MI43 – Final

1. Interface simplifiée

Une interface est connectée à bus de type USB. On ne s'intéresse pas ici aux spécificités de ce bus. Le système informatique auquel appartient l'interface considéré est de type 'device'. On souhaite envoyer des données à un autre système informatique ('l'hôte') .

1) Description (sans entrer dans les détails des transactions du bus)

A intervalles régulier, l'hôte génère une transaction de type IN vers notre système. L'interface répond et envoie un paquet de données vers l'hôte. Pour que la transaction soit un succès, les données émises par l'interface doivent être présente dans le buffer de l'interface avant le début transaction IN, dans le cas contraire, la transaction est abandonnée et elle sera retentée plus tard.

On s'intéresse ici uniquement à l'envoi des données, on suppose les initialisations faites.

2) Principe d'envoi des données pour le logiciel

- déplacement d'un paquet de données de la mémoire de l'application vers le buffer de l'interface
- définition du nombre de données transférer (packet_size)
- validation du buffer (validate_buffer)
- Le transfert des données se produit au moment ou l'hôte lance une transaction (instant inconnu)
- lorsque toute les données ont été transférée, l'interface génère une interruption pour signaler que l'envoi est terminé.
- Le buffer de l'interface est alors vidé (flush_buffer) afin de procéder au transfert d'un autre packet
- Déplacement d'un autre paquet de données …..

3) Description des registres utiles de l'interface.

- *status* : adresse 0xFFFFFF00, registre en lecture seul.
  - bit 0 : à 1 signal qu'un transfert à eu lieu, si les interruptions de l'interface ont été autorisée, ce bit signale également qu'une requête d'interruption est en cours. Remis à 0 après une opération *flush_buffer*
- *control* : adresse 0xFFFFFF04, registre 32 bits en lecture écriture
  - bit 0 : à 0 interruptions non autorisées, à 1 interruptions autorisées
  - bit 1 : bit *validate_buffer*, lorsque mis à 1 les données transférées dans l'interface sont validées, se remet automatiquement à 0
  - bit 2 : bit *flush_buffer*, lorsque mis à 1 le buffer de l'interface est vidé, se remet automatiquement à 0
- *packet_size* : adresse 0xFFFFFF08, registre 32 bits en lecture écriture, taille du paquet à transférer
- *data* : adresse 0xFFFFFF0C, registre 32 bits en écriture seul permettant de remplir le buffer de l'interface

Après l'initialisation tous les registres sont à 0. la taille du packet maximal qui peut être stocké dans l'interface est MAX_PACKET_SIZE.

4) Environnement et pilote

Le système est développé avec le système d'exploitation multitâche FreeRTOS. Les applications disposent pour envoyer des données à l'hôte de la fonction suivante :

*void write (char * data, unsigned int len) ;*

Cette fonction est une fonction bloquante, elle ne revient que lorsque toutes les données ont été transférées. D'autres tâches tournant sur le système, il convient que les attentes inévitables lors du transfert se fasse de manière passives (n'utilisant pas de temps CPU). Pour ce faire le pilote effectuant le transfert doit utiliser des sémaphore binaires :

- appel de la fonction utilisateur par l'application
- lancement du transfert du premier paquet de données
- attente passive d'une interruption
- lancement du transfert du second paquet de données
- ….
- lancement du dernier paquet de données
- attente passive d'une interruption

- retour de la fonction utilisateur

la routine de traitement des interruptions est appelée à chaque interruption de l'interface, elle à le prototype suivant :

void isr(void)

5) Questions

1. Écrire le code commenté du pilote

- routine utilisateur
- routine de traitement des interruptions

détailler clairement les variables utilisées et les opérations réalisées par les différentes routines.

2. Quel problème peut présenter dans un environnement multitâche comme ici, l'appel par plusieurs tâche de la routine utilisateur *write* ? Comment y remédier ?

2. Questions de cours (6pts)

1. Quel est le rôle du contrôleur VIC pour un processeur ARM7 tel qu'utilisé en TP.

2. Expliquer les différentes manières possible pour un système d'exploitation de reprendre la main sur une tâche en cours d'exécution.

3. Expliquer pourquoi une tâche d'un OS tel que FreeRTOS s'exécutant sur un système tel que celui utilisé en TP peuvent corrompre les données d'autres tâches. Comment ceci est-il évité pour des processus s'exécutant avec des OS tels que linux ou windows.