Jacob Fontaine
Philippe Spino

Spip2401
Fonj1903

Rapport App2

Présente à :
JEAN LAVOIE

January 30, 2018

Contents

1 Introduction.

Ce présent rapport a pour but de présenter les éléments de conception du système d'acquisition de données(SAD) développé par notre équipe. Le SAD a été conçu à l'aide d'un processeur embarqué LPC1768, soit la coquille de base du SAD. Le contenue de ce rapport explique les ressources mémoire et du processeur utilisé, les algorithmes de gestion des acquisitions, la synchronisation des tâches utilisées, une analyse des problèmes de conception et de déverminage lors du montage du prototype, une description des méthodes concernant la validation des fréquences d'échantillonnages des données et une analyse du prototype le plus optimal. De plus, une discussion sur les temps d'exécution des codes base pour un inclinomètre, à la demande de la compagnie métrologie numérique.

2 Utilisation des ressources matérielles.

Pour ce présent prototype, deux LCP1768 sont utilisés. Un des microcontrôleurs est programmé pour émettre de façon aléatoire des données soit numériquement ou bien analogiquement. Il est nommé DataSpammer. L'autre microcontrôleur utilise l'interface principale du SAD. Le DataSpammer, est réglé avec une fréquence d'opération mobile, c'est-à-dire qu'elle peut être changée selon l'utilisateur. Pour le montage, nous avons 2 mbed qui sont branchés ensemble et le DataSpammer envoie ses données aléatoires via 2 port GPIO et la fréquence d'opération passe de 10ms a 500ms. Sur le microcontrôleur qui a l'interface primaire, les leds 1 et 2 servent a indiquer un changement de voltage sur les pattes analogiques p19 et p20. Les leds 3 et 4 servent a indiquer un événement numérique. Les pattes p21 et p22 sont utilisées. Nous utilisons au maximum 976 bytes de Stack.

L'interface principale utilise 4 tâches pour faire la gestion des acquisitions, une tâche pour chaque type d'entrée, une tâche pour l'horloge temps-réel(RTC) et une tâche pour la gestion de la propagation. Pour propager les données acquises, une structure, nommée Event, est utilisée. Cette structure contient un byte non signé, un entier de 32bit et une struct temporel. Pour propager les Event, une file globale, qui contient des Event, de type First In First Out(FIFO) est utilisé. Pour la gestion de l'accès mémoire de cette file, un mutex global est utilisé. La file est utilisée par les deux tâches d'acqusitions numérique et analogique. Par la suite, pour l'acquisition du temps lors d'une capture de données, une file globale FIFO est utilisée pour une demande du temps. Un mutex est aussi utilisé pour la gestion de l'accès à cette file.

3 Algorithmes d'acquisitions de données.

L'algorithme développé pour l'acquisition des données est divisé en 4 tâches différentes. Numérique, Analogique et le RTC. La tâche pour l'acquisition numérique étant en priorisation haute, l'analogique en priorisation moyenne et le reste, soit la tâche pour le RTC, en priorisation basse.

3.1 Numérique

La tâche pour l'acquisition des données numériques vérifie s'il y a un changement de bit. Le temps prit pour qu'un bit puisse changer et être considéré un changement est de l'ordre des 50 ms. La fréquence d'opération de cet algorithme est 100ms. Pour ce faire, une tâche::wait est utilisé. Cela permet d'être assuré que la tâche va être à une fréquence de 100ms. Une boucle itérative sur 20 tours regarde les valeurs des pattes d'entrées digitales à toutes les 5ms, lorsque les valeurs rester inchangé après une itération, un conteur est

incrémenté. Si dès que le conteur atteint 10(10 * 5ms = 50ms),l'algorithme a fait une demande au mutex pour assurer l'accès a la file et évité les interlockage. Ensuite, une réquisition du temps est faite à la tâche du RTC. Le mutex du temps est acquis et un Event est créé avec le changement et le temps, et on l'ajoute à la file des Event. Une fois, l'Event dans la file, les mutex sont libérés.

3.2 Analogique

La tâche pour l'acquisition des données analogique vérifie s'il y a un changement de la moyenne des 5 dernières lectures. La définition d'un changement est lorsque la nouvelle moyenne est $\pm$ 12.5% de l'ancienne moyenne. La lecture des pattes analogique opère à 50ms et envoi la moyenne recalculée au 250ms. Une boucle itérative sur 5 tours additionne les valeurs lues et attend 50ms. Une fois 5 échantillons additionnés ensemble, soit 250ms plus tard, la comparaison est faite. Si une nouvelle valeur est acceptée, une acquisition du mutex de la file des Event est appelée et le mutex du temps est acquis. Un Event est créé et inséré dans la file. Ensuite les mutex sont relâchés.

3.3 L'horloge temps réel (RTC)

La tâche pour l'acquisition de l'étampe de temps s'exécute à une fréquence d'opération de 50ms. Elle demande l'accès à la file du temps en tentant d'acquérir le mutex de la file du temps. Ensuite, elle crée une variable time_t et l'insère dans la file de temps. L'algorithme relâche ensuite le mutex de la file de temps.

4 Synchronisation des tâches CPU.

Pour la synchronisation des ressources et des tâches, des mutexs sont utilisés. Les tâches ont un ordre de priorité allant de haute à basse. Pour s'assurer que les tâches seront bien synchrones au départ, il faut fixer le processus créé par la fonction main à une haute priorité, partir les tâches avec leur priorité respective et ensuite remettre le processus de la fonction main à une priorité normale. Cela permet de bien s'assurer que les opérations critiquent, tel que le démarrage des tâches soit bien exécuté dans la fonction main. Ensuite, dans les tâches, l'utilisation de la fonction Thread::wait(ms). Cela nous permet de fixer la fréquence d'opération aux valeurs que l'on veut obtenir. elle permet aussi de réguler l'accès aux mutex

5 Déverminage

Il est certain que l'utilisation de l'oscilloscope était primordiale pour déverminer cet APP. Une difficulté se présentait donc devant le fait que nous devions toujours travailler dans un local contenant cet outil, car nous voulions valider nos données en entrées et en sortie. De plus, nous avons remarqué qu'il était difficile de savoir ce que contenaient les files. Par exemple, nous avions un problème avec un timestamp, car il changeait de valeur entre 2 thread après être passé par une file. Nous ne pouvions voir que l'adresse du timestamp, et non sa valeur. Pour faciliter les choses, il serait intéressant d'utiliser plus le système d'expression de Keil, qui permet d'analyser le résultat d'une série de variables.

6 Validation des fréquences d'échantillonnage.

Les fréquences d'échantillonnages étant de 100ms et de 250ms pour le numérique et l'analogique respectivement, les informations trop vites, ce qui pourrait par exemple créer des évènements invalides alors qu'ils sont bien valides. En plus de faire le bon nombre de wait, comme expliquer dans notre algorithme plus haut, nous pouvions vérifier nos échantillons en allumant les leds sur réception d'un échantillon valide. Par contre, cela étant peu précis, nous avons également développé un autre programme qui permettait d'envoyer les données à un rythme précis. Cela, coupler à notre invite de commande, permettait de valider que nous avions bien reçu, par exemple, les 3 échantillons valides envoyés. Ainsi, nous avons pu confirmer le fonctionnement de nos fréquences.

7 Performances des algorithmes métrologie.

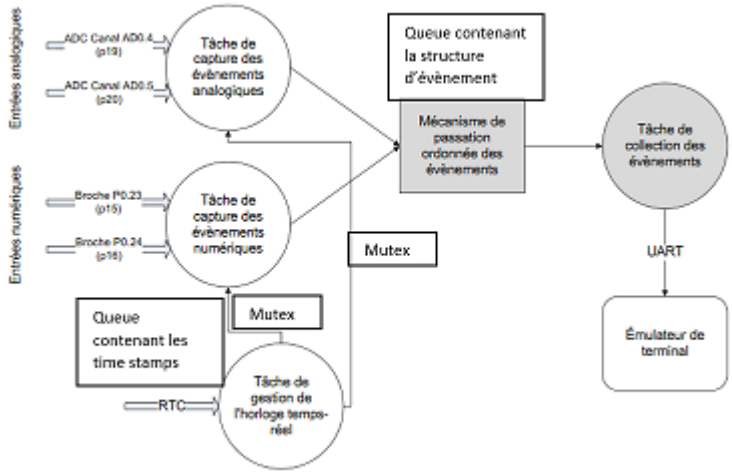
En ordre du plus rapide au moins rapide, nous avons le calcul 4 ensuite le 1, le 3, et finalement le 2. L'algorithme est le plus rapide, car il utilise acos_table, qui fonctionne comme une table de correspondance. Les données du calcul acos sont déjà faites, donc on peut simplement aller chercher les valeurs au lieu de les calculer chaque fois. Ensuite, l'algorithme 1 est plus rapide que le 2 et 3, car en déclarant sa variable module en float, il gagne en vitesse face a à déclaration de la constante 1024.0, qui est vue comme un double. Si la constante avait été déclarée comme un float (1024.0f), la vitesse de l'algorithme 1 et 2 serait identique. Finalement, le 3 est plus rapide que le 2, car même s'il fait un calcul mathématique de plus, il fonctionne en float au lieu d'en double. Mais en faisant justement 2 calculs mathématiques, l'algorithme 3 est plus lent que le 1, qui ne fait lui q'un seul calcul

8 Conclusion.

Au cours de cette expérience magnifique de développement, nous avons pu peaufiner nos connaissances en gestion de Thread, par exemple grâce au Mutex et principes de priorité. Également, nous avons dû faire l'analyse des performances de différents algorithmes, ce qui nous a fait comprend les subtilités des vitesses d'exécutions.

9 Schémas.

9.1 Schéma bloc.



9.2 Diagramme de séquence