The cx23416 can produce (and the cx23415 can also read) raw YUV output. The format of a YUV frame is specific to this chip and is called HM12. 'HM' stands for 'Hauppauge Macroblock', which is a misnomer as 'Conexant Macroblock' would be more accurate.

The format is YUV 4:2:0 which uses 1 Y byte per pixel and 1 U and V byte per four pixels.

The data is encoded as two macroblock planes, the first containing the Y values, the second containing UV macroblocks.

The Y plane is divided into blocks of 16x16 pixels from left to right and from top to bottom. Each block is transmitted in turn, line-by-line.

So the first 16 bytes are the first line of the top-left block, the second 16 bytes are the second line of the top-left block, etc. After transmitting this block the first line of the block on the right to the first block is transmitted, etc.

The UV plane is divided into blocks of 16x8 UV values going from left to right, top to bottom. Each block is transmitted in turn, line-by-line.

So the first 16 bytes are the first line of the top-left block and contain 8 UV value pairs (16 bytes in total). The second 16 bytes are the second line of 8 UV pairs of the top-left block, etc. After transmitting this block the first line of the block on the right to the first block is transmitted, etc.

The code below is given as an example on how to convert HM12 to separate Y, U and V planes. This code assumes frames of 720x576 (PAL) pixels.

The width of a frame is always 720 pixels, regardless of the actual specified width.

If the height is not a multiple of 32 lines, then the captured video is missing macroblocks at the end and is unusable. So the height must be a multiple of 32.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static unsigned char frame[576*720*3/2];
static unsigned char framey[576*720];
static unsigned char frameu[576*720 / 4];
static unsigned char framev[576*720 / 4];

static void de_macro_y(unsigned char* dst, unsigned char *src, int dstride, int
w, int h)
{
    unsigned int y, x, i;

    // descramble Y plane
    // dstride = 720 = w
```

```
    // The Y plane is divided into blocks of 16x16 pixels
    // Each block in transmitted in turn, line-by-line.
    for (y = 0; y < h; y += 16) {
        for (x = 0; x < w; x += 16) {
            for (i = 0; i < 16; i++) {
                memcpy(dst + x + (y + i) * dstride, src, 16);
                src += 16;
            }
        }
    }
}

static void de_macro_uv(unsigned char *dstu, unsigned char *dstv, unsigned char
*src, int dstride, int w, int h)
{
    unsigned int y, x, i;

    // descramble U/V plane
    // dstride = 720 / 2 = w
    // The U/V values are interlaced (UVUV...).
    // Again, the UV plane is divided into blocks of 16x16 UV values.
    // Each block in transmitted in turn, line-by-line.
    for (y = 0; y < h; y += 16) {
        for (x = 0; x < w; x += 8) {
            for (i = 0; i < 16; i++) {
                int idx = x + (y + i) * dstride;

                dstu[idx+0] = src[0];  dstv[idx+0] = src[1];
                dstu[idx+1] = src[2];  dstv[idx+1] = src[3];
                dstu[idx+2] = src[4];  dstv[idx+2] = src[5];
                dstu[idx+3] = src[6];  dstv[idx+3] = src[7];
                dstu[idx+4] = src[8];  dstv[idx+4] = src[9];
                dstu[idx+5] = src[10]; dstv[idx+5] = src[11];
                dstu[idx+6] = src[12]; dstv[idx+6] = src[13];
                dstu[idx+7] = src[14]; dstv[idx+7] = src[15];
                src += 16;
            }
        }
    }
}

/*---------------------------------------------------------------------------*/
int main(int argc, char **argv)
{
    FILE *fin;
    int i;

    if (argc == 1) fin = stdin;
    else fin = fopen(argv[1], "r");

    if (fin == NULL) {
        fprintf(stderr, "cannot open input\n");
        exit(-1);
    }
    while (fread(frame, sizeof(frame), 1, fin) == 1) {
        de_macro_y(framey, frame, 720, 720, 576);
```

```
        de_macro_uv(frameu, framev, frame + 720 * 576, 720 / 2, 720 / 2, 576 /
2);
        fwrite(framey, sizeof(framey), 1, stdout);
        fwrite(framev, sizeof(framev), 1, stdout);
        fwrite(frameu, sizeof(frameu), 1, stdout);
    }
    fclose(fin);
    return 0;
}
```
