HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD

www.heig-vd.ch

Auteurs: Richard Ludovic
Passuello Arthur
Professeur: Romuald Mosqueron (RMQ)
Assistant: Anthony Convers (ACS)
Salle: A07

ASP-A

LABORATOIRE 3: SDCARD
Rapport

5 juin 2017

Table des matières

1 Introduction

Le présent rapport présente notre travail pour le 3ème laboratoire. Il s'agit de réaliser des fonctions bas et de haut niveau pour lire et écrire dans une carte SD. Voici la liste des commandes envoyées à la carte pour l'implémentation des fonctionnalités désirées :

Numéro	Nom	Argument	Réponse
CMD0	GO_IDLE	No	Null
CMD2	SEND_CID	No	136 bits, Content of CID/CSD register ;128 LSB
CMD3	SEND_REL_AD	No	48 bits, RCA in [31 :16]
ACMD6	SET_BUS_WDT	[1 :0] : width	48 bits, card status, cmd index, ..
CMD6	SWITCH_FUN	[3 :0] :access mode	48 bits, card status, cmd index, ..
CMD7	SEL_CARD	[31 :16] : RCA	48 bits, card status, cmd index, ..
CMD8	SEND_IF_CD	[11 :0] : voltage and check	48 bits, card interface condition
CMD 9	SEND_CSD	[31 :16] : RCA	136 bits, Content of CID/CSD register :128 LSB
CMD10	SEND_CID	[31 :16] : RCA	136 bits, Content of CID/CSD register :128 LSB
CMD12	STOP_TRANS	No	48 bits, card status, cmd index, ..
CMD17	READ_BLOCK	[31 :0] : data adress	48 bits, card status, cmd index..
CMD18	READ_BLOCKS	[31 :0] : data adress	48 bits, card status, cmd index..
CMD24	WRITE_BLOCK	[31 :0] : data adress	48 bits, card status, cmd index..
CMD25	WRITE_BLOCKS	[31 :0] : data adress	48 bits, card status, cmd index..
ACMD41	SENd_OP_CON	host capacity, VDD	48 bits, card status, cmd index..
CMD55	APP_CMD	[31 :16] : RCA	48 bits, card status, cmd index..

2 Étape 1 : Initialisation de la carte SD

2.1 Objectifs

Comprendre le mécanisme d'initilisaton de la carte ainsi que d'envoi de commande.

2.2 Imlémentation

Toutes les méthodes d'initialisation sont fournies avec la donnée. On remarque que les opérations suivantes sont effectuées :

- **mmc1_pad_conf** : configure les pins de la carte pour placer le lecteur SD en entrée/sortie (globalement)
- **mmchs_conf** : initialise le controleur de la DM3730 le MMCHS1 (horloge, interface, software reset, lignes de commande et données, le comportement en mode *idle* ainsi que les registres de flag)
- **sdhc_init** : prépare la carte SD en configurant correctement les flags de MMCHS_CON, en engageant l'initialisation, puis en configurant les différents paramètres (voltage, etc...) décrits dans les commandes ci dessus. Elle lit ensuite les informations essentielles de la carte (adresse, CSD, CID, ...), et place la carte dans le mode *data transfer* et l'état *Transfer*

3 Étape 2 : Lecture et Affichage des informations de la carte SD

3.1 Objectifs

Est demandé d'implémenter les méthodes suivantes :

— **read_product_name()** qui récupère le nom de la carte.

— **read_card_size()** qui récupère la taille de la carte.

3.2 read_product_name

Les informations qu'il est demandé de lire depuis la carte ont été placées dans la variable globale $\mathrm{cid\_reg}$ lors de l'initialisation. Il s'agit d'un tableau de type vulong etde longueur 4. Chacun de ses éléments comporte donc 32 bits d'informations, représentant ainsi la réponse sur 128 bits de la carte. Seuls les bits [103..64] de cette réponse représentent le nom de la carte (5 char). La réponse est placée dans un tableau de pointeurs sur char mais la méthode ne retourne rien.

Pour cette partie, il est donc nécessaire de *parser* des mots de 32 bits en mots de 8 bits au moyen de masques sélectionnant à chaque fois 8 bits parmis les 32 et les places dans le tableau passé en argument. Le premier char trouve dans les 8 LSB de $\mathrm{cid\_reg}[3]$, les 4 autres sont dans $\mathrm{cid\_reg}[2]$). Enfin, le caractère de fin de ligne est placé en fin de tableau.

3.3 read_product_size

Les informations qu'il est demandé de lire depuis la carte ont été placées dans la variable globale $\mathrm{csd\_reg}$ lors de l'initialisation. Il s'agit d'un tableau de type vulong et de longueur 4. Chacun de ses élément comporte donc 32 bits d'informations, représentant la réponse sur 128 bits de carte. Seuls les bits [69..48] de cette réponse représentent la taille de la carte.

Ainsi, la valeur de $\mathrm{C\_SIZE}$ est répartie dans $\mathrm{csd\_reg}[1]$ dont les 16 MSB en font partie, et dans $\mathrm{csd\_reg}[2]$ dont les 6 LSB en font partie.
Une fois la valeur de $\mathrm{C\_SIZE}$ extraite et placée dans une variable size, l'opération suivante est nécessaire pour obtenir le chiffre retourné par la méthode qui sera la taille de la carte en KiBytes :

$$size\_f = (size + 1) * SD\_BLOCK\_LENGTH$$

4 Remarques sur les lectures/écritures

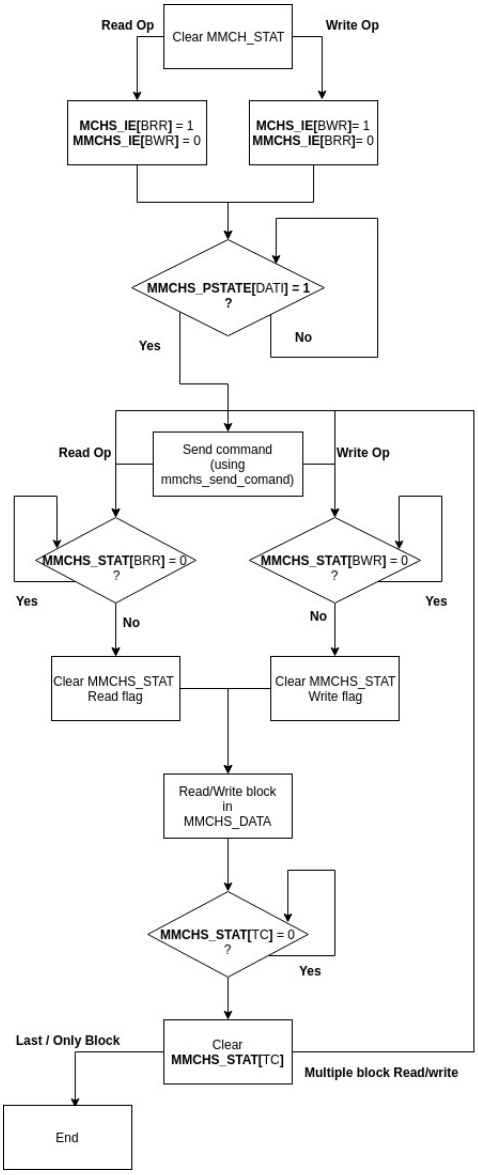
La carte SD doit être vue comme une machine à état. En effet celle-ci, lors de sa mise sous tension, est en *card identification mode*, il est donc nécessaire de l'initialiser au moyen d'un dialogue commande/réponse avant de commencer à échanger des données avec celle-ci.

Fort heureusement, les méthodes fournies et décrites dans l'**Étape 1** effectuent pour nous ces opérations et laissent la carte dans l'état *Transfer* du mode *data transfer*.
Depuis ce mode, il nous suffit alors de d'envoyer la commande correspondante à l'opération que nous voulons effectuer pour chercher l'état *Receive-data* ou *Sending-data* et commencer à échanger avec la carte.

Au delà des données elles-même, il est aussi nécessaire de lire et d'écrire dans des registres de la DM3730, plus précisément ceux du MMCHS (MultiMedia Card High-Speed host controller) afin de préciser au système le type d'opération voulue et d'obtenir des informations sur son état comme décrit ci dessous :

	MMCHS_STAT	RW
BWR	Write 1 : clear status | Read 1 : Ready to write, 0 : Not ready	bit[4]
BRR	Write 1 : clear status | Read 1 : Ready to read, 0 : Not ready	bit[5]
TC	Write 1 : clear status | Read 1 : data transfer complete, 0 : Not	bit[1]
	MMCHS_IE	RW
BWR_EN	0 : Buffer Write Interrupt disabled, 1 : Enabled	bit[4]
BRR_EN	0 : Buffer Read Interrupt disabled, 1 : Enabled	bit[5]
	MMCHS_PSTATE	R
CMDI	0 : Using mmci_cmd line allowed, 1 : Not allowed	bit[0]
DATI	0 : Using mmci_dat lines allowed, 1 : Not allowed	bit[1]
BWE	0 : Buffer is ready to be written, 1 : Not ready	bit[10]
BRE	0 : Buffer is ready to be read 1 : Not ready	bit[11]
	MMCHS_DATA	RW
DATA	contains 32 bits of data, access granted by BWE & BRE above	bit[31..0]

Ainsi, pour chaque transfert de donnée, on observe le déroulement suivant :



5 Étape 3 : lire un bloc

5.1 Objectifs

Implémenter la méthode **mmchs_read_block** qui permet de lire un bloc de donnée indiqué par l'argument block et de placer la donnée à l'adresse pointée par l'argument data

5.2 Implémentation

La méthode commence par placer les bonnes valeurs dans les registres du MMCHS1, puis, lorsque la ligne MMCHS_DATI est disponible, la commande 17 (*read single block*) est envoyée. Lorsque le buffet est prêt et que la ligne MMCHS_DATA contient l'information voulue, la donnée est lue mot par mot au moyen d'une boucle.
Lorsque le transfert est terminé, le flag *TransferComplete* de MMHS_STAT est cleared et la fonction retourne.

6 Étape 4 : lire plusieurs blocs

6.1 Objectif

Implémenter la méthode **mmchs_read_multiple_block** qui permet de lire nblocks blocs indiqués par l'argument block et de placer la donnée à l'adresse pointée par data.

6.2 Implémentation

Les registres du MMCHS1 sont écris et lus de manière analogue a la fonction précédement décrite. C'est cependant la commande 18 (*read multiple blocks*) qui est envoyée à la carte. La méthode entre ensuite dans une boucle au début de laquelle, à chaque début de bloc, le flag *BRR* de MMCHS_STAT est lu en boucle puis cleared afin de vérifier la disponibilité de la carte pour la lecture (ie que la donnée est bien présente sur la ligne MMCHS_DATA), avant de procéder au transfert comme précédemment.

Lorsque le transfert est terminé, et après avoir clear le flag *TC* de MMCHS_STAT, la commande 12 (*stop transmission*) est envoyée à la carte et la méthode retourne.

7 Etape 5 : écrire un bloc

7.1 Objectifs

Implémenter la méthode **mmchs_write_block** qui permet d'écrire dans un bloc de donnée indiqué par l'argument block la donnée à l'adresse pointée par l'argument data

7.2 Implémentation

La méthode commence par placer les bonnes valeurs dans les registres du MMCHS1, puis, lorsque la ligne MMCHS_DATI est disponible, la commande 24 (*write single block*) est envoyée. Lorsque le buffet est prêt et que la ligne MMCHS_DATA est prete a recevoir l'information, la donnée est écrite mot par mot au moyen d'une boucle.
Lorsque le transfert est terminé, le flag *TransferComplete* de MMHS_STAT est cleared et la fonction retourne.

8 Étape 6 : écrire plusieurs blocs

8.1 Objectif

Implémenter la méthode **mmchs_write_multiple_block** qui permet d'écrire nblocks blocs indiqués à l'endroit indiqué par l'argument block la donnée à l'adresse pointée par data.

8.2 Implémentation

Les registres du MMCHS1 sont écris et lus de manière analogue a la fonction précédement décrite. C'est cependant la commande 25 (*write multiple blocks*) qui est envoyée à la carte. La méthode entre ensuite dans une boucle au début de laquelle, à chaque début de bloc, le flag *BWR* de MMCHS_STAT est lu en boucle puis cleared afin de vérifier la disponibilité de la carte pour l'écriture (ie que le buffer est bien prêt a recevoir des informations sur la ligne MMCHS_DATA), avant de procéder au transfert comme précédemment.

Lorsque le transfert est terminé, et après avoir clear le flag *TC* de MMCHS_STAT, la commande 12 (*stop transmission*) est envoyée à la carte et la méthode retourne.

9 Remarques sur la librairie FatFs et le menu fourni

La suite du laboratoire sera plus haut niveau. En effet, une librairie nommée FatFs nous est fournie. Cette librairie implémente ses propres fonctions bas niveau, et donc, initialise la carte correctement et respecte les conventions du protocole de communication (machine d'état). Elle permet de lire et d'écrire sur une carte utilisant le système de fichier FAT32.
Voici la liste des types de variable, définis par librairie, qui sont utilisés dans les étapes suivantes :

- **FATFS** : correspond au système de fichier FAT32. Ce type de variable est utilisé de manière générale pour monter et démonter la carte SD (avec la fonction f_mount)
- **FIL** : contient un fichier
- **FILINFO** : contient les informations relatives à un fichier (metadata). Les composants de cette struct sont détaillé à l'étape 10.
- **DIR** : contient un dossier
- **WORD** : simple redéfinition d'un unsigned short

De plus, un code permettant d'afficher un menu à option et de naviguer à l'intérieur nous a été fourni. Les méthodes que nous écrirons par la suite seront utilisées pour mettre en oeuvre les fonctionnalités proposées par ce menu.

Pour les prochaines étapes, notre carte aura donc été formatée (avec windows) pour utiliser le système de fichier FAT32

10 Étape 7 : lister les fichiers de la carte SD

10.1 Objectifs

Il est demandé d'implémenter la méthode suivante :

- **scan_files(char *path)** qui scan et affiche le contenu d'un dossier (fichiers et dossiers) et ce, de manière récursive.

10.2 scan_files

La fonction scan_files ouvre tout d'abord le dossier donné en paramètre ("" pour la racine de la carte) grâce à la fonction f_opendir. Elle parcourt ensuite tous les fichiers/dossiers présents dans le répertoire ouvert. Grâce à la variable de type FILINFO renvoyée par la fonction f_readdir, elle est capable de déterminer si c'est un fichier ou un dossier. Si c'est un fichier, elle affiche son nom sur l'écran de la REPTAR (avec la fonction fb_print_string, écrite dans le labo précédent). Si un dossier, elle affiche son nom et s'appelle récursivement avec le chemin de ce nouveau dossier en paramètre.

11 Etape 8 : lire un ficher de la carte SD

11.1 Objectifs

Il est demandé d'implémenter la méthode suivante :

— **read_files(char *file_name)** qui lit un fichier et affiche son contenu sur l'écran de la REPTAR

11.2 read_files

La fonction read_file commence par effacer l'écran de la REPTAR. Elle ouvre ensuite le fichier dont le chemin et le nom sont donnés en paramètre en mode lecture, grâce à la fonction f_open. Le flag F̈A_OPEN_EXISTINGp̈ermet de créer une erreur si le fichier n'existe pas. Une fois l'ouverture du fichier effectuée, elle utilise f_read pour en lire le contenu. Elle lit 1 cluster à la fois, soit 8 blocs de 512 bytes, et l'affiche sur l'écran de la REPTAR. La fonction F_read s'arrête si elle détecte la fin du fichier et renvoie le nombre de bytes lus. En comparant ce nombre de byte avec la taille d'un cluster, il est possible de déterminer si nous sommes arrivés à la fin du fichier ou non. Une fois la lecture du fichier terminée, la fonction read_files ferme le fichier grâce à la fonction f_close.

12 Étape 9 : écrire un fichier dans la carte SD

12.1 Objectifs

Il est demandé d'implémenter la méthode suivante :

— **create_file(char *file_name, uchar *data, ulong nbyte))** qui lit un fichier et affiche son contenu sur l'écran de la REPTAR

12.2 create_files

On commence par vérifier que le nombre de bytes à lire ne dépasse pas la limite d'un cluster (8 blocs de 512 bytes). On ouvre ensuite le fichier en mode écriture avec f_open et le flag "FA_CREATE_ALWAYS" qui écrasera le fichier s'il existe déjà. On écrit ensuite les caractères reçu en paramètre dans le fichier et on le ferme. Le fonctionnement de cette fonction a été vérifié en lisant avec un PC un fichier écrit de cette manière.

13 Étape 10 : afficher les informations d'un fichier de la carte SD

13.1 Objectifs

Il est demandé d'implémenter la méthode suivante :

— **print_file_info(char *file_name)** qui lit les metadata d'un fichier et qui en affiche les détails sur l'écran de la REPTAR

13.2 la structure FILINFO

la fonction print_file_info exploite toutes les données contenu dans une struct de type "FILINFO". En voici la liste :

- **fname** : contient le nom du fichier
- **fsize** : contient la taille du fichier en byte
- **fdate** : contient la date de la dernière modification comme suit : bits 0 à 4 pour le jour, bits 5 à 8 pour le mois et bits 9 à 15 pour l'année
- **ftime** : contient l'heure de la dernière modification comme suit : bits 0 à 4 pour les secondes, bits 5 à 10 pour les minutes et bits 11 à 15 pour l'heure
- **fattrib** : contient la liste des attributs du fichier sous forme de flags (AM_DIR, AM_RDO, AM_HID, AM_SYS and AM_ARC)

13.3 print_file_info

Après avoir effacé l'écran, on récupère les informations d'un fichier grâce à la fonction f_stat. On affiche ensuite toutes les infos (voir ci-dessus), ligne par ligne, sur l'écran de la REPTAR.

14 Conclusion

Ce laboratoire était long mais a permis d'appréhender la gestion d'une carte SD en bas niveau comme en haut niveau. Cette fois encore, le gros du travail a été la lecture et la compréhension de la documentation. Nous avec remarquer qu'avec l'entrainement, cela est de plus de plus rapide.