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 plu

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

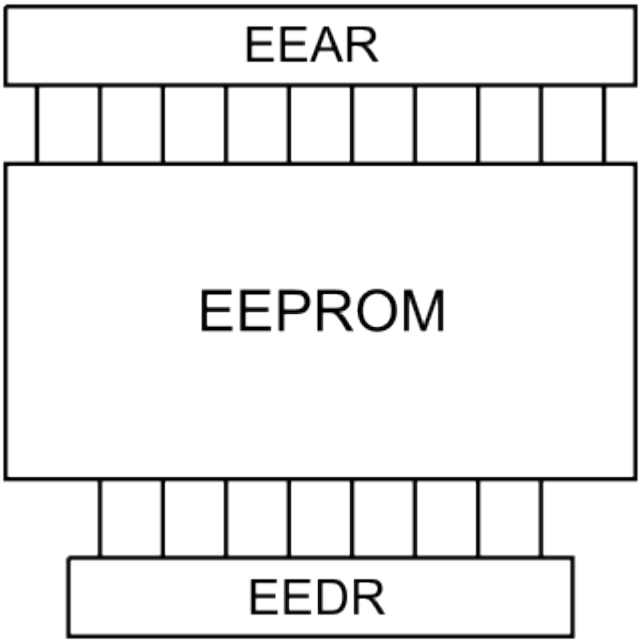
Une particularité de la mémoire Flash est son organisation en blocs. S'il est possible d'écrire une val moire précise, il n'est pas possible d'effacer une position mémoire seule. Les effacements se font d'un bloc varie considérablement d'un microcontrôleur à un autre, parfois seulement 64 octets, jus kB.

LES MÉMOIRES EEPROM

Certains microcontrôleurs disposent d'un troisième type de mémoire, en plus de la Flash et de la R EEPROM. Ce sont aussi des mémoires non-volatiles, mais chaque position mémoire peut être écrit

ment des autres. On trouve ce type de mémoire dans les microcontrôleurs AVR, dont l'ATmega328, isé dans les Arduino.

L'accès à l'EEPROM interne d'un microcontrôleur est généralement très simple. Des registres so (EEAR) et pour la donnée (EEDR), en lecture ou en écriture. Sur les AVR, deux registres donnent l' données :



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 écritur
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 l par Write Enable. EEMPE doit être activé juste avant l'activation du fanion EEPE (EEprom Write d'écriture. Son activation ne dure que quelques cycles d'horloge : il est automatiquement remis à z

possible s'il n'est pas actif. Son but est de rendre improbable une écriture accidentelle en EEPROM. 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 lecture est immédiate, il faut se souvenir que l'écriture prend un temps relativement long, de l'ord des.

Accès à la mémoire Flash par programmation

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

Sur les MSP430, l'architecture de Von Neumann rend facile l'accès à la mémoire Flash. La lecture s quant 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 e 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 fa 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 d'effacement est activé. Mais il est nécessaire de désactiver au préalable le fanion qui bloque toute é

```
// 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
FCTL3 = FWKEY + LOCK; // Set LOCK bit
FCTL1 = FWKEY; // Clear WRT bit
```

Souvent, des librairies sont disponibles pour faciliter ce travail. Pour les AVR, construits sur une a
librairie `Pgmspace.h` est utilisée.

Limite du nombre de cycles d'écriture

Toutes les mémoires non-volatiles ont une limite dans le nombre d'écritures ou d'effacements. Pour
vent de 10'000 cycles. S'il ne s'agit que de programmer le microcontrôleur, ce nombre semble très g
10'000 fois un même microcontrôleur ?

Toutefois... à l'occasion du MOOC Comprendre les microcontrôleurs, proposé par l'EPFL sur Coursera
au point un système de correction de devoirs. Il a corrigé à ce jour plus de 70'000 devoirs. Les micro
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'
100'000 cycles ou davantage. Mais attention, ce nombre est très vite atteint si certaines précautio
marquez par exemple qu'en écrivant une valeur chaque seconde, on atteint la limite peu après un jo

Mémoires externes

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

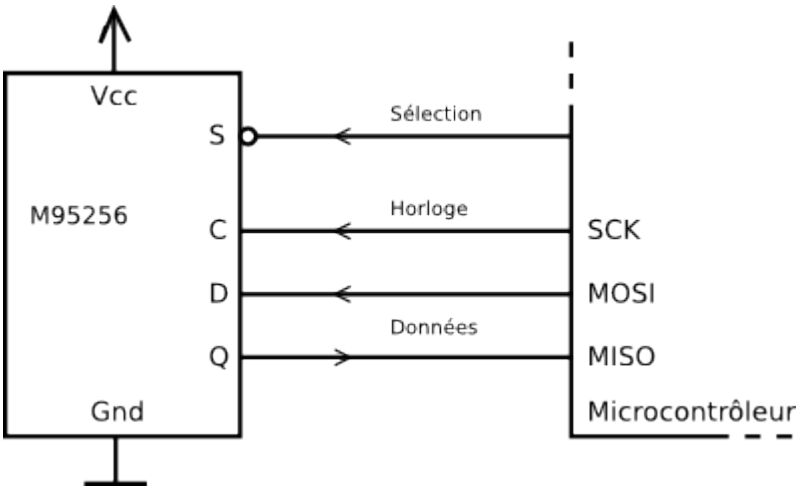
– les mémoires EEPROM, souvent disponibles dans des boîtier à 8 broches (DIL 8 ou modèles corres
– les cartes SD (ou microSD).

Au moyen d'une autre technologie, les circuits horloges temps réel offrent souvent quelques pos
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, STm
logue entre ces mémoires et le microcontrôleur se fait en série, le plus souvent par des signaux aux

Par exemple, la mémoire M95256 de STmicro a une capacité de 256 kb. Attention, la lettre b est en
bit. En d'autre termes, c'est une mémoire de 32 kB (kilo Bytes). Elle utilise les signaux SPI. Voici un s



Mise en œuvre d'une EEPROM

Pour dialoguer avec ces circuits pour lire ou écrire des valeurs en mémoire, on trouve facilement
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 contrô
ou pour SPI. Pour pouvoir utiliser ces contrôleurs, il faut impérativement utiliser les broches choisi
cant. 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
sur les broches d'entrée-sortie. Dans ce cas, on a toute liberté dans le choix des broches. L'usage d
deux avantages. D'une part, le travail de programmation est simplifié. Mais surtout, ces contrôleu
des vitesses de transfert plus élevées. Pour les afficheurs à LED de grande taille, c'est un aspect très

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

Cartes SD

Pour des capacités plus grandes, on choisira des cartes SD. Elles utilisent en interne des mémoires
s'occupe de gérer l'accès aux blocs. Il gère également les mauvais blocs, qui peuvent souvent appa
moires.

Les cartes SD seront aussi utilisées lorsqu'il est nécessaire de déplacer la mémoire pour la connecte
afficheur à LED, on peut envisager de mettre le contenu à afficher dans une cartes SD à partir d'un
dans l'afficheur.

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

Système de fichier

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

Pour faciliter l'accès aux données des fichiers en lecture ou en écriture, de nombreuses librairies so
titFat, *SdFat*, etc.

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

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