POUSSET Pierre-Alexandre
TSMA2
2008 - 2009

Construction d'un Direct To Disk

SOMMAIRE

Introduction

L'idée de ce mémoire ne m'est pas apparue d'un bloc. Le sujet a évolué au fur et à mesure de l'avancement des travaux. L'idée première était de traiter de la MAO sous Linux, ce qui est vite apparu assez fastidieux, et trop orienté vers l'informatique. Puis, ayant eu vent de projets de surfaces de contrôle et d'instruments midi, m'est venu l'idée de pouvoir piloter un programme linux à partir de contrôles hardware. Mes recherches m'ont mené à un logiciel d'enregistrement qui n'a pas besoin d'interface graphique, donc pas d'écran, pas de souris... et pourquoi pas, pas de clavier ? A partir de là il était clair que se passer de station informatique complète était la vocation de mon projet, qui devait se résumer à une carte mère, un support de stockage une interface audionumérique, quelques contrôles sous forme de boutons poussoirs, et de quoi afficher quelques informations essentielles, le tout permettant de réaliser ce que je croyais être « une sorte » de Direct To Disk. Puis je me suis posé la question de savoir quelle était la différence entre mon projet, et n'importe quelle machine audionumérique. Réponse ? Aucune. Une machine dédiée dispose uniquement des ressources dont elle a besoin, un PC est fait pour un plus grand nombre d'applications, mais le principe reste le même. Finalement je me suis retrouvé avec projet qui m'a mené vers son sens premier, et non un sujet qui m'a mené vers un projet : je suis parti de la MAO sous Linux pour atterrir sur une démystification de la technologie numérique.

Décomposons ce travail en trois axes :

- Une première partie traitera de la conception et de la fabrication du Direct To Disk, rédigée en quasi temps réel, afin de clarifier le processus de conception et de mettre en avant que la complexité de ce projet n'est qu'une apparence.
- Dans une seconde partie nous établirons un parallèle avec les machines dédiées, au niveau du matériel, et de la conception.
- Une troisième et dernière partie contiendra des considérations plus générales sur la technologie numérique en guise de conclusion.

I/ La construction du Direct To Disk

Ce projet peut paraître assez ambitieux. Il n'en est rien.

Pour mettre en oeuvre ce projet nous allons créer un système Linux embarqué. Qu'est-ce qu'un système embarqué ? C'est un système allégé au point de ne contenir que les outils indispensables au fonctionnement d'une machine à laquelle il sera dédié. Par exemple, dans notre cas, nous n'avons pas besoin d'interface graphique, qui pèse par ailleurs un poids considérable. Celà ne veut pas dire non plus que des outils ne doivent pas être ajoutés. Toujours dans notre cas, nous utiliserons un afficheur LCD, notre système doit donc être capable de le gérer.

I.A/ Etablissons tout d'abord un cahier des charges de notre système :

Au niveau matériel :

- En premier lieu, il nous faut un matériel pour accueillir notre système : un PC. Comme nous construisons un système embarqué, et donc sensé être extrêmement léger, nous n'avons pas besoin d'une configuration ultramoderne ; du matériel de récupération devrait faire l'affaire.
- Une interface réseau : notre système doit pouvoir facilement transférer nos enregistrements vers une autre station de travail
- Un port parallèle : c'est grâce à lui que nous allons manipuler notre interface utilisateur.
- Un port PCI, USB, Firewire ou autre pour accueillir une interface audio.
- Le système doit disposer d'un système d'affichage pour communiquer avec l'utilisateur : un simple afficheur LCD de caractères suffira.
- Nous avons également besoin d'un moyen de contrôle de la part de l'utilisateur. Nous allons donc devoir construire un clavier matriciel, pour des raisons que nous verrons plus tard. Les commandes seront : Play, Record, Pause, Forwind, Rewind, plus : 4 touches directionnelles pour naviguer dans les menus et deux touches de fonction pour valider ou choisir des options, soit 11 contrôles au total.
- Nous souhaitons pouvoir enregistrer 2 pistes, et bien évidemment les écouter. Nous avons donc besoin d'une interface audio possédant deux entrées et deux sorties (notre système est très facilement modulable et pourra évoluer par la suite en multipiste).
- Un affichage des niveaux, soit via l'afficheur LCD, soit via des LEDs.

Au niveau logiciel :

- Un noyau Linux compilé temps réel
- Un sequenceur pouvant être piloté par une application tierce
- Un gestionnaire pour notre affichage LCD et notre clavier
- Notre application qui gèrera l'affichage et le clavier pour l'interfacer avec le sequenceur et nous offrira divers paramètres comme le choix de la fréquence d'échantillonnage, nous permettra d'exporter nos fichiers, etc...
- Un pilote pour l'interface audio
- Un utilitaire de gravure, si nous intégrons un graveur CD
- Un plug-in de Limiter

Et de manière plus optionnelle :

- Un deuxième disque dur pour contenir nos données plutôt qu'un seul disque système
- Un graveur CD pour peut-être mettre notre projet sur support de manière rapide.

I.B/ Constructions de l'interface utilisateur

Evaluons le coût des opérations :
On peut facilement récupérer un vieux PC, ainsi que du bois ou de la tôle pour construire un boîtier : 0€
Une façade de rack 3U : 15€
12 bouton poussoirs carrés "tableau" : environ 40€
Quelques diodes et résistances : 2€
Une plaque d'essais à pastilles de petite dimension : 10€
Un afficheur LCD 2x16 rétroéclairé : 15€
Connecteur SubD-25 : 2€
Soit moins de 100€, facilement réductible en utilisant des contrôles moins agréables. Il faut ajouter à celà le prix d'une interface audio, prix qui déterminera à lui seul la qualité de notre Direct To Disk. Nous choisirons une interface bas de gamme, chez Behringer pour ne pas le citer.

I.B.1/ L'affichage

Notre interface se compose d'un afficheur LCD et d'un clavier 12 touches. Commençons par le plus amusant : l'affichage. Une rapide recherche sur le net nous apprend qu'il est facile de connecter un afficheur LCD au port parallèle, et que le microcontrolleur est intégré à l'afficheur : nous n'avons besoin que de fil pour relier l'afficheur au PC ! Notre afficheur est un DataVision 16422-NRB. Il faut ensuite réfléchir à la manière de le piloter, seulement... impossible de

trouver la référence du microcontrolleur dans les datasheets ! Nos recherches nous disent que le microcontrolleur HD44780 s'est plus ou moins démocratisé, et le nombre de broches correspond.

Nous avons aussi appris au cours de ces même recherches qu'un utilitaire de gestion d'afficheur LCD s'appelle "lcdproc" (en fait le gestionnaire est compilé sous le nom LCDd, lcdproc est un client pour LCDd qui affiche des informations sur le CPU, le traffic réseau, etc...). On se farcit la doc qui nous apprend diverses choses, principalement de bonnes nouvelles, mais nous reviendrons dessus plus tard.

LCDd, pardon, lcdproc est très simple à utiliser puisque c'est un démon (une application qui tourne en tâche de fond mais reste prête à communiquer avec l'utilisateur) qui utilise un port de l'hôte local (127.0.0.1 port 13666 par défaut). Ainsi pour lui faire afficher ce que l'on souhaite il nous suffit d'écrire sur ce port, ce qui est simplissime quel que soit le langage de programmation utilisé. L'outil telnet permettra donc de faire un rapide test.

La configuration consiste à éditer /etc/LCDd.conf, qui est par ailleurs très bien commenté. La configuration de lcdproc passe princinpalement par le choix du driver (du microcontrolleur). Parmis les choix possible, ncurses. Pour les néophytes, ncurses est une librairie d'affichage de texte en mode console. Autrement dit, ce driver nous permet de simuler un afficheur LCD pour programmer en toute tranquilité.
Testons donc lcdproc avec telnet (j'ai symbolisé le texte rentré avec « > » et le retour de LCDd par « < ») :

```
# telnet localhost 13666
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
> hello
< connect LCDproc 0.5.2 protocol 0.3 lcd wid 16 hgt 2 cellwid
5 cellhgt 8
```

Arrivé à ce point LCDproc nous indique qu'un client est connecté, ce qui est bon signe

```
> screen_add monecran
< success
< listen monecran
> widget_add monecran montitre title
< success
> widget_set monecran montitre "Hello World !!!"
```

Un grand classique en programmation

```
< success
```

Effectivement la sortie virtuelle nous affiche "Hello World !!!" en défilant (plus de 16 caractères étant donné que title rempli le texte encadré de deux blocs et deux espaces : "## Hello World ##")

```
> widget_add monecran montexte string
< success
> widget_set monecran montexte 1 2 "J'afficheSurLCD!"
< success
```

Une fois de plus, c'est effectivement un success. Un jeu d'enfant !

Il est temps de tester notre véritable afficheur. Il nécessite une alimentation en 5V, disponible sur le bloc d'alimentation d'un PC, mais ne disposant la plupart du temps que d'un portable, il va falloir la trouver ailleurs, autrement dit dans un port USB. Ceci a le mérite de permettre de déconnecter notre interface plus facilement et de l'adapter n'importe ou.

Paramétrons donc lcdproc pour qu'il utilise le driver HD44780 avec un afficheur 2X16, raccordons notre interface, lançons lcdproc (le client, qui affiche des infos système), et :

Le mystère est levé : mon pc tourne sous linux Ubuntu

Nous avons de quoi être heureux. Nous savons maintenant que notre afficheur contient bien un microcontroller HD44780, que notre brochage était le bon, et que le développement de la partie affichage risque de se révéler simplissime. Nous avons également appris à utiliser lcdproc grâce à la lecture d'une partie de la documentation. Le seul point négatif est que selon l'éclairage et l'angle, l'écran réflectif n'est pas toujours net, nous allons donc investir dans un modèle rétroéclairé.

Et pour les plus curieux voici quelques infos techniques supplémentaires :

Le brochage :

LCD	Pin	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	Control	Vss	Vcc	Vo	RS*	RW*	E/S*	Data 0	D1	D2	D3	D4	D5	D6	Data 7
SubD	Control	Gnd	/	Gnd	RS	Gnd	E/S	Data 0	D1	D2	D3	D4	D5	D6	Data 7
	Pin	18	/	19	16	20	1	2	3	4	5	6	7	8	9

*RS = Register Select
*RW = Read/Write
*E/S = Enable/Strobe

L'insertion d'un potentiomètre entre les broches 1 et 3 permet un contrôle du contrast. Nous l'avons reliée directement à la masse pour un contraste maximum. Le modèle rétroéclairé contient deux broches 15 et 16 supplémentaires qui correspondent respectivement à la cathode et à l'anode du rétroéclairage, il suffira de relier 15 à 1 et 16 à 2. Il sera aussi possible de le connecter à la broche 17 du SUBD-25 pour contrôler l'éclairage de manière logicielle.
La connexion se fait ici en mode 8 bits, on peut également contrôler l'afficheur sur 4 bits (8 bits par deux paquets de 4) en ne connectant que D4, D5, D6 et D7. Le codage sur 8 bits permet un affichage plus rapide, et nous n'avons pas besoin de libérer 4 bits de data.

Les commandes envoyées à LCDd :
hello : permet d'envoyer un premier paquet à LCDd, en clair lui montrer qu'on est là

screen_add monecran : initialise un nouvel « écran », une nouvelle page d'affichage et lui attribue le nom monecran.

widget_add monecran montitre title : ajoute un objet titre sur la page monecran et lui attribue le nom montitre. L'objet titre se caractérise par un encadrement sombre et est toujours affiché sur la première ligne.

widget_set monecran montitre « Hello World !!! » : affecte la valeur « Hello World !!! » à l'objet montitre, et donc l'affiche

widget_add monecran montexte string : ajoute un objet texte (string signifie une chaîne de caractère) sur la page monecran et lui attribue le nom « montexte »

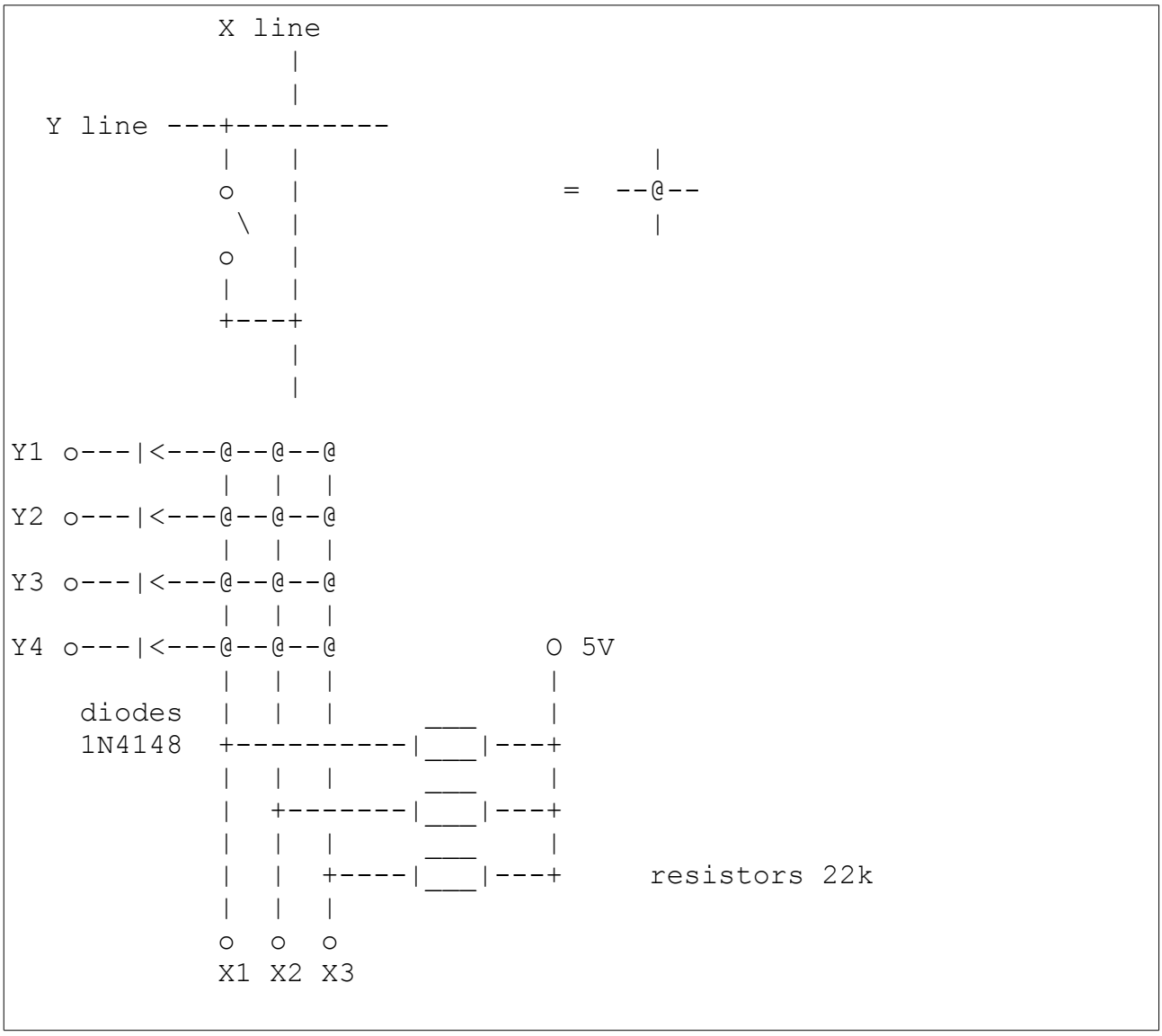
widget_set monecran montexte 1 2 « J'afficheSurLCD! » : affecte la valeur « J'afficheSurLCD! » à l'objet montexte. Il est nécessaire de préciser à

partir de quel emplacement on souhaite afficher le texte, ici colonne 1 ligne 2. Pour un affichage de texte sur 2 lignes il faut donc 2 objets string.
Une erreur dans une commande nous renvoie grossièrement « Huh ???»

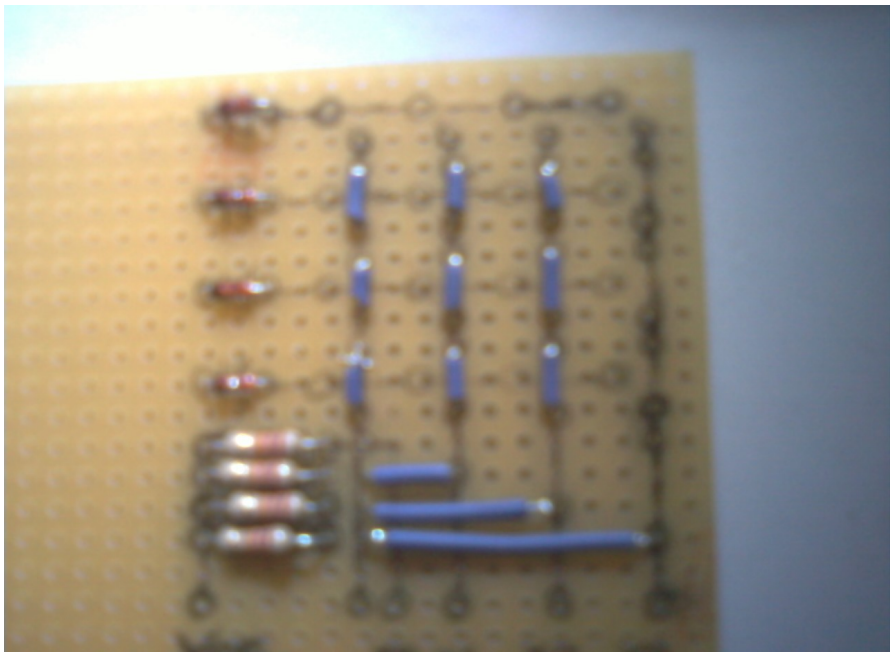
I.B.2/ Les contrôles

Passons maintenant à la partie clavier. La documentation de Lcdproc nous a apporté une bonne nouvelle : il est capable de gérer un clavier matriciel possédant jusqu'à 20 touches. Un clavier matriciel est une grille avec une touche à chaque intersection, l'appui sur une touche laisse passer le courant entre une ligne et une colonne : pour 16 touches on code l'information sur 8 points.

Voici ce que nous donne la doc :



Et voici ce que nous donnent nos petits doigts de fée :



J'ai choisi de faire une version 16 touches, permettant d'ajouter de nouveaux contrôles par la suite.
Etant piloté par lcdproc, le clavier se connecte également à la fiche Sub-D 25 :

Clavier	Pin	Y1	Y2	Y3	Y4	X1	X2	X3	X4
Sub-D 25	Pin	2	3	4	5	10	11	12	13
	Control	D0	D1	D2	D3	nACK	BUSY	PAPEREND	SELIN

Le clavier matriciel 16 touches renvoie par défaut :

	X1	X2	X3	X4
Y1	1	2	3	A
Y2	4	5	6	B
Y3	7	8	9	C
Y4	*	0	#	D

La modification du fichier de configuration de lcdproc nous permet d'avoir des messages plus clairs. Enfin, pour utiliser les contrôles il faut les déclarer à lcdproc (l'hôte). L'appui sur une touche renverra alors le message « key » suivi du code correspondant.

Il n'y a pas grand chose à dire de plus sur le clavier. L'ensemble de l'interface

étant gérée par lcdproc, elle est reliée à un PC uniquement par une prise parallèle, ce qui va permettre la programmation plus simple, en le connectant sur n'importe quel PC. Pour le plaisir des yeux, jetons tout de même un coup d'oeil aux jolis poussoirs :

L'interface utilisateur est donc terminée, si ce n'est la finition et la fixation définitivement des éléments.

I.C/ La mise en oeuvre de la partie logicielle

La prochaîne étape consiste à mettre un cerveau dans notre boîte, qui serait sans cela totalement insipide.

I.C.1/ Un point moins amusant : la programmation

Pour enregistrer, mon choix s'est porté sur le programme Ecasound : ce logiciel fonctionne sans interface graphique, uniquement en mode console. Le piloter est assez complexe (il faut plusieurs lignes de commandes pour lui dire de créer une nouvelle chaîne audio, avec une entrée logicielle ou matérielle, une sortie logicielle ou matérielle ou un fichier de sortie, gérer les fichiers audio disposés sur une même piste est un véritable enfer, positionner un « fader » est d'une complexité absurde, et ne parlons pas des plug-ins...), cependant cette particularité permet de facilement le piloter à partir d'un autre

programme sans avoir à le modifer. Il propose pour cela un mode démon dont nous avons déjà vu le principe, et surtout l'ECI (Ecasound Control Interface).

L'ECI est une bibliothèque d'instructions, c'est à dire des nouvelles commandes qui viennent se greffer à un langage de programmation, permettant de piloter simplement ecasound à partir de nombreux langages (petite liste exhaustive : C / C++, Perl, Python, Php, Ruby, Emacs). Pour ma part j'ai choisi le Perl, un langage interprété (bien qu'il existe maintenant un compilateur appelé PerlCC), pour sa structure inspirée du C et sa grande souplesse d'utilisation.

Notre programme n'est qu'une interface logicielle entre le logiciel d'enregistrement et nos contrôles et afficheur. Il interprète l'appui sur une touche et envoie l'instruction correspondante à ecasound.

Il me paraît inutile de m'attarder lourdement sur ce point, cependant je vais donner pour les béotiens une brève explication de ce qu'est la programmation, et les grandes lignes du programme.

Programmer consiste tout d'abord à apprendre un langage de programmation, c'est à dire des mots clés correspondant à des instructions, et une syntaxe qui ne doivent contenir aucune erreur pour être correctement compris par le compilateur et/ou l'interpréteur. Un compilateur permet de transformer un langage facilement compréhensible par l'homme en code machine contenu dans un fichier exécutable, le .exe de Windows ; une suite de 0 et de 1 directement comprise et traitée par le processeur. L'interpréteur fait la même chose à ceci près qu'il n'envoie pas le résultat dans un fichier mais directement au processeur, ainsi le programme reste lisible de tous (à moins d'être compressé) et consiste en un langage de haut niveau extrêmement clair, mais est plus lourd à exécuter, ce qui selon le type et les fonctionnalités d'un programme n'est pas forcément un handicap vu la puissance de calcul actuelle (les jeux flash par exemple ne sont pas compilés). A noter qu'un compilateur passe par un langage intermédiaire, l'ASM ou assembleur, qui revient à écrire directement en langage binaire, la seul différence est qu'au lieu d'écrire un octet (autrement dit son symbole correspondant dans la table ASCII) on écrit un mot clé. Chaque instruction d'un langage de haut niveau (loin du langage processeur) correspond à une ou plusieurs instructions assembleur. Le compilateur supprime les commentaires, transforme le langage en assembleur, et passe la main a un compilateur asm. L'asm était un passage obligatoire avant l'apparation des langages de haut niveau, eux-mêmes programmés en asm. Par exemple le compilateur C a d'abord été écrit puis compilé en asm. Une fois le compilateur existant, il a été réécrit en C puis compilé en quelque sorte par lui-même. Fin de cette grande parenthèse.

Un langage est constitués d'instructions et d'une syntaxe, donc. Il existe 3 grands types d'outils : les variables, les tests logiques, et les boucles. Les variables permettent de stocker des valeurs de plusieurs types, les test logiques testent des variables (si a est plus grand que b alors faire ceci), et les boucles portent bien leur nom puisqu'elle permettent d'exécuter en boucle un bloc d'instructions (jusqu'à une instruction permettant de sortir de cette boucle, ou de la fin de cette boucle définie par un nombre de passages). Une fois ces outils assimilés, l'art de programmer consiste à les utiliser de sorte à transformer une logique humaine en une logique limitée par un faible nombre d'outils. Par exemple si l'on prend une suite de nombre variants entre 0 et 100, il est facile sans y penser de les classer par ordre croissant. Rentrer ces nombres dans un programme et faire en sorte que le programme les classe est une tout autre chose. Il faut retrouver la logique qui nous permet de les classer alors qu'on l'a fait sans même y penser. On pourrait par exemple commencer par trouver le nombre le plus petit de la liste par une suite interminable de tests, recommencer avec une suite débarassée du nombre trouvé précédemment, jusqu'à ce que la liste soit vide (en réalité la solution la plus simple s'appelle l'algorithme récursif de tri rapide mais passons...).

Le gros de notre programme est une boucle principale qui teste en permanence l'appui sur une touche. Si une touche est appuyée, le programme quitte la boucle et va à la fonction correspondante puis revient à la boucle. La fonction bouton_pause, par exemple, teste une variable qui contient l'état du Direct To Disk. Si la variable contient « pause », il ne se passe rien, si elle contient « lecture », alors la variable passe à « pause », l'ordre est donné à ecasound de se mettre en pause, lcdproc affiche « Stopped » sur l'écran LCD. Tout ceci après avoir testé si un projet avait bien été créé, l'envoi de l'ordre de se mettre en pause alors qu'il n'y a pas matière à le faire aurait provoqué un bug. Ainsi chaque comportement doit être prévu, et la logique du programme doit être sans faille. Ce qui est extrêmement intéressant, ou pourrait être extrêmement intéressant si la programmation n'impliquait pas la rédaction du code source : si réfléchir à une logique est passionnant, toutes les étapes nécessaires à sa mise en oeuvre, comme créer des variables en vérifiant qu'elles n'existent pas déjà, tout ce qui est protocole pour commander un autre programme ; sont incroyable lourdes.

Pour finir sur cet aspect qu'est la programmation, une petite anecdote dont on peut tirer beaucoup de leçons. A un certain point de l'avancement du programme je me suis retrouvé confronté à un comportement que je n'avais pas prévu. Pour les fonctions avance rapide et retour arrière, on garde enfoncée la touche correspondante jusqu'à avoir atteint le point souhaité. Rien de spécial, si ce n'est qu'au bout d'un certain temps d'appui le programme s'arrêtait de lui-même. En fait lcdproc envoyait en permanence l'état d'appui sur cette touche, et à chaque fois que le programme recevait l'état il envoyait

l'ordre de reculer ou d'avancer de 0,5ms. Hors les états d'appui s'empilaient en mémoire dans ce qu'on appelle justement une pile, et qui a ses limites. Au bout d'un certain temps elle saturait, disons grossièrement qu'à chaque fois qu'elle retirait le message tout en bas de la pile elle en recevait 10 nouveaux. Après deux jours de galères et avoir seulement réussi à multiplier le temps avant saturation par deux au prix d'un ralentissement considérable de la réactivité, et le traitement après relâchement de la touche des messages encore dans la pile (autrement dit lorsqu'on relâchait avance rapide, le Direct To Disk continuait à avancer pendant un certain temps), c'est Nico de la section ISMA, totalement inculte en matière de programmation et informatique un peu poussée, qui a trouvé la solution : « Tu n'as qu'à appuyer qu'une seule fois sur Avance Rapide, et pour l'arrêter tu appuies sur Pause ». Tout simplement. Et bien évidemment, cela fonctionne très bien comme ça.

I.D/ Un point vraiment pas drôle : la construction d'un système embarqué

Voici le cerveau du Direct To Disk :

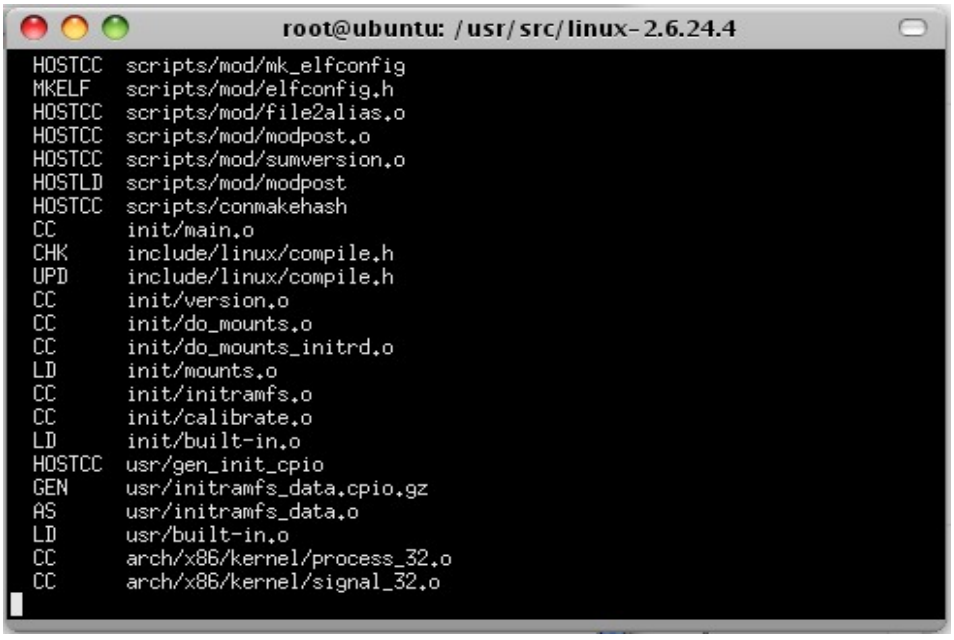
HP Vectra VL 400 :
Pentium III 800Mhz
128Mo RAM
Disque Dur 10 Go

L'intérêt de construire un système embarqué est la rapidité de démarrage et son faible poids. Nous pouvons utiliser ce deuxième paramètre pour optimiser la rapidité d'exécution : si le système est suffisamment petit, il pourra être intégralement chargé dans la mémoire vive. En fait, le bootloader de Linux nous permet de créer un ramdisk, un disque virtuel dans la ram. En clair, au démarrage, nous allons recopier notre partition système dans la mémoire vive et c'est ici que tout notre système fonctionnera : pour chercher ou éditer un fichier système, il faudra aller chercher dans la ram et non dans le disque dur, dont nous ne verrions d'ailleurs pas les fichiers. Outre la rapidité d'exécution, c'est aussi la vitesse d'accès au disque dur qui est optimisée : une fois le système chargé à l'allumage du PC, plus besoin d'accéder a la partition système, il ne reste que notre partition de données. Hors, notre direct-to-disk étant un deux pistes, il ne pourra que lire ou enregistrer, mais pas les deux à la fois, à la manière d'un DAT ou graveur CD. Autrement dit, notre disque dur ne fonctionnera dans le même temps qu'en lecture ou qu'en écriture. Autant dire que notre processeur cadencé à 800Mhz est une bête de course pour notre système. En résumé, notre système embarqué optimise : rapidité au démarrage, rapidité d'exécution, espace de stockage, temps d'accès aux supports de stockage.

Nous devons pour celà construire notre système allégé. Il faut partir d'une distribution lourde et enlever tout le superflux : interface graphique, logiciels inutiles même les plus petits, supprimer les démons inutiles, éradiquer les modules comme la gestion de périphériques inutilisés, etc.

Il faut ensuite installer perl et audio-ecasound pour exécuter notre programme, lcdproc pour gérer notre afficheur et notre clavier, ecasound pour enregistrer l'audio, et alsa pour gérer l'audio et l'interface audio.

Il faudra également compiler un noyau Linux le plus léger possible qui devra gérer l'accès aux disques IDE, le port parallèle, le réseau, les interfaces USB, les drivers ALSA, et surtout, en TEMPS REEL.
La compilation d'un noyau Linux est très simple : on télécharge la source, on la décompresse, on rentre dans le répertoire on tape « make xconfig » dans un terminal, ce qui nous ouvre une fenêtre qui nous permet de cocher/décocher tous les modules dont nous avons besoin. La principale difficulté réside dans le nombre d'options, constamment croissant avec les versions, et leurs commentaires parfois très vagues, du type « Option telle option : active ou désactive telle option. Cochez Oui à moins que vous soyiez sûr de ce que vous faîtes » ou « Cochez Non si vous n'avez pas compris pas les explications ». Il ne reste plus qu'à taper « make dep » et « make bzImage » et regarder défiler ce genre de choses pendant un moment interminable en étant très attentif au moindre problème :



15 minutes plus tard...

Le noyau résultant pèse moins d'1Mo, sans la lourdeur d'Ecasound, notre Direct To Disk pourrait tenir sur une disquette.

Après avoir transféré le tout sur notre PC de récupération, notre direct-to-disk est maintenant fonctionnel, ne reste plus qu'à le baptiser : le DansTonDisk v1.0 !

II/ Le DansTonDisk, gadget ou véritable machine dédiée ?

Parce qu'il est construit sur la base d'une carte mère de PC et qu'il tourne sous Linux, notre DansTonDisk a l'air d'un gadget pour geek bricoleur. Pourtant, qu'est-ce qu'un PC ?

- Une carte mère avec principalement :
  * Un processeur, avec des lignes d'entrées/sorties de données
  * Une RAM, pour contenir le programme en fonctionnement, y accéder rapidement et y stocker les infos temporaires dont il a besoin
  * Une ROM pour contenir le programme lorsque le PC est éteint et pouvoir le charger à l'allumage
- Et des périphériques divers comme un clavier, un écran, ou bien absolument n'importe quoi : ce sont simplement divers capteurs et récepteurs, tous gérés par le processeur par le biais d'un driver (qui n'est qu'un greffon de programme car un PC est modulaire par nature).

Et, qu'est-ce qu'un système dédié ?

- Un microcontrôleur, qui est un circuit intégré comprenant :
  * Un processeur RISC (à jeu d'instructions réduit) et ses lignes d'entrées/sorties
  * Une RAM
  * Une ROM
- Et des capteurs et récepteurs divers tous gérés par le processeur du microcontrôleur.

En outre un microcontrôleur peut disposer d'une horloge interne, d'entrées numériques, de convertisseurs A/N, d'unités de traitement USB...

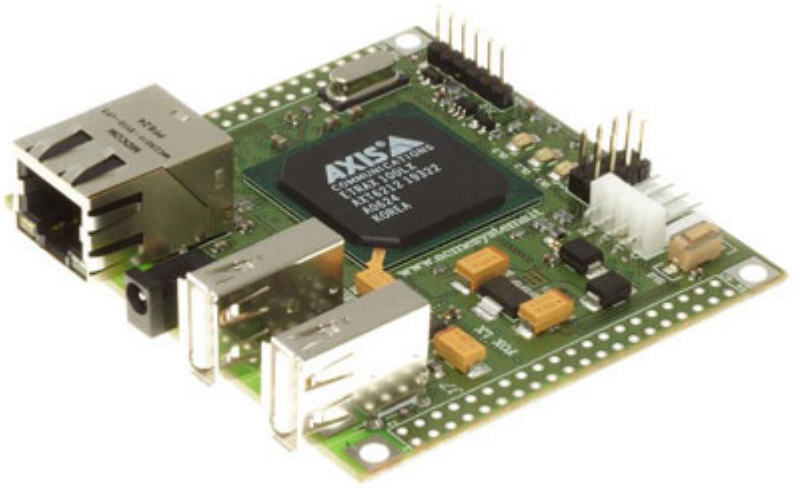
Mis à part le prix et les possibilités, il n'y pas vraiment de différence. Simplement, le système dédié comprend le minimum vital : ce dont il a besoin et uniquement ce dont il a besoin.

Autrement dit, utiliser un afficheur LCD et un clavier matriciel reliés à un microcontroller, des convertisseurs A/D et D/A et un contrôleur de disque dur ou de mémoire flash, et enfin s'affranchir de centaines de lignes d'assembleur et de C plutôt que de Perl aurait été exactement la même démarche (certes en version « légèrement » plus lourde). Pour l'expérimentation, et pour commencer quelque part, il était plus simple de partir d'un PC...

A l'inverse, le fait qu'elle possède une unité centrale déportée qui est

clairement un compatible PC fait-il de la fabuleuse Mackie D8b une « sous console » ? La SSL C200 tourne sous Linux. En cas de bug ou de besoin de mise-à-jour, il va falloir s'affranchir de quelques commandes UNIX : il ne manque à cette console que d'être estampillée d'un pingouin. Innovason conseille d'installer son Sensoft sur un PC externe prêt à prendre le relai en cas de problème sur le PC interne... Certes tout ce monde contrôle des DSP, mais d'une part les besoins ne sont pas les même que pour un deux pistes, et d'autres part c'est bien l'ordinateur qui décide des algorithmes à traiter et est donc le cerveau, ce qui fait d'un DSP un périphérique presque comme un autre.

Une solution intermédiaire entre le système dédié et le PC aurait été la carte mère embarquée, une sorte de version allégée de PC de taille réduite sans port classique parallèle ou série mais simplement des lignes d'entrées /sorties, mémoire flash, pas de refroidissement. Le processeur est un RISC (à jeu d'instructions réduit) et l'avantage de ce système est de faire tourner Linux, donc possibilité de créer facilement le programme, tout en complexifiant l'exercice en se rapprochant d'un microcontroller classique.



La Fox board, carte mère pour système embarqué

Notre Direct To Disk n'a donc finalement rien d'un gadget. En revanche son système est quelque peu lourd au vu des fonctions qu'il a à remplir.

III/ Quelque considérations d'ordre plus général en guise de conclusion

Finalement la technologie numérique est loin d'être si austère qu'on le prétend. On en parle souvent en la comparant à un ordinateur, c'est en fait exactement un ordinateur. Et elle est de plus en plus accessible, en matière de conception et de réalisation, au commun des mortels.

Les microcontrôleurs, notamment les PIC édités par la société Microchip, sont relativement simples à mettre en oeuvre. Ils disposent d'un certain nombre de modules internes, il suffit de choisir le modèle adapté à ses besoins, construire la carte permettant de le programmer (ou l'acheter), et de consulter la documentation. Celle-ci donne souvent des exemples d'utilisation, et nul besoin d'avoir l'âme d'un apprenti sorcier pour connecter quelques LEDs ou des boutons poussoirs, et l'on peut oublier la lourdeur et la complexité de l'électronique analogique. L'on a juste besoin de savoir souder pour relier des modules entre eux, de connecter le microcontrôleur à son PC et de programmer. Des compilateurs C sont maintenant à disposition, donc même l'aspect programmation est simplifié, sans compter les librairies de fonctions mises à dispositions (ce sont des morceaux de programmes déjà faits que le programmeur a juste à appeler en lui passant ses paramètres).

On peut ainsi citer le projet MidiBox (www.ucapps.de , « Non-commercial DIY Projects for MIDI Hardware Geeks »), qui consiste en un module de base constitué principalement d'un PIC 18F452, et de multiples modules d'entrées / sorties, par exemple un module AIN (Analog In Module) constitué uniquement de 4 multiplexeurs, un module MF (Motor Fader) permettant comme son nom l'indique de gérer 8 fader motorisés. De plus le module de base, le Core, permet d'écrire le programme directement à partir du port midi. Chaque module étonne par sa simplicité de mise en oeuvre, et une fois de plus l'essentiel de la conception revient à la programmation, encore simplifiée par les bibliothèques de fonctions proposées. Ainsi il devient facile de créer son propre instrument midi totalement original, en utilisant des schémas tout faits (les typons sont disponibles) et du code prémâché, la seule question à se poser étant de savoir quelque capteur utilisé pour créer quelque chose de nouveau. On peut aussi parler du MidiBox LC, une surface de contrôle possédant 8 fader motorisés, 8 V-Pots, etc, qui gère le protocole HUI, Logic control... Le tout toujours d'une simplicité déconcertante, du moins par rapport à l'imaginaire. Il est vrai que les typons destinés à etre assemblés par un humain et les boîtiers DIL contribuent à mieux appréhender cette relative simplicité. Lorsqu'on relie 3 MidiBox LC, et qu'on paye l'usinage et la sérigraphie d'une plaque d'aluminium, avec un peu de patience et de minutie, on peut arriver à fabriquer ceci par soi-même :

La Command24 du pauvre

Et l'auteur de ce projet n'a fait que suivre à la lettre les instructions d'un projet déjà fini, il s'est seulement contenté de le designer.

Une fois de plus tout ceci n'est « qu'une » interface de contrôle. Mais est-ce vraiment plus compliqué lorsqu'il s'agit de traiter de l'audio ? Réponse : pas vraiment. Une étape supplémentaire à récemment été franchie avec l'arrivée des dsPic : un microcontrôleur PIC basé sur un DSP 16 bits. Le dsPic peut contenir également quelques convertisseurs A/N (2 entrées 2 sorties) et les composants qui le contrôleront ou qui seront contrôlés ne seront une fois de plus pas des plus complexes à relier correctement. Les outils permettant de développer un programme pour ces dsPICs offrent la possibilité de développer uniquement en C, des bibliothèques de fonctions des plus simples aux plus complexes sont fournies (payantes). Et comme à son habitude Microchip proposent des kits tout faits, ne demandant qu'à être programmés par un geek en puissance. Ainsi l'on trouve d'ores et déjà des projets fonctionnels d'analyseur de spectre, de filtres simples (coupe-bas)... De là à connecter quelques V-Pots, quelques poussoirs, transposer quelques algorithmes et

formules pour programmer un EQ, il n'y a qu'un pas.

En résumé, entre allumer quelques LEDs via le port parallèle au moyen de quelques ligne de BASIC, et fabriquer son propre effet numérique, le fossé n'est pas si grand. Peut-être un jour un membre de la génération digitale concevra sa propre console numérique ?