Compte rendu TME 1 - PERI

1 INTRODUCTION

Le but de ce TME était de nous faire piloter un écran LCD (HD44780) en mode utilisateur. Dans un premier temps il nous a fallu configurer les GPIOs pour cela on pouvait reprendre la librairie libgpio. Puis coder des fonctions basiques pour faire fonctionner l'écran de sorte qu'il affiche « Hello World ! », puis dans un second temps, réaliser du monitoring en temps réél via le fichier /proc/loadavg. Pour ce faire nous avions une carte Raspberry Pi relié directement à l'écran LCD.
Au niveau des fonctions basiques il nous a été demandé de coder une fonction les fonctions suivantes :

- lcd_strobe, pour générer un front descendant
- lcd_write_4bit_value, qui envoie une valeur de 4 bits sur le bus de l'afficheur LCD.
- lcd_write_value, qui envoie une valeur de 8 bits en reprenant la function précédente.
- lcd_send_4bit_cmd, qui envoie une commande sur 4 bits à l'afficheur LCD.
- lcd_send_cmd, qui envoie une commande sur 8 bits à l'afficheur LCD.
- lcd_send_data, qui envoie une donnée à l'afficheur LCD.
- lcd_init, qui initialize les GPIOs.
- lcd_deinit, qui vide la mémoire de l'afficheur et déinitialise les GPIOs.

2 FONCTIONNEMENT DE PRINCIPE

2.1 CONFIGURATION DES GPIOS

La première chose à faire a été de configurer les GPIOs. Pour cela on a dû se référer à la documentation donné dans le TME. Pour pouvoir réaliser cet affichage, on a mis tous les GPIOs en sortie (output). Ainsi, le signal RS de l'afficheur correspondait au GPIO 18, le signal EN au GPIO 23 et les signaux D0, D1, D2, D3 correspondait respectivement aux GPIOs 4, 17, 22, 27.
Pour qu'une valeur soit prise en compte par l'afficheur LCD, il a fallu coder la fonction « lcd_strobe » qui était chargée de créer un front descendant du signal EN. C'est-à-dire garder le signal EN à 1 pendant une demi période (50 µsec) puis le remettre à 0 pendant une demi-période.
La demi période était caractérisé par une attente faite par la fonction usleep.
Pour cela, il n'y a pas vraiment eu de difficulté majeure, on a utilisé deux fois la function "gpio_update" de la libgpio entre un udelay de 50 µsec.
Par soucis de propreté, dans le code tous les GPIOs ont été définis par #define.

2.2 FONCTIONS DE BASES

Le principal problème est que l'afficheur LCD est en mode 4 bits. Or les données et les commandes sont transmises sur 8 bits, il nous a donc fallu coder une fonction « lcd_write_4bit_value ». Cette fonction prend un argument un mot et via une boucle for de 4 itérations, transmet le mot en

faisant un décalage de 1 à chaque fois. La variable globale contient l'ensemble des bits data à savoir D0, D1, D2, D3. A la fin de cette function, la fonction permettant de générer un front descendant est appelé pour que justement ces bits de données soit prises en compte.

La fonction "lcd_write_value" permet d'envoyer des mots de 8 bits, pour cela elle appelle deux fois la fonction précédente, en faisant un décalage de 4 bits lors du premier appelle.

Maintenant que nous pouvons envoyer des mots de 8 bits à notre afficheur LCD, la prochaine difficiculté a été de pouvoir distinguer les commandes et les données qui étaient envoyé à l'afficheur, c'est pour cela que les fonctions. On distingue donc deux fonctions pour l'envoie de commande "lcd_send_4bit_cmd" et "lcd_send_cmd". Ces deux fonctions sont très similaires dans le sens où elles appellent toutes les deux la fonction gpio_update pour signaler qu'on envoie une commande avec le signal RS (c'est le signal qui gére la distinction entre les commandes et les données). Enfin selon le cas où l'ont est dans un envoi de 4 bits ou 8 bits, ces fonctions appellent respectivement "lcd_write_4bit_value" et "lcd_write_value" avec en argument la commande en question.

La aussi pour rendre le code plus souple, on a prit la decision de séparer au maximum les commandes, c'est pour cela que dans le code on a de nombreux #define CMD_X, pour rendre le débuggage plus simple.

L'envoi de donnée quant elle se fait via la fonction "lcd_send_data", qui prend un argument un mot de 8 bits et qui appelle la fonction gpio_update en passant en argument ce mot de 8 bits. Aprés cela, la donnée est envoyée grâce à la fonction "lcd_write_value".

Pour terminer il nous a fallu coder la fonction d'initialisation "lcd_init", celle qui nous a pose le plus de soucis.

3 IMPLEMENTATION ET RESULTATS

L'implémentation des premières fonctions s'est faite assez rapidement. Le plus long fut d'implémenté la fonction "lcd_init". Notre afficheur LCD n'affichait pas le fameux "hello World !" au début se contentant de caractére illisible. C'est pourquoi on a revu notre code afin de bien détailler l'envoi de commande (d'où les nombreux define dans le code). Une fois cela fait on s'est apercu que le code était plus simple à lire et nous avons pu régler notre probléme de cette manière.

Pour ce qui est du monitoring en temps réél, nous avons commence et fait une première ébauche. C'est à dire que l'on ouvre le fichier /proc/loadavg et avec un while on parcourt le fichier tant que l'on est pas à la fin. Il nous reste à replacer le curseur en début de fichier pour le relire et renvoyer le tout sur l'afficheur pour avoir une actualisation et ainsi suivre en temps réél la mise à jour du fichier.

4 CONCLUSION

Ce premier TME nous a permis de nous familiariser avec la configuration des GPIOs et de faire une application basique pour un afficheur LCD en mode utilisateur. C'est un bon entrainement avant de passer à la création de pilote de périphérique en mode noyau que nous verrons lors des prochains TME.