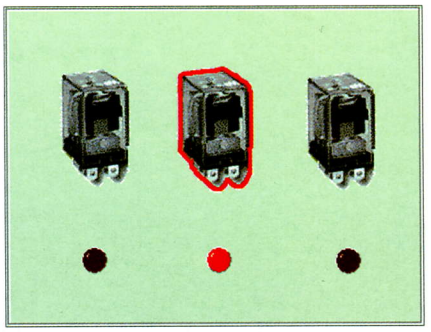

Figure 5. Affichage sur LCD tactile

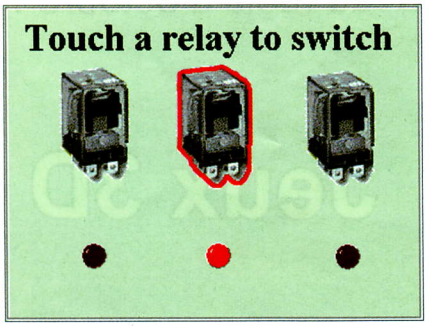

Figure 6. Ajout de texte en police fixe

Le principe est ici exactement le même que pour la fonction I2Cwrite avec l'appel à la fonction `ioctl` avant la fonction `read`.

Pilotage par touchscreen

Nous rappelons que le but de l'application est de pouvoir piloter des relais d'une carte d'extension I2C à l'aide d'un écran LCD tactile. Le mécanisme est celui d'un va-et-vient, donc à chaque nouvel appui sur le LCD, le relais concerné sera tantôt ON, tantôt OFF.

Dans la boucle `while(1)` du main, il nous faut donc périodiquement inspecter l'écran tactile, et modifier l'affichage si besoin tout en transmettant l'ordre d'activation de relais en I2C (Listing 10).

La première chose qui peut surprendre, ce sont les calculs liés aux coordonnées X et Y. En effet, bien souvent les écrans tactiles ont besoin d'être calibrés, pire encore dans notre cas, les coordonnées X et Y ont besoin d'être inversées (cas d'une rotation de l'écran du mode portrait à paysage). Nous n'avons pas voulu alourdir le code avec une étape préliminaire de calibrage, aussi nous effectuons le calcul directement avec une mise à l'échelle par rapport à notre écran 320x240. On définit de la même façon des zones graphiques pour les différents objets (`Widgets Relais`) et on appelle la fonction `ToggleRelay` selon l'appuie qui a été détecté. Bien entendu, il serait préférable de créer un widget qui examine lui-même l'appui détecté pour déterminer l'action à suivre. Nous serions alors davantage dans le cas d'une programmation orientée objet où le widget possède les données et les fonctions qui définissent les actions et décisions du widget.

Ceci peut être réalisé en C pour tous ceux qui se sentent à l'aise avec les pointeurs de fonctions (inclure alors le pointeur dans la structure widget et initialiser le pointeur dans la fonction `main`).

Enfin dernière étape, qui n'est pas évidente en soit mais indispensable : vider le tampon du driver de touchscreen ! En effet la plupart des drivers stockent la liste des appuis qui ont été détectés, et ceci dans le but de pouvoir tracer l'historique du stylet sur le LCD (dans le cas d'un logiciel de dessin par exemple).

Dans notre cas il faut flusher cet historique pour être prêt à détecter le prochain appui.

Entrées télécommandes

Pour terminer il faut être capable de lire les entrées télécommandes en provenance du périphérique I2C à l'adresse 0x23. Juste avant de flusher le buffer du touchscreen on peut en profiter pour lire les entrées télécommandes (Listing 11). La fonction `SwitchLedsON` est similaire à celle des fonctions liées au relais (Listing 12). Encore une fois nous n'avons ici besoin que de modifier l'identifiant de l'image et non pas de recharger l'image complètement en mémoire.

Application finale

Finalement, l'application fonctionne simplement de la façon suivante :

- lorsqu'on appuie sur une image d'un relais, celui-ci se colle ou se décolle alternativement
- lorsqu'une entrée télécommande survient, la LED correspondante s'allume

Pour compléter l'application, on pourrait ajouter par exemple du son, en envoyant un fichier wav à l'interface */dev/sound/dsp* par exemple.

Une amélioration simple avec la librairie Nano-X serait l'ajout d'un texte en haut de l'écran (Figure 6) comme ceci. Il suffit pour cela de créer une variable `font` et de l'associer aux variables `MainWindow` et `gc` (Listing 13). La police *Times35* est inclue de façon native dans le nano-X que nous utilisons, mais mieux encore nous pourrions utiliser une police *TrueType* en chargeant par exemple la police *times.ttf*.

N'oubliez pas pour cela de linker la librairie *libttf.a* à votre projet. Enfin, cerise sur la gâteau, nous pourrions aussi faire de l'affichage distant, c'est-à-dire de lancer le serveur Nano-X sur une station ou un système embarqué distant. En effet, Nano-X est un serveur avec lequel on communique par socket. Il peut donc être utilisé sur la même plate-forme comme sur 2 plate-formes distantes. Vous pourrez donc ainsi afficher le contenu graphique du `Frame Buffer` sur un PC distant pour lequel nano-X aura été recompilé en natif.

Conclusion

Nous espérons que cet article vous aura convaincu de la simplicité qu'il est très simple de réaliser une application graphique embarquée. Les outils logiciels qui existent au jour d'hui sur de telles cibles permettent de réaliser rapidement des applications, voir même dans certains cas, de porter des applications pour PC grâce à l'utilisation d'une librairie SDL adaptée... mais nous verrons tout cela dans un prochain article !

Sur le réseau

- Site de uClinux, version embarquée de Linux pour microcontrôleurs : *www.uclinux.org*
- Site officiel du serveur nano-X : *www.microwindows.org*
- Les cartes de développement SIMTEC : *http://www.simtec.co.uk*
- Forum français lié au développement d'applications embarquées Linux : *http://www.pragmatux.net*

Concernant l'auteur

Ingénieur en électronique et informatique, Xavier Montagne a travaillé en France et aux Etats-Unis sur des projets de défense et de super-calculateurs. Spécialiste en systèmes temps réels et en Linux embarqué, il est aujourd'hui directeur technique de la société Pragmatec. Auteur du noyau temps réel PICos18 et intervenant auprès d'écoles d'ingénieurs et de formations universitaires, il croit avant tout en l'efficacité des solutions GPL et Open Sources.