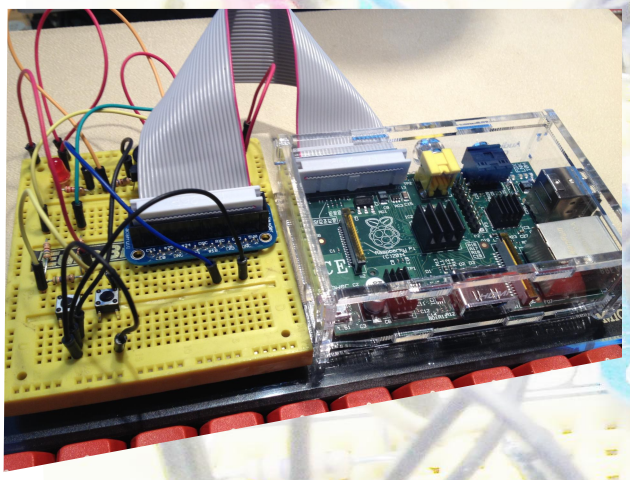
maxa.c) la fréquence monte en gros à 120 kHz. La fréquence réelle varie car le Raspberry Pi fait d'autres choses qui suspendent momentanément la boucle - maintenir l'horloge à jour, gérer l'activité réseau, et autres processus système et utilisateur.

Le Programme

Comme promis précédemment, voici le programme http://ryniker.ods.org/raspberrypi/MagPi/gpio_control.c qui exécute les opérations nécessitant le privilège root pour qu'un utilisateur normal puisse utiliser les broches du GPIO. Les commentaires au début du programme expliquent comment le compiler



et l'installer. Une fois installé (par root), du fait qu'il dispose du droit "setuid", le programme s'exécute avec le userid de root, il a donc le privilège requis pour lire ou écrire une broche du GPIO, et configurer les permissions requises pour les fichiers utilisés pour contrôler cette broche.

Les programmes qui s'exécutent avec le privilège root devraient n'être écrits que par des programmeurs vraiment paranoïaques. La plupart du code de gpio_control.c ne fait que vérifier si les arguments sont vraisemblables , et essaye d'être informatif si quelque chose d'inattendu arrive.

Pour utiliser gpio_control pour contrôler la broche 23 de sorte que toutes les manipulations de broches mentionnées plus tôt ne requièrent pas le privilège root, exécutez simplement :

```
gpio_control 23 export
```

gpio_control peut être facilement configuré, avant compilation, pour permettre l'accès du GPIO à tous les utilisateurs, ou aux seuls utilisateurs figurant dans un groupe donné. Chacune des 54 broches du GPIO peut être configurée individuellement pour autoriser ou interdire l'accès.

Le Raspberry Pi utilise la broche 16 du GPIO pour contrôler la DEL verte "Status OK". Si quelqu'un tente d'accéder à la broche 16 du GPIO, l'opération échoue car le noyau utilise cette ressource :

```
ryniker@raspberrypi:~$ gpio_control 16
 export
export failed: Device or ressource busy
```

Les autres programmes du noyau peuvent gérer les broches du GPIO, ce qui les rend indisponibles pour les autres utilisateurs. C'est bien. Un petit souci pourrait venir d'un utilisateur qui allumerait/éteindrait la DEL d'état, mais que se passe-t-il pour le pilote I2C du noyau ? Il peut facilement subir des plantages aléatoires si les broches qu'il utilise sont modifiées de façon incompréhensible pour lui.

Le noyau mémorise l'état des broches du GPIO. Par exemple, supposons qu'une broche est utilisée, configurée par l'utilisateur comme sortie, puis libérée. Les fichiers de l'espace utilisateur disparaissent, mais la broche reste une broche de sortie et conserve la dernière valeur reçue. Si cette broche est utilisée à nouveau, les fichiers de l'espace utilisateur sont recréés pour refléter l'état sauvegardé.

La commande echo est pratique pour être utilisée dans des scripts shell, éventuellement en ligne de commande, mais Python est plus adapté pour écrire de vrais programmes. La douzaine de lignes de gpio23-max.py vous en fournit un exemple simple.

Maintenant que j'ai exposé les éléments de base du contrôle du GPIO, cette facilité peut être utilisée pour remplacer la "boucle infinie", dans laquelle un programme lit sans cesse la valeur d'un signal d'entrée et exécute une opération lorsqu'il change, par un programme bien plus efficace qui ne s'exécute que quand la valeur du signal change. Avec une seule entrée et rien d'autre à faire jusqu'au changement de sa valeur, une boucle pourrait faire l'affaire. Quoiqu'il en soit, cette boucle consommerait 100% des ressources CPU, et ainsi concurrencerait agressivement tout autre programme qui aurait besoin d'une ressource du Raspberry Pi.

On peut introduire un délai dans la boucle, disons une commande "sleep 0.5" pour