EEPROM

Utiliser la mémoire EEPROM interne de l'Arduino

fonctionnement d une EEPROM

https://electroniqueamateur.blogspot.com/2020/01/utiliser-la-memoire-eeprom-interne-de.html

Les cartes Arduino disposent d'une mémoire EEPROM (“Electrically-Erasable Programmable Read-Only Memory”): il s'agit d'un endroit où vous pouvez stocker des informations qui demeureront disponibles même après que l'Arduino ait été mis hors tension pendant un certain temps, ou après que vous ayez téléversé un nouveau sketch dans l'Arduino.

La mémoire EEPROM pourrait servir, par exemple, à conserver en mémoire un numéro permettant d'identifier de façon unique une carte Arduino, les préférences de l'utilisateur, les paramètres de calibration d'un capteur, etc.

Attention, cependant, à une contrainte importante: le nombre d'écritures sur une même adresse de la mémoire EEPROM est limitée à environ 100 000. Ça peut sembler énorme à première vue, mais tout dépend de l'utilisation que vous en faites... Si vous écrivez continuellement une nouvelle information une fois par heure, la mémoire EEPROM devrait théoriquement pouvoir tenir le coup pendant un peu plus de 11 ans. Mais cette durée de vie passe à environ 2 mois si l'information est réécrite une fois par minute, et à environ une journée si l'information est mise à jour à chaque seconde! Précisons toutefois que cette estimation de 100 000 écritures est généralement considérée comme extrêmement sévère: dans la vraie vie, l'EEPROM de votre Arduino demeurera probablement fonctionnel beaucoup plus longtemps.

La mémoire EEPROM interne de la carte Arduino Uno est de 1 ko, ce qui signifie qu'on dispose de 1024 adresses (numérotées de 0 à 1023) pouvant chacune stocker un octet (donc un nombre situé entre 0 et 255). La taille disponible dépend du modèle de carte: elle est de 4 ko pour l'Arduino Mega, par exemple.

La bibliothèque EEPROM

Pour écrire ou lire l'information sur la mémoire EEPROM interne de l'Arduino, nous utiliserons la bibliothèque EEPROM. Aucune installation n'est nécessaire, puisqu'elle est présente par défaut dans l'IDE Arduino. Il est toutefois important de déclarer cette bibliothèque au début de votre sketch:

La bibliothèque est accompagnée d'une bonne quantité d'exemples fort instructifs, que vous trouverez dans le menu “Fichier / Exemples”.

Lire un octet: la méthode EEPROM.read()

Pour lire l'une des 1024 valeurs stockées dans l'EEPROM, on utilise la méthode EEPROM read:

```
EEPROM.read(adresse)
```

...où l'adresse passée en paramètre est un nombre situé entre 0 et 1023 (s'il s'agit d'une carte

Arduino Uno). Ainsi, pour lire la valeur enregistrée à l'adresse 55, on pourrait écrire:

```
valeur = EEPROM.read(55);
```

À titre d'exemple, je vous suggère de jeter un oeil sur l'exemple intitulé “EEPROM_read()”, accessible dans l'IDE Arduino par le menu “Fichier / Exemples / EEPROM” . Ce sketch affiche dans le moniteur série la valeur stockée à chaque adresse de la mémoire EEPROM. Si rien n'a jamais été enregistré dans votre mémoire EEPROM, chacune des adresse contient initialement la valeur 255.

Écrire un octet: la méthode EEPROM.write()

Pour écrire une information dans l'EEPROM, on peut utiliser la méthode EEPROM.write():

```
EEPROM.write(adresse, valeur)
```

...où “adresse” est l'adresse à laquelle nous désirons écrire l'information (c'est un nombre entre 0 et 1023 pour l'Arduino Uno), et “valeur” est l'information que nous désirons enregistrer à cet endroit (un nombre entre 0 et 255).

Par exemple, l'instruction suivante enregistrera le nombre “23” à l'adresse numéro 4 de l'EEPROM:

```
EEPROM.write(4,23);
```

Si, après avoir exécuté cette instruction, j'exécute à nouveau l'exemple “EEPROM_read()”. je constate que l'adresse 4 contient maintenant le nombre 23, plutôt que le nombre 255 qu'il contenait auparavant.

Vous pourriez en principe pouvoir faire tout ce que vous voulez en vous limitant à l'utilisation des méthodes EEPROM.write() et EEPROM.read(). La bibliothèque EEPROM met toutefois à notre disposition quelques méthodes supplémentaires, afin de nous simplifier la vie.

Écrire un octet, mais seulement si nécessaire: la méthode EEPROM.update()

Comme je le mentionnais au tout début de cet article, le nombre total d'écritures d'une mémoire EEPROM est limité. Il serait donc dommage de réduire inutilement la durée de vie d'une mémoire EEPROM en y écrivant une valeur identique à celle qui s'y trouvait déjà.

La méthode EEPROM.update() est donc identique à la méthode EEPROM.write(), sauf que la nouvelle valeur sera écrite uniquement si elle est différente de la valeur déjà présente à cette adresse.

```
EEPROM.update(adresse, valeur)
```

Par exemple, l'instruction EEPROM.update(22,144) inscrira la valeur “144” à l'adresse “22”, mais seulement si la valeur déjà stockée à l'adresse 22 est différente de 144.

Personnellement, je ne vois aucune raison valable de ne pas utiliser EEPROM.update() plutôt qu'EEPROM.write() chaque fois que vous désirez écrire une valeur en mémoire.

Écrire une variable de n'importe quel type:

```
EEPROM.put()
```

Les 3 méthodes que nous avons explorées jusqu'à maintenant impliquent l'enregistrement ou la lecture d'un octet: un nombre à 8 bits dont la valeur se situe entre 0 et 255.

Imaginez que vous désirez stocker en mémoire EEPROM l'état d'une entrée analogique de l'Arduino: il s'agit d'une valeur à 10 bits pouvant aller de 0 à 1023, ce qui est trop grand pour être exprimé par un seul octet.

Pour contourner ce problème, vous pourriez diviser votre mesure par 4 avant de la stocker en mémoire (ce qui aurait pour effet négatif de diminuer la résolution de votre mesure), ou encore utiliser les opérateurs bit à bit pour séparer manuellement le nombre à 10 bits en deux octets différents.

Mais rien de tout ça ne sera nécessaire si vous utilisez la méthode EEPROM.put(), spécialement conçue pour stocker facilement une variable de n'importe quel type (int, long, float, etc.) ou même une structure (struct) constituée de plusieurs types différents.

La syntaxe est similaire à celle d'EEPROM.write():

```
  EEPROM.put(adresse, valeur)
```

Dans l'exemple ci-dessous, j'enregistre une variable de type “long” contenant le nombre 123456789 à l'adresse 4 de l'EEPROM:

Lire une variable de n'importe quel type: la méthode EEPROM.get()

Le complément de la méthode EEPROM.put() est EEPROM.get(), qui nous permet de lire sur l'EEPROM la valeur d'une variable de n'importe quel type.

La syntaxe est:

```
  EEPROM.get(adresse, valeur)
```

Par exemple, dans le sketch ci-dessous, je récupère la valeur de la variable de type “long” précédemment enregistrée à l'adresse 4, et je l'affiche dans le moniteur série.

Lors de l'exécution de ce sketch, le moniteur série a affiché 123456789, puisque j'avais précédemment enregistré cette valeur grâce à la méthode EEPROM.put().

Mais attention! Puisque tout à été fait de façon automatique par les méthodes EEPROM.put() et EEPROM.get(), il serait dangereux d'oublier que notre variable de type “long” est un nombre à 32 bits. Elle ne peut donc pas avoir été enregistré uniquement à l'adresse 4, qui ne peut contenir que 8 bits.

Pour en avoir le coeur net, j'utilise encore une fois l'exemple EEPROM_read, fourni avec l'IDE Arduino, qui présente dans le moniteur série le contenu de toutes les adresses:

Comme vous pouvez le constater, lorsque j'ai demandé de stocker la variable de type long à l'adresse 4 par la méthode EEPROM.put(), 4 octets ont été modifiés: ceux situés aux adresses 4, 5, 6 et 7.

L'adresse 4 contient le nombre décimal 21, ou 00010101 en binaire. L'adresse 5 contient le nombre décimal 205, soit 11001101 en binaire. L'adresse 6 contient le nombre décimal 91, ou 01011011 en binaire. L'adresse 7 contient le nombre décimal 7, soit 00000111 en binaire.

En mettant ces 4 octets bout à bout pour obtenir un nombre à 32 bits, en commençant par l'adresse 7

et en terminant par l'adresse 4, nous obtenons: 00000111010110111100110100010101 , qui correspond à la valeur décimale 123456789.

Il aurait donc été catastrophique, après avoir stocké notre variable à l'adresse 4, d'en enregistrer une autre à l'adresse 5, en oubliant que cette adresse est déjà occupée!

Utiliser l'EEPROM comme un tableau d'octets: l'objet EEPROM[]

Un objet EEPROM[] vous permet d'écrire et lire des octets comme si la mémoire EEPROM était un tableau d'octets.

Ainsi, l'expression “valeur = EEPROM[5]” aura le même effet que l'expression “valeur = EEPROM.read(5)”,

et l'expression “EEPROM[7] = 123” aura le même effet que l'expression “EEPROM.update(7, 123)”.

Connaître la taille de l'EEPROM grâce à

```
EEPROM.length()
```

Tel que précisé un peu plus haut, la taille de la mémoire EEPROM n'est pas la même pour tous les modèles d'Arduino. La méthode EEPROM.length() retourne cette taille (en octets), ce qui permet à votre sketch d'utiliser la totalité de la mémoire disponible, peu importe le modèle de carte utilisé.

Exemple de programme de test

EEPROM_Structure.ino

```
//------------------Test structure  -- EEPROM -------------
typedef struct  {
  char a[32];
  int b;
  float c;
  String d;
  bool e;
} sensor;

 char g[20] = " ";

void setup() {
  Serial.begin(115200);

  sensor mySensor;

  char  g[] = "bonjour ca va bien";

  /*
  for (int i=0; i <7 ;i++){
  mySensor.a[i]  =  g[i];
  }
```

```
  */
  strcpy(mySensor.a,g);
  mySensor.b = 255;
  mySensor.c= 20.4;
  mySensor.d = "autre nom";
  mySensor.e = 1;

  Serial.println(sizeof(g));
  Serial.println("--------------");
  Serial.println(sizeof(mySensor.a));
  Serial.println("--------------");
   for (int j = 0 ; j < sizeof(g) ; j++ ) {
       Serial.print(mySensor.a[j]);
   }
  Serial.println(" ");
  Serial.println(mySensor.b);
  Serial.println(mySensor.c);
  Serial.println(mySensor.d);
  Serial.println(mySensor.e);

}

void loop() {}
```

Tableau_pointeur.ino

```
#include <stdio.h>

int main(void) {

    // Pointeur sur tableau de char
    char * ptr = "Foobar";
    printf("sizeof(ptr) = %d\n", sizeof(ptr));

    // Tableau de char
    char array[] = "Foobar";
    printf("sizeof(array) = %d\n", sizeof(array));
    return 0;
```

EEPROM et esp32

Liens web

EEPROM sur Locoduino

EEPROM sur Arduino

EEPROM I2C 2024LC512 64 Ko

From:
http://chanterie37.fr/fablab37110/ - **Castel'Lab le Fablab MJC de Château-Renault**

Permanent link:
http://chanterie37.fr/fablab37110/doku.php?id=start:arduino:eeprom&rev=1608315735

Last update: **2023/01/27 16:08**