MI43 – Final

Documents autorisés : documentation constructeurs uniquement et transparent de cours
Durée 2h. Tout code non expliqué sera considéré non valable

1. Interruptions (8)

On s'occupe ici que de la gestion d'une requête d'interruption utilisant le gestionnaire de priorité AIC de L'AT91.

1. Rappeler ce qu'il se passe lorsqu'une requête d'interruption de priorité 5 arrive sur le contrôleur AIC.
2. Comment est prise en compte cette requête par le logiciel et quelle sont les conséquences au niveau du contrôleur AIC
3. Donner le code assembleur arm d'une routine de traitement des interruptions avec le cahier des charges suivant

   Traitement par une routine externe appelée *isr*
   Le traitement doit pouvoir être interrompu par le traitement d'une requête d'interruption plus prioritaire
   On souhaite que tout le traitement soit terminé avant de prendre en compte des requêtes d'interruptions de priorité inférieures ou égales
4. Donner le code d'initialisation de l'AIC si l'interruption est l'irq0. Cette requête d'interruption est de type niveau bas et on souhaite une priorité de 5.

2. Envoi de données série utilisant l'USART de l'AT91 (12)

On utilise l'usart 1 de l'at91pour envoyer des données via la liaison série. L'application poste les envois de données en appelant une fonction appelée *us1_write*. Cette fonction prend en paramètre une adresse sur les données à transmettre et le nombre d'octets à transmettre. Cette fonction est chargée le mettre en place l'envoi de données et peut revenir avant que toutes les données n'ai fini d'être transmises (fonction non bloquante). On suppose dans un premier temps que lorsque cette fonction est appelée, aucun envoi de données n'est en cours.

Pour que cette fonction soit non bloquante, il est nécessaire d'utiliser les requêtes d'interruption que peut générer l'usart.

Dans cet exercice il s'agit donc d'écrire la routine d'envoi que l'on nommera *us1_write* et la routine de traitement des interruptions que l'on nommera *us1_isr*. La routine de traitement des interruptions gère les requêtes d'interruption pour la réception de données également. On ne s'en occupe pas dans cet exercice, cette partie sera laissée en blanc, mais cela doit être pris en compte pour l'écriture du code.

On considère ici que l'usart est paramétrée avec l'émission désactivée (UX_TXDIS) ainsi que les interruptions pour l'émission désactivée, on ne s'occupe donc pas du paramétrage de l'USART (vitesse, parité, ...), ni de l'AIC. De même on ne s'occupe pas de la routine de traitement des interruptions assembleur en charge de l'AIC et des sauvegardes. La routine de traitement des interruptions *us1_isr* ne s'occupe que de l'interface USART et peut être écrite en C.

prototypes :

```
void us1_write(char * data, unsigned int len);
void us1_isr(void);
```

1) Envois simples

1. Donner le code de la fonction *us1_write*
2. Donner le code de la fonction *us1_isr*

2) Gestion de requêtes multiples

Afin que l'appel de la fonction *us1_write* puisse se faire alors qu'une émission est en cours et pour qu'elle reste toujours non bloquante, on souhaite travailler en utilisant des files d'attente. Le principe consiste à placer des requêtes d'émission dans une liste chaînée ou un tableau. Ces requêtes correspondent à des bon de travail qui seront traités les uns après les autres :

- Lors d'un appel de la fonction *us1_write*, si une requête est en cours de traitement, une nouvelle requête est créée et cette dernière est placée dans le tableau ou la liste chaînée pour un traitement ultérieur sinon une émission est mise en place tout de suite.
- En fin d'émission la routine de traitement des interruptions met en place une nouvelle émission si une requête est en attente.

Structure de données exemple (peut être modifié selon les besoins)

```
#typedef struct  write_request_s{
        char * data;                        /* adresse des données à envoyer */
        unsigned int len;                   /* nombre d'octets à envoyer */
        struct write_request next_request;  /* pointeur vers la requêtes suivante, NULL si pas de requêtes suivante */
} write_request_t;

write_request_t request_tab[5];             /* tableau de 5 requêtes */
write_request_t * write_request = NULL;     /* pointeur sur la requête en cours par ex  (allocation :  write_request = & request_tab[0] )*/
```

1. Détailler les modifications à apporter aux fonctions *us1_isr* et *us1_write* pour gérer ces requêtes multiples.
2. Donner les codes des fonction *us1_isr* et *us1_write* modifiées.