R. DARDENNE, M. VAN DROOGENBROECK. **Acquisition vidéo au moyen d'une caméra IEEE 1394**. *Linux Magazine France*, N°69, pages 54–59, février 2005.

Acquisition vidéo au moyen d'une caméra IEEE 1394

Renaud DARDENNE, R.Dardenne@ulg.ac.be
Marc VAN DROOGENBROECK, M.VanDroogenbroeck@ulg.ac.be
Université de Liège, Belgique

14 décembre 2004

Résumé

Pour l'acquisition vidéo grand public, la norme USB 1.1 est devenu le choix le plus courant en matière de standard d'interconnexion. Pour des applications à plus haut débit, les utilisateurs préfèrent recourir à des cartes d'acquisition vidéo dédiées ou à des normes d'interfaçage génériques, permettant des transferts à débit plus élevé. Cet article explique comment se servir de la norme IEEE 1394, aussi appelée *FireWire* ou *i.Link*, pour l'acquisition de séquences vidéo.

Introduction

Deux approches s'affrontent en matière d'acquisition vidéo. On a d'une part les cartes d'acquisition, qui transforment un ou plusieurs signaux généralement analogiques en signal numérique et, d'autre part, l'approche consistant à réaliser l'acquisition des données brutes d'une caméra à sortie numérique via une interface standardisée comme les normes USB 1.1, USB 2.0 ou IEEE 1394 (également appelée FireWire dans la suite de ce document).

L'avantage de la seconde approche est de fonctionner sans carte d'acquisition dédiée. En contrepartie, l'absence de cette carte dédiée se traduit par l'impossibilité de traiter le signal sur la carte ; tous les traitements devront être réalisés par le processeur central. S'équiper de caméras numériques conformes à la norme USB 2.0 ou FireWire est une question de choix mais remarquons que (1) la norme FireWire permet différentes topologies (l'USB ne permet que des configurations en étoile) et dispose d'un standard bien défini et respecté par la plupart des fabriquants (IIDC 1.30), et (2) le débit le plus élevé est fourni par la norme FireWire.

C'est donc un choix qui semble s'imposer mais qui, à l'usage, présente quelques difficultés d'implémentation. La figure 1 montre une caméra numérique ayant une sortie FireWire.

FIG. 1 – Exemple de caméra IEEE 1394 (série Marlin de la société ALLIED).

Norme	Débit théorique
IEEE 1394a-S100	100 Mbit/s
IEEE 1394a-S200	200 Mbit/s
IEEE 1394a-S400	400 Mbit/s
IEEE 1394b-S800	800 Mbit/s
IEEE 1394b-S1200	1200 Mbit/s
IEEE 1394b-S1600	1600 Mbit/s
IEEE 1394b-S3200	3200 Mbit/s

TAB. 1 – Débits théoriques des variantes de la norme IEEE1394.

1 Quelques explications sur la norme FireWire

1.1 Le bus IEEE 1394

Fondamentalement, la norme IEEE 1394 n'est rien de plus que la description d'un bus. Mais, comme il s'agit d'un bus ayant une vitesse typique de 400 Mbit/s, la description prend plus que quelques pages ...

Le bus IEEE 1394 est un bus série Plug and Play capable de supporter simultanément jusqu'à 63 périphériques. Le débit est fonction de la variante de la norme que l'on considère. Le tableau 1 reprend les différentes variantes.

Parmi toutes ces normes, la norme IEEE 1394a-S400 est la plus répandue, ce qui correspond à un débit maximum de 400 Mbit par seconde.

Le bus 1394, dont l'architecture est reproduite à la figure 2, peut fonctionner selon 2 modes distincts :

– *asynchrone*. Ce mode de transfert est basé sur une transmission de paquets à intervalles de

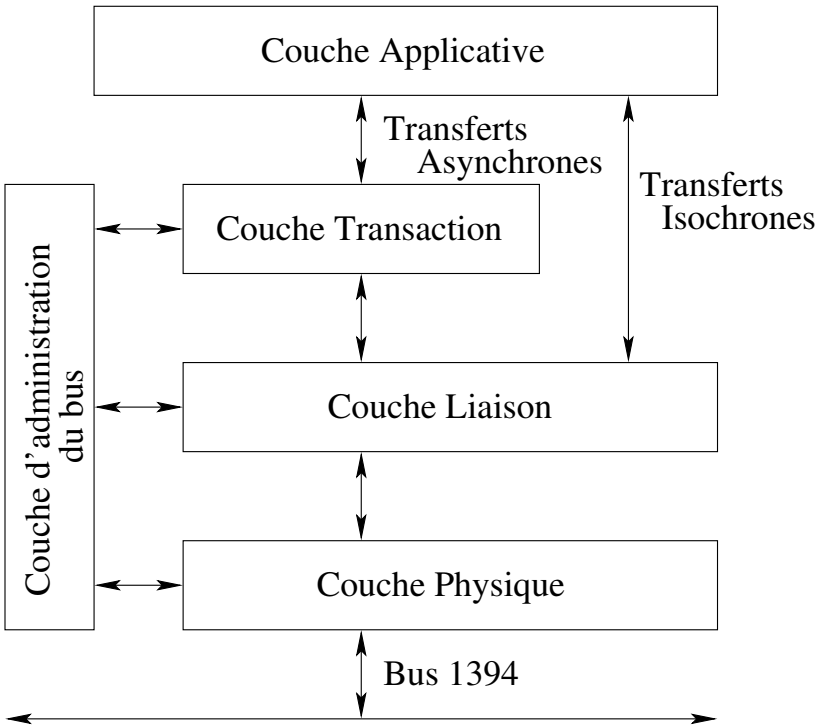

FIG. 2 – Architecture des protocoles IEEE 1394 (d'après [7]).

temps variables. Cela signifie que l'hôte envoie un paquet de données et attend de recevoir un accusé de réception du périphérique. Si l'hôte reçoit un accusé de réception, il envoie le paquet de données suivant, sinon le paquet est transmis à nouveau après un certain délai.

– *isochrone*. Ce mode permet l'envoi de paquets de données de taille fixe à intervalle de temps régulier (cadencé grâce à deux fils d'horloge). De cette façon, aucun accusé de réception n'est nécessaire, ce qui garantit un débit fixe et donc une bande passante. De plus, étant donné qu'aucun accusé n'est nécessaire, l'adressage des périphériques est simplifié et la bande passante économisée permet de gagner en vitesse de transfert. Les communications isochrones sont prioritaires par rapport aux asynchrones ce qui permet de garantit la bande passante pour des applications du type transmission vidéo temps-réel ; ce qui n'est pas le cas des transactions asynchrones transitant par la couche transaction. Si le débit est théoriquement égal à 400 Mbit/s, le cadencement des paquets d'une transmission isochrone fait que le débit utile ne dépassera pas 256 Mbit/s. En pratique, on est donc loin de la vitesse promise !

1.2 Transmission vidéo par IEEE 1394

La transmission de signaux vidéo n'est pas explicitement couverte par des normes de la famille IEEE 1394. C'est le groupement d'industriels appelé *1394 Trade Association* qui s'est chargé de définir une norme spécifiquement dédicacée à la transmission vidéo. C'est précisément le sous-groupe *Instrumentation & Industrial Digital Camera* (IIDC) qui a développé un protocole pour la transmission de signaux de caméras ; il s'agit du protocole *Digital Camera* (DCAM). Au passage, notons que la plupart des auteurs font indifféremment référence à l'IIDC et la norme DCAM pour qualifier le protocole de transmission. DCAM prévoit :

1. la transmission asynchrone d'informations de paramétrisation des caméras (taille, système de couleur, luminosité, balance des blancs, etc).
   Un logiciel comme Coriander [5] vous permettra de vous familiariser avec la panoplie de paramètres ajustables par DCAM (voir illustration 3).
2. la transmission isochrone de flux vidéo *non comprimés*.

Il existe également un format appelé DV (pour *Digital Video*), à ne pas confondre avec les formats DVCAM et DVPRO, ... (vous suivez toujours ?). Ce format est utilisé notamment par les caméscopes numériques à interface IEEE 1394 mais, à la grande différence avec DCAM, il définit le transfert de flux vidéo *comprimés*. Comprimer ou ne pas comprimer ? la polémique continue de faire rage entre constructeurs.

Dans cet article, nous nous contenterons de décrire l'acquisition suivant le protocole DCAM ; sans compression donc. Examinons donc de plus près le nombre de caméras utilisables sur un même bus.

Prenons comme exemple une caméra capable de fournir 30 images au format progressif, en couleurs (sur 24 bits) et de taille 640x480. Par caméra, cela représente un débit (en bits) de 30x24x640x480, soit 221 Mbit/s.

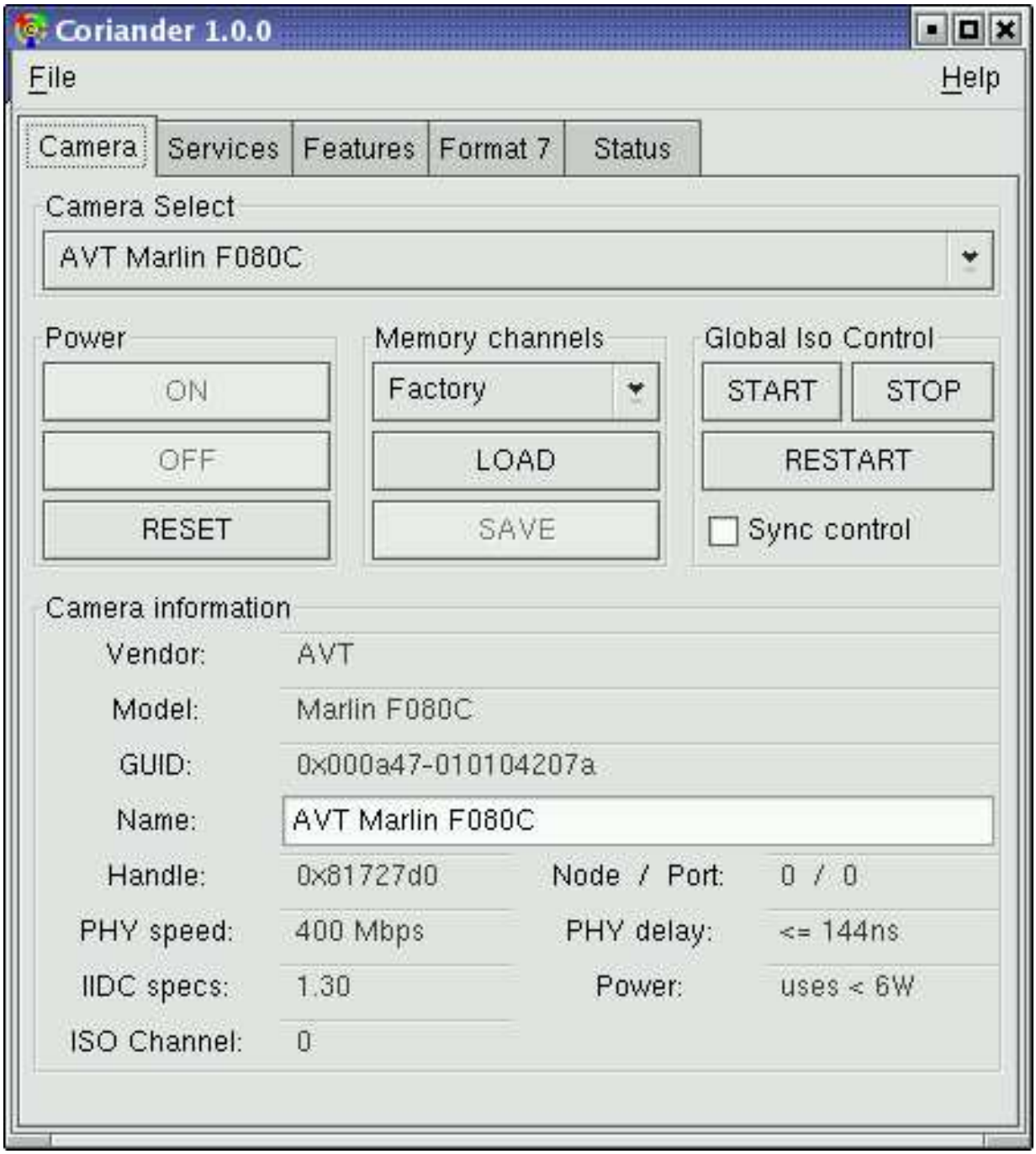

FIG. 3 – Fenêtre du logiciel Coriander.

	RGB	YUV (4 :2 :2)	YUV (4 :1 :1)	Monochrome (4 :2 :2)
Taille de l'image	640x480	640x480	640x480	640x480
Octet par pixel	3	2	1,5	1
Nombre d'images par seconde	30	30	30	30
Débit	221 Mbit/s	147 Mbit/s	111 Mbit/s	74 Mbit/s

TAB. 2 – Calcul explicite de débits pour certains formats vidéo.

A priori, il n'est donc pas possible d'utiliser 2 caméras couleurs sur une carte FireWire ; en tous cas à la cadence nominale de 30 images par seconde. Mais c'est oublier un peu vite que les formats vidéo sont multiples. En effet, le format *RVB* (*RGB* en anglais), qui consiste à attribuer un octet par composante de couleur rouge, verte ou bleue par pixel, n'est pas la représentation la plus efficace. Il se fait que le système visuel humain est plus sensible au signal de luminance qu'aux signaux de couleurs. Comme le système RVB n'en tire aucun profit, certains préfèrent utiliser le système *YUV* (ou *YCbCr*) qui a l'avantage de séparer une composante moyenne des signaux RVB, appelée *luminance* (Y), de deux signaux de couleurs (U, V ou Cb, Cr) appelés *chrominances*. Le mérite de ce changement de représentation est avant tout de permettre une représentation des chrominances avec moins d'échantillons que celle de la composante Y car l'œil y est moins sensible. Cela a conduit à définir une série de systèmes YUV en fonction de l'échantillonnage des composantes YUV, dont voici les principaux :

4 :4 :4 Ce schéma préserve la grille d'échantillonnage originale ; il y a donc un échantillon de luminance et deux échantillons de chrominance par pixel.

4 :2 :2 C'est le format obtenu en conservant toutes les lignes mais seulement un échantillon de chrominance sur 2 par ligne. Autrement dit, par échantillon de luminance, il y a soit un échantillon U, soit un échantillon V. Cela représente une moyenne de 2 octets par pixel, contrairement au schéma 4 :4 :4 qui nécessite 3 octets par pixel.

4 :1 :1 Basé sur le même principe d'échantillonnage, on ne garde ici qu'un échantillon par chrominance sur 4. Autrement dit, pour deux échantillons de luminance, il y a, au total, un échantillon de chrominance. Le débit total est donc 1,5 celui de la luminance.

Nous sommes maintenant en mesure de calculer les débits pour quelques formats vidéo numériques typiques. Ces débits sont repris dans le tableau 2.

Le débit maximum typique d'une carte FireWire est limité à 400 Mbit/s. Hélas, un fonctionnement isochrone du bus –indispensable pour des transmissions continues de signaux vidéo– limite le débit utile à 256 Mbit/s, ce qui signifie qu'il n'est pas possible de brancher 2 caméras au format YUV 4 :2 :2 sur une même carte sans réduire le nombre d'images par seconde. Bien entendu, rien n'empêche de brancher plusieurs caméras sur différentes cartes FireWire ; ainsi, 2 cartes permettront d'acquérir simultanément les signaux de plusieurs caméras, même à haut débit. Notons au passage qu'avec des caméras monochromes, il est possible d'accroître leur nombre ou

d'augmenter la taille de l'image, ce qui peut convenir pour des applications industrielles plus exigeantes.

Des améliorations (IEEE 1394b) ont été prévues pour porter la bande passante de 400 Mbit/s à 800, 1600 et même à 3200 Mbit/s. Signalons tout de même que ces débits seront limités par la bande passante du bus du processeur. En effet, le bus PCI est limité à 1 Gbit/s. Il faudra dès lors attendre une généralisation de ses successeurs (PCI-X, PCI-Express) avant de pouvoir pleinement bénéficier de ces capacités.

2 Enfin du code ...

Cette fois, nous entrons dans le vif du sujet. Pour l'acquisition, il faut configurer les périphériques, installer des librairies et écrire un peu de code. Voici comment procéder.

2.1 Configuration des périphériques

Cette section décrit ce dont le système a besoin pour pouvoir acquérir en toute sérénité :

1. Tout d'abord disposer d'une carte 1394 (!). Il en existe de deux types : celles respectant la norme OHCI et celles ne la respectant pas. Inutile de dire que les cartes appartenant à la première catégorie sont mieux supportées sous linux. On peut facilement en vérifier la présence à l'aide de `/sbin/lspci -vvx`.
2. La présence d'un noyau récent (`uname -r`) ne peut que vous éviter des écueils : au minimum supérieur à la version 2.4.20, un noyau 2.6 étant plus que conseillé pour éviter quelques beaux plantages et profiter des dernières améliorations parmi lesquelles l'amélioration du sous-système d'entrée/sortie, du sous-système des modules et périphérique unifié ainsi que le support du Hot Plug.
3. Ensuite, il faut vérifier la présence des périphériques adéquats dans `/dev` : `/dev/video1394` et `/dev/raw1394` doivent tous deux être présents et posséder respectivement les numéros de majeurs/mineurs suivants : (171/0) et (171/16). Attention : ces valeurs ne sont correctes que sur des systèmes dont le noyau est supérieur à 2.4.20. Voici, dans le cas d'un noyau 2.6.4 les entrées dans `/dev` d'un système sur lequel il est prévu de brancher quatre caméras :
   ```
   crw-rw-rw- 1 root 171, 0 Sep 15 2003 /dev/raw1394
   crw-rw-rw- 1 root 171, 16 Sep 15 2003 /dev/video1394/0
   crw-rw-rw- 1 root 171, 17 Sep 15 2003 /dev/video1394/1
   crw-rw-rw- 1 root 171, 18 Sep 15 2003 /dev/video1394/2
   crw-rw-rw- 1 root 171, 19 Sep 15 2003 /dev/video1394/3
   ```
   Certains systèmes paranoïaques empêchent la lecture par un utilisateur de ces périphériques. Il convient donc d'ajuster les permissions.

4. Les drivers 1394 sont habituellement chargés en tant que modules. Il faut en vérifier le chargement correct : les modules requis sont ieee1394, ohci1394, raw1394 et video1394. Le système doit normalement charger de lui-même les deux premiers (à vérifier avec la commande `dmesg`) si une carte est présente. raw1394 et video1394 doivent quant à eux être chargés à la main. Il suffit d'ajouter les lignes suivantes à la fin du fichier `/etc/rc.d/rc.local`
```
/sbin/modprobe raw1394
/sbin/modprobe video1394
```
et de vérifier que tout s'est bien passé (à l'aide de `lsmod`).

Que sont raw1394 et video1394 ? Ce sont des drivers de haut niveau. Les fonctions de raw1394 sont appelées lorsqu'un hôte est ajouté ou supprimé, lorsque les drivers de bas niveaux envoient des données en mode isochrone, etc. video1394 implémente un accès matériel via un canal DMA pour le réception et l'envoi de données isochrones, ce qui signifie que le processeur ne doit pas prendre en charge les opérations de copie des données vers la mémoire centrale.

Un éventuel problème peut survenir en conséquence du chargement du module eth1394 au démarrage, empêchant le chargement des autres drivers. eth1394 est un module qui, comme son nom l'indique ? !, permet le transport de datagrammes IPv4 sur un bus IEEE1394. Cette possibilité a été décrite dans le Request For Comments (RFC) numéro 2734. Notons toutefois que le module ne supporte pas le protocole complet.

Lorsque toutes ces opérations ont été réalisées. On peut vérifier que tout s'est bien passé en branchant les caméras et en lançant l'utilitaire `systool`, faisant partie du package sysfsutils, qui permet de donner une foule de renseignements sur la structure et le contenu du système de fichier virtuel `/sys` (uniquement noyaux 2.6) reflétant la configuration réelle de la machine.

2.2 Librairies (API) nécessaires

Il existe des librairies correspondantes aux 2 drivers de haut niveau que sont raw1394 et video1394 : libraw1394 [3] et libdc1394 [2]. Toutes deux sont disponibles sur sourceforge, mais elles sont toujours en développement.

La librairie libraw1394 (version 1.1.0) inclut tout ce qu'il faut pour récupérer les données brutes via le bus IEEE 1394 et même pour la norme IEEE 1394b à 800 Mbit/s ! La librairie libdc1394 est indispensable pour ne pas consommer de ressources CPU (ce qui est possible grâce à l'accès DMA) et, comme elle est mise à jour régulièrement, il convient de télécharger la dernière version (1.0.0). Remarquons que l'installation des librairies ne fait pas toujours correctement appel à `ldconfig` ; il est donc parfois nécessaire de mettre à jour soi-même le cache des librairies pour l'édition de liens.

2.3 Un exemple d'acquisition multi-caméras en C

Pour cet exemple, nous avons choisi une acquisition multi caméras car cette approche est plus générique sans pour autant augmenter de manière significative la complexité du code. Les ins-

tructions comportent les phases suivantes.

2.3.1 Inclure les librairies

Sans quoi rien ne fonctionnera :

```
#include <libraw1394/raw1394.h>
#include <libdc1394/dc1394_control.h>
```

2.3.2 Vérifier que tout est prêt

Même si après la lecture des précédents paragraphes, tout devrait être opérationnel, nous allons tout de même vérifier le bon chargement des modules. Puisque libdc1394 est une implémentation plus efficace de l'acquisition, vous pouvez vous demander pourquoi libraw1394 est nécessaire. La réponse est que la librairie définit un type que nous allons recroiser par la suite : `raw1394handle_t`. Nous allons commencer par vérifier que les drivers sont bien chargés et que nous avons les permissions suffisantes pour accéder aux périphériques en lecture :

```
// Déclaration pour raw1493
raw1394handle_t handle;
struct raw1394_portinfo ports[MAX_PORTS];

// Assigner un "handle"
handle = raw1394_new_handle();
if (handle==NULL) {
  fprintf( stderr, "Unable to aquire a raw1394 handle\n\n"
     "Please check \n"
     " - if modules 'ieee1394','raw1394' and 'ohci1394' are loaded \n"
     " - if you have read/write access to /dev/raw1394\n\n");
  exit(1);
}
handle = NULL;
```

2.3.3 Compter le nombre de ports

Une fois cette vérification d'usage effectuée, nous savons que tout ce qui est nécessaire à l'acquisition est en place. Nous allons poursuivre en vérifiant le nombre de cartes FireWire présentes sur la machine ; dans la terminologie FireWire, une carte est appelée "port" :

```
// Déclarations pour raw1394
raw1394handle_t handle;
struct raw1394_portinfo ports[MAX_PORTS];
int numPorts;
```

```
// Assigner un "handle"
printf("Asking new handle ...\n");
handle = raw1394_new_handle();
if (handle==NULL) {
  fprintf( stderr, "Unable to aquire a raw1394 handle\n\n");
  exit(1);
}

// Compter le nombre de ports
printf("Getting the number of ports ...\n");
numPorts = raw1394_get_port_info(handle, ports, numPorts);
printf("[ %2d ]",numPorts);
raw1394_destroy_handle(handle);
handle = NULL;
```

où `MAXPORT` est une macro spécifiant le nombre maximal de cartes potentiellement connectées à la machine.

2.3.4 Compter le nombre de nœuds sur chaque port

Par pure commodité nous supposerons que certaines variables sont globales :

```
nodeid_t *camera_nodes;
raw1394handle_t handles[MAX_CAMERAS];
dc1394_cameracapture cameras[MAX_CAMERAS];
int camCount=0;
```

`camera_nodes` est une structure qui nous permet de compter les nombre de caméras effectivement connectées, `handles` est un tableau de descripteurs que nous utiliserons sans cesse, `cameras`, qui est du type `dc1394_cameracapture`, contient toutes les données sur les caméras y compris le buffer de données associé et `camCount` sera le nombre de caméras effectivement connectées.

```
int numPorts;
int p=0;
raw1394handle_t handle;
int found=0;

// Pour chaque port
for (p = 0; p < numPorts; p++) {
  // Demander un handle
  handle = raw1394_new_handle();
  raw1394_set_port(handle, p );
```

```
  // Demander le nombre de caméras connectées
  camera_nodes = dc1394_get_camera_nodes(handle, &camCount, 1);

  // Libérer le handle
  raw1394_destroy_handle(handle);
  handle = NULL;
  found += camCount;
}
// found contient le nombre de caméras conectées
```

L'appel à la fonction `dc1394_get_camera_nodes` permet de remplir la structure `nodeid_t *camera_nodes` avec les données propres à chaque camera qui seront utilisées lors de l'initialisation de l'acquisition, ainsi que de montrer une description de la caméra trouvée. A priori donc, après acquisition, le programme ira chercher les caractéristiques et les données d'une caméra à l'intérieur de cette structure pour effectuer les traitements. En voici la définition complète :

```
typedef struct __dc1394_cam_cap_struct
{
    nodeid_t node;
    int channel;
    int frame_rate;
    int frame_width, frame_height;  /* Largeur, hauteur */
    int * capture_buffer;           /* Buffer de données */
    int quadlets_per_frame;
    int quadlets_per_packet;
    /* components needed for the DMA based video capture */
    const unsigned char * dma_ring_buffer;
    int dma_buffer_size;
    int dma_frame_size;
    int num_dma_buffers;
    int dma_last_buffer;
    int num_dma_buffers_behind;
    const char * dma_device_file;
    int dma_fd;
    int port;
    struct timeval filltime;
    int drop_frames;
} dc1394_cameracapture ;
```

2.3.5 Fixer les paramètres de l'acquisition

C'est, en nombre de lignes de code, la partie la plus longue ; c'est ici que nous allons fixer les paramètres comme la taille des images, le nombre d'images par seconde, l'espace de couleurs

utilisé, etc. Avant de nous lancer dans le code, nous allons faire un petit rappel des différents formats existants. Le standard IIDC 1.30 définit 7 formats se distinguant par la résolution maximale que chacun peut atteindre. Chacun de ces formats se subdivise en modes spécifiant une résolution et un espace de couleur précis. Chacun de ces modes se subdivise à son tour afin de spécifier le nombre d'images par seconde qui peut être compris entre 1.875 et 60 image/s. Bien sûr, toutes les caméras ne supportent pas tous les modes à tous les débits. Référez-vous donc au manuel de votre matériel pour connaître les modes supportés. La librairie libdc1394 comporte bien des appels (cf. `dc1394_control.h`) permettant de demander aux caméras les modes qu'elles supportent, mais nous ne les utiliserons pas pour ne pas surcharger le code. La description des premiers modes est fournie dans le tableau 3.

Mode	Description
0	160x120 YUV (4 :4 :4)
1	320x240 YUV (4 :2 :2)
2	640x480 YUV (4 :1 :1)
3	640x480 YUV (4 :2 :2)
4	640x480 RGB
5	640x480 Y (Mono)
6	640x480 Y (Mono16)

Format 0

Mode	Description
0	800x600 YUV (4 :2 :2)
1	800x600 RGB
2	800x600 Y (Mono)
3	1024x768 (4 :2 :2)
4	1024x768 (RGB)
5	1024x768 Y (Mono)
6	800x600 Y (Mono16)
7	1024x768 Y (Mono16)

Format 1

TAB. 3 – Description des premiers modes des formats 0 et 1.

```
#define DROP_FRAMES 0

// Définition des débits et des modes supportés par toutes les caméras
int fps = FRAMERATE_30; // 30 images/sec
int res = MODE_640x480_MONO;
int formatGeneral = FORMAT_VGA_NONCOMPRESSED;
char *device_name=NULL;

// iso transmission
unsigned int channel;
unsigned int speed;

int i, p;
// Pour chaque port
for (p = 0; p < portCount; p++) {
  // Pour chaque caméra
  for (i = 0; i < thisCamCount; i++) {
    handles[numCameras] = dc1394_create_handle(p);
    if (handles[numCameras]==NULL)
```

```
      perror("Unable to aquire a raw1394 handle\n");
    // Obtenir un canal ISO
    if (dc1394_get_iso_channel_and_speed(handles[numCameras],
        cameras[numCameras].node,&channel, &speed) != DC1394_SUCCESS) {
      printf("unable to get the iso channel number\n");
      cleanBus();
      exit(-1);
    }
    // Fixer les paramètres
    if (dc1394_dma_setup_capture(handles[numCameras],
        cameras[numCameras].node, i+1, formatGeneral, res, SPEED_400, fps,
        NUM_BUFFERS ,DROP_FRAMES, device_name, &cameras[numCameras])
          != DC1394_SUCCESS) {
      perror("Unable to setup capture");
      cleanBus();
      exit(-1);
    }

    // Commencer la transmission ISO
    if (dc1394_start_iso_transmission(handles[numCameras],
cameras[numCameras].node) != DC1394_SUCCESS) {
      perror("unable to start camera iso transmission\n");
      cleanBus();
      exit(-1);
    }
  }
  dc1394_free_camera_nodes(camera_nodes);
}
```

Il vous faut évidemment veiller à ce que votre caméra supporte le bon nombre d'images par secondes (*framerate*) à la résolution demandée.

Pour une description complète des appels aux différentes fonctions de la librairie, nous ne pouvons que vous conseiller de lire les commentaires qui se trouvent dans le fichier `dc1394_control.h` de libdc1394. Il n'existe malheureusement aucune documentation officielle à ce jour ...

2.3.6 Acquérir

Une fois tous les paramètres fixés aux bonnes valeurs, acquérir est une simple formalité. Imaginons par exemple que l'on fasse une simple inversion de dynamique, c'est-à-dire que toute valeur `x` du buffer est transformée en `255-x` :

```
// Acquérir
dc1394_dma_multi_capture(cameras, camCount);
// cameras[x].capture_buffer contient les données
```

```
// Effectuons un traitement simple sur la deuxième caméra
int taille;
int *output; // Ne pas oublier d'allouer ce buffer de sortie!

taille = cameras[1].frame_width * cameras[1].frame_height * 3;
// Utilisation du buffer
for (i=0 ; i< taille ; i++)
  *(output +i) = 255 - *(cameras[1].capture_buffer +i);

// Rendre le buffer au driver
for (i = 0; i < camCount; i++)
  dc1394_dma_done_with_buffer(&cameras[i]);
```

Quelques remarques :

- Avant de pouvoir réutiliser le buffer, il faut signaler qu'on a fini de l'utiliser.
- L'exemple présuppose que l'image soit acquise au format YUV 4 :4 :4. Si ce n'est pas le cas, il faut très souvent commencer par une conversion d'espace de couleurs ou adapter le nombre d'octets à parcourir par pixel. Des routines de conversion efficicaces, de YUV vers RGB par exemple, sont disponibles sur Internet : Coriander en possède pour les cas les plus fréquents.
- La librairie interdit que l'on écrive dans le buffer d'entrée. Dès lors, il convient de déclarer et d'allouer un buffer de sortie.
- Le buffer d'entrée est du type `(int *)`. Si, comme d'usage courant, on se contente de valeurs de pixels comprises entre 0 et 255, un octet (de type `unsigned char`) aurait pu suffire ; on gaspille donc 2 octets de mémoire par pixel !
  Mais il y a une explication à cela : certaines caméras FireWire, supportées sous linux, travaille sur une dynamique de 10, voire 12 bits par composante. Le type `unsigned char` ne conviendra plus de toutes façons. C'est donc par compromis que les concepteurs de la librairie ont choisi le type entier pour les données.

2.3.7 Une gestion propre de la sortie

Lorsque pour une raison ou pour une autre un appel a échoué, un simple appel à `exit()` est un peu barbare. C'est dans cette section que se justifie le plus la déclaration globale des variables `handles`, `cameras` et `camCount` : nous commençons par arrêter la transmission isochrone, nous arrêtons les caméras, nous libérons le canal DMA et les handlers.

```
void cleanBus()
{
  int i;
  printf("\n Stopping ISO transmissions");
  for (i=0; i < camCount; i++)
    if (dc1394_stop_iso_transmission(handles[i], cameras[i].node )
```

```
          != DC1394_SUCCESS) {
        perror(" *** unable to stop camera iso transmission ***");
        exit(-1);
      }

  printf("\n Cameras PowerOff");
  for(i=0 ; i< camCount; i++)
    if (dc1394_camera_off(handles[i], cameras[i].node) != DC1394_SUCCESS)
      perror(" *** PowerOff Failed ***");

  printf("\n Releasing DMA channels and freeing handlers");
  for (i=0; i < camCount; i++) {
    dc1394_dma_unlisten( handles[i], &cameras[i] );
    dc1394_dma_release_camera( handles[i], &cameras[i]);
    dc1394_destroy_handle(handles[i]);
  }
}
```

Cette fonction sans argument peut être aussi positionnée sur le signal de Ctrl+C.

2.3.8 Afficher l'image

Cette section sort un peu du cadre propre de l'acquisition IEEE 1394, mais une visualisation des images issues des caméras peut être réalisée très aisément à l'aide de GTK2 ou de SDL.

- GTK est sous licence GNU LGPL et permet de créer très rapidement des interfaces graphiques. GTK+ existe sur plusieurs plates-formes : plates-formes UNIX-like, Windows, BeOs. GTK+ est également utilisable avec plusieurs langages de programmation.
- SDL permet de réaliser des programmes qui utilisent des images, des sons, les lecteur de CD-ROM, qui communiquent en réseau etc. La bibliothèque est très populaire pour des applications multimédia comme les jeux vidéo, les démos et les émulateurs. Elle peut également être utilisée avec plusieurs langages de programmation, comme le C, le Perl, etc.

Vous pouvez consulter d'autres articles parus dans ce journal pour des explications et des exemples de programmes de visualisation basés sur les librairies GTK et SDL.

À vous de jouer maintenant !

Références

[1] Installation des drivers IEEE 1394, `http://www.videredesign.com/support_linux_kernel.htm/`.

[2] Librairie libdc1394, `http://sourceforge.net/projects/libdc1394`.

[3] Librairie libraw1394, http://sourceforge.net/projects/libraw1394.

[4] Liste des caméras IEEE 1394, http://www.tele.ucl.ac.be/PEOPLE/DOUXCHAMPS/ieee1394/cameras/.

[5] Logiciel coriander (par Damien Douxchamps), http://www.tele.ucl.ac.be/PEOPLE/DOUXCHAMPS/ieee1394/coriander/.

[6] Ressources IEEE 1394 pour linux, http://www.linux1394.org.

[7] http://www.hlp.fr/francais1/developp/ieee1394/ieee1394.html.