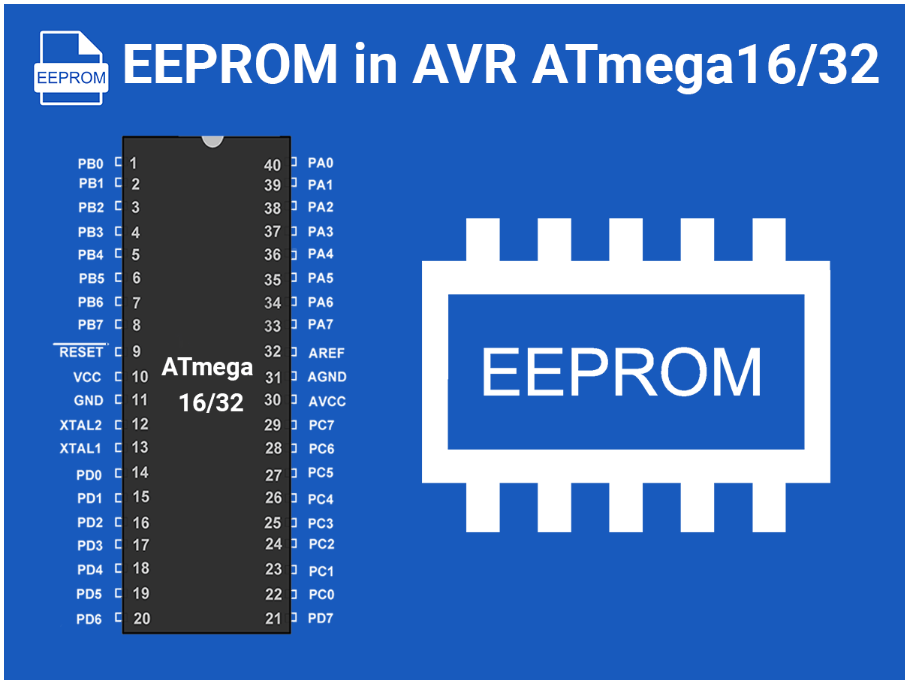
EEPROM in AVR ATmega16/ATmega32



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.

AVR ATmega16 contain 512 bytes of data EEPROM memory. It is organised 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.

Let's see how to access it.

Access over EEPROM memory is made through three registers.

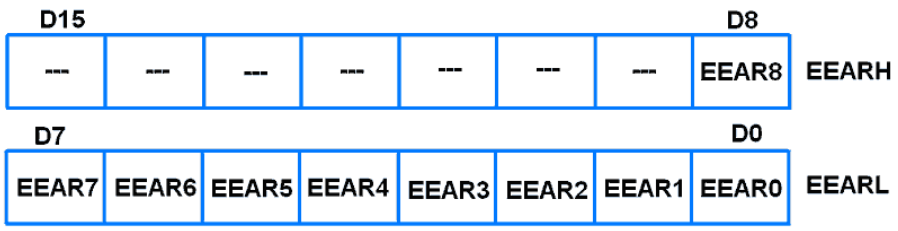
- **EEAR** (EEPROM Address register).
- **EEDR** (EEPROM Data register).
- **EECR** (EEPROM Control register).

EEPROM Address Register (EEAR)

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

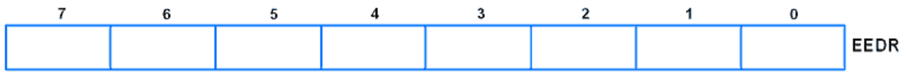
In EEAR, lower 9-bits are used to specify the address and remaining are reserved and will always read as zero.

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


D15
D8
--- --- --- --- --- --- --- EEAR8
EEARH
D7
D0
EEAR7 EEAR6 EEAR5 EEAR4 EEAR3 EEAR2 EEAR1 EEAR0
EEARL

EEAR should be written properly before memory access.

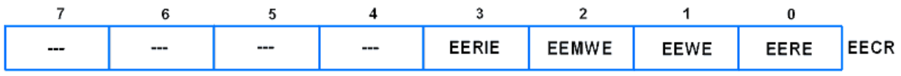
EEPROM Data Register (EEDR)



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

EEPROM Control Register (EECR)

EECR contains control bits to get access over EEPROM.



Bit 7:4:

These bits are reserved and always read as zero.

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 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.

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.

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,

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 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.

In order to prevent unintentional EEPROM writes, the procedure needs to be followed as EEWE must be written to one within the next four-cycle after EEMWE set to one.

Instead of creating EEPROM functions, AVR GCC already provides EEPROM functions. So let's take a look at the EEPROM library.

Using avr-libc EEPROM library function

Atmel studio with GCC compiler provides their avr-libc inbuilt libraries for EEPROM access.

We just need to include the header file of EEPROM in the main program file e.g.

```
#include <avr/eeprom.h>  /* Include AVR EEPROM header file */
```

The header file contains all EEPROM access functions given below,

```
uint8_t eeprom_read_byte (const uint8_t *__p);     /* Read one by
                              EEPROM address */
uint16_t eeprom_read_word (const uint16_t *__p);   /* Read one 16-bit
                              word from EEPROM
                              address */
uint32_t eeprom_read_dword (const uint32_t *__p);          /* Read one 3
                              double word from
                              EEPROM address*/
float eeprom_read_float (const float *__p);        /* Read one float
                              value from EEPROM
                              address */
void eeprom_read_block (void *__dst, const void *__src, size_t __n);/* Read
                              a block of n bytes
                              from EEPROM address
                              to SRAM */
void eeprom_write_byte (uint8_t *__p, uint8_t __value);   /* Write a byte value
                              to EEPROM address */
void eeprom_write_word (uint16_t *__p, uint16_t __value);/* Write a word to
                              EEPROM address */
void eeprom_write_dword (uint32_t *__p, uint32_t __value);/* Write a 32-bit
                              double word to EEPROM
```

Above functions are used to write/read EEPROM. Also there is support for 8, 16, 32-bit data operation with different data types like integer, float, double.

The update function is mostly preferred for writing (updating) EEPROM,

Difference between write function and update function

Update function is able to check whether new data content is different from old data content held at that address. If it differs, then only it writes new data content to that address. This saves write cycles in the case of same data content which helps in maintaining EEPROM lifespan.

Examples

Writing and Reading Byte value

Suppose we want to write a **55** value to address **64** in EEPROM, then we can write it as,

```
#include <avr/eeprom.h>  /* Include AVR EEPROM header file */

void main ( void )
{
        uint8_t ByteOfData ;

        ByteOfData = 0 x55 ;
        eeprom_update_byte (( uint8_t *) 64, ByteOfData );
}
```

Read a value from address **64**, in variable uint8_t value

```
#include <avr/eeprom.h>  /* Include AVR EEPROM header file */

void main ( void )
{
        uint8_t value;
        value = eeprom_read_byte ((const uint8_t*)64);    /* Read value fro
                            64 address */
}
```

Similarly, for reading and writing integer values we can use uint16_t.

Writing and Reading float value

Let's write float value 22.52 at address 20 in EEPROM

```
#include <avr/eeprom.h>  /* Include AVR EEPROM header file */

void main ( void )
{
        float floatvalue;

        floatvalue = 22.52 ;
        eeprom_write_float (( float *) 20, floatvalue);
}
```

The float value requires 4 bytes.

Reading float value from address location 20

```
#include <avr/eeprom.h>  /* Include AVR EEPROM header file */

void main ( void )
{
float floatvalue;

floatvalue =eeprom_read_float(( float *) 20);
}
```

Writing and Reading Block value

Writing an array block "EEPROM TEST" in EEPROM at address 0x0, and reading it back from EEPROM. After reading it is also printed on LCD16x2 for testing.

Program for EEPROM

```
/*
ATmega16_EEPROM.c
http://www.electronicwings.com
*/

#define F_CPU 8000000UL                 /* Define frequency here its 8MHz */
#include <avr/io.h>             /* Include avr std header file */
#include <util/delay.h>             /* Include delay header file */
#include <avr/eeprom.h>             /* Include AVR EEPROM header file */
#include <string.h>             /* Include string header file */
#include "LCD_16x2_H_file.h"          /* Include LCD header file */

int main()
{
        char R_array[15],W_array[15] = "EEPROM TEST";
        LCD_Init();
        memset(R_array,0,15);
        eeprom_busy_wait();                  /* Initialize LCD */
        eeprom_write_block(W_array,0,strlen(W_array));    /* Write W_arr
                                from EEPROM address 0 */
        eeprom_read_block(R_array,0,strlen(W_array));    /* Read EEPR(
```

Components Used

Powered By (https://www.mouser.in?utm_source=electronicswings&utm_medium=display&utm_campaign=mouser-componentslisting&utm_content=0x0)

ATmega 16 ATmega 16	X 1	(https://www.mouser.in/ProductDetail/Microchip-Technology-Atmel/ATMEGA16L-8PU?qs=%2Fha2pyFaduiGCJtTvs2wv8fVZbVAalLu7Iq%2FgITS0tALAx6fMenLvg%3D%3D&utm_source=electronicswings&utm_medium=display&utm_campaign=mouser-componentslisting&utm_content=0x0) Datasheet (/components/atmega-16/1/datasheet)
Atmega32 Atmega32	X 1	(https://www.mouser.in/ProductDetail/Microchip-Technology-Atmel/ATMEGA32-16PU?qs=aqrrBurbvGdpkmgj7RWmsQ%3D%3D&utm_source=electronicswings&utm_medium=display&utm_campaign=mouser-componentslisting&utm_content=0x0) Datasheet (/components/atmega32/1/datasheet)

Downloads

ATmega16 EEPROM proteus simulation file	Dow nloa d (/api/download/platform-attachment/128)
ATmega16 EEPROM Project file	Dow nloa d (/api/download/platform-attachment/314)

Comments

Comment

harish (/users/harish/profile)
2018-12-23 17:05:04

sir i want know how Write W_array from EEPROM address 20 using block

Reply Like 1

authorized (/users/authorized/profile)
2018-12-31 21:18:38

@harish:
i think you need to write like,
eeprom_write_block(W_array, 20, strlen(W_array));
to write eeprom block from 20 address

Reply Like 2

harish (/users/harish/profile)
2019-01-04 19:07:49

ya but i am not able to write it20 address

Reply Like

authorized (/users/authorized/profile)
2019-01-04 20:29:23

Ok then try with argument type i.e.
eeprom_write_block(W_array, (void*)20, strlen(W_array));

Reply Like 1

riddheshmore311 (/users/riddheshmore311/profile)
2022-01-04 12:57:03

Why your application is using 16Bytes of RAM?

Reply Like

About Us (/about)
Business Offering (/business-services)

Connect On:
Facebook(https://www.facebook.com/electronicwings)

LinkedIn(https://www.linkedin.com/company/electronicwin