CHAPITRE 7 EEPROM dans AVR ATmega16 / ATmega32

Introduction

L'EEPROM est une mémoire à lecture seule programmable effaçable électriquement. Il s'agit d'une mémoire de type non volatile car elle contient les données même lorsque l'ordinateur est hors tension.

Le principal avantage de cette mémoire est que le contrôleur peut lire, modifier / écrire cette mémoire dans une application d'exécution. Ainsi, la mémoire EEPROM peut être utilisée pour stocker des valeurs de capteur, des paramètres importants, etc. sans craindre de perte, même en cas de panne de courant.

L'ATMEGA32 contient 1024 octets de données dans la mémoire EEPROM. Elle est organisée comme un espace de donné séparé, dans lequel des octets simples peuvent être lus et écrits. L'EEPROM a une endurance d'au moins 100,000 cycles d’écriture/effacement. Il est organisé comme un espace de données séparé dans lequel un seul octet peut être lu et écrit.

L'accès via la mémoire EEPROM s'effectue via trois registres.

* **EEAR** (registre d'adresses EEPROM).
* **EEDR** (registre de données EEPROM).
* **EECR** (registre de contrôle EEPROM).

 **Registre d'adresses EEPROM (EEAR)**

Atmega32 a un registre EEAR 16 bits qui est utilisé pour adresser l'emplacement de la mémoire EEPROM.

Dans EEAR, les 10 bits les plus bas sont utilisés pour spécifier l'adresse. Les autres sont réservés et seront toujours lus à zéro.

Il possède deux registres de 8 bits, EEARH et EEARL. EEARL contient le premier bit d'adresse 8 et EEARH contient le dernier bit 10 d'adresse, comme indiqué sur la figure.



EEAR doit être écrit correctement avant l'accès à la mémoire.

 **Registre de données EEPROM (EEDR)**

****

L'EEDR contient des données à écrire / lire à l'emplacement adressé par EEAR lors d'une opération d'écriture / lecture.

 **Registre de contrôle EEPROM (EECR )**

EECR contient des bits de contrôle permettant d’avoir accès via EEPROM.



**Bit 7: 4:**

        Ces bits sont réservés et toujours lus à zéro.

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

**1 = l'** écriture 1 active EERIE si le bit I dans SREG est activé.

**0 = l'** écriture zéro désactive EERIE

Il génère une interruption constante lorsque EEWE est effacé

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

       Ce bit est activé par le maître pour l'opération d'écriture.

**1** = configurer EEWE dans un délai de quatre cycles d'horloge écrira des données sur l'EEPROM

**0** = Le réglage de EEWE n'aura aucun effet.

Lorsque EEMWE a été écrit dans un logiciel par un programme, le matériel remet le bit à zéro après quatre cycles d'horloge. Il est donc indispensable de définir le bit EEWE dans les quatre cycles d'horloge après le paramétrage d'EEMWE pour effectuer l'opération d'écriture avec succès.

**Bit 1** -  **EEWE** : EEPROM Write Enable

   1 = autorisation d'écriture.

**0** = écriture désactivée.

Si vous définissez EEWE alors que EEAR et EEDR contiennent l'adresse et les données correctes et que le bit EEMWE est activé, effectuez une opération d'écriture.

**Bit 0** - **EERE:** **EEPROM Read Enable**

    **1** = autorisation de lecture.

     **0** = lecture désactivée.

Le réglage de EERE alors que EEAR contient l’adresse appropriée permet la lecture.

Séquence d'écriture EEPROM

1. Attendez que EEWE devienne nul.
2. Attendez que SPMEN (Mémoire programme activé) dans SPMCR devienne nul.
3. Écrivez l'adresse EEPROM dans EEAR.
4. Ecrire des données EEPROM à EEDR.
5. Ecrivez un 1 logique dans le bit EEMWE en écrivant un zéro dans EEWE dans EECR.
6. Dans les quatre cycles d'horloge suivant le réglage de EEMWE, écrivez 1 logique dans EEWE.

 Séquence de lecture EEPROM

1. Attendez que EEWE devienne nul.
2. Écrivez l'adresse EEPROM dans EEAR.
3. Ecrivez-en un sur EERE pour activer l'opération de lecture à partir de l'adresse spécifiée.
4. Lire le registre EEDR.

Pour éviter les écritures EEPROM non intentionnelles, vous devez suivre cette procédure, car EEWE doit être écrit dans l'un au cours du prochain cycle, après la définition de EEMWE sur 1

**Utilisation de la fonction de bibliothèque EEPROM avr-libc**

Le studio Atmel avec le compilateur GCC fournit ses bibliothèques intégrées apr-libc pour un accès EEPROM.

Nous avons juste besoin d'inclure le fichier d'en-tête de l'EEPROM dans le fichier du programme principal, par exemple

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

Le fichier d’en-tête contient toutes les fonctions d’accès EEPROM indiquées ci-dessous,

uint8\_t eeprom\_read\_byte (const uint8\_t \*\_\_p); /\* Read one byte from 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 32-bit 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 address \*/

void eeprom\_write\_float (float \*\_\_p, float \_\_value); /\* Write float value to EEPROM address \*/

void eeprom\_write\_block (const void \*\_\_src, void \*\_\_dst, size\_t \_\_n);/\* Write a block of bytes to EEPROM address \*/

void eeprom\_update\_byte (uint8\_t \*\_\_p, uint8\_t \_\_value);/\* Update byte value to EEPROM address \*/

void eeprom\_update\_word (uint16\_t \*\_\_p, uint16\_t \_\_value);/\* Update word val to EEPROM address \*/

void eeprom\_update\_dword (uint32\_t \*\_\_p, uint32\_t \_\_value);/\* Update a 32-bit double word value to EEPROM address \*/

void eeprom\_update\_float (float \*\_\_p, float \_\_value); /\* Update float value to EEPROM address \*/

void eeprom\_update\_block (const void \*\_\_src, void \*\_\_dst, size\_t \_\_n);/\* Up- date a block of bytes to EEPROM address \*/

Les fonctions ci-dessus sont utilisées pour écrire / lire une EEPROM. Il existe également une prise en charge des opérations de données 8, 16 et 32 ​​bits avec différents types de données tels que integer, float, double.

La fonction de mise à jour est principalement recommandée pour l'écriture (mise à jour) de l'EEPROM,

**Différence entre la fonction write et la fonction update**

La fonction de mise à jour permet de vérifier si le nouveau contenu de données est différent de l'ancien contenu de données conservé à cette adresse. S'il diffère, alors il écrit uniquement le nouveau contenu de données à cette adresse. Cela évite les cycles d'écriture dans le cas d'un contenu de données identique, ce qui aide à maintenir la durée de vie de l'EEPROM.

###  **EEPROM carte Arduino**

Pour manipuler des données en mémoire EEPROM, il est nécessaire d'utiliser la [bibliothèque EEPROM](https://www.arduino.cc/en/Reference/EEPROM) disponible de base avec le logiciel de développement Arduino.

Pour utiliser la bibliothèque EEPROM dans un programme Arduino, il suffit d'ajouter cette ligne en début de programme :

|  |  |
| --- | --- |
| 1 | #include *<EEPROM.h>* |

### **Écrire dans la mémoire EEPROM**

Écrire un octet en mémoire ce fait au moyen de la fonction [EEPROM.write()](https://www.arduino.cc/en/Reference/EEPROMWrite).

|  |  |
| --- | --- |
| 1 | EEPROM.write(int adresse, byte valeur) |

La fonction [EEPROM.write()](https://www.arduino.cc/en/Reference/EEPROMWrite) accepte deux paramètres obligatoires : l'adresse mémoire de l'octet à écrire (débutant à 0) et la valeur de l'octet en question (entre 0 et 255). La fonction ne retourne aucune valeur.

### **Lire depuis la mémoire EEPROM**

Lire un octet en mémoire ce fait au moyen de la fonction [EEPROM.read()](https://www.arduino.cc/en/Reference/EEPROMRead).

|  |  |
| --- | --- |
| 1 | byte EEPROM.read (int adresse) |

La fonction [EEPROM.read()](https://www.arduino.cc/en/Reference/EEPROMRead) accepte un paramètre obligatoire : l'adresse mémoire de l'octet à lire (débutant à 0). La fonction retourne la valeur de l'octet à l'adresse en question (entre 0 et 255).

*PS : Vous pouvez lire une mémoire EEPROM autant de fois que vous le souhaitez, seule l'écriture réduit sa durée de vie.*

Par exemple, voici comment lire le contenu des 256 premiers octets de mémoire :

|  |  |
| --- | --- |
|  1 2 3 4 5 6 7 8 91011121314 | #include *<EEPROM.h>*void setup() { Serial.begin(9600); **for** (int i = 0; i < 256; i++) { byte valeur = EEPROM.read(i); Serial.print(i); Serial.print(" = "); Serial.println(valeur); }}void loop() {}  |

### **Mettre à jour les données efficacement**

Imaginons que la mémoire EEPROM de votre carte Arduino contient la suite d'octets suivante : 1 2 3 4 5 6.

Le temps passe et vous voulez maintenant changer ces valeurs par 1 2 3 7 8 9, comme vous pouvez le remarquer, certaines valeurs sont identiques, il serait donc idiot de perdre un cycle d'écriture pour écrire une valeur identique à la précédente.

C'est pourquoi il existe la fonction [EEPROM.update()](https://www.arduino.cc/en/Reference/EEPROMUpdate). Celle-ci est identique à la fonction EEPROM.write() mais vérifie d'abord la valeur en mémoire avant de faire une écriture. Si la valeur est différente, l'écriture est réalisée, sinon, la fonction ne fait rien.

*PS : Idéalement, cette fonction devrait toujours être utilisée, en remplacement de EEPROM.write(), car celle-ci fonctionne de manière identique, mais évite les écritures inutiles.*

|  |  |
| --- | --- |
| 1 | EEPROM.update (int adresse, byte valeur) |

La fonction [EEPROM.update()](https://www.arduino.cc/en/Reference/EEPROMUpdate) accepte deux paramètres obligatoires : l'adresse mémoire de l'octet à écrire (débutant à 0) et la valeur de l'octet en question (entre 0 et 255). La fonction ne retourne aucune valeur.

### **Connaitre la taille de la mémoire EEPROM depuis le programme**

Cela n'est pas documenté officiellement, mais depuis Arduino 1.0.x, il est possible de connaitre la taille de la mémoire EEPROM via la fonction : EEPROM.length().

La fonction EEPROM.length() retourne un nombre entier qui correspond à la taille de la mémoire EEPROM.

Exemple :

|  |  |
| --- | --- |
|  1 2 3 4 5 6 7 8 91 | #include *<EEPROM.h>*void setup() { Serial.begin(9600); Serial.print("Taille EEPROM : "); Serial.println(EEPROM.length());}void loop() {}  |

Référencées :

1/ Fiche technique ATmega16 et ATmega32

2/ www.arduino.cc

3/ <https://www.carnetdumaker.net/articles/stocker-des-donnees-en-memoire-eeprom-avec-une-carte-arduino-genuino/>

4/ http://www.electronicwings.com