

Contents

1	Autour du projet :	3
1.1	La carte à puce :	3
1.2	Programmation et spécifications :	3
1.3	Les endianness	5
2	Mise en place du programme :	6
2.1	Algorithme:	6
2.2	Implémentation:	7
3	Manipulation :	12
4	Notion d'anti-arrachement :	12
4.1	Connexion et transaction :	12
4.2	Arrachement ou <i>tearing</i>	12
4.3	Les méthodes d'anti-arrachements et leurs nécessités:	13

Rapport TP: Programmation carte à puce

TP encadré par L. Demange

May 9, 2021

1 Autour du projet :

Le projet de TP vu pendant les séances de programmation de carte à puce est la réalisation d'un programme en langage C qu'on a nommé "projet portemonnaie" qui consiste à manipuler un solde sur 2 octets au sein d'un système embarqué qui est la carte à puce.

Voici les étapes qu'on a vu :

d'abord faire la vérification en amont du montant de débit/crédit avant toute opération, ensuite faire que les données soient entrées dans la mémoire EEPROM en assurant la possibilité de faire quelques actions de base sur la gestion de l'encaissement (crédit), du décaissement (débit) et finalement apprécier le solde dans l'EEPROM.

Enfin, cette carte sera insérée dans le programmeur de carte pour que le programme soit compilé dans la carte puis dans le lecteur pour afficher les résultats des commandes à envoyer dans la carte par le fichier "hello.script".

Les matériels utilisés sont une carte à puce programmable, un programmeur de carte et un lecteur de carte.

1.1 La carte à puce :

La carte à puce a une mémoire inscriptible et stocke les données à la différence de la RAM. C'est l'EEPROM. On dit qu'elle est persistante.

Tout au long de ce TP, nous allons manipuler des données entre l'EEPROM et la RAM. Les commandes pour mettre des variables dans l'EEPROM sont :

- écriture : `uint8_t ee_chiffre EEMEM; //déclaration de la variable ee_chiffre de taille 1 octet dans l'EEPROM.`
- lecture : `EEPROM_READ_BYTE(&ee_chiffre);`

1.2 Programmation et spécifications :

L'objectif sera d'écrire un programme qui va permettre à la carte de dialoguer avec le lecteur.

On a travaillé principalement avec les fichiers "hello.c" et "io.c" pour voir l'architecture de la programmation ainsi que la syntaxe qu'exige le système embarqué.

1. Dans le fichier hello.c, on a vu :

- La déclaration des fonctions d'entrée et de sortie définies dans le fichier "io.c".
- la procédure qui renvoie l'**ATR (Answer to Reset ou réponse à l'initialisation)** qui donne au lecteur des informations sur le fonctionnement de la puce.

L'octet le plus important dans l'ATR est le premier. Il vaut soit 0x3b pour dire que la puce fonctionne en convention directe, soit 0x3F pour

signaler que la puce fonctionne en convention inverse c'est à dire du **MSB (most significant bit)** vers le **LSB (less significant bit)**. Une fois que la puce a envoyé son ATR, elle ne parle plus tant qu'elle n'est pas sollicitée par le lecteur.

Pour interagir avec la puce, le lecteur lui envoie un **APDU (Application Protocol Data Unit)**, c'est à dire une séquence de 5 octets :

- le premier est la classe (**CLA**) de la commande
- le deuxième octets est l'*instruction* dans la classe (**INS**)
- le 3ème et 4ème octets sont les *paramètres* **P1** et **P2** de l'instruction
- et le 5ème octets **P3** est généralement la *taille* de la réponse attendue.

Un **APDU** est de la forme : **CLA INS P1 P2 P3 xx** , xx est la data en nombre d'octets défini au préalable.

Grâce à l'octets de classe, la carte sait à quelle application l'APDU s'adresser. La fonction switch () du programme principal va appeler la fonction indiquée pour chaque cas du "case" associé.

- émission de la version qui définit la vérification de la taille du data ainsi que la taille attendue, son acquittement et enfin l'émission des données. Dans cette partie sera donnée les status word de l'erreur relative à la taille : **SW=0x6c** ou le **SW=0x90** si tout est correcte dans ce bloc.
- commande de réception de données : elle est définie par la vérification de la taille de celle-ci. Quand la carte à puce a reçu un APDU, elle donne une réponse formalisée par xx xx xx xx SW1 SW2 soit SW1 SW2 qui est la réponse qui donne toujours le statut de l'émission de celle-ci. La convention sur le statut de réponse est donnée dans cette partie du code. La ligne est ensuite suivie de l'acquittement ainsi que du statut à afficher.

La problématique rencontrée est la taille en unité Hexadécimale. La solution est donnée par le message d’erreur après lecture de la carte. En effet, le lecteur nous affichera la taille des données *à insérer* au lieu de ceux qu’on a préalablement inséré si la taille n’est pas correcte. Il suffit de faire la modification dans le script.

Pour résumer, dans chaque bloc de fonction, on a la structure suivante :

```
Void fonction():
{
    Vérification de la taille avec les status word
    acquittement
    Déclaration des variables
    Les conditions avec les status word associés.
}
```

2. le programme principal :

- commence par un “int main(void):”
- suivi des commandes sur l’initialisation des ports
- procédure ATR
- boucles de traitement de commandes (switch()) avec ses codes erreurs en fonction des status word.

Dans le fichier io.c, on a vu le mode de communication input/output entre la carte à puce et le lecteur. On constate que les entrées/sorties sont synchronisés sur l’horloge externe. Ce programme utilise le compteur asynchrone TCNT2 pour fonctionner y compris lorsque le cpu fonctionne sur l’horloge interne à 8 MHz. On y trouve le programme sur l’envoi d’un bit sur le lien série, l’envoi d’un octet, la lecture et la réception d’un octet avec ses spécifications.

1.3 Les endianness

On a vu les 2 types d’endianess qui est une notion importante du sens de la lecture dans la carte, il y a :

1. le Big Endian (BE): de gauche à droite.
2. le Little Endian (LE): de droite à gauche

Par exemple: la valeur 7 (sept) est représentée par 0x07 en BE ou 0x70 en LE.

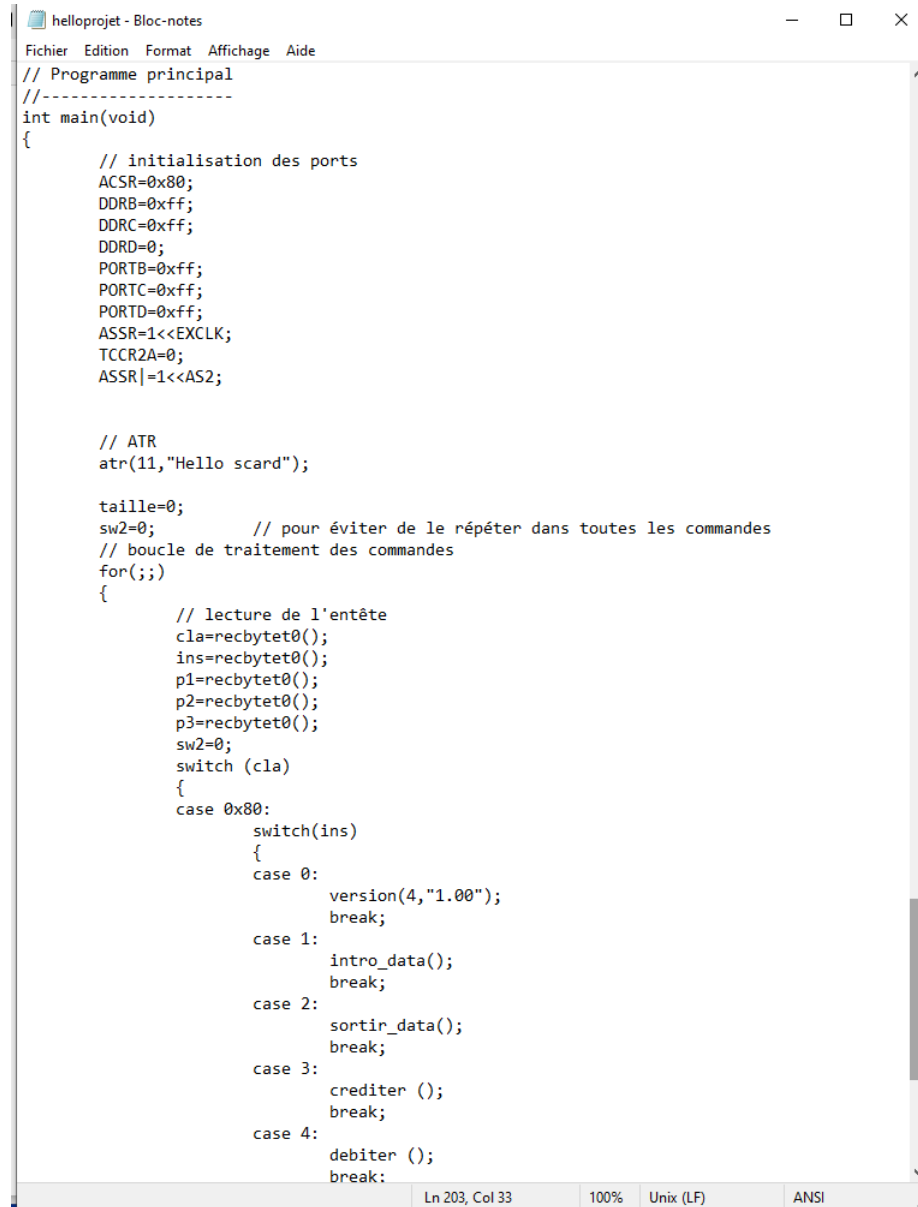
2 Mise en place du programme :

2.1 Algorithme:

1. Entrée : $x1$, $x2$, ee_solde
2. Sortie : $solde = (x1 \ll 8) + x2$ en BE
3. on appelle le solde ee_solde dans l'EEPROM
4. On met en place la variable 2 octets "nouveau_solde" :
 $(uint16_t \text{ nouveau_solde} = (x1 \ll 8) + x2)$
5. Vérification de la taille
6. Fonction créditer:
 - Si la somme du solde avec le nouveau solde est plus petite que le nouveau solde
 - alors la transaction n'est pas autorisée.
7. Fonction débiter
 - Si la différence du solde avec le nouveau solde est plus grande que le nouveau solde
 - Alors la transaction n'est pas autorisée.
8. Sinon nouveau solde est égale à la somme du solde et du nouveau solde
9. On envoie le nouveau solde dans l'EEPROM avec la fonction `sortir_solde()`.

2.2 Implémentation:

Fonction main:



```
// Programme principal
//-----
int main(void)
{
    // initialisation des ports
    ACSR=0x80;
    DDRB=0xff;
    DDRC=0xff;
    DDRD=0;
    PORTB=0xff;
    PORTC=0xff;
    PORTD=0xff;
    ASSR=1<<EXCLK;
    TCCR2A=0;
    ASSR|=1<<AS2;

    // ATR
    atr(11,"Hello scard");

    taille=0;
    sw2=0;        // pour éviter de le répéter dans toutes les commandes
    // boucle de traitement des commandes
    for(;;)
    {
        // lecture de l'entête
        cla=recbytet0();
        ins=recbytet0();
        p1=recbytet0();
        p2=recbytet0();
        p3=recbytet0();
        sw2=0;
        switch (cla)
        {
            case 0x80:
                switch(ins)
                {
                    case 0:
                        version(4,"1.00");
                        break;
                    case 1:
                        intro_data();
                        break;
                    case 2:
                        sortir_data();
                        break;
                    case 3:
                        crediter ();
                        break;
                    case 4:
                        debiter ();
                        break;
                }
            break;
        }
    }
}
```

Ln 203, Col 33 100% Unix (LF) ANSI

Interprétation:

La fonction main donne la liste des instructions que la fonction peut appeler.

Fonction créditer:



```
void crediter ()
{
    if(p3!=2) //2octets
    {
        sw1=0x6c;
        sw2=2; //taille 2 octets
        return;
    }
    sendbytet0(ins);

    uint8_t x1=recbytet0();
    uint8_t x2=recbytet0();
    uint16_t solde;
    uint16_t somme;

    somme= (x1<<8) + x2; //big endian
    eeprom_read_block(&solde, &ee_solde, 2);

    if (solde > solde + somme) //crediter
    {
        sw1=0x6c; //taille incorrecte
        sw2=2;
        return;
    }
    else
    {
        solde += somme;
        eeprom_write_block(&solde, &ee_solde, 2);
        sw1=0x90;
        sw2=00;
    }
}
```

Interprétation:

Pour donner l'instruction "créditer", INS sera le "case 3", pour ajouter "40" on a le script : 80 **03** 00 00 02 **00 40**.

Fonction débiter:

```
void debiter ()
{
    if(p3!=2)    //2octets
    {
        sw1=0x6c;
        sw2=2;  //taille 2 octets
        return;
    }
    sendbytet0(ins);

    uint8_t x1=recbytet0();
    uint8_t x2=recbytet0();
    uint16_t solde;
    uint16_t somme;

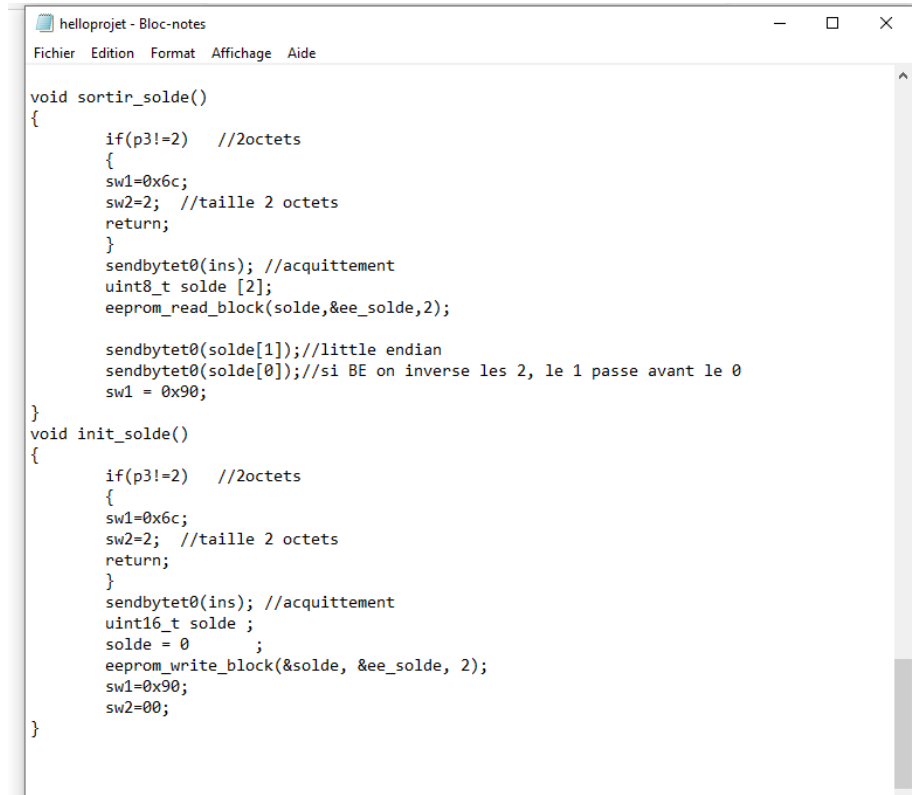
    somme= (x1<<8) + x2; //big endian
    eeprom_read_block(&solde, &ee_solde, 2);

    if (solde < solde - somme) //debiter
    {
        sw1=0x6c; //taille incorrecte
        sw2=00;  //taille attendue
        return;
    }
    else
    {
        solde -= somme;
        eeprom_write_block(&solde, &ee_solde, 2);
        sw1=0x90;
        sw2=00;
    }
}
```

Interprétation:

Pour donner l'instruction "débiter", INS sera le "case 4", pour supprimer "30"
on a le script : 80 **04** 00 00 02 **00 30**.

Fonction sortir_solde:



```
void sortir_solde()
{
    if(p3!=2) //2octets
    {
        sw1=0x6c;
        sw2=2; //taille 2 octets
        return;
    }
    sendbytet0(ins); //acquittement
    uint8_t solde [2];
    eeprom_read_block(solde,&ee_solde,2);

    sendbytet0(solde[1]); //little endian
    sendbytet0(solde[0]); //si BE on inverse les 2, le 1 passe avant le 0
    sw1 = 0x90;
}

void init_solde()
{
    if(p3!=2) //2octets
    {
        sw1=0x6c;
        sw2=2; //taille 2 octets
        return;
    }
    sendbytet0(ins); //acquittement
    uint16_t solde ;
    solde = 0 ;
    eeprom_write_block(&solde, &ee_solde, 2);
    sw1=0x90;
    sw2=00;
}
```

Interprétation:

Pour donner l'instruction "sortir_solde", INS sera le "case 5", pour afficher le solde on a le script: 80 **05** 00 00 **02**. (instruction 5, taille de données 2)

```
hello - Bloc-notes
Fichier Edition Format Affichage Aide
# script de test pour le programme hello world
# usage :
# > scat hello.script

reset

# lecture de la version
#80 00 00 00 00          # avec une mauvaise taille -> sw = 6c05
#80 00 00 00 $sw2        # avec p3=sw3

# introduction de données
#80 01 00 00 01 d
#80 01 00 00 01 a
#80 01 00 00 02 01 02 03          # p3 incorrect

#80 01 00 00 20

# lecture des données          # p3 trop grand
#80 02 00 00 01

#tentative ajout de 40
80 03 00 00 02 00 40

#debiter 30
80 04 00 00 02 00 30

#sortir solde
80 05 00 00 02

#la réponse sera 20 00 au lieu de 00 20

exit # fin du script

Ln 1, Col 1    100%    Unix (LF)    UTF-8
```

Le programme fonctionne correctement.

Les status word utilisés sont :

- 0x6c: code erreur taille incorrecte
- 0x6d: code erreur ins inconnu
- 0x6e: code erreur classe inconnue.
- 0x90 : pas d'erreur

3 Manipulation :

Tous les programmes que nous verrons seront écrits en langage C avec un éditeur de texte puis seront appelés une fois que la carte sera insérée dans le programmeur de carte pour lui transmettre les instructions. Une fois insérée, on lance les commandes suivantes pour compiler notre programme et programmer notre carte :

1. `cd c:/programmation_carte/`
2. `make`
3. `make progcarte`

On corrige les erreurs s'il y a. Sinon on retire la carte du programmeur et on l'insère dans le lecteur de carte pour visualiser le résultat.

Lorsque la carte est prête, on lance la commande:

- `scat helloscript`

On a l'affichage du solde restant sur la carte.

4 Notion d'anti-arrachement :

4.1 Connexion et transaction :

La connexion entre la carte et le terminal est établie dans le lecteur de carte. Une carte à puce sans contact (technologie NFC) est sans source d'énergie propre. Il ne fonctionne que lorsqu'il est télé-alimenté par le champ du lecteur.

Elle effectue une suite d'actions pour tester son état, puis elle émet une suite d'octets appelée "réponse à la remise à zéro ou ATR". Cette réponse est normalisée. L'ATR est composée de 33 octets maximum, qui sont répartis en cinq champs que l'on a vu précédemment.

La fin de la *connexion* carte/terminal est marquée par le retrait de la carte du lecteur, appelé **arrachement**. Dans un usage normale, le porteur est informé que la connexion est terminée et qu'il peut retirer sa carte.

Il arrive également que la carte soit complètement avalée par le lecteur et restituée par ce dernier au terme de la connexion. Dans les deux cas, l'arrachement de la carte ne doit pas poser de problème. Cependant, un arrachement intempestif par erreur (distraction, impatience) ou volontaire (tentative de fraude) représente une situation suffisamment fréquente pour être prise en compte en tant qu'événement habituel.

4.2 Arrachement ou *tearing*

Ainsi, pour effectuer une transaction, la carte doit être insérée dans le lecteur et laissée en place jusqu'à la fin de la transaction. Lorsque la carte a été retirée

avant la fin de la transaction, cela peut provoquer une écriture corrompue du contenu stocké en mémoire.

Supposons par exemple que si la carte stocke un nombre de points ou de jetons, dont la valeur initiale est 12345. Le terminal souhaite l'incrémenter de 1 pour le porter à 12346. Si la carte ne dispose pas de mesure "**anti-arrachement ou anti-tearing**" et qu'elle s'est arrachée au moment précis où sa logique interne réécrit le compteur en mémoire, la valeur finale peut être 12300, ou 123FF, ou 0000, ou etc.

Le compteur sera associé à une somme de contrôle ou à un contrôle d'intégrité et d'authenticité cryptographique (**CMAC**) pour déterminer à la prochaine transaction s'il est correct ou non.

4.3 Les méthodes d'anti-arrachements et leurs nécessités:

- La méthode la plus simple pour résoudre ce problème est d'utiliser une technologie de "*mirroring*", c'est à dire de doubler toute la mémoire de stockage et d'écrire sur une zone puis sur l'autre. Si l'écriture n'est pas menée à son terme l'ancienne valeur est toujours présente dans une des deux zones qui sert ainsi de backup et permet de restaurer un état cohérent à la mise sous tension suivante. La taille de la mémoire de stockage doit être doublée, cela peut avoir un impact sur le coût de la carte.
- Pour les cartes ne disposant pas d'anti-tearing, il est possible d'implémenter le même principe de "*mirroring*" au niveau applicatif, en stockant toutes les données en double. En contrepartie, la taille de la mémoire utile est divisée par deux et le temps de transaction est doublé.
- Une autre méthode consiste à doter la puce d'un condensateur et d'une mémoire tampon c'est à dire lors d'une opération d'écriture, elle n'est pas effectuée directement sur la mémoire de stockage non volatile, mais mise en attente provisoirement dans la mémoire tampon. Lorsque toutes les données attendues ont été reçues, la carte vérifie l'état de son alimentation et le niveau de charge de son condensateur. Si son condensateur a accumulé suffisamment d'énergie, elle sait qu'elle peut désormais commencer l'opération d'écriture car elle pourra la mener à son terme même en cas de tearing. Par contre si l'alimentation n'est plus présente et que le condensateur ne dispose pas d'assez d'énergie, l'écriture n'aura simplement pas lieu.

Ces deux méthodes peuvent se combiner pour amener sur certaines cartes, à un mécanisme transactionnel puissant. Au lieu de protéger des opérations d'écriture individuelles, l'anti-tearing couplé au mécanisme transactionnel va assurer l'atomicité d'un enchaînement de plusieurs écritures. Soit elles sont toutes effectuées avec succès, soit le contenu précédent est préservé.

- Les cartes sans contact, de moyenne et haute gamme, intègrent généralement un mécanisme anti-tearing qui protège systématiquement toutes leurs métadonnées: zones de configurations, clés cryptographiques, organisation du système de fichiers.

Par contre, l'anti-tearing pour protéger les zones de données n'est pas forcément systématique. Le cas échéant, il faut l'activer fichier par fichier lors du formatage initial.

References

<https://www.springcard.com/fr/blog/news/the-anti-tearing-concept-and-mechanisms>