Microprocesseur

TP6 :UART / interruptions avancées

Ce TP reste focalisé sur la mise en œuvre de la liaison UART, mais cette fois-ci, nous allons nous intéresser à son utilisation en interruptions. Dans toutes les questions qui suivent, il vous est demandé de réaliser une animation de type *heartbeat* en scrutation pour bien garantir que vos interruptions ne monopolisent pas le temps processeur.

Les pages 66 et 67 de la datasheet du microcontrôleur contiennent la liste des sources d'interruption (IRQ) et des vecteurs. Pour rappel, plusieurs IRQs peuvent se partager le même vecteur d'interruption. C'est le cas des modules UART qui sont associés à trois IRQ (Réception, Emission, Erreurs), mais un seul vecteur (donc une seule fonction).

1) Interruptions en réception

a) Pour s'échauffer.

Avant même de traiter les données, réalisez un programme qui compte le nombre de caractères reçus par l'UART. Pour afficher le nombre de caractères, vous pouvez utiliser les afficheurs 7-segments ou l'écran LCD. Notez toutefois que les fonctions de gestion de l'écran LCD souffrent d'une latence élevée due à l'utilisation de fonctions d'attente bloquantes. Cette fonction doit bien entendu se faire en interruption.

b) Réalisation d'un echo.

A ce stade, vous pouvez réaliser une fonction d'écho (plus ou moins fantaisiste selon votre convenance). Par souci de simplicité, il est possible de négliger le test pour savoir s'il est possible d'envoyer des données. En effet, les caractères transitent dans les deux sens à la même vitesse. La réception limite donc le flux d'envoi, et le canal d'envoi sera toujours libre.

c) Interpréteur de commandes.

Cette fois, l'objectif est de réaliser un interpréteur de commandes. La fonction d'interruption doit recevoir les données et exécuter les commandes suivantes :

- 'C' + <num2> : positionne le curseur de l'écran LCD (0 à 15 ligne 1, 16 à 31 ligne 2)
- 'T' + <num2> + texte : affiche les <num> caractères suivants sur l'écran LCD
- 'S' + <num4> : affiche la valeur fournie sur l'afficheur 7 segments

<num2> et <num4> indiquent respectivement des entiers décimaux à 2 et 4 caractères. Cette fonctionnalité doit être entièrement effectuée dans l'interruption UART (ce qui est très moche car la gestion de l'écran LCD effectue des attentes bloquantes, c'est pas grave, on tapera les commandes à la main, donc l'envoi est très lent)

2) Interruptions en émission

a) Envoi d'un buffer.

L'intérêt des interruptions en envoi est de ne pas bloquer le processeur lors des attentes d'envoi des caractères. Ainsi, le fait de pouvoir envoyer une nouvelle donnée déclenche l'interruption, et cette dernière effectue l'envoi.

L'envoi en interruption doit donc se faire avec un bloc de données. Une variable globale de type *char* * définit où se trouve ce bloc, une autre variable globale de type *int* definit le nombre de caractères à envoyer. La fonction d'interruption, lorsqu'elle se déclenche, envoie le carcatère pointé, incrémente le pointeur et décrémente le compteur.
Quelle est l'action particulière à réaliser lorsqu'il n'y a plus de caractères à envoyer ?

Réalisez un programme qui envoie 'Bonjour le monde' ou toute autre phrase de plus de 12 caractères, à chaque seconde. L'attente des secondes s'effectuera en scrutation.

b) Utilisation d'une file d'attente.

Pour obtenir un comportement plus exploitable, il est intéressant d'utiliser un file d'attente (ou FIFO, pour First In First Out). Le rôle de l'interruption sera alors de vider la FIFO en envoyant son contenu au fur et à mesure des possibilités de la liaison. Le programme d'application pourra se contenter de remplir cette FIFO pour envoyer des données (l'interruption se chargeant de l'envoi réel). L'avantage est qu'il n'est pas nécessaire d'attendre la fin de l'envoi d'un bloc de données pour envoyer le bloc suivant.

Parmi les nombreuses implémentations possibles de FIFO, la plus simple est la FIFO circulaire :

- un tableau contient les élements à envoyer
- un entier (idx_w) contient l'indice d'écriture (le numero de la première case vide)
- un autre entier (idx_r) contient l'indice de lecture (le numero de l'élément le plus ancien)
- un dernier entier (nelts) contient le nombre d'éléments.

Écrire une donnée consiste à écrire dans tableau[idx_w], incrémenter idx_w et incrémenter nelts.
Lire une donnée consiste à lire tableau[idx_r], incrémenter idx_r et décrémenter nelts.
Bien entendu quelques précautions s'imposent :

- si idx_w (ou idx_r) dépasse la taille du tableau, il doit être remis à 0 (c'est le côté circulaire).
- Si nelts == 0, il n'y a aucun élement dans la FIFO, il faut donc refuser la lecture
- si nelt est égal à la taille du tableau, la FIFO est pleine, il faut donc au choix :
    - refuser l'écriture en mettant l'emetteur en pause (appel bloquant)
    - perdre l'élément le plus récent
    - écraser l'élément le plus ancien
    - crier très fort que tout est cassé

Rq : entre idx_w, idx_r et nelts, une des trois valeurs est superflue car elle peut être retrouvée à partir des deux autres modulo la taille du tableau. Mais ici, on va se la faire cool, on va pas se stresser pour économiser une variable (c'est mal ?).

Reprenez la fonctionnalité du a), en utilisant une FIFO pour les données en attende d'envoi. Il est conseillé d'écrire une fonction spécifique pour écrire dans la FIFO, voire deux : une pour les caractères isolés, une autre pour des chaines de caractères ou des buffers.

3) Si on se faisait un pilote de périphérique ?

a) fonctions de base.

Il ne manque pas grand-chose pour concevoir un pilote de périphériques complet :

- Gérer une FIFO pour les données en envoi
- Avoir une fonction de configuration

Ben du coup, y'a plus qu'à ….

b) questions philosophiques.

- La cohérence des structures de FIFO, on en parle ?
- Faut-il utiliser les FIFOs matérielles ?
- Que faut-il inclure comme fonctionnalités dans les interruptions ?