

Mémoires permanentes

Pierre-Yves Rochat, EPFL

rév 2015/09/18

Types de mémoires

Tous les microcontrôleurs disposent de deux types de mémoire :

- la mémoire Flash, prévue principalement pour recevoir le programme, qui est permanente
- la mémoire vive (RAM = Random Access Memory), qui perd son contenu lorsque le circuit n'est plus alimenté.

Pour commander des enseignes et afficheurs à LED, on placera souvent en mémoire non volatile non seulement le programme proprement dit, mais aussi les informations sur le contenu qui sera visualisé. En effet, les séquences d'une enseigne ou les textes d'un afficheur doivent être mémorisés de manière non-volatile, pour leur assurer un fonctionnement correct après une coupure de courant.

Une particularité de la mémoire Flash est son organisation en blocs. S'il est possible d'écrire une valeur dans une position mémoire précise, il n'est pas possible d'effacer une position mémoire seule. Les effacements se font toujours par bloc. La taille d'un bloc varie considérablement d'un microcontrôleur à un autre, parfois seulement 64 octets, jusqu'à plusieurs dizaines de kB.

Les mémoires EEPROM

Certains microcontrôleurs disposent d'un troisième type de mémoire, en plus de la Flash et de la RAM. Il s'agit de mémoires EEPROM. Ce sont aussi des mémoires non-volatiles, mais chaque position mémoire peut être écrite et effacée indépendamment des autres. On trouve ce type de mémoire dans les microcontrôleurs AVR, dont l'ATmega328, bien connu pour être utilisé dans les Arduino.

L'accès à l'EEPROM interne d'un microcontrôleur est généralement très simple. Des registres sont prévus pour l'adresse (EEAR) et pour la donnée (EEDR), en

lecture ou en écriture. Sur les AVR, deux registres donnent l'accès aux adresses et aux données :

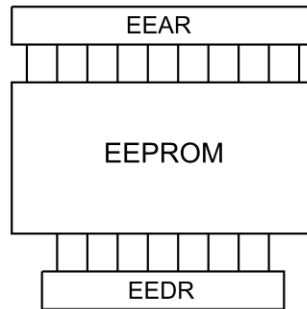


Figure 1: Accès à l'EEPROM sur les AVR

Voici les instructions pour lire et écrire dans la mémoire :

```
// Lecture en EEPROM :
EEAR = adresse; // l'adresse est donnée
EECR = (1<<EERE); // le fanion de lecture est activé
valeur = EEDR; // lecture de la valeur

// Ecriture en EEPROM :
while (EECR & (1<<EEPE)) {} // attente de la fin d'une éventuelle écriture précédente
EEAR = adresse; // l'adresse est donnée
EEDR = valeur; // la valeur est donnée
EECR = (1<<EEMPE); // autorise une écriture (Master Write Enable)
EECR = (1<<EEPE); // lance le cycle d'écriture (Write Enable)
```

Le fanion EEMPE signifiait EEprom Master Program Enable. Mais les lettres PE sont traduites dans la documentation actuelle par Write Enable. EEMPE doit être activé juste avant l'activation du fanion EEPE (EEPROM Write Enable) qui lance le cycle d'écriture. Son activation ne dure que quelques cycles d'horloge : il est automatiquement remis à zéro. Aucune écriture n'est possible s'il n'est pas actif. Son but est de rendre improbable une écriture accidentelle en EEPROM. L'activation des EEMPE et de EEPE doivent en effet se suivre de près pour qu'une écriture soit possible.

Le fanion EEPE est remis à zéro automatiquement par le microcontrôleur lorsque le cycle d'écriture est terminé. Alors que la lecture est immédiate, il faut se souvenir que l'écriture prend un temps relativement long, de l'ordre de quelques millisecondes.

Accès à la mémoire Flash par programmation

Pour y placer le programme, la mémoire Flash du microcontrôleur est accédée de l'extérieur, au moyen d'un programmeur. S'il reste de la place dans la mémoire Flash, il est possible de l'utiliser depuis le programme pour y placer des données. La manière de le faire dépend de l'architecture du microcontrôleur. Il faut se référer à la documentation du microcontrôleur pour avoir les détails.

Sur les MSP430, l'architecture de Von Neumann rend facile l'accès à la mémoire Flash. La lecture se fait simplement en indiquant l'adresse de la position mémoire :

```
// Lecture en Flash :  
uint8_t *pointeur; // pointeur dans la Flash  
pointeur = (uint8_t *) 0x1040; //place l'adresse dans le pointeur  
uint8_t valeur = *pointeur;
```

On remarque le trans-typage de l'adresse, indiquée ici en hexadécimal, vers le type du pointeur, qui est dans ce cas un pointeur vers des valeurs 8 bits non-signées (uint8_t *).

L'écriture se fait de la même manière. Il faut toutefois précéder l'écriture par la désactivation du fanion qui bloque l'écriture en mémoire Flash. Son rôle est de rendre improbable des effacements accidentels de la mémoire.

```
// Ecriture en Flash :  
FCTL3 = FWKEY; // Clear Lock bit  
*pointeur = valeur; // écrit la valeur dans la Flash  
FCTL3 = FWKEY + LOCK; // Set LOCK bit
```

L'effacement s'effectue par bloc. Il s'effectue lorsqu'une écriture est faite dans la zone du bloc alors que le fanion d'effacement est activé. Mais il est nécessaire de désactiver au préalable le fanion qui bloque toute écriture en mémoire Flash.

```
// Effacement d'un bloc de la mémoire Flash  
FCTL1 = FWKEY + ERASE; // Set Erase bit  
FCTL3 = FWKEY; // Clear Lock bit  
*pointeur = 0; // lance un cycle d'effacement du bloc, la valeur donnée n'a pas d'importance  
FCTL3 = FWKEY + LOCK; // Set LOCK bit  
FCTL1 = FWKEY; // Clear WRT bit
```

Souvent, des bibliothèques sont disponibles pour faciliter ce travail. Pour les AVR, construits sur une architecture de Harvard, la bibliothèque `PgmSpace.h` est utilisée.

Limite du nombre de cycles d'écriture

Toutes les mémoires non-volatiles ont une limite dans le nombre d'écriture ou d'effacements. Pour les Flash, la limite est souvent de 10'000 cycles. S'il ne s'agit que de programmer le microcontrôleur, ce nombre semble très grand : qui va programmer 10'000 fois un même microcontrôleur ?

Toutefois... à l'occasion du MOOC Comprendre les microcontrôleurs, proposé par l'EPFL sur Coursera dès 2013, nous avons mis au point un système de correction de devoirs. Il a corrigé à ce jour plus de 70'000 devoirs. Les microcontrôleurs ont donc été changés de temps en temps, pour éviter des pannes.

Les EEPROM sont souvent capables d'un nombre de cycles d'écriture plus grand que les Flash. L'ordre de grandeur est de 100'000 cycles ou davantage. Mais attention, ce nombre est très vite atteint si certaines précautions ne sont pas prises. Remarquez par exemple qu'en écrivant une valeur chaque seconde, on atteint la limite peu après un jour de fonctionnement.

Mémoires externes

Il est possible d'ajouter de la mémoire non-volatile à un microcontrôleur en utilisant des circuits mémoire externes. On choisira un circuit en fonction de la taille des données à mémoriser. Les solutions les plus courantes sont :

- les mémoires EEPROM, souvent disponibles dans des boîtiers à 8 broches (DIP 8 ou modèles correspondant en SMD)
- les cartes SD (ou microSD).

Au moyen d'une autre technologie, les circuits horloges temps réel offrent souvent quelques positions mémoires RAM. Or celles-ci sont rendues non-volatiles par la pile qui maintient le circuit en fonctionnement permanent.

EEPROM I2C ou SPI

Plusieurs fabricants proposent des familles de mémoires EEPROM, tels que Microchip, Atmel, STmicro, Winbond, etc. Le dialogue entre ces mémoires et le microcontrôleur se fait en série, le plus souvent par des signaux aux normes I2C ou SPI.

Par exemple, la mémoire M95256 de STmicro a une capacité de 256 kb. Attention, la lettre b est en minuscule, ce qui signifie bit. En d'autres termes, c'est une mémoire de 32 kB (kilo Bytes). Elle utilise les signaux SPI. Voici un schéma de mise en œuvre :

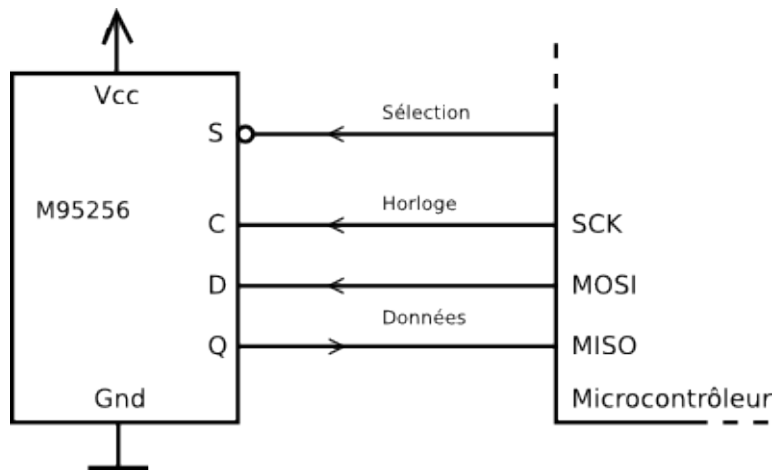


Figure 2: Mise en œuvre d'une EEPROM

Pour dialoguer avec ces circuits pour lire ou écrire des valeurs en mémoire, on trouve facilement sur Internet des librairies prêtes à l'emploi. La lecture des sources permet d'en comprendre le fonctionnement.

Il faut être attentif à un point important. Presque tous les microcontrôleurs sont équipés de contrôleurs spécialisés pour I2C ou pour SPI. Pour pouvoir utiliser ces contrôleurs, il faut impérativement utiliser les broches choisies à cet effet par le fabricant. On n'est donc pas libre de choisir les broches.

Bien entendu, il est possible de programmer les procédures I2C ou SPI en *bit banging*, c'est-à-dire en agissant directement sur les broches d'entrée-sortie. Dans ce cas, on a toute liberté dans le choix des broches. L'usage des contrôleurs intégrés a deux avantages. D'une part, le travail de programmation est simplifié. Mais surtout, ces contrôleurs permettent d'atteindre des vitesses de transfert plus élevées. Pour les afficheurs à LED de grande taille, c'est un aspect très important.

On trouve des mémoires EEPROM dont les capacités vont de quelques dizaines d'octets jusqu'à des dizaines de méga-octets.

Cartes SD

Pour des capacités plus grandes, on choisira des cartes SD. Elles utilisent en interne des mémoires Flash, mais un contrôleur s'occupe de gérer l'accès aux blocs. Il gère également les mauvais blocs, qui peuvent souvent apparaître avec ce type de mémoires.

Les cartes SD seront aussi utilisées lorsqu'il est nécessaire de déplacer la mémoire pour la connecter à un PC. Dans le cas d'un afficheur à LED, on peut envisager

de mettre le contenu à afficher dans une cartes SD à partir d'un PC, puis de placer la carte dans l'afficheur.

La capacité des cartes SD est très importante, aujourd'hui jusqu'au TB (1 tera octet = 1'000'000'000'000 octets).

Système de fichier

Pour placer ou lire des données sur une carte SD, on choisit généralement d'utiliser le format spécifique d'un système de fichiers (*file system*). On bénéficie ainsi d'un accès aux données par fichier. On peut aussi utiliser les dossiers, appelés aussi répertoires. Le standard FAT32 est le plus souvent utilisé, vu qu'il est compatible avec tous les systèmes d'exploitation.

Pour faciliter l'accès aux données des fichiers en lecture ou en écriture, de nombreuses librairies sont disponibles : *FatFS*, *PetitFat*, *SdFat*, etc.

A titre d'exemple, la librairie ***PetitFat*** fournit les primitives suivantes :

Procédure	Rôle
<code>pf_mount:</code>	Monter un volume
<code>pf_open:</code>	Ouvrir un fichier
<code>pf_read:</code>	Lire des données dans un fichier
<code>pf_write:</code>	Écrire des données dans un fichier
<code>pf_lseek:</code>	Déplacer le pointeur de lecture ou d'écriture
<code>pf_opendir:</code>	Ouvrir un dossier
<code>pf_readdir:</code>	Lire le contenu d'un dossier.