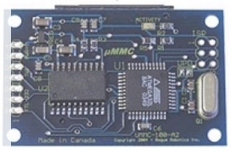
STI2D Projet SIN

PSoC Mise en œuvre de la sauvegarde de données sur carte SD

Sommaire

⇦ Retour au sommaire

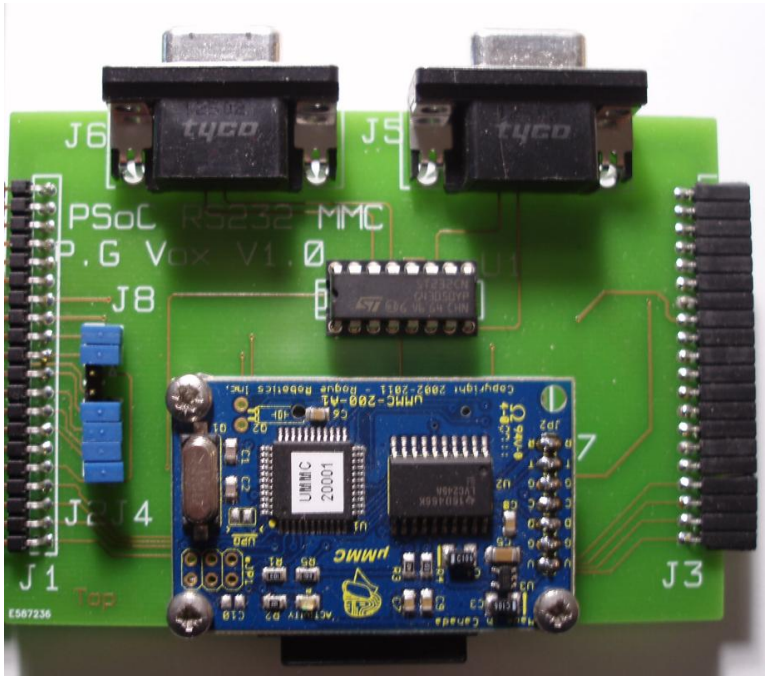
STI2D Projet SIN




La carte d'interface PSoC_RS232_MMC

La carte MMC de Rogue Robotics est implantée sur une carte d'interface spécifique permettant l'emploi de trois liaisons séries en supplément de la liaison série déjà présente sur la platine PSoCVox.

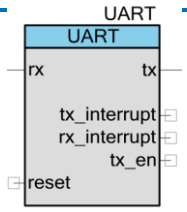
Trois d'entre elles sont au standard RS232 et la quatrième est au format TTL Level pour piloter directement la carte MMC



STI2D Projet SIN

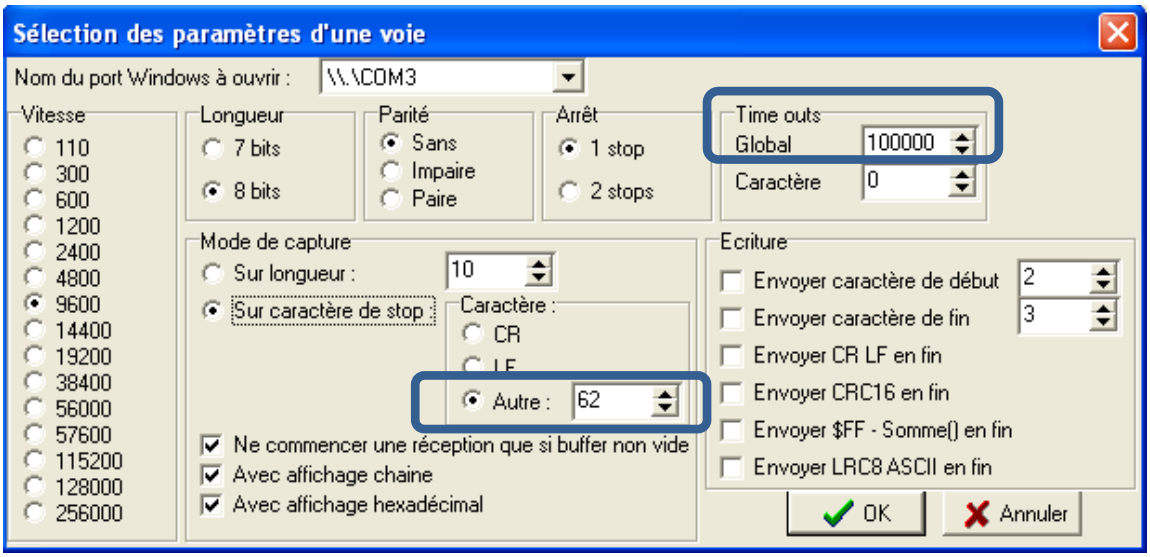



Utilisation de la carte MMC de Rogue Robotics



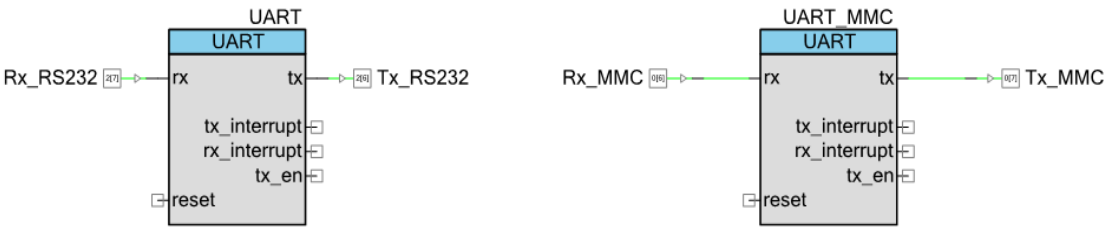
Configuration de TestComVox

Pour faciliter l'observation de la bonne exécution des requêtes vers la carte MMC nous allons utiliser le programme TestComVox. Pour avoir un affichage correct le mode de capture est réglé sur le caractère de stop 0x3E soit 62, la durée du timeout est aussi augmentée et enfin le port série physique est choisi en fonction du poste informatique, ici com3.



Deux UART sont utilisées, la première dénommée UART est relié au PC, la deuxième est reliée en Serial TTL Level à la carte MMC via l'adaptateur PSoC_RS232_MMC_MyAVR

Configuration de PSoC Creator pour la liaison MMC

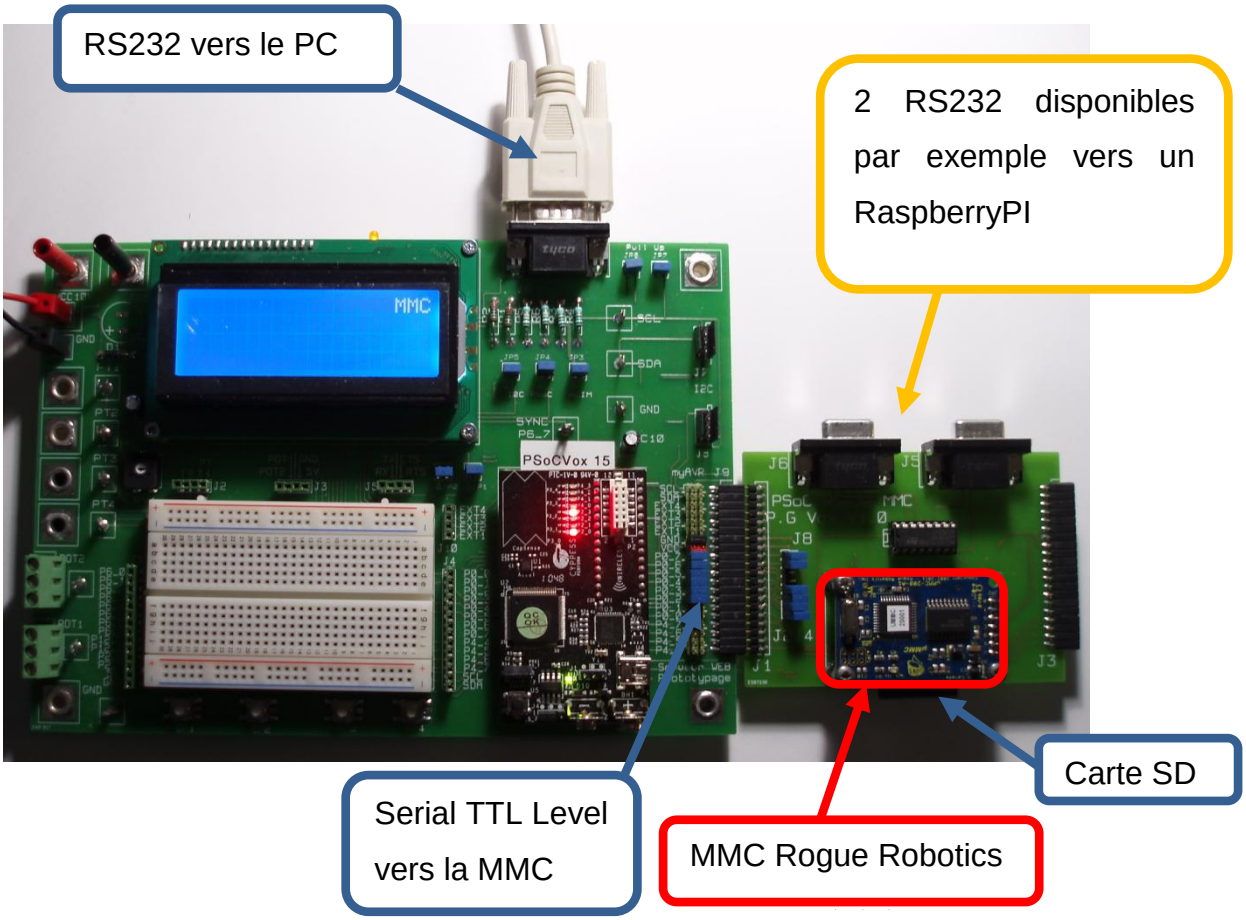


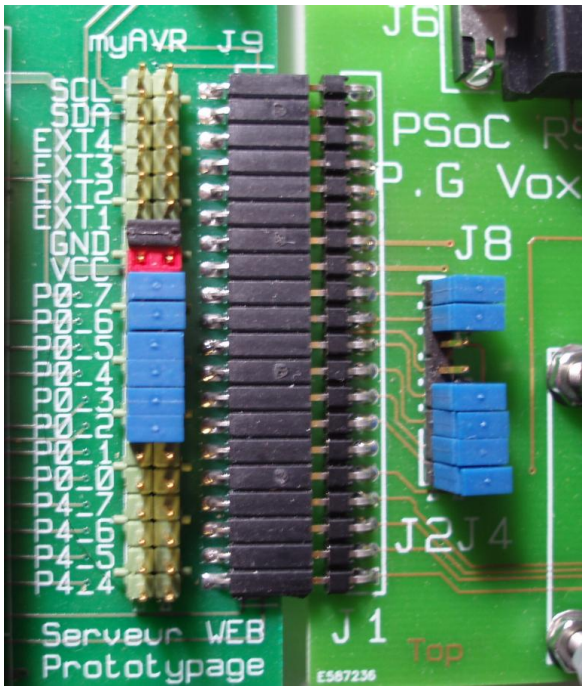
STI2D Projet SIN

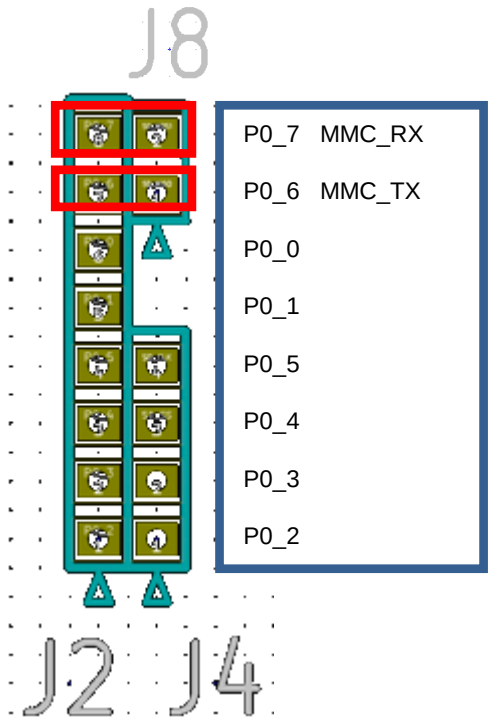




La carte MMC en action avec une platine PSoCVox



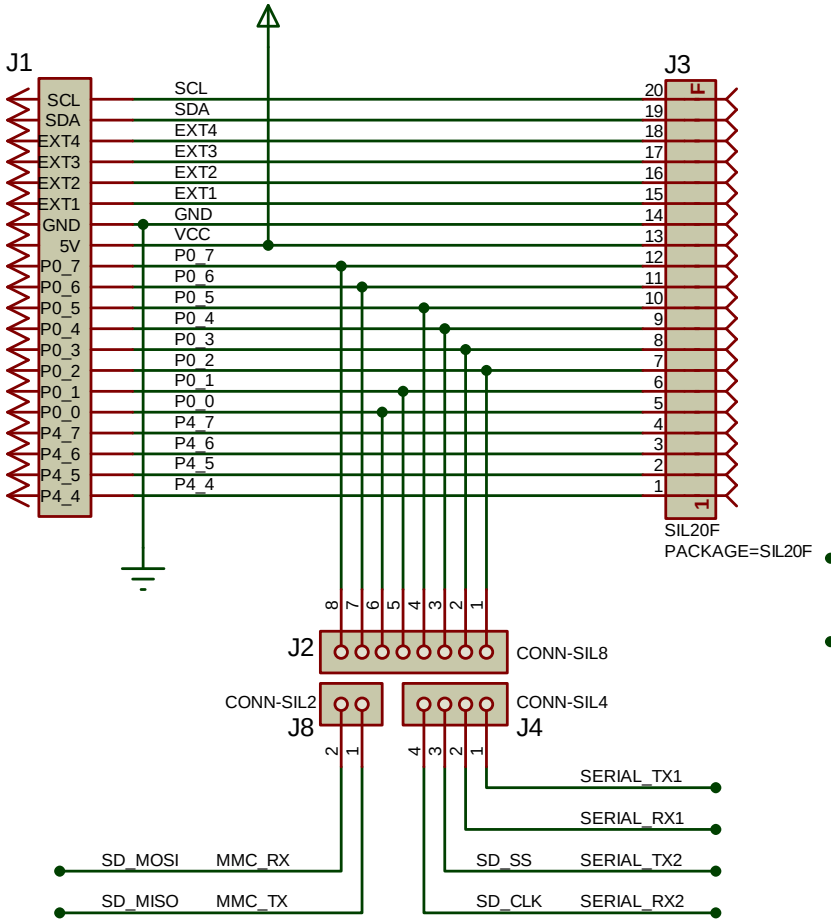


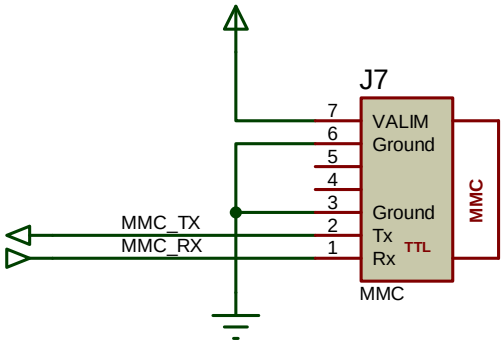


STI2D Projet SIN



Le schéma de l'interface PSoC_RS232_MMC





SD TTL Level

STI2D Projet SIN




Essais logiciels

Lecture d'un fichier de données

Appui sur BP1 lecture de 7 valeurs dans le fichier . Les valeurs sont sélectionnées avec la table d'entiers ListeAlire

```
int ListeALire[10]={1,12,13,14,24,31,6};
int NombreALire=7;
```

☞ le résultat lu est-il correct ?

Réponse de la carte MMC

Trace TestComVox

Nombres lus dans le fichier

```
1: 3E[>]<OK>
14: 20 20 20 20 20 30 2E 30 30 30 0D 0A 20 3E[     0.000..  >]<OK>
14: 20 20 20 34 37 30 2E 30 30 30 0D 0A 20 3E[   470.000..  >]<OK>
14: 20 20 20 20 37 30 2E 30 30 30 0D 0A 20 3E[    70.000..  >]<OK>
14: 20 20 20 38 34 30 2E 30 30 30 0D 0A 20 3E[   840.000..  >]<OK>
14: 20 20 20 33 30 30 2E 30 30 30 0D 0A 20 3E[   300.000..  >]<OK>
14: 20 20 20 34 38 30 2E 30 30 30 0D 0A 20 3E[   480.000..  >]<OK>
14: 20 20 20 36 37 30 2E 30 30 30 0D 0A 20 3E[   670.000..  >]<OK>
```

E1000.txt | Essai.txt

1	0.000
2	30.000
3	860.000
4	200.000
5	270.000
6	670.000
7	310.000
8	160.000
9	370.000
10	420.000
11	80.000
12	470.000
13	70.000
14	840.000
15	50.000
16	290.000
17	910.000
18	360.000
19	770.000
20	320.000
21	690.000
22	840.000
23	710.000
24	300.000
25	160.000
26	320.000
27	460.000
28	240.000
29	820.000
30	270.000
31	480.000
32	140.000

```
321         // Lecture du fichier ligne par ligne soit
322         // 10 octets pour le nombre + 2 pour CR LF = 12 octets
323         if ( BP1_Push == true )
324         {
325             OuvertureFichierLecture(1,NomFichierFloat10);
326             DepartElement=1;
327             for(i=0;i<NombreALire;i++)
328             {
329                 NumeroElement=ListeALire[i];
330                 NombreLu=LectureFichierFloat10(NumeroElement,10);
331                 Somme=Somme+NombreLu;
332                 // Attendre pour laisser le temps à Testcomvox
333                 // de tracer les bytes envoyés
334                 CyDelay(20);
335             }
336             FermetureFichier(1);
337         }
```

Le début du fichier de 1000 floats

STI2D Projet SIN





Ajout de données dans un fichier

Appui sur BP2 ouverture d'un fichier existant en mode Append puis ajout de 10 entiers dans ce fichier :

☞ commenter les résultats obtenus présentés ci-dessous.

```
341         //  Ouverture d'un fichier en écriture en mode Append
342         if ( BP2_Push == true )
343         {
344             OuvertureFichierAppend(2,NomFichierEssai);
345             for (i=0; i<10; i++)
346             {
347                 sprintf(tstr,"%6d",i*10);
348                 UART_MMC_PutString("W 2 8");
349                 UART_MMC_WriteTxData(0x0D);
350                 UART_MMC_PutString(tstr);
351                 UART_MMC_WriteTxData(0x0D);
352                 UART_MMC_WriteTxData(0x0A);
353             }
354             FermetureFichier(2);
355         }
```

```
1: 3E[>]<OK>
1: 3E[>]<OK>
1: 3E[>]<OK>
1: 3E[>]<OK>
1: 3E[>]<OK>
1: 3E[>]<OK>
1: 3E[>]<OK>
1: 3E[>]<OK>
1: 3E[>]<OK>
1: 3E[>]<OK>
```

```
48    70
49    80
50    90
51     0
52    10
53    20
54    30
55    40
56    50
57    60
58    70
59    80
60    90
```

Write File

W *fh bytes*

- *fh* is a file handle (1 - 4)
- *bytes* is the number of bytes to be written. You must send this number of bytes to return to the command prompt.

Description
You can write up to 512 bytes at a time with the ***Write*** command. If the bytes parameter is omitted, then 512 bytes will be expected on the incoming serial stream. Data is accepted and written to the card directly (there is no escape sequence).
By default, there is no time-out for how long it takes to send the bytes to the µMMC. This means that the µMMC will wait indefinitely for all the bytes to be sent (unless the power is removed, or a time-out value – setting 1 – has been set using the ***Settings*** command).
If you assign a value to the time-out setting using the ***Settings*** command, then the ***Write*** command will terminate, write the accepted bytes to the file, and return to the command prompt; no error will be returned.
If the file has been opened for append, the ***Write*** command will append all bytes to the end of the file.

Example

```
>W 1 18{cr}
13:22:02 ADC1=4.9V>
```




Calcul de la moyenne de 1000 données numériques

Appui sur BP4 calcul de la somme de tous les nombres du fichier E1000.txt

```
364         // Calcul de la somme des 1000 nombres flottants du fichier
365         // E1000.txt et calcul de la moyenne pour vérification
366         if ( BP4_Push == true )
367         {
368             OuvertureFichierLecture(1,NomFichierFloat10);
369             DepartElement=1;
370             Somme=0.0;
371             for(i=1;i<1001;i++)
372             {
373                 NumeroElement=i;
374                 NombreLu=LectureFichierFloat10(NumeroElement,10);
375                 Somme=Somme+NombreLu;
376                 CyDelay(20);
377             }
378             FermetureFichier(1);
379             sprintf(tstr,"%10.3f",Somme/1000.0);
380             CharLCD_Position(3,0);
381             CharLCD_PrintString(tstr);
382         }
```

STI2D Projet SIN





Les fonctions utilisées dans le projet PSoC

La fonction d'ouverture du fichier en lecture seule

```
75  /*****************************************************************************************/
76  void OuvertureFichierLecture(int NumeroHandle,char NomFichier[])
77  /*****************************************************************************************
78
79   Fonction qui ouvre un fichier présent sur la MMC en lecture
80
81   Paramètre en entrée    : le numéro du handle que l'on veut utilisé de 1 à 4 pour la MMC
82
83                          : le nom du fichier par exemple E1000.txt  ! syntaxe DOS 8.3
84
85   Attention il n'y a pas de contrôle ni timeout
86
87  ************************************************************** P.G 2013 1.0 ********/
88  {
89      char tstr[20];
90      char CommandeOuverture[20];
91
92      // Construction de la commande
93      // O 1 R /toto.txt
94      strcpy(CommandeOuverture,"O ");
95      sprintf(tstr,"%1d",NumeroHandle);
96      strcat(CommandeOuverture,tstr);
97      strcat(CommandeOuverture," R /");
98      strcat(CommandeOuverture,NomFichier);
99      // Envoi de la requête à la carte MMC
100     UART_MMC_PutString(CommandeOuverture);
101     UART_MMC_WriteTxData(0x0D);
102     // Attente de la réponse
103     do ch_MMC = UART_MMC_GetChar(); while(ch_MMC!=0x3E);
104 }
```

STI2D Projet SIN





La fonction d'ouverture du fichier en mode append

```
106  /*****************************************************************************************/
107  void OuvertureFichierAppend(int NumeroHandle,char NomFichier[])
108  /*****************************************************************************************
109
110   Fonction qui ouvre un fichier présent sur la MMC en mode append
111
112   Paramètre en entrée    : le numéro du handle que l'on veut utilisé de 1 à 4 pour la MMC
113
114                          : le nom du fichier par exemple E1000.txt  ! syntaxe DOS 8.3
115
116   Attention il n'y a pas de contrôle ni timeout
117
118  ********************************************************************* P.G 2013 1.0 ********/
119  {
120      char tstr[20];
121      char CommandeOuverture[20];
122
123      // Construction de la commande
124      // O 1 A /toto.txt
125      strcpy(CommandeOuverture,"O ");
126      sprintf(tstr,"%1d",NumeroHandle);
127      strcat(CommandeOuverture,tstr);
128      strcat(CommandeOuverture," A /");
129      strcat(CommandeOuverture,NomFichier);
130      // Envoi de la requête à la carte MMC
131      UART_MMC_PutString(CommandeOuverture);
132      UART_MMC_WriteTxData(0x0D);
133      // Attente de la réponse
134      do ch_MMC = UART_MMC_GetChar(); while(ch_MMC!=0x3E);
135  }
```

STI2D Projet SIN




La fonction de fermeture de fichier

```
139  /*****************************************************************************************/
140  void FermetureFichier(int NumeroHandle)
141  /*****************************************************************************************
142
143   Fonction qui ferme un handle de fichier préalablement ouvert
144
145   Paramètre en entrée   : le numéro du handle que l'on veut utilisé de 1 à 4 pour la MMC
146
147   Attention il n'y a pas de contrôle ni timeout
148
149  ********************************************************************* P.G 2013 1.0 ********/
150  {
151      char tstr[20];
152      char CommandeFermeture[20];
153
154      // Construction de la commande
155      strcpy(CommandeFermeture,"C ");
156      sprintf(tstr,"%1d",NumeroHandle);
157      strcat(CommandeFermeture,tstr);
158      // Envoi de la requête à la carte MMC
159      UART_MMC_PutString(CommandeFermeture);
160      UART_MMC_WriteTxData(0x0D);
161      // Attente de la réponse
162      do ch_MMC = UART_MMC_GetChar(); while(ch_MMC!=0x3E);
163
164  }
```

STI2D Projet SIN





Exemple complet de traitement : lecture d'un fichier de 1000 floats écrits sous forme Ascii

Etude de la structure du fichier

Le format est constitué d'un nombre de type float 10.3 pour chaque ligne. Une image binaire du début du fichier est donnée ci-dessous :

```
      00 01 02 03  04 05 06 07  08 09 0A 0B  0C 0D 0E 0F  0123456789ABCDEF
0000: 20 20 20 20  20 30 2E 30  30 30 0D 0A  20 20 20 20   0.000..
0010: 33 30 2E 30  30 30 0D 0A  20 20 20 38  36 30 2E 30  30.000..   860.0
0020: 30 30 0D 0A  20 20 20 32  30 30 2E 30  30 30 0D 0A  00..   200.000.
0030: 20 20 20 32  37 30 2E 30  30 30 0D 0A  20 20 20 36   2
0040: 37 30 2E 30  30 30 0D 0A  20 20 20 33  31 30 2E 30  70.0
0050: 30 30 0D 0A  20 20 20 31  36 30 2E 30  30 30 0D 0A  00..
0060: 20 20 20 33  37 30 2E 30  30 30 0D 0A  20 20 20 34
0070: 32 30 2E 30  30 30 0D 0A  20 20 20 20  38 30 2E 30  20.000..    80.0
0080: 30 30 0D 0A  20 20 20 34  37 30 2E 30  30 30 0D 0A  00..   470.000..
0090: 20 20 20 20  37 30 2E 30  30 30 0D 0A  20 20 20 38    70.000..   8
00A0: 34 30 2E 30  30 30 0D 0A  20 20 20 20  35 30 2E 30  40.000..    50.0
00B0: 30 30 0D 0A  20 20 20 32  39 30 2E 30  30 30 0D 0A  00..   290.000..
```

Dump binaire

Vision du fichier E1000.txt

Editeur Notepad++

1	0.000
2	30.000
3	860.000
4	200.000
5	270.000
6	670.000
7	310.000
8	160.000
9	370.000
10	420.000
11	80.000
12	470.000
13	70.000
14	840.000
15	50.000
16	290.000
17	910.000
18	360.000
19	770.000
20	320.000
21	690.000
22	840.000
23	710.000
24	300.000
25	160.000

☞ Donner le nombre d'octets par ligne ?

☞ Quels sont les deux derniers octets de chaque ligne ?

☞ Que signifient t-ils ?

Analyse de la réponse de la carte MMC

La carte MMC répond à une requête de lecture de 12 octets par une séquence de 14 octets déterminer la signification de chacun des octets de cette trame :

```
14: 20 20 20 34 37 30 2E 30 30 30 0D 0A 20 3E[    470.000.. >]<OK>
```

Trame retour de la carte MMC

STI2D Projet SIN





La fonction lecture fichier float

```
173  //#define TAILLE_BLOC_DATA 10
174  //#define TAILLE_BLOC_DATA_LG       (TAILLE_BLOC_DATA+1)
175  //#define TAILLE_BLOC_DATA_CR_LF    (TAILLE_BLOC_DATA+2)
176  //#define TAILLE_BLOC_REPONSE_MMC   (TAILLE_BLOC_DATA+4)
177
178  /*****************************************************************************************/
179  float LectureFichierFloat10(int NumeroElement,int TailleBloc)
180 /*****************************************************************************************
181
182   Fonction qui lit un fichier texte composé d'un unique nombre de type float par
183   ligne de fichier texte. Chaque nombre est codé sur 10 caractères et est suivi de
184   CR et LF. Attention il n'y a aucun contrôle de requête de lecture au delà de la fin
185   du fichier.
186
187   Paramètre en entrée    : le numéro de la ligne du fichier à lire (première ligne n°1)
188
189                            la taille du bloc sans compter le CR et LF de chaque fin de
190                            ligne
191
192   Retour de la fonction : le nombre float lu
193
194  ********************************************************************* P.G 2013 1.0 ********/
```

Cette fonction réalise la lecture d'un nombre dans le fichier E1000.txt

Elle devra être adaptée en fonction de vos besoins, néanmoins le format fixe facilite le traitement des données car il permet un accès direct à n'importe quelle data enregistrée dans le fichier il suffit de se placer au bon endroit.

Le driver de la carte SD réalise cela très simplement car il est possible de préciser l'adresse de lecture dans la requête read.

Le stockage des nombres sous formes de chaine de caractères permet de s'affranchir des représentations binaires des nombres propres à chaque machine. De plus la vérification, voire la modification ou la création de fichiers est possible avec un éditeur de type Notepad++.

STI2D Projet SIN




```
195 {
196     char  CommandeLecture[20];
197     char  CommandeLectureUneData[20]="R 1 12 ";
198     char  ChaineValeur[20];
199     int   AdresseElement;
200     float NombreLu=0.0;
201     int TAILLE_BLOC_DATA;
202     int TAILLE_BLOC_DATA_LG;
203     int TAILLE_BLOC_DATA_CR_LF;
204     int TAILLE_BLOC_REPONSE_MMC;
205 
206     TAILLE_BLOC_DATA=TailleBloc;
207     TAILLE_BLOC_DATA_LG=(TAILLE_BLOC_DATA+1);
208     TAILLE_BLOC_DATA_CR_LF=(TAILLE_BLOC_DATA+2);
209     TAILLE_BLOC_REPONSE_MMC=(TAILLE_BLOC_DATA+4);
210 
211     // Calcul de l'adresse de l'élément, chaque nombre float dans le fichier
212     // occupe 10 caractères + CR + LF donc 12 au total.
213     // Le premier commence à 1
214     // Le suivant est stocké à partir de 13 ...
215     AdresseElement=(NumeroElement-1)*TAILLE_BLOC_DATA_CR_LF + 1;
216     // On convertit l'adresse calculée en chaine de caractères
217     sprintf(tstra,"%4d",AdresseElement);
218     // On convertit également le numéro de l'élément
219     sprintf(tstrc,"%4d",NumeroElement);
220     // On recopie dans la chaine CommandeLecture la requête de lecture
221     // dans la MMC soit "R 1 12 "
222     strcpy(CommandeLecture,CommandeLectureUneData);
223     // On ajoute ensuite à la fin l'adresse (concaténation)
224     strcat(CommandeLecture,tstra);
225     // Envoi de la requête de lecture
226     // La réponse fera 2 caractères de plus '0x20' et '0x3E'
227     // soit un espace ' ' et le caractère d'acquitement 0x3E.
228     UART_MMC_PutString(CommandeLecture);
229     UART_MMC_WriteTxData(0x0D);
230 
```

STI2D Projet SIN





```
232     // Réception des caractères en provenance de la MMC
233     // on attend TAILLE_BLOC_REPONSE_MMC caractères
234     count_MMC=0;
235     do
236     {
237         // Réception d'un caractère sur l'UART en provenance
238         // de l'UMMC
239         ch_MMC = UART_MMC_GetChar();
240         if ( ch_MMC > 0 )
241         {
242             // Recopie sur l'UART pour debug
243             UART_WriteTxData(ch_MMC);
244             // Accumulation des caractères dans tampon_MMC
245             tampon_MMC[count_MMC] = ch_MMC;
246             count_MMC++;
247         }
248     }
249     while(count_MMC<(TAILLE_BLOC_REPONSE_MMC));
250
251     // On supprime les derniers caractères non significatifs en plaçant au bon
252     // endroit le caractère de fin de chaine '\0'
253     // La chaine reçue est constituée ainsi :
254     // 10 caractères pour le float
255     // CR
256     // LF
257     // espace
258     // 3E fin de requête MMC
259     // Le '\0' est placé à la place du CR
260     tampon_MMC[count_MMC-4]='\0';
261     // On recopie la chaine résultante dans ChaineValeur soit 11
262     // caractères au total
263     strncpy(ChaineValeur,tampon_MMC,TAILLE_BLOC_DATA_LG);
264     // Conversion en float
265     NombreLu=atof(ChaineValeur);
266     // Re-conversion en chaine pour contrôle
267     sprintf(tstrb,"%9.3f",NombreLu);
268     // On retourne la valeur lue dans le fichier au format float
269     return(NombreLu);
270
271
272 }
```

☞ Vous pouvez vous penchez avec le plus grand intérêt sur le détail des manipulations des chaines de caractères.

⇦