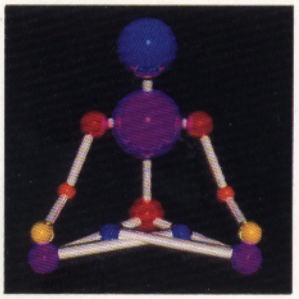
IRIS universe

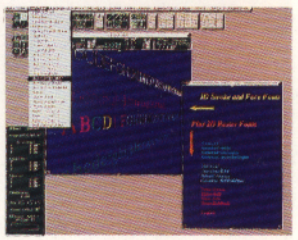
THE MAGAZINE OF VISUAL COMPUTING

NUMBER TWENTY

p. 56



p. 26



p. 16

FEATURES

ON THE COVER
Cover image was produced with SoftImage software on an IRIS Indigo at Metamorphose AG.

DEPARTMENTS

IRIS universe

THE MAGAZINE OF VISUAL COMPUTING

EDITOR Anne-Marie Gambelin

CONTRIBUTING EDITOR (International) Crispin Littlehales

DEPARTMENT EDITOR Gaye Graves

COPY EDITOR Paula Martin

EDITORIAL CONSULTANTS Douglas Cruickshank, Mark Compton

DESIGN Diéguez Design

DIGITAL IMAGES Susan Moran, Doug Morgan

TYPESETTING Lauren Langford Typography

COLOR SEPARATIONS Color Response, Inc.

PRINTING The Press, Inc.

EDITORIAL Silicon Graphics, Inc.
2011 N. Shoreline Blvd.
Mountain View, CA
94039
415/335-1293

CIRCULATION Silicon Graphics, Inc.
2011 N. Shoreline Blvd.
Mail Stop 415
Mountain View, CA
94039
415/335-1293
Fax: 415/968-3579

ADVERTISING Ardith Lowell
A.J. Lowell and Associates
2755 Campus Dr. #247
San Mateo, CA
94403
415/341-9681

ISSN 1061-6608

IRIS Universe: The Magazine of Visual Computing, is published quarterly by Silicon Graphics, Inc. and is dedicated exclusively to the needs and interests of the visual computing community. Please send address changes to *IRIS Universe*, Silicon Graphics, 2011 North Shoreline Boulevard, Mail Stop 415, Mountain View, CA 94039-7311. Subscriptions are available upon request to qualified users. Fill out the postage-paid subscription card in this issue. Correspondence regarding editorial (press releases and product announcements) should be sent to Editor, *IRIS Universe*, 2011 North Shoreline Boulevard, Mail Stop 415, Mountain View, CA 94039-7311. Letters to the *IRIS Universe* or its editors become the property of the magazine and are assumed to be intended for publication.

EDITOR'S NOTE

The emergence of a new technology promises to have an impact that may rival that of the printing press, the telephone and the motion picture.

—Douglas Cruickshank

I do not feel that Mr. Cruickshank's statement is an exaggeration. For indeed, what faces us is a revolution—one that will cause each of us to redefine the way we communicate and live. Previous revolutions in communication mediums yielded leaps in accessibility of information and increased efficiency of communication—not unlike what we will witness with digital media.

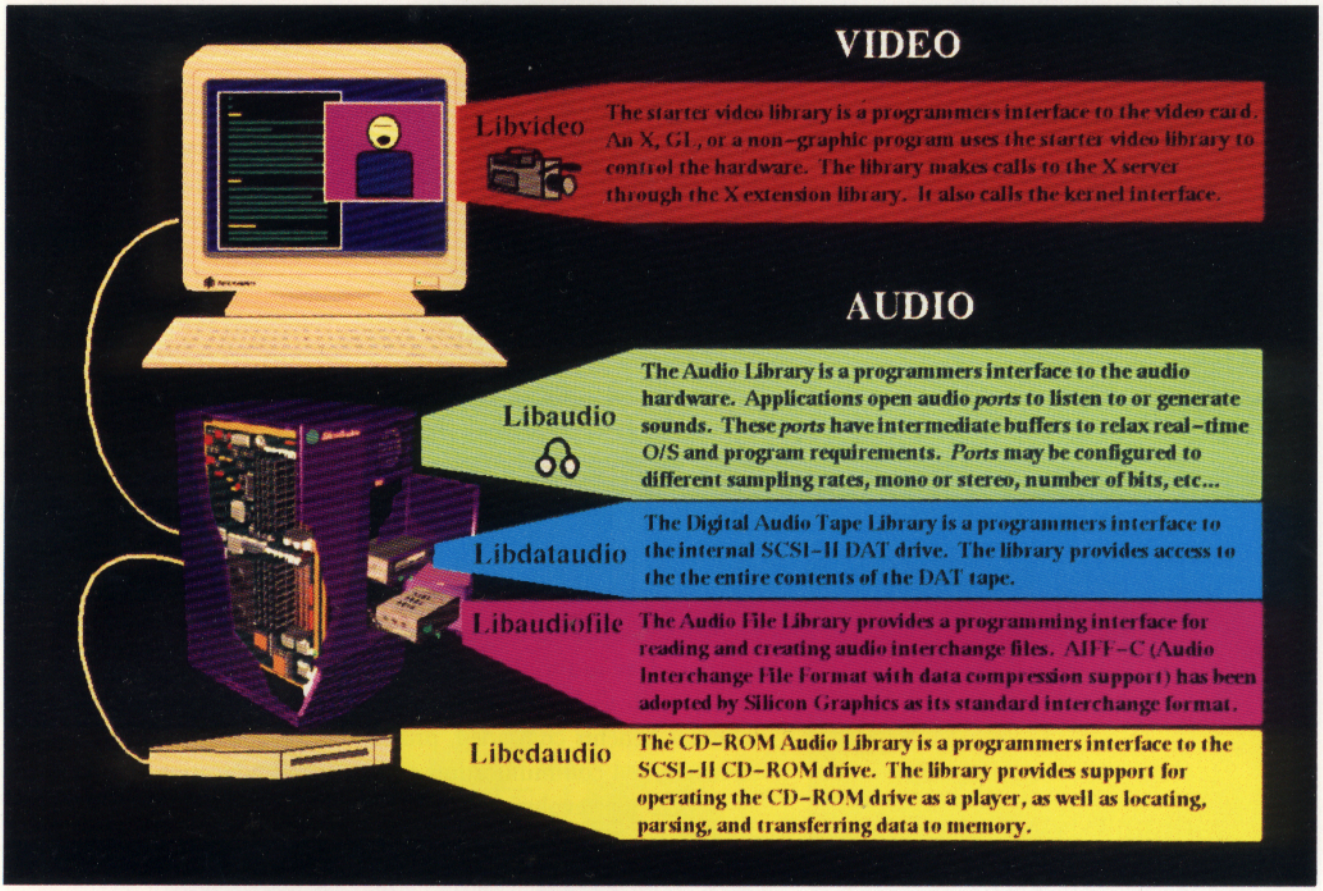
The arrival of the printing press and mass printed medium accelerated the spread of knowledge through accessibility. What previously was privy only to clergy and aristocracy was now within the grasp of the common man. Additionally, the invention of the telephone accelerated the method by which that information could be shared, further promoting the commerce and progress of mankind. Business transactions and personal affairs that took weeks by post were accomplished relatively instantaneously via telephone. Furthermore, the advent of the motion picture made all of us more sophisticated before our time—sharing ideas and information visually, efficiently and effectively. Today, with digital media, all three of these mediums, and their inherent benefits combined, promise to give birth to a new world of networked communication and universal education.

Like the revolutions that these mediums have ignited in the past, so too will the digital media revolution forever change the way we communicate and live. What new worlds await our discovery with increased volume and efficiency of knowledge gain? As our ability to share information accelerates, so too does our progress as mankind. One can only attempt to fathom where the technology of digital media will take us in the realms of science and culture. Perhaps the knowledge we gain will be what we need to avert the path of self-destruction that scientists, who most likely will be the first to benefit from this new medium, have been bringing to our collective awareness.

—Anne-Marie Gambelin

AUDIO & VIDEO

SOFTWARE DEVELOPMENT ENVIRONMENT



The Audio and Video Programming Environment is a collection of code libraries, sound libraries, and tools designed for developers of audio, MIDI, and video software, or developers seeking to integrate these media into their existing applications.

Typical audio programs access analog or digitally recorded sound data that is either input directly to the IRIS or stored on disk, digital-audio tape, or CD. They then manipulate the data and output the results live, to disk, or to tape. MIDI programs read, process, and produce MIDI data streams which in turn are interpreted by MIDI devices such as synthesizers, drum machines, and sequencers distributed across a MIDI network. Typical video applications include displaying live (camera), taped (S-VHS or Composite), or hard-disk (Compressed) video in a window, or recording graphics from a window to tape. The Audio and Video Programming Environment provides all the necessary tools to create such programs to run on all Silicon Graphics workstations supporting audio and video hardware.

By Paul Lacombe

AUDIO LIBRARY

Silicon Graphics' Audio Library (AL) provides a programming interface for real-time digital-audio I/O streams. The AL is a standard development-software component on the IRIS Indigo and Personal IRIS 4D/35 workstations, which include DAT-quality digital-audio subsystems. In addition, the AL will supply a standard library interface to the digital-audio subsystems on future Silicon Graphics platforms.

The AL promotes the philosophy that it is preferable to develop digital-audio code on a powerful general-purpose CPU running a familiar operating system than to develop code for specialized DSP hardware. RISC-processor technology continues to evolve rapidly, and a portable application written in a high level programming language can benefit from increased CPU performance with a minimum amount of work by the application's developer.

AL 1.0 consists of twenty-six "C" function calls. The basic library data structure is a port, which is an input or output channel for digital-audio sample data. A unique feature of the AL is that it allows multiple UNIX processes to share audio hardware. For example, two or more processes can open input ports and simultaneously process input data from a micophone; two or more processes can play audio tracks from disk and their audio channels will be mixed automatically to a single stereo output.

To write a more complex application than those above, it helps to understand the software model the library presents. Note that there are four types of routines in the library. (Look in "/usr/include/audio.h" to see the name prototypes of all the "Audio Library" routines. For some more ambitious sample programs look in "/usr/people/4Dgifts/examples/libaudio".) Watch for AL 2.0 in August 1992.

AUDIO FILE LIBRARY

The Audio File Library (AF—release date 8/92) provides a programming interface for reading and creating audio-interchange files. AIFF-C (extended Audio Interchange File Format with data-compression support) has been adopted by Silicon Graphics as its standard audio-file format. The Audio File Library allows you to write programs quickly which access audio-sample data stored in AIFF-C files, as well as non-audio data, such as sampler-configuration parameters, text strings, and application-specific information. The AF supports the original AIFF format in addition to the extended AIFF-C format.

AIFF/AIFF-C files contain chunks of various types of data including audio-sample data, text, sampler-configuration parameters, and application-defined data. Some of the chunks are optional; others are required to appear in every AIFF/AIFF-C file. AIFF-C includes some additional chunks not supported

A simple audio program to play 10 seconds of 16-bit, 44.1kHz, stereo audio from a file:

```
#include <sys/fcntl.h>
#include <audio.h>

main(int argc, char **argv)
{
        short buffer[44100 * 10 * 2];
          /* 10 seconds at 44.1KHz stereo */
        ALport audio_output1;
        char *file;
        int input;

    file = argv[1];
    input = open(file, O_RDWR);
    read(input, buffer, sizeof(buffer));
    audio_output1 = ALopenport("audio_output_port1",
      "w", 0);
    ALwritesamps(audio_output1, &buffer, sizeof(buffer)/2);
    while (ALgetfilled(audio_output1) 0)
        sginap(1);
    ALcloseport(audio_output1);
}
```

Open a file using the command argument for a file name, read the file into a buffer, open a port to write samples to (again choosing the defaults), write the samples out the port, loop until the port is drained, then close the port.

Here is a "hello world" program that obtains data from an AIFF-C file using Audio File Library calls (assume here that the input file contains 48 KHz stereo data):

```
#include <audio.h>
#include <audiofile.h>
main( )
{
    /* 3 seconds of 48 KHz stereo 16-bit sample data */
    short sampbuf[3 * 48000 * 2];
    long n;
    ALport outport;
    AFfilehandle playfile;

    playfile = AFopenfile("hellofile.aiff", "r", 0);
    outport = ALopenport("hello", "w", 0);

    /* read buffer of sound from file */
    n = AFreadsamps(playfile, AF_DEFAULT_TRACK, sampbuf,
      sizeof(sampbuf));

    /* play buffer of sound */
    ALwritesamps(outport, buf, n);

    /* wait until all sound has played */
    while(ALgetfilled(p) > 0)
      sginap(1); /* sleep for 1/60 of a second */

    AFclosefile(playfile);
    ALcloseport(outport);
}
```

by AIFF, such as the Sound Accelerator Chunk. See the AIFF-C specifications for a detailed description of the different kinds of chunks which may appear in AIFF/AIFF-C files, and a summary of the differences between the older and newer file formats.

The Audio File Library interface has been designed with enough generality so that it will be able to accommodate future versions of AIFF-C, and may be extended to support other file formats as well. An audio file is viewed logically as a collection of audio tracks, instrument maps, and miscellaneous data chunks. An audio track is a stream of audio samples (possibly a multi-channel, e.f. stereo, sample stream). An instrument map contains sample loops along with additional information about how the samples should be played back. Miscellaneous chunks include MIDI data, copyright/author strings and other text, and application-defined information.

AIFF-C allows at most one audio track (Sound Data Chunk) per file. Various audio-track configuration parameters (number of channels, sample rate, number of bits per sample, total number of samples) are stored in the AIFF-C Common Chunk and Sound Data Chunk. There may be one or more marker structures associated with the audio track (these are stored in the Marker Chunk). Markers provide pointers to sample locations within an audio track. These pointers are useful for marking loop endpoints, for attaching comments to specific sample locations, and so on.

An AIFF-C file may optionally contain an instrument map (Instrument Chunk). This map has storage locations for sample loops. The Audio File Library API has been generalized to allow an arbitrary number of loops per instrument, although AIFF-C currently limits you to two—a sustain loop and release loop. Loops are described in terms of marker structures. In order to read a loop from a file, you first determine which marker structures contain the end-points for a given loop, and then where each of the markers points into the audio track.

The Audio File Library allows you to open and read an existing audio file, or to create a new audio file. An AFfilehandle is the basic library data structure, which you use for reading/writing sample data, getting/setting audio-track configuration parameters, accessing instrument-map information, etc.

You use an AFfilesetup structure to configure a new file before you create it by calling the library-file open routine, AFopenfile(). The library writes out an AIFF-C header structure when you open a new file, and updates various fields in the header (such as the total sample count for the audio track) when you call the library-close routine, AFclosefile(). There is currently no library-edit mode (read/write access for existing files), since changing the size of any of the header fields will generally require you to relocate the contents of the audio track.

CD-ROM AUDIO LIBRARY (CD)

The special Silicon Graphics version of the Toshiba 3301 CD-ROM drive can transfer digital-audio data across the SCSI bus into system memory. The CD-ROM audio library provides support for audio compact discs in the CD-ROM drive. The library has three sections: support for operating the CD-ROM drive as a player, support for locating and transferring digital-audio data into computer memory, and support for parsing and understanding the content of that data.

```
#include <sys/types.h>
#include <cdaudio.h>

main( )
{
    CDPLAYER *cd = CDopen
        ("/dev/scsi/sc0d710", "r");
    CDSTATUS status;

    if (cd && CDgetstatus(cd, &status)) {
      CDplay(cd, status.first, 1);
      CDclose(cd);
      exit(0);
    }
    exit(1);
}
```

The CD-ROM drive can be operated as a CD player delivering analog audio to the drive's headphone and stereo line out-jacks. The program in the sidebar causes the CD-ROM drive with SCSI ID 7 to play, starting with the first track on the CD.

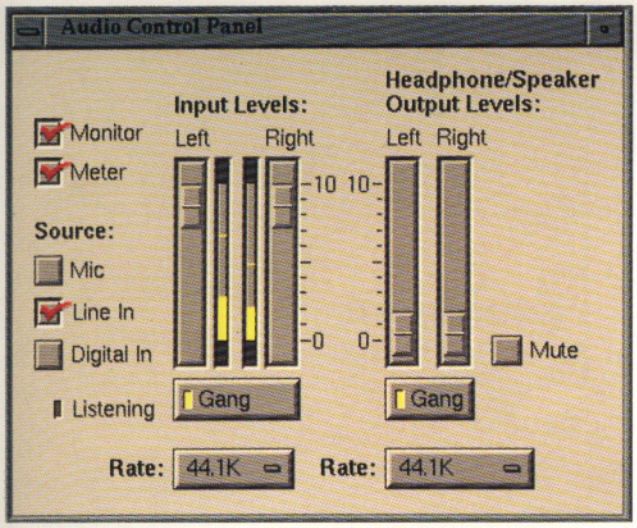
DIGITAL AUDIO TAPE LIBRARY (DT)

The Archive Python DAT (Digital Audio Tape) drive has the capability of reading and writing audio tapes compatible with consumer and professional DAT recorders. Silicon Graphics is the first in the industry to provide DAT over the SCSI bus. Users can also use the DAT drive to backup their hard disks (1.3 Gbytes—60 meters, and 2.0 Gbytes—90 meters), using standard UNIX commands like tar(1), or Silicon Graphics System Manager—Backup and Restore utility. There are two components to the support: the kernel SCSI tape driver and libdataudio.

The kernal tape driver provides the ability to switch between audio and regular data mode, reposition the tape, search for time codes, and perform 150X normal speed search or rewind, among other functions.

The digital-audio tape library contains a group of utility functions for handling some elements of the sub code data and it contains support for parsing and understanding the complete content of the digital audio data returned by a read(2).

The following example opens the tape drive, switches it to audio mode then constantly reads DTFRAMEs from the tape and hands them to the parser. It has one callback set up to play the audio data. The parser will call this function with the audio data which it will have byte-swapped and de-emphasized if the tape's pre-emphasis bit is turned on. The example uses another callback to keep track of the sampling frequency.



MIDI

The Musical Instrument Digital Interface (MIDI) Library was unavailable at the time of this printing, however, applications can transmit and receive MIDI data using the IRIS Indigo, 4D/30 and 4D/35 serial ports and three IRIX system calls. The MIDI Library will provide time-stamping information not inherent in the MIDI protocol, among other high level routines.

All that is required to connect MIDI equipment to the Indigo, 4D/30, and 4D/35 is any Macintosh/MIDI interface with a serial port cable (usually included), and a MIDI cable available at most music stores.

The following program sets up serial port 2 as a MIDI port to receive and print out MIDI data. Essentially, three ioctl system calls perform the function of setting the port to RAW, RS-422, and External Clock (MIDI boxes have built-in clocks, usually 1MHz).

The infinite while loop performs printf's until the program is halted with <CTlrl C>.

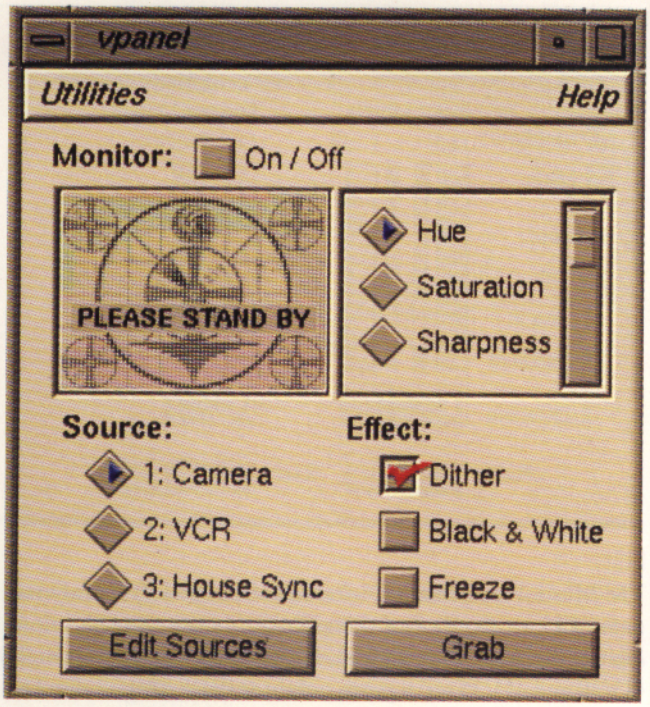
STARTER VIDEO LIBRARY (VL)

The Starter Video Library is a high level programming interface for applications that lets users integrate and access the features and functions of the Starter Video hardware. The library provides a fundamental set of calls to open a video window, select various parameters of the Starter Video hardware, grab a frame of YUV or RGB data, define an output window for simultaneous input and output among others. It is designed to allow easy

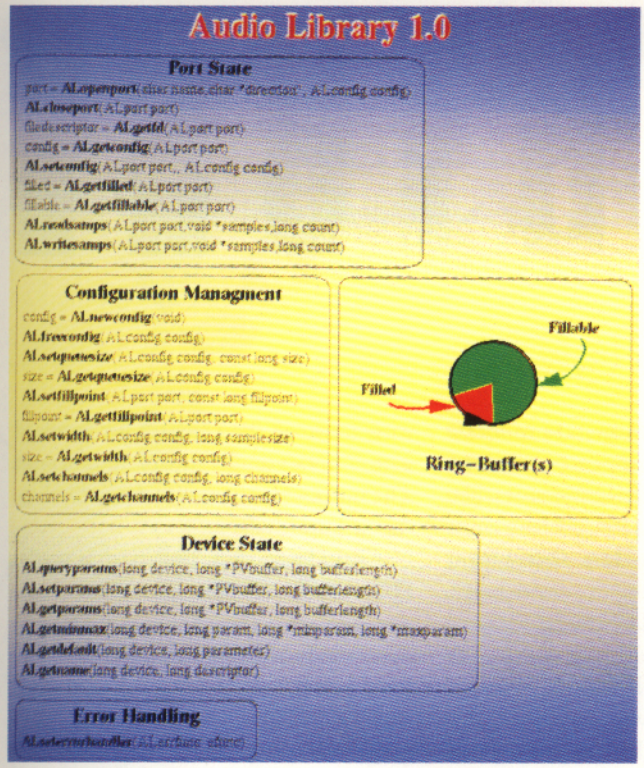
```
#include <sys/types.h>
#include <sys/fcntl.h>
#include <audio.h>
#include taudio.h

static int sampsperframe = DTDA_NUMSAMPS48K;
ALport audioport;

playaudio(void *arg, DTDATATYPES type, short *audio)
{
   ALwritesamps(audioport, audio, sampsperframe);
}

frequency(void *arg, DTDATATYPES type, int *freq)
{
   switch (*freq) {
      DT_FREQ48000:
         sampsperframe = DTDA_NUMSAMPS48K;
         break;
      DT_FREQ44100:
         sampsperframe = DTDA_NUMSAMPS44K;
         break;
      DT_FREQ32000:
         sampsperframe = DTDA_NUMSAMPS32K;
         break;
   }
}

main( )
{
int *tape = open("/dev/nrtape", O_RDONLY);
DTPARSER *dtp = DTcreateparser();
DTFRAME buf[4];
struct mtop mt_com;
int i, n;

audioport = ALopenport("DAT Test", "w", 0);
if (dtp) {
DTsetcallback(dtp, dt_audio,
            (DTCALLBACKFUNC)playaudio, 0);
DTsetcallback(dtp, dt_sampfreq,
            (DTCALLBACKFUNC)frequency, 0);
}else
   exit(1);
if (tape = 0) {
   mt_com.op = MT_AUD;
   mt_com.count = 1;
   ioctl(tape, MTIOCTOP, &mt_com);
   for (;;) {
      n = read(tape, buf, sizeof(buf));
      if (n < 0) {
         /* report error */
         exit(2);
      }
      if (n == 0) /* We're at the end of the tape */
         break;
      for (i = 0; i <4; i++)
         DTparseframe(dtp, &buf[i]);
   }
   exit(0);
}
exit(3);
} /* end main */
```

integration of NTSC or PAL video that is independent of the windowing system, whether a X or GL program. The Starter Video Library is designed to provide developers with a complete programming environment on an IRIS Indigo configured with Starter Video.

Applications for using the Starter Video Library range from animation assembly to motion analysis. Motion analysis can benefit from using the sequential capture feature to grab at timed intervals either RGB or YUV data files. The files then either can be compressed (see compression library) or left as binary files that can be looped with 4Dgifts "makemovie" to create an animated movie sequence. Another application for the Starter Video Library is rendering 24-bit images or animations into the video-frame buffer for frame-by-frame output to a VTR. This routine reconfigures the video board to do still-frame output of 24-bit images from the composite, S-video, or RGB video ports. An application can also make calls to define different states of the hardware, providing graphics overlay on top of video or video overlay on top of graphics. This capability is especially useful for placing titling or animations over a live-video input with simultaneous NTSC- or PAL-composite, S-video, and RGB output. The other hardware state provides a chroma-key feature by




```
#include <stdio.h>
#include <fcntl.h>
#include <sys/termio.h>
#include <sys/stropts.h>
#include <sys/z8530.h>

unsigned char buf[1000];
int midifile;

int main(int argc, char **argv)
{
    int n;
    struct strioctl str;
    struct termio t;
    int arg;
    int count = 0;

    if ((midifile=open("/dev/ttyd2",O_RDWR)) < 0) {
        perror("Couldn't open /dev/ttyd2");
        exit(1);
    }

    t.c_iflag = IGNBRK;
    t.c_oflag = 0;
    t.c_cflag = B9600|CS8|CREAD|CLOCAL|HUPCL;
    t.c_lflag = 0;
    t.c_line = 1;
    t.c_cc[VINTR] = 0;
    t.c_cc[VQUIT] = 0;
    t.c_cc[VERASE] = 0;
    t.c_cc[VKILL] = 0;
    t.c_cc[VMIN] = 1;
    t.c_cc[VTIME] = 0;
    ioctl(midifile, TCSETAF, &t);

    str.ic_cmd = SIOC_RS422;
    str.ic_timout = 0;
    str.ic_len = 4;
    arg = RS422_ON;
    str.ic_dp = (char *)&arg;
    if (ioctl(midifile, I_STR, &str) < 0) {
        perror("Can't ioctl RS422");
        exit(1);
    }
    str.ic_cmd = SIOC_EXTCLK;
    str.ic_timout = 0;
    str.ic_len = 4;
    arg = EXTCLK_32X;
    str.ic_dp = (char *)&arg;
    if (ioctl(midifile, I_STR, &str) < 0 {
        perror("Can't ioctl EXTCLK");
        exit(1);
    }

    while(1) {
        n = read(midifile, buf, 1);
        if (count++ < 8) {
            printf("%x\t", buf[0]);
        } else {
            count = 0;
            printf("%x\n", buf[0]);
        }
        fflush(stdout);
        if (n)
            write(midifile, buf, n);
    }
}
```



which a range of pixel color values, up to 256, can be selected and then "keyed" against the live-video stream. An important point to remember with the video overlay and underlay mode is that the live video written to the raster is dithered 8-bit RGB pixels and the graphics displayed on screen are also the same.

The software components to control the Starter Video board are the kernel driver, the X server, the X extension library, and the Starter Video Library. The kernel driver provides access to the hardware and directly controls the registers on the board. The driver is loaded into the kernel by the lboot procedure when you install it from tar tape or CD-ROM disc. The higher-level software uses the kernel interface to control the board, with user level applications hidden from it. The X server manages the video window and controls where the video window is placed on the screen. When moving a window it will automatically update the correct video position. The X server also manages the color maps that are required to support video. The X extension library is an interface that the Starter Video Library uses to have the X server work correctly with video. User-level programs must be linked with the extension library to display video on the screen. The Starter Video Library is the programmers' interface to the Starter Video board when using an X, GL or a non-graphic application. The library makes calls to the X server through the X extension library and also calls the kernel interface. Any application that uses video must be linked with /usr/lib/libvideo.a.

PROSONUS SOUND LIBRARY

IRIX 4.0.1 includes 10Mbytes of professional quality sound samples, licensed from the Prosonus Sound Library. The directory /usr/lib/sounds/prosonus contains a collection of music and sound files created by Prosonus especially for the Silicon Graphics IRIS Indigo, Personal IRIS 4D/35 or 4D/30 computer system. These files are a small selection of the music, sound effects, and instrument samples that will be available on CD-ROM through Silicon Graphics Software Express and Prosonus.

For more information concerning Prosonus CD-ROM products, call Software Express at 1 (800) 800-4SGI or contact Prosonus at:

Prosonus
11126 Weddington Street
North Hollywood, CA 91601
(800) 999-6191 or (818) 766-5221
(818) 766-6098

CONCLUSION

In tradition with the Graphics Library (GL) and Digital Media's commitment to its Iris Partners, the company will continue to evolve these digital media libraries. Digital Media values the suggestions of its developers and encourages them to provide feedback. The second major release of the Audio Library will ship this summer and many of the new features were implemented in response to existing developers' input. Direct inquiries to John Barco or Paul Lacombe, Digital Media Marketing.

COMPRESSION LIBRARY

The Digital Media Developer Kit will include a Compression Library as part of the package. This library is designed to give developers easy access to a standard API that requires almost no changes to the application to take advantage of various software compression algorithms.

Initially, a JPEG still-frame compression and decompression algorithm, a Silicon Graphics proprietary algorithm for compression and decompression of video streams, and a MPEG video-decompression algorithm primarily for CD-ROM playback rates will be bundled.

To get an idea of the performance rates on an Indigo system, the Silicon Graphics proprietary decompressor will run up to 320 by 240 at 15 frames per second with a data rate of about 864KB/s (6 bits/pixel). The MPEG decompressor will run up to 200 by 160 at 10 frames per second with a data rate of about 20KB/s (0.5 bits/pixel).

Included with the Digital Media Tool Suite is the first client application called the Movie Player. The Movie Player will enable the user to view sequences of compressed video that also can be synchronized with an audio track.

The Compression Library will be available with the release of the Digital Media Developer Kit, expected in early summer 1992.