RAPPORT PORTE MONNAIE ELECTRONIQUE

I – Introduction

Le projet du porte monnaie électronique a pour objectif de proposer une carte a puces contenant des informations sur le client et son solde selon la norme ISO 7816. Le but étant de gérer les entrées/sorties d'argent sur le compte du client ainsi que de de sécuriser les transactions de façon fiable. Pour cela, nous avons implémenté un certains de nombre de fonctions (7 en tout)pour y parvenir. Nous commençons tout d'abord a introduire des variables globales pour mettre en forme les entêtes des commandes de la carte i.e cinq variable(uint8_t cla, ins, p1, p2, p3) statique de type 'uint8_t' (1 octet de 2 chiffres hexadécimal) qui correspondent a la classe,l'instruction ainsi qu'aux 3 paramètres de la commande.Des variables EEPROM sont utilisées pour sauvegardés les données nécessaire dans la mémoire non volatils.
A cela, on ajoute deux autres variables de type 'uint8_t' qui explicite si la requête s'est bien passé ou pas (uint8_t sw1, sw2). On inclut un fichier d'en-tête qui nous fournit deux fonctions qui permet de recevoir et d'envoyer des octets (void sendbytet0 et uint8_t recbytet0).

Et enfin , nous avons une fonction main qui fait exécuter n'importe quelle requête à la carte depuis un fichier .script. Dans le main, on commence premièrement à initialiser les ports,ensuite on fait envoyer un ATR a la carte (procédure requise lors de du branchement de la carte au lecteur) .Dans une boucle for a l'infini , on reçoit les commandes décrites dans le fichier script ,une commande se présente sous la forme de 5 entêtes qui sont des octets qu'on mets dans les cinq variables de la RAM décrites ci-dessus. Grâce à un 'switch case' , on teste de quelle classe est la requête et de quelle type d'instruction décrit-t-elle pour appeler la fonction adéquate.La carte envoie un acquittement suivi du STATUS WORD 9000 si elle sait traité la requête sinon un code d'erreur (la valeur du STATUS WORD dépend de l'entête qui a posé problème).

II– Les commandes de la carte

Le nombres des commande sont en nombre de cinq :

a) Introduction des donnés personnel de la personne

A l'aide de la fonction ' void intro_perso() ' ,on introduit les données de la personne dans la mémoire EEPROM tels que son nom et son solde dans son compte actuel.
Pour cela, on commence à tester si le troisième paramètre de la commande correspondant à la taille des données entrante est valide ou pas. Si c'est le cas , on envoie un acquittement (sendbytet0 de l'octet ' ins') puis on traite la requête en effectuant tout d'abord un recbytet0 pour recevoir le nom de la personne.Par exemple si on veut mettre le prénom 'Ali' , on va faire correspondre chaque lettre du prénom a son code ASCII qui vaut un octet , sachant que recbytetv0 renvoie un octet , on va à l'aide d'une boucle for rentrer la totalité du prénom dans une variable (qu' on va nommer data)de la RAM qui représentera un tableau d'octets de taille au moins 3 dans ce cas là (De façon général , on déclare la variable data de manière globale en lui fixant une taille maximum). Puis on effectue , un transfert vers la mémoire EEPROM à l'aide des fonctions suivantes :

```
*eeprom_write_block(data,ee_nom,p3) ;
*eeprom_write_byte(&ee_taille_nom,p3);
```

en prenant le soin de déclarer au préalable les variables globales 'ee_nom' et 'ee_taille_nom' suivi de l'inscription EEMEM (signifie que ce sont des variables de l'EEPROM).On réalise la même chose avec une variable solde qu'on initialise à zéro.On fini par mettre la STATUS WORD à 9000.

b) Lecture des donnés personnel depuis la mémoire EEPROM

Même raisonnement que la commende précédente, on utilise la fonction void lecture_perso() (On teste si la taille des données qu'on veut extraire correspond bien à la valeur du troisième paramètre , on envoie ensuite un acquittement ect..).Sauf qu'ici on lit dans la mémoire EEPROM , au lieu d'écrire à l'aide de la fonction suivante :

- eeprom_read_block(data,ee_nom,p3);

Ici , on ne récupère que le nom de la personne.Pour récupérer son solde, on utilisera une autre commande qui appellera la fonction void lecture_solde().

Ensuite, on envoie ces donnés requis au lecteur via un sendbytet0 :

```
*  for(i=0;i<p3;i++){
        sendbytet0(data[i]);
   }
```

c) Créditer dans le solde du client

On utilise une commande qui fait appelle a la fonction void credit(). On fait le choix ici que le solde ne dépasse pas un certains seuil qui est 65 535 car on veut faire entrer que des nombre de deux octets (en faites, 65 535 est le nombre maximal qu'on peut écrire sur 2 octets) dans le solde. On adopte la convention BIG ENDIAN i.e le 1er octet est l'octet du poids le plus fort.Cette fonction effectue la procédure habituelle à savoir le test de la taille du nombre qu'on veut ajouter au solde (ici ça sera 2) ainsi que l'acquittement si on passe le test . Le seul nouveau cas ici est qu'on va additionner les données qu'on rentre au solde en vérifiant qu'on ne dépasse le seuil lors de cette opération.Pour cela on s 'est pris en deux temps :

```
uint8_t data[2];
eeprom_read_block(data,ee_solde,2);
uint8_t ajout[2];
for(i=0;i<2;i++){
        ajout[i]=recbytet0();
}
int err;
err = 256*((int)data[0] + (int)ajout[0]) + (int)data[1] + (int)ajout[1];
if(err>65535){
        sw1=0x61;
        return;
}
```

Ici , on ajoute l'octet du poids le plus fort du solde ainsi que des données entrante et on multiplie le tout par 256 pour se retrouver avec un nombre de 2 octets , ensuite on ajoute a ce dernier les octets des poids les plus faibles (à savoir ici (int)data[1] + (int)ajout[1]) puis on test si le résultat obtenue est supérieur à 65 535. Malheureusement, lors des test , quand on faisait en sorte de dépasser le seuil dans une des commandes , la commande passer le test d'erreur et donc rajouter le nombre entrant au solde. On a dû être contraint de changer notre code pour le suivant :

```
uint8_t data[2];
eeprom_read_block(data,ee_solde,2);
uint8_t ajout[2];
for(i=0;i<2;i++){
        ajout[i]=recbytet0();
}
uint8_t somme[2];
somme[1] = data[1] + ajout[1];
somme[0] = data[0] + ajout[0] + (data[1] + ajout[1] < data[1] ? 1 : 0);
if(somme[0]<data[0] || (somme[0]==data[0] && somme[1]<data[1])){
        sw1=0x61;
        return;
}
```

Ici,on prend le faite que si l'addition de 2 octets dépasse la valeur max d'un octets i.e 255 alors la valeur de la somme repart à zéro. La somme de deux nombre et le test sont d'ailleurs basés sur cette propriété : si la somme des octets des poids les plus faibles dépasse 255 alors on retient la retenue qu'on ajoute a la somme des octets des poids les plus forts sinon on ajoute rien.

Une fois le test passé , on remet le nouveaux solde dans l'EEPROM et on met le STATUS WORD à 90 00.

La fonction qui permet de débiter (void debit()) fait exactement l'inverse que la fonction précédente. Le seuil ici sera 0 et non plus 65 535 et le test du dépassement de seuil sera de la même forme que le précédent :

```
uint8_t somme[2];
somme[1] = data[1] - ajout[1];
somme[0] = data[0] - ajout[0] - (data[1] - ajout[1] > data[1] ? 1 : 0);
if(somme[0]>data[0] || (somme[0]==data[0] && somme[1]>data[1])){
        sw1=0x61;
        return;
}
```

III– La sécurisation de la carte

Jusqu'ici , nous avons implémenté toutes les méthodes nécessaire à notre porte monnaie virtuel.Reste à voir comment bien sécuriser les transactions. En effet,lors d'un crédit ou d'un débit, il est susceptible qu'il y est un arrachement de la carte au moment de ces opérations.Et donc pourrait se retrouver dans un cas ,par exemple lors d'un crédit d'une

somme , i.e l'ajout d'un nombre à, un nombre déjà présent dans l'EEPROM, qu'une partie soit additionné et l'autre non.Pour éviter de tomber dans cette situation là,on va essayer d'écrire en EEPROM sur transactions.On veut que la carte reste dans un état cohérent lors d'un arrachement imprévisible de la carte i.e revenir à l'état où les choses n'avait pas débuté. Une des solutions est de faire des transactions i.e faire des changement d'états sous des conditions ACID :
-A : Atomicité i.e écrire les données en même temps , en une opération
-C : Cohérence i.e l'état du système doit rester cohérent pendant toute la durée de vie de la carte quoi qu'il arrive
-I : Isolation i.e ce qui se passe pendant la transaction est isolé du monde extérieur
-D :Durabilité qui est lié à la cohérence

Cette solution repose sur une mémoire tampon, on veut écrire des données dans l'EEPROM mais avant cela on les mets d'abord en mémoire tampon, cette tâche sera réalisé grâce a la fonction engager() et ensuite cette transaction va falloir l a valider i.e on va écrire dans la mémoire tampon(qui est lui même dans l'EEPROM) vers sa destination finale dans l'EEPROM (fonction valider()).
On introduit un nouveau type ne RAM qui correspondra à l'état , ainsi la fonction valider() s'appuiera sur cet état : soit la mémoire tampon contient des données alors l'état est plein et on valide la la transaction (puis on remet l'état à vide) ou soit elle ne contient pas alors l'état est vide , la fonction valider() ne fait rien.

La fonction engager() prend en entrée la taille 'n 1' des donnés à introduire dans l'EEPROM qui va écrite depuis l'adresse de la source en RAM 's1' à destination de l'adresse en EEPROM 'd1' ensuite on a 'n2' la taille de la deuxième données a introduire dans 'd2' ect.. Ces données doivent être traité ensemble de manière atomique pour cela la fonction engager() prend en entrée un nombre variable de paramètres.

La fonction valider() n'a pas de paramètre , elle consiste simplement à valider la transaction qui a été engagé.

Donc les fonctions engager() et valider() font office de remplacement aux fonctions eeprom_write_byte() et eeprom_write_block() utilisés auparavant

Il faut aussi qu'au lancement du programme,dans le main , avant la boucle 'for' , valider une éventuelle transactions qui aurait été interrompue au préalable.

Donc le tour est joué, il nous manque plus qu'à tester.

IV – Stratégies de test

Pour le test , on a commencer par tester si la fonction 'void intro_perso() ' fonctionner.Pour cela on a essayer d'introduire le nom 'Carl' via le script comme tel :

```
# perso nom
80 00 00 00 04 "carl"
```

ceci nous renvoie le résultat suivant :

- 80 00 00 00 04 "carl"
  0.000 < 80 00 00 00 04 63 61 72 6c ....carl
  0.108 > 90 00 .
  exécution normale

qui signifie que l'opération s'est bien déroulé.Ensuite nous avons testé la lecture du nom ainsi que du solde avec pour chacun des cas la taille correct et la taille incorrect du paramètre p3 en entrée (rappel sur la taille du solde qui est sur deux octets) :

lecture nom
80 01 00 00 03 taille erronée
80 01 00 00 04 taille correct de 'Carl'
lecture solde
80 02 00 00 03 taille erronée
80 02 00 00 02 taille correct du solde présent dans l'EEPROM

nous obtenons le résultats suivant :

* 80 01 00 00 03
0.108 < 80 01 00 00 03 ....
0.116 > 6c 04 l.
P3 incorrect, 04 attendu
* 80 01 00 00 04
0.116 < 80 01 00 00 04 ....
0.136 > 63 61 72 6c 90 00 carl.
exécution normale
* 80 02 00 00 03
0.136 < 80 02 00 00 03 ....
0.152 > 6c 02 l.
P3 incorrect, 02 attendu
* 80 02 00 00 02
0.152 < 80 02 00 00 02 ....
0.168 > 00 00 90 00 renvoie que notre solde est nulle suivi du SW.
exécution normale

Tout ce passe comme prévue. Ensuite, on ajoute la valeur 10 000 à notre solde, 27 10 en hexadécimal, une fois avec le bon paramètre p3 et une fois avec la mauvaise taille de p3 puis on lit le solde de nouveau :

credit
80 03 00 00 03 27 10
80 03 00 00 02 27 10
80 02 00 00 02

Nous obtenons le résultat suivant :

- 80 03 00 00 03 27 10
  0.168 < 80 03 00 00 03 27 10 ....'.

erreur **valeur de p3 non valide**
* 80 03 00 00 02 27 10
0.168 < 80 03 00 00 02 27 10 ....'.
0.224 > 90 00 .
exécution normale
* 80 02 00 00 02 # lecture solde
0.224 < 80 02 00 00 02 ....
0.236 > 27 10 90 00 Le solde a bien été introduit ! '..
exécution normale

Puis on ajoute successivement 30 583 jusqu'à dépasser le seuil qui est à 65 535 :
lecture solde
80 03 00 00 02 77 77
80 02 00 00 02
lecture solde
80 03 00 00 02 77 77
80 02 00 00 02

Nous obtenons ceci :
* 80 03 00 00 02 77 77 Nous ajoutons 30 583 à 10 000
0.236 < 80 03 00 00 02 77 77 ....ww
0.292 > 90 00 .
exécution normale
* 80 02 00 00 02 # lecture solde
0.292 < 80 02 00 00 02 ....
0.304 > 9e 87 90 00 Nous avons bien ici 30 583+10 000=9e 87 < ff ff
exécution normale
* 80 03 00 00 02 77 77 Nous ajoutons 30 583 à 9e 87 qui est supérieur à 65 535
0.304 < 80 03 00 00 02 77 77 ....ww
0.320 > 61 00 Affiche un message d'erreur a.0 octets de données toujours disponibles
* 80 02 00 00 02 # lecture solde
0.320 < 80 02 00 00 02 ....
0.336 > 9e 87 90 00 On relit le solde et on s'aperçoit que l'opération précédente à été annulé

Nous faisons la même chose pour le débit :
debit
80 04 00 00 03 27 10
80 04 00 00 02 27 10
80 02 00 00 02 # lecture solde

80 04 00 00 02 77 77
80 02 00 00 02 # lecture solde

80 04 00 00 02 77 77
80 02 00 00 02 # lecture solde

et tout se passe de façon nickel :
* 80 04 00 00 03 27 10
0.336 < 80 04 00 00 03 27 10 ....'.
erreur **valeur de p3 non valide**
* 80 04 00 00 02 27 10
0.336 < 80 04 00 00 02 27 10 ....'.
0.392 > 90 00 .
exécution normale
80 02 00 00 02 # lecture solde
0.392 < 80 02 00 00 02 ....
0.404 > 77 77 90 00 ww
exécution normale
* 80 04 00 00 02 77 77
0.404 < 80 04 00 00 02 77 77 ....ww
0.460 > 90 00 .
exécution normale
* 80 02 00 00 02 # lecture solde
0.460 < 80 02 00 00 02 ....
0.472 > 00 00 90 00 ..
exécution normale
* 80 04 00 00 02 77 77 On essaye de retirer alors qu'il y'a rien sur notre compte
0.476 < 80 04 00 00 02 77 77 ....ww
0.496 > 61 00 Message d'erreur a.0 octets de données toujours disponibles
* 80 02 00 00 02 # lecture solde
0.496 < 80 02 00 00 02 ....
0.508 > 00 00 90 00 Compte reste à zéro