

A Video Display System for an Automatic Cue Generator

by

Matthew G. Sexton


S.B., Massachusetts Institute of Technology (1996)


Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the

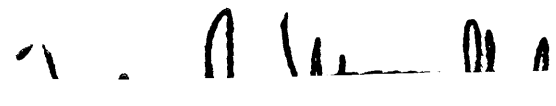
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1997

© Massachusetts Institute of Technology 1997. All rights reserved.

Author 
Department of Electrical Engineering and Computer Science
February 10, 1997

Certified by 
Louis D. Braid
Henry Ellis Warren Professor of Electrical Engineering
Thesis Supervisor

Accepted by 
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MAR 21 1997

Eng.

LIBRARIES

A Video Display System for an Automatic Cue Generator

by

Matthew G. Sexton

Submitted to the Department of Electrical Engineering and Computer Science
on February 10, 1997, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Deaf individuals who rely solely upon lipreading to understand the speech of others encounter difficulties because not all phonemes can be distinguished just by viewing the speaker's face. The system of Manual Cued Speech supplements lipreading by using the coded shape and position of the speaker's hand to disambiguate the phonemes appearing on his lips. The Automatic Cue Generator is a computer based system being developed by the Sensory Communications Group for automatically determining these cues from the speech waveform and presenting them to the lipreader.

This thesis describes the design and implementation of the video display system for the Automatic Cue Generator. The display system is responsible for overlaying a graphic image of a hand on the video image of a speaker given the cue sequence determined by an automatic speech recognizer. A time delayed video image of the speaker is used to accommodate the processing latency of the speech recognition. By delaying the video input from a camera for a period of two seconds in its internal buffer, the display system can synchronize the timing of the cue with the movement of the speaker's lips. The video image is then displayed on a conventional television monitor.

Thesis Supervisor: Louis D. Braida

Title: Henry Ellis Warren Professor of Electrical Engineering

Acknowledgments

I have had the great pleasure of collaborating on the Autocuer¹ project with Paul Duchnowski and David Lum, from whom I have learned the many pleasures of swaggering about the lab while wielding baseball bats and canes. While I can't say that these implements greatly advanced the progress of the project, they certainly made it more enjoyable. Of course, I did regularly pepper the both of them with questions on every aspect of the design of the system, and I am thankful for their great restraint in providing answers and not banishing me from their offices.

There were many others who provided invaluable assistance in the development of this thesis. In particular, Jeanie was always prepared to give me yet another photocopy of her T-shirt to add to the multitude scattered throughout my office, none of which I could ever seem to locate. And the whole seventh floor group certainly helped maintain my sanity with the never ending schedule of skiing, movies, and beer excursions.

Finally, I owe Lou Braida my sincere thanks. If it were not for his willingness to offer a summer job to a passerby in an elevator, I would probably still be lost somewhere in the stacks of MIT's libraries, cursing each book as I reshelved it. He allowed me to pursue those projects that I found interesting, and even saw fit to entrust me with this thesis. A large part of my career at MIT has been shaped by his guidance and I am grateful.

¹The rightful name of the project, and that by which I shall always refer to it—not the curmudgeonly Automatic Cue Generator.

Contents

1	Introduction	8
1.1	Cued Speech	8
1.2	Automatic Cueing Systems	9
1.3	Automatic Cue Generator	10
1.4	ACG System Design	11
2	Video System Design	13
2.1	Hardware Architecture	14
2.1.1	Video Acquisition	16
2.1.2	Computer Interconnect	16
2.1.3	Peripheral Interconnect	16
2.1.4	Video Digitization and Video Formats	17
2.1.5	Host Interface	17
2.1.6	FIFO Delay and Overlay	18
2.1.7	Display	18
2.1.8	NTSC Conversion	19
2.2	Software Architecture	19
3	Frame Grabber Kernel Module	21
3.1	Hardware Interface	21
3.2	Software Interface	22
3.3	<code>interface.c</code>	23
3.3.1	Memory Buffer	24

3.3.2	Interrupt Handling	24
3.3.3	System Call Handlers	25
3.4	bt819.c	27
3.5	saa7192.c	28
3.6	icd2061a.c	28
3.7	fpga.c	29
3.8	saa7116.c	29
3.9	iic.c	32
4	Display Output	34
4.1	Display Card	35
4.1.1	Hardware Interface	35
4.1.2	Software Interface	35
4.2	968.c	36
4.3	ti3026.c	37
4.4	NTSC Conversion	38
5	Control and Overlay	39
5.1	Real Time System Control	39
5.1.1	Synchronization	39
5.1.2	Fault Tolerance	40
5.2	Cue Overlay	41
5.3	Software Modules	42
5.3.1	acg.c	42
5.3.2	cue.c	45
5.3.3	timer.c	46
5.3.4	display.c	46
5.3.5	grab.c	47
5.3.6	socket.c	48
5.3.7	terminal.c	48

6 Discussion	49
6.1 Base System Improvements	49
6.1.1 Video Hardware	49
6.1.2 Computer Synchronization	50
6.1.3 Real Time Scheduling	50
6.1.4 Graphical Interface	51
6.2 Cueing Improvements	51
6.2.1 Cue Presentation	52
6.2.2 Automatic Cue Positioning	52
Bibliography	53

List of Figures

- 2-1 Automatic Cue Generator 14
- 2-2 Video Display System 15
- 2-3 Circular Delay Buffer 18
- 2-4 Software Module Organization 20

- 3-1 Frame Grabber Hardware Block Diagram 22
- 3-2 Frame Grabber Software Modules 23

- 4-1 Diamond Stealth Display Card 34

- 5-1 Display System Software Module Block Diagram 42
- 5-2 Main Control Loop 44

Chapter 1

Introduction

For hearing impaired individuals who derive little benefit from hearing aids, communication using spoken language is frustrating at best [12]. Even if a deaf individual successfully learns to speak, he or she still faces the difficult task of understanding the speech of others using only the visible action of the face as a guide. This difficulty arises from there often being several distinct speech elements that can be associated with a single visual pattern on the face. The single visual pattern provides no reliable way for the speechreader (lip-reader) to uniquely identify which of several possible sounds was spoken, and so the speechreader must instead use word patterns and contextual clues to help derive meaning. But even the most highly experienced speechreaders typically can only follow half of what is said in a conversation. And in the controlled experiments that are used to measure how much the speechreader actually understands, performance is often even lower [29].

1.1 Cued Speech

To resolve the ambiguity inherent in speechreading alone, Orin Cornett at Gallaudet University developed a “Manual Cued Speech” (MCS) system [8] in which the shape of the speaker’s hand and its position relative to his face provide supplementary information about the phonemes in his speech. Phonemes are the basic linguistic units of speech that are combined to form words. Sequential phonemes can be clustered

into consonant-vowel pairs that are partially described by the cues. Each hand shape represents a group of consonants, one of which is the consonant of the phoneme pair. A similar mapping is used for vowels, as indicated by the position of the hand. Each group of consonants or vowels can have several members because the members of a group are easily distinguished from one another by their appearance on the speaker's face, and so can be represented by the same hand shape or position. Phonemes that can not be distinguished visually are classified as a viseme [16], often called a visual phoneme. The phonemes of a particular viseme group, such as /p,b,m/, are placed into separate cue groups. Through the combination of speechreading and the reception of cues, all phonemes can be distinguished.

Unfortunately, using Manual Cued Speech requires that the speaker constantly move and reshape his hand as he speaks [20]. The hand produces eight shapes and moves through five main hand positions. In addition, diphthongs are signified by the movement of the hand between two of the main positions.

Experience has shown that few individuals other than family members undertake the task of learning to produce MCS. Therefore, a system for *automatically* producing the cues could be of great benefit for hearing impaired individuals. It would permit them to effectively communicate with any speaker of their language, without the need for a transliterator.

1.2 Automatic Cueing Systems

The implementation of an automatic system for cue generation has been considered in the past. Such a system is generally based on an Automatic Speech Recognizer (ASR) that uses the acoustic speech signal to determine which phoneme is being produced. Once the recognition is completed, the cue is presented visually to the deaf individual, to be used in conjunction with speechreading.

One attempt, from a group at Gallaudet University and the Research Triangle Institute [29] attempted to produce a wearable device that would supplement the speechreading of the live speaker. However, due to fundamental problems in several

aspects of the device, the Gallaudet-RTI group met with only limited success.

First, the recognizer had fairly poor performance. It detected only about two-thirds of the phonemes in words, and of those, it accurately identified only 54 percent.

Second, the manner in which cues were displayed was different from that of MCS. To meet the constraints of portability, a pair of seven-segment LEDs was used to present cues. In addition, the composition of the cue groups themselves were changed to better match the limitations of the recognizer.

Finally, the LED display was designed to be used as the cue receiver watched the live speaker's face. Since the ASR did not recognize a phoneme until it had been completely uttered, which is a limitation fundamental to any system, its output followed the facial actions in time. Therefore, a presented cue was delayed with respect to the lip movements to which it corresponded, and so the cue information did not necessarily correspond to what was seen on the lips. The effects of the desynchronization of the cue and lip movement information were investigated in a separate study by Bratakos [4], which showed that if the cue data were delayed by even 100 msec, the cue receiver would suffer significant reductions in his or her ability to understand the speech.

1.3 Automatic Cue Generator

The Sensory Communications Group at MIT has sought to address many of these problems in its realization of an Automatic Cue Generator (ACG). The ASR system, using a Hidden Markov Modeling system based on a modified version the HTK Toolkit of Entropic Labs [13], has achieved recognition rates of up to 80% for off-line processing, and 70% in real-time. With sufficiently fast computation, it is anticipated that the 80% mark will be achievable in real-time as well. Bratakos [4] has shown that cues produced by a recognizer with 80% accuracy can provide a significant improvement in speech reception relative to speechreading alone.

In order to resolve the problems encountered by the Gallaudet-RTI group with the presentation of the cues, the cue receiver will watch a video image of the speaker

on a monitor, instead of watching the speaker himself. The use of video permits the introduction a time delay into the video stream. By delaying the video image of the speaker, sufficient time is allowed for the speech recognition to be performed, and the cue and the speaker's image to be presented in synchrony. The delay fundamentally must be at least the length of the longest frequently occurring consonant-vowel pair, typically about three-quarters of a second. A delay of one to two seconds is used in practice to allow for recognizer overhead, and should not be more than a minor impediment to conversation. In a setting such as a classroom lecture, there should be little effect at all.

Using a video image has the added benefit that the cues that form the basis of the manual system can still be used. Through a video overlay process, pictures of the various hand shapes can simply be drawn on the video image of the speaker in the appropriate position in response to the output of the recognizer. Using the natural cueing system will allow experienced cue receivers to use the automatic system without any additional training.

1.4 ACG System Design

Several aspects of the design of the Automatic Cue Generator were constrained by the availability of appropriate hardware and software platforms. The Automatic Speech Recognition software package that provided the requisite performance was based upon a modified version of the HTK Toolkit that ran on a DEC Alpha workstation under the Digital Unix operating system. The HTK software running on the Alpha was, at the time, the most readily available system that was stable and supported and could provide real-time recognition with an acceptable level of accuracy. In an effort to reserve the Alpha's computational power for the recognition proper, the audio capture and preprocessing was performed on an Ariel DSP-96 Digital Signal Processing (DSP) board [2]. The board digitized the incoming analog audio signal and parameterized the speech waveform for recognition. The DSP-96 functioned as a plug-in peripheral for an Intel processor based workstation running the Linux operating system. As this

DSP setup had been used in many other applications in the Sensory Communication Group, it had a large body of development tools available [26, 18], and so could be relatively quickly and easily incorporated into the system.

This document describes the design and implementation of the video display system of the Automatic Cue Generator of the Sensory Communications Group. Technical descriptions of the project at both the hardware and software level are given. In addition, recommended work for the future is discussed.

Chapter 2

Video System Design

As discussed in the preceding chapter, the video system had to meet several performance criteria in order for the Automatic Cue Generator to properly synchronize the cues with the facial movements of the speaker. The minimal video system had to provide a fixed video delay of up to two seconds, and in response to directives from the ASR, overlay on each video image a picture of a hand of a particular shape and position. Unfortunately, there were no systems that performed a similar function upon which it could be modeled. Application specific hardware systems only existed for implementing one or two of the necessary tasks and were difficult to interface with one another. A video delay line [23], for example, could provide the delay, but offered no dynamic controllability and no access to the video frames before the end of the delay period.

Therefore, a new system was designed, with a general purpose computer as the platform on which all video handling and processing would take place. A large variety of configurations and peripherals would be available to modify and augment the system, more so than for a hardware system designed specifically for the display. While a dedicated system might be more efficient, it would not be as flexible as a programmable computer.

Figure 2-1 shows the role of the cue display system in the overall Automatic Cue Generator design. A camera trained on the speaker provides input to the video and audio preprocessing computers. The audio system digitizes and processes the audio

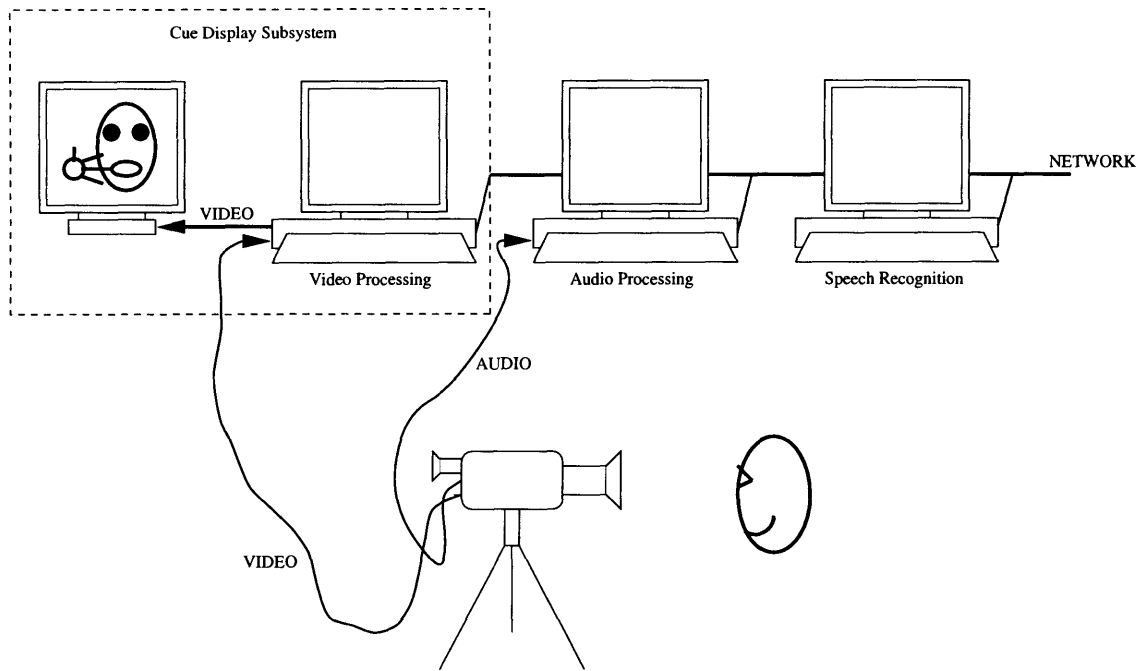


Figure 2-1: Automatic Cue Generator

waveform and sends its data over the network to the speech recognition computer. Once the speech recognizer has identified a cue using that data, it sends the cue information over the network to the cue display system. The display system uses that information to overlay cues on the video frames that it has stored in its delay buffer. The modified video is displayed on a television monitor.

2.1 Hardware Architecture

Both the input and output video images had to be in NTSC format [9] for compatibility with standard video hardware such as cameras, monitors, and recording devices. In NTSC video, the standard format for North America, Japan, and much of South America, 60 fields of video are transmitted per second; however, an individual field of video does not represent an entire image. Instead, fields are generated by interlaced scanning; fields contain either all even lines or all odd lines of an image. Odd fields and even fields are alternately scanned and transmitted. Since consecutive

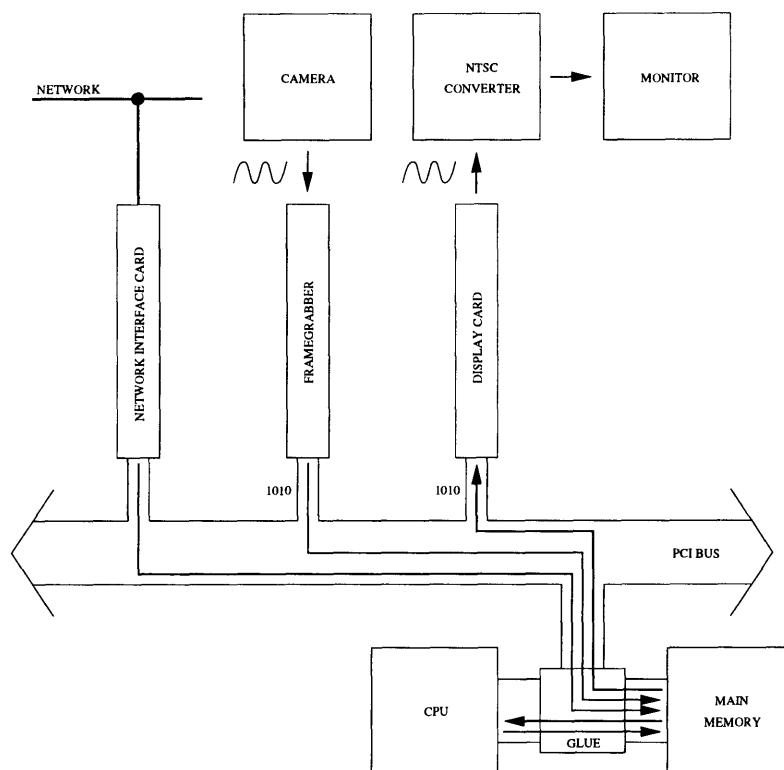


Figure 2-2: Video Display System

fields can be combined, or 'de-interlaced,' to form complete images, NTSC signals are also described as delivering 30 frames per second. This description is somewhat inaccurate because displaying consecutive fields at the same time would introduce unwanted motion artifacts and is not done in practice.

A block diagram of the video display system is presented in Figure 2-2, which also shows the paths of video and network data flow. The use of a general purpose computer required that the computer be outfitted with specialized hardware designed for use with the high bandwidth traffic of continuous video streams. Essentially, what the frame grabber and display card components do in combination with the computer's standard hardware is to capture each video field in a digital format, store it in memory for the delay period, and then reconvert it to an NTSC signal for display. While a frame is in memory, the CPU receives a message about the cue from the ASR computer over their network connection, and superimposes the appropriate cue on the image.

2.1.1 Video Acquisition

Any standard video camera that produces an NTSC video signal can be used to monitor the speaker. The Sony DCR-VX1000 Digital Video Camera Recorder [27] was selected on the basis of its exceptional picture quality and its ability to record images and sound on digital tape. The recordings could then be used at a later time to recreate the speaker's visual and audio input to the system.

2.1.2 Computer Interconnect

The network connection was the standard 10 Mbps Ethernet cabling that served the Sensory Communications Group for its general computing purposes. The rate of data exchange between the three computers is sufficiently low that interference by other network traffic is not a serious concern. The audio preprocessing system transmits about 10 KB of data per second, and the ASR computer only sends about 100 bytes per second.

2.1.3 Peripheral Interconnect

Since video sources can easily produce 35 MB of data per second, a high speed bus for connecting various peripheral and system components was essential. Further, as can be observed in Figure 2-2, each video datum travels on the bus twice, once on its way into main memory, and a second time on its way from main memory to the display card after the delay period. Thus, the bandwidth requirements for the hardware in a particular video mode are doubled, as there are effectively two separate video streams over the same bus.

The 32-bit wide Peripheral Connect Interface (PCI) bus [19] met this need for high bandwidth, as it offered continuous data transfer rates of up to 120 MB/sec. Just as important, peripherals using the PCI bus can use a very simple and powerful Direct Memory Access (DMA) protocol to exchange blocks of memory with one another and with main memory, without the need for intervention by the CPU. This feature is essential in relieving the CPU of the burden of transferring data throughout the

system.

2.1.4 Video Digitization and Video Formats

In order for the computer to be able to accept a video signal as input, it must first be converted into a digital format. Since no hardware existed for accessing the digital data produced by the camera, a frame grabber performs the digitization for the computer, by sampling an NTSC video signal into digital frames at a specified resolution and color depth, and making them available on the system bus for transfer to main memory. The maximum resolution typically supported by frame grabbers, and the minimum necessary to preserve adequate image quality, is 640 horizontal pixels by 480 vertical pixels.

The available color depth is defined by how many bits are used for each component of the Red-Blue-Green coding of a pixel's color. There are two popular pixel formats for encoding the color information. The 555 RGB format uses 5 bits for each color component, and so a two byte word can hold the fifteen bits of video data for each pixel. The 888 RGB format uses a four byte word¹ for its 24 bits of pixel data. In a 640 by 480 image, these two formats require 600 KB and 1200 KB per frame, respectively. And in a continuous video stream, more than 17 MB and 35 MB must be transferred each second.

The Oculus TC*i* frame grabber from Coreco Inc. [7] provided the necessary resolution and both color depths. In addition, it was able to utilize the full bandwidth of the PCI bus, independently transferring frames into memory at up to 120 MB/sec through the use of its DMA controllers.

2.1.5 Host Interface

The 440FX PCI chip set provides the interface between the PCI bus, system memory, and the Intel Pentium Pro processor CPU. As illustrated in Figure 2-2, the 440FX accelerator and controller chips, labelled as "GLUE," allow for very high bandwidth

¹The three bytes of data are padded with a null byte to enable pixel word alignment.

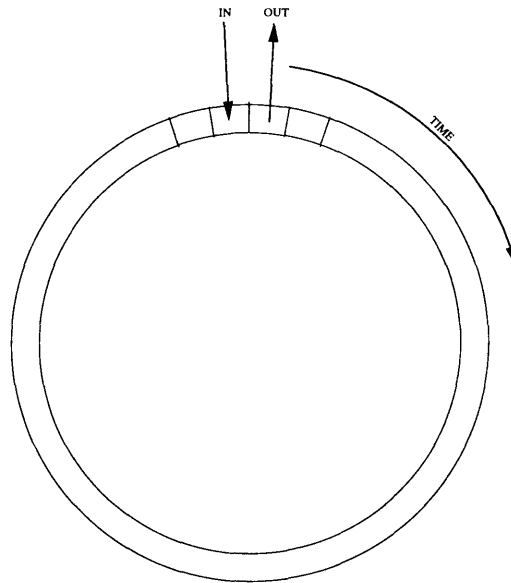


Figure 2-3: Circular Delay Buffer

access to memory through a 64-bit, 66 MHz host bus. Such high bandwidth permits concurrent access to memory for image storage, modification, and retrieval.

2.1.6 FIFO Delay and Overlay

The two seconds of video delay is accomplished through the use of a circular buffer in the computer's main memory, as shown in Figure 2-3. The buffer is sufficiently large that it can hold the 60 frames, or 120 fields, in two seconds of video. A frame is stored in a element of the buffer for the specified delay period, overlaid with a hand shape, and removed for display. That element of the buffer then immediately receives another frame for delay.

2.1.7 Display

A standard computer display card produces the analog video signal from the stream of digital video data from the buffer in memory. A digital image is transferred into the video card's on-board frame buffer memory, from which it is converted into a VGA format [14] analog signal appropriate for display on a VGA computer monitor.

As different manufacturers' video cards were considered, it became apparent that whichever card was chosen, it would be the most limiting factor in the performance of the entire video display system. The card used, the Stealth 64 Video VRAM from Diamond Multimedia Systems, Inc. [11], had no DMA controller, forcing the CPU to perform all data transfers to the card. Furthermore, even though it had the fastest bus interface available, the CPU could only transfer data into its display buffer at 33 MB/sec. Such a low transfer rate meant that 555 RGB was the best color depth that could be supported, as the transfer of frames from memory to the display card would even then take up more than half the PCI bus bandwidth.

Fortunately, the much faster transfer rate of the frame grabber enabled the CPU to accomplish its transfer to the display card despite the disproportionately long transfer time. The following calculations show the allocation of the bus over the course of one second for a 640x480 video stream, with two bytes per pixel:

$$\begin{array}{rcl}
 \text{CPU to Display Card :} & (640 * 480 * 30 * 2)\text{bytes} * \frac{1 \text{ sec}}{33 \text{ MB}} & = .53\text{sec} \\
 \text{Frame Grabber to RAM :} & (640 * 480 * 30 * 2)\text{bytes} * \frac{1 \text{ sec}}{120 \text{ MB}} & = .15\text{sec} \\
 & & \hline
 & & .68\text{sec}
 \end{array}$$

The frame grabber is so fast that, in total, the bus is in use for only .68 seconds of that one second period. Clearly, with only 68% of the bus bandwidth taken, the functioning of the system is possible.

2.1.8 NTSC Conversion

An output video signal capable of being displayed on a standard television monitor and recorded by a conventional VCR clearly adds desirable flexibility to the system. Therefore, a scan converter from AITech International [1] is used to convert the VGA signal into an NTSC signal. A scan converter is a stand-alone hardware product and requires no programming or special hardware accommodations in the rest of the system.

2.2 Software Architecture

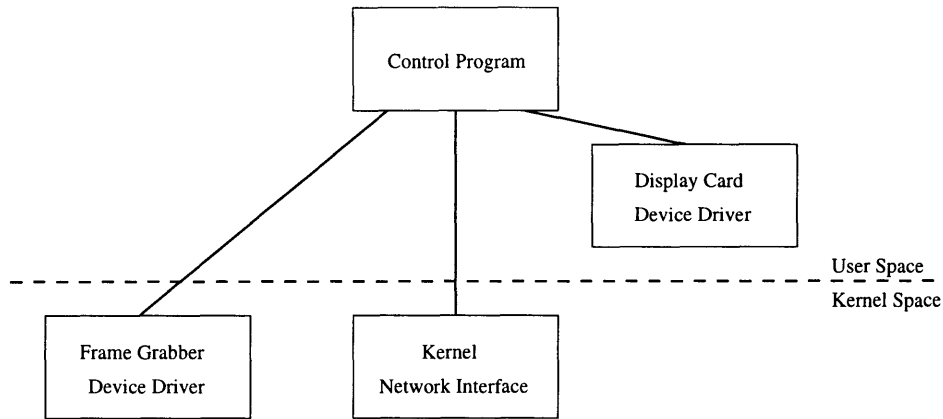


Figure 2-4: Software Module Organization

The flexibility of the hardware was significantly augmented by the use of the Linux operating system, a freely distributed version of Unix. The source code of the Linux kernel is publicly available, which with technical information on a particular piece of hardware, permits the coding of a device driver for it. As neither Coreco nor Diamond supplied Linux drivers for their products, this feature was essential for incorporating their devices into the video display system.

Since Linux is a version of the Unix operating system, a very structured and well defined environment existed for the development and integration of the controlling software with the device drivers. Figure 2-4 shows the relationship between the several software modules under Linux. At the top is the controlling software, which interacts with the user, manages the hardware interfaces, and performs the overlay of cues on the video images. The controlling software primarily relies upon the kernel level device driver controlling the frame grabber, the user space driver for the display card, and the kernel's network interface for communicating with the other computers of the Automatic Cue Generating system. The details of the implementation of the software modules are discussed in the succeeding chapters.

Chapter 3

Frame Grabber Kernel Module

The integration of the Oculus TC*i* frame grabber from Coreco Inc. into the video display system for the Automatic Cue Generator required that a device driver be written for it under the Linux operating system. This chapter provides a background on the structure of the board and a description of the software module that was written to control it.

3.1 Hardware Interface

The Oculus TC*i* frame grabber is a peripheral card for Intel x86 based computer systems that plugs into an expansion slot on the PCI bus. It accepts an NTSC signal for digitization and then uses its DMA controllers to transfer the image data into main memory over the system PCI bus. As shown in Figure 3-1, the board also has its own PCI bus, which connects to the system bus through a bridge chip. Because the two buses can operate independently of one another, the frame grabber board has the capability of transferring data to components attached to its secondary bus, such as memory or processors, without affecting system bus activity. This feature is not utilized in the display system because appropriate components are not available and because the PCI bridge chip is incapable of supporting certain simultaneous transfers

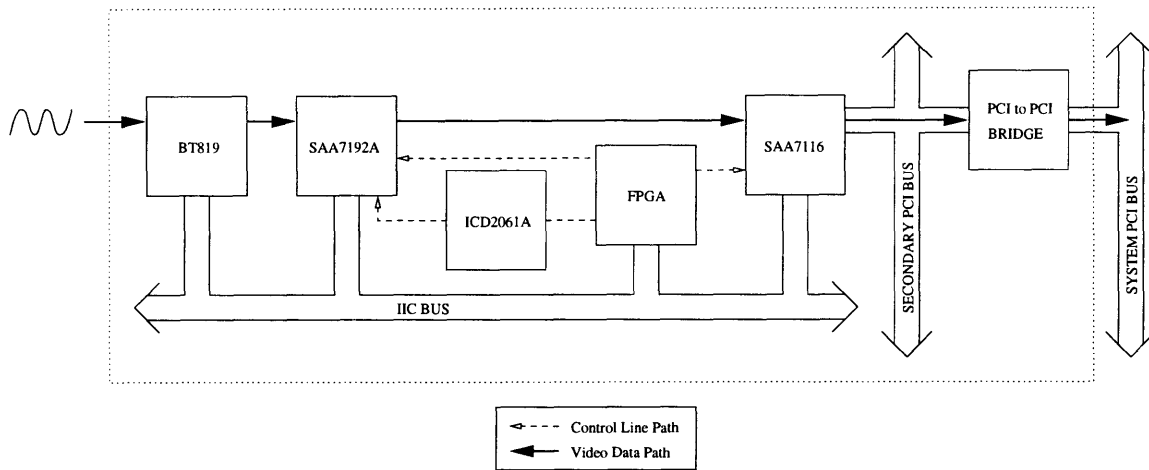


Figure 3-1: Frame Grabber Hardware Block Diagram

across it.¹

The frame grabber interfaces with the PCI bus through the SAA7116 chip [21], which is responsible for transferring the digital video frames into memory. The registers of the SAA7116 are mapped into the address space of the computer so that it can be easily controlled by a program. The SAA7116 in turn controls several other video hardware components through the Inter-Integrated Circuit (I²C) interface [15], a two-line serial bus and protocol originally developed by Philips for the company's television video chips. The Bt819 and the SAA7192A, which digitize and color adapt the video, respectively, are both on the I²C bus, as is the Field Programmable Gate Array (FPGA) that provides additional control over these hardware elements. The ICD2061A is a programmable clock chip that provides timing information for the SAA7192A.

3.2 Software Interface

The device driver for the Oculus TC*i* frame grabber was implemented as a loadable module for the Linux 2.0.x kernel series [3]. Kernel modules can be dynamically

¹The bridge chip does not arbitrate between a peripheral device and the CPU when both attempt to transfer data over it. Instead, it terminates the data transfer of the device and grants unrestricted access to the CPU.

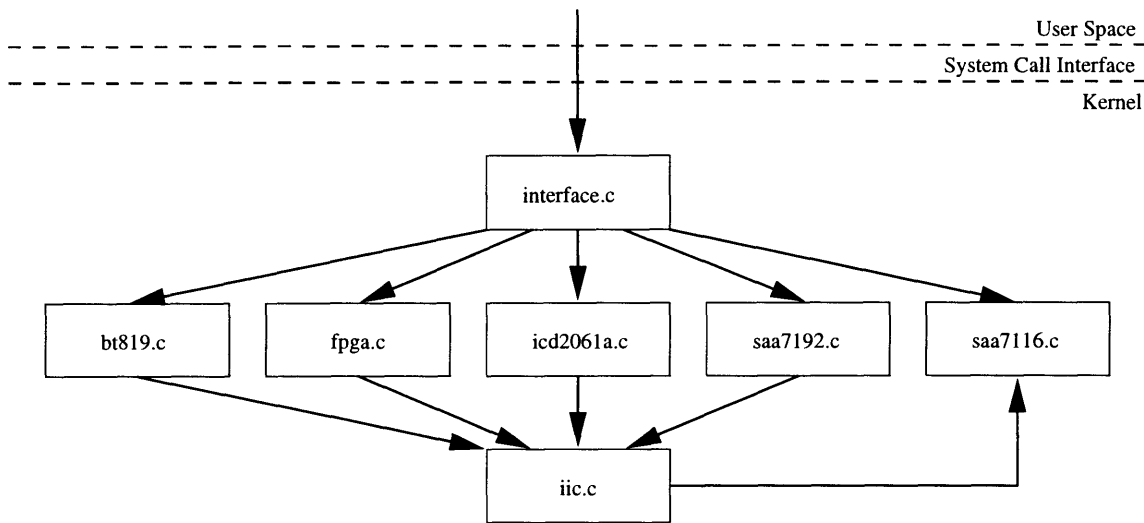


Figure 3-2: Frame Grabber Software Modules

inserted into the running kernel, permitting quicker development and debugging because the system does not have to be rebooted with each revision. Within the kernel module source code, related functions were grouped together into separate files, each typically named for the hardware chip that they control. The calling hierarchy of these software modules is shown in Figure 3-2. Each of the following sections discusses the functioning of an individual source file and documents in detail its external functions.

3.3 interface.c

The standard Unix system calls by which a user program controls the frame grabber through its special device file are implemented in routines defined in `interface.c`. Under the direction of commands from the user program, these functions control the hardware components and interact with the various kernel features for resource acquisition and management.

3.3.1 Memory Buffer

The kernel module allocates the memory for the FIFO video delay buffer. As discussed in Subsection 2.1.6, the buffer is circular, so that its store and remove pointers are incremented sequentially through all the elements and wrap from the end to the beginning. The elements are indexed so that they may be easily referenced in commands issued by the user program.

Each buffer element requires 600 KB of memory to store the 640 by 480 pixel images at a 15 bit color depth. For use with the Oculus board's DMA controller, the memory must be physically contiguous, but the kernel's standard contiguous memory allocation function, `kmalloc()`, cannot allocate more than 128 KB. Therefore, Matt Walsh's `bigphysarea()` function was patched into the kernel. This function reserves a block of contiguous memory of a specified size at system boot-up, so that it cannot be allocated by any other kernel routines.

Memory allocation is further complicated by another limitation of the board's DMA controller (see the SAA7116 section below). Its range checking feature, which detects if the frame grabber attempts to write to an address above the specified image transfer window in memory, uses only 22 bits² to specify the end of the DMA transfer window; the upper 10 bits are taken from the register that specifies the start address of the image frame. Because the upper 10 bits of the start and stop addresses are the same, the image frame must not cross a 4 megabyte (2^{22}) address boundary in memory. Therefore, in a 4 megabyte region of memory, the six frames that fully fit use only 3.5 megabytes, resulting in a significant waste of memory. Thus, the 60 frames of buffer that implement a two second delay require the allocation of more than 42 MB of memory, of which only 35 MB is actually used for video.

3.3.2 Interrupt Handling

After each video frame is transferred into memory, the Oculus board generates a hardware interrupt, which is handled by a function in `interface.c`. In the interrupt

²The SAA7116 data sheet states that 24 bits are used, but that claim was not born out in practice.

service routine (ISR), the address registers describing where the DMA controller will store the next video frame are updated to point to the next element in the delay buffer. The ISR also checks to see if the transfer of the captured video frame was prematurely terminated due to the DMA controller having insufficient time on the PCI bus. The DMA controller uses only a 1 KB FIFO to buffer the live digital data from the PCI bus. If the FIFO fills before access to the bus is granted, image data is lost and a field corruption flag is set. This flag's setting causes `ioctl()` to return an error for the `WAIT_FOR_FRAME` command (see Subsection 3.3.3).

3.3.3 System Call Handlers

The system calls for controlling the frame grabber, `open()`, `close()`, and `ioctl()`, are routed to the following three routines defined in the kernel module.

```
static int tci_open(struct inode *inode, struct file *file);
```

The system command `open()` passes control on to `tci_open()` in order to allocate system resources, such as memory and interrupts, for each module and to call its initialization routine. The function `tci_open` also sets up a driver information data structure that is stored between calls to record the state of the device. One element of this structure, the `tci_info` structure, points to the memory allocated for the other modules; its address is passed as an argument in all calls to the other modules.

```
static void tci_release(struct inode *inode, struct file *file);
```

This routine, called by the system command `close()`, frees those resources that have been acquired and ensures that the board is no longer capturing video frames.

```
static int tci_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg);
```

The bulk of the interface with the user space program is accomplished through the `ioctl()` command, which in turn calls `tci_ioctl()`. The arguments `cmd`

and arg are specified by the user program in the call to `ioctl()`, and determine which function is performed and any additional parameters, respectively. A call to `ioctl()` takes the form

$$\text{ioctl}(\underline{fd}, \underline{cmd}, \underline{arg})$$

The complete list of `ioctl()` cmd codes, which are defined in the header file `tci.h`, are as follows:

`TCI_GET_BUFFER_ADDRESS` Get the kernel address for the image frame in the buffer that has index *arg, and store it in arg.

`TCI_SET_PIXEL_STYLE` Set the pixel style to be either `RGB_555`, `RGB_888`, or `YUV_555`.

`TCI_SET_H_CAPTURE` Set the horizontal resolution of images to be captured to be arg pixels.

`TCI_SET_V_CAPTURE` Set the vertical resolution of images to be captured to be arg pixels. If the resolution is set to be no greater than 240 pixels, then only the even fields of the video are used, with half the scaling as for using both even and odd fields.

`TCI_SET_DELAY` Set the number of images in the video delay buffer to be arg.

`TCI_GRAB_ONE` Grab one frame of video into memory, and copy it to the buffer beginning at arg.

`TCI_GRAB_ALL` Continuously grab video frames into the delay buffer.

`TCI_WAIT_FOR_FRAME` While doing a continuous grab of video frames, block until the current frame is completely captured, and return its buffer index in arg.

`TCI_STOP_GRAB` Stop grabbing video frames.

`TCI_SET_COMPOSITE_VIDEO_MODE` Set the system to digitize a composite video source.

`TCI_SET_YC_VIDEO_MODE` Set the system to digitize an S-Video source.

`TCI_SET_CAMERA_0` Choose camera input 0.

`TCI_SET_CAMERA_1` Choose camera input 1.

`TCI_SET_CAMERA_2` Choose camera input 2.

3.4 bt819.c

The Bt819 Video Capture Processor [6, 5] performs the actual digitization of the video signal and is able to vary a number of parameters for the captured image. All parameters for a particular video mode must be set explicitly; the default values are undefined. The macros mentioned in the descriptions are defined in the header file `bt819.h`.

```
void init_BT819_registers(struct tci_info *tci);
```

The Bt819 chip is reset into its default state.

```
void BT819_set_video_mode(struct tci_info *tci, int format);
```

While the Bt819 can accept both NTSC and PAL signals, only NTSC is currently implemented as a `format`.

```
void BT819_set_camera(struct tci_info *tci, int camera);
```

The `camera` type is either `YC` for S-Video or `COMPOSITE` for composite.

```
void BT819_set_mux(struct tci_info *tci, int source);
```

The Bt819 has three different camera inputs, of which the first two are of interest. Video `source CAMERA_0` is used for composite video input and `CAMERA_1` is used for S-Video input.

```
void BT819_set_h_scale(struct tci_info *tci, int resolution, int format);
```

The horizontal resolution of a capture image will be `resolution` pixels. `format` must be NTSC.

```
void BT819_set_v_scale(struct tci_info *tci, int resolution, int format);
```

The vertical resolution of a capture image will be `resolution` pixels. `format` must be NTSC.

```
void BT819_set_h_crop(struct tci_info *tci, int start, int active, int format);
```

The image, after scaling as specified in `BT819_set_h_scale()` will be cropped in the horizontal direction from the image starting at pixel `start` and extending for `active` pixels. `format` must be NTSC.

```
void BT819_set_v_crop(struct tci_info *tci, int start, int active, int format);
```

The image, after scaling as specified in `BT819_set_v_scale()` will be cropped in the vertical direction from the image starting at pixel `start` and extending for `active` pixels. `format` must be NTSC.

3.5 saa7192.c

The SAA7192A Digital Color Space Converter [22] transforms the 16-bit YCrCb format digital data from the Bt819 into either a 24-bit RGB format or a 16-bit YUV format. In this application the data is transformed into the RGB format for compatibility with the display card. The Color Space Converter also provides multiplicative Look-Up Tables (LUTs) for each color component in order to effect gamma correction. This feature, however, is disabled (see Section 3.7).

```
void init_SAA7192_registers(struct tci_info *tci);
```

The video chip is initialized to RGB output mode.

```
void SAA7192_set_mode(struct tci_info *tci, int mode);
```

For use with the Bt819 data, `mode` must be either `F422_M` or `F422_NM`. The first enables the matrix that converts its input to RGB, and the second disables it (for YUV output).

3.6 icd2061a.c

The SAA7192A Converter chip requires an external clocking signal to set the rate at which it should acquire its data. Because images can be captured at different horizontal resolutions, the rate at which the Bt819 produces valid data is variable, and so the ICD2061A Dual Programmable Graphics Clock Generator [10] is used to synchronize the two chips.

```
void init_ICD2061A_registers(struct tci_info *tci);
```

The clock is set to `NTSC_SQ_FREQ` MHz, for use with full resolution images.

```
void ICD2061A_set_clock(struct tci_info *tci, double freq);
```

The clock is set to produce a frequency of freq MHz. Typically, this value is determined as a fraction of `NTSC_SQ_FREQ`, in which the multiplier of the standard is equal to the ratio of the horizontal resolution to 640, i.e.:

$$\underline{freq} = NTSC_SQ_FREQ * \underline{horiz_resolution} / 640$$

3.7 fpga.c

A Field Programmable Gate Array (FPGA) contains several registers that are used to program the clock chip and to provide additional input to the SAA7116.

```
void init_FPGA_registers(struct tci_info *tci);
```

The FPGA is initialized to default values.

```
void FPGA_set_pixel_line_length(struct tci_info *tci, int length);
```

In order to do its DMA address calculations, the SAA7116 PCI interface chip must know the length of each scan line in the image. This value is set by the function to length, as measured in pixels.

```
void FPGA_set_lut_use(struct tci_info *tci, BOOL use);
```

Whether or not the SAA7192 uses its color Look Up Tables (LUTs) to modify the video signal is determined by the truth value of use.

3.8 saa7116.c

All interaction between the host computer and the TC*i* frame grabber is mediated by the SAA7116 Digital Video to PCI Interface chip [21]. By mapping its registers into the computer's address space, the SAA7116 provides the low level interface to the I²C bus on which `iic.c` is built (see Section 3.9). This register mapping also provides access to the DMA controllers for transferring captured video frames into memory. The video frames that the SAA7116 receives from the SAA7192 are interlaced, meaning

that it receives separate even and odd fields. The SAA7116 is programmed to store consecutive even and odd fields together in memory as complete images to permit their easier overlay and manipulation.

```
void init_SAA7116_registers(struct tci_info *tci);
```

Prepare the SAA7116's data FIFO, initialize the I²C bus, and set up other defaults.

```
void SAA7116_set_DMA_E_address(struct tci_info *tci, unsigned int address);
```

Set the destination in memory of the even field to be address.

```
void SAA7116_set_DMA_O_address(struct tci_info *tci, unsigned int address);
```

Set the destination in memory of the odd field to be address.

```
void SAA7116_set_DMA_E_end(struct tci_info *tci, unsigned int address);
```

Sets the highest address in memory to which the even field can be transferred to be address.

```
void SAA7116_set_DMA_O_end(struct tci_info *tci, unsigned int address);
```

Sets the highest address in memory to which the odd field can be transferred to be address.

```
void SAA7116_set_DMA_E_stride(struct tci_info *tci, unsigned int stride);
```

The transfer address is incremented by stride at the end of each line of the even field.

```
void SAA7116_set_DMA_O_stride(struct tci_info *tci, unsigned int stride);
```

The transfer address is incremented by stride at the end of each line of the odd field.

```
void SAA7116_grab_one_even(struct tci_info *tci);
```

Grab one even field of video into memory.

```
void SAA7116_grab_one_odd(struct tci_info *tci);
```

Grab one odd field of video into memory.

```
void SAA7116_grab_even(struct tci_info *tci);
```

Continuously grab even fields of video into memory.

```
void SAA7116_grab_odd(struct tci_info *tci);
```

Continuously grab odd fields of video into memory.

```
void SAA7116_reset_int(struct tci_info *tci);
```

Reset the interrupt generating circuitry after an interrupt has been received to prepare for the next one.

```
int SAA7116_check_int_E(struct tci_info *tci);
```

Return true if an interrupt was triggered for the end of the even field.

```
int SAA7116_check_int_O(struct tci_info *tci);
```

Return true if an interrupt was triggered for the end of the odd field.

```
int SAA7116_check_corrupt_E(struct tci_info *tci);
```

Return true if the even field transfer was aborted due to a FIFO overflow.

```
int SAA7116_check_corrupt_O(struct tci_info *tci);
```

Return true if the odd field transfer was aborted due to a FIFO overflow.

```
void SAA7116_stop_int(struct tci_info *tci);
```

Stop generating any interrupts.

```
void SAA7116_int_even(struct tci_info *tci);
```

Generate an interrupt at the end of a captured even field.

```
void SAA7116_int_odd(struct tci_info *tci);
```

Generate an interrupt at the end of a captured odd field.

```
void SAA7116_int_start(struct tci_info *tci);
```

Generate an interrupt at the beginning of all fields, captured or not.

```
void SAA7116_stop_grab(struct tci_info *tci);
```

Stop transferring fields to memory.

```
void SAA7116_set_pixel_mode(struct tci_info *tci, int mode);
```

Set the pixel mode to be either RGB24pa, RGB16pa, or YUV16pa.

```
void SAA7116_put_SDA(struct tci_info *tci, BOOL bit);
```

Set the SDA (data) line of the I²C bus to be bit.

```
void SAA7116_put_SCL(struct tci_info *tci, BOOL bit);
```

Set the SCL (clock) line of the I²C bus to be bit.

```
BOOL SAA7116_get_SDA(struct tci_info *tci);
```

Return the value on the SDA (data) line of the I²C bus.

```
BOOL SAA7116_get_SCL(struct tci_info * tci);
```

Return the value on the SCL (clock) line of the I²C bus.

```
void SAA7116_raise_SDA(struct tci_info *tci);
```

Set the value of the SDA (data) line of the I²C bus to be one.

```
void SAA7116_lower_SDA(struct tci_info *tci);
```

Set the value of the SDA (data) line of the I²C bus to be zero.

```
void SAA7116_raise_SCL(struct tci_info *tci);
```

Set the value of the SCL (clock) line of the I²C bus to be one.

```
void SAA7116_lower_SCL(struct tci_info *tci);
```

Set the value of the SCL (clock) line of the I²C bus to be zero.

3.9 iic.c

The I²C bus interface [15] is implemented on top of the functionality of the SAA7116 chip. The bus protocol is used to program the Bt819, SAA7192, and FPGA chips.

```
void I2C_read(struct tci_info *tci, unsigned char address, unsigned char  
*bytes, int length);
```


An array of character sized data of length length is read from the device with address address on the I²C bus and written to memory beginning at address bytes.

```
void I2C_write(struct tci_info *tci, unsigned char address, unsigned char  
*bytes, int length);
```

An array of character sized data starting at bytes, and of length length, is written to the device with address address on the I²C bus.

Chapter 4

Display Output

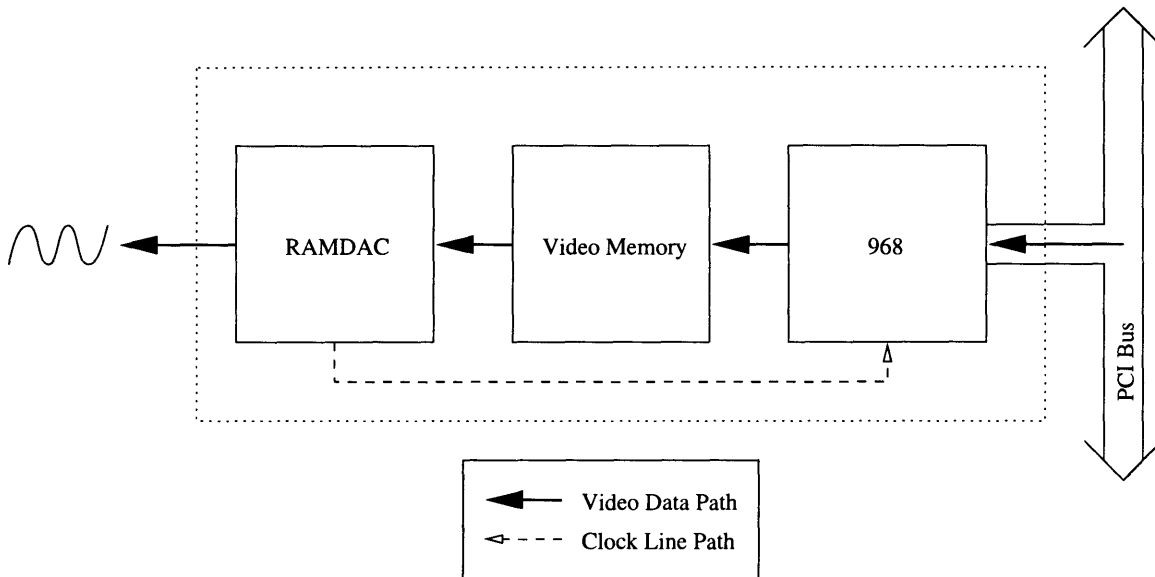


Figure 4-1: Diamond Stealth Display Card

The images that are produced through the overlay of the cues on the captured and delayed video are presented for viewing by the cue receiver on a television monitor. The conversion of those images from digital format in memory to an analog NTSC signal is accomplished through the combination of a computer display card and a VGA to NTSC scan converter.

4.1 Display Card

4.1.1 Hardware Interface

The Stealth 64 Video VRAM (Diamond Multimedia Systems) is a conventional VGA computer display card. It transforms the digital image in its frame buffer into an analog signal for display on a VGA computer monitor. As shown in Figure 4-1, there are three main hardware components on the Stealth board. The first is the Vision968 Multimedia Accelerator chip [24, 25], which interfaces the board with the PCI bus and provides the programming registers by which the various display modes are set. The second is the two megabytes of video memory (VRAM), which is accessed through the 968 and which stores an image for display. Finally, there is the RAMDAC (RAM Digital to Analog Converter), a Texas Instruments TVP3026 [28] that converts the image in VRAM into an analog signal. The TVP3026 reads the words of image data sequentially from VRAM, looping back to the beginning at the end of the image. The number of times the TVP3026 reads an entire image from memory each second is its “refresh rate.” In addition, the TVP3026 provides timing clocks for the 968.

4.1.2 Software Interface

Unlike the kernel module device driver for the Oculus TC*i*, the driver for the Diamond Stealth was implemented as a user space library [17]. The driver does not use DMA or interrupts and so requires no special functionality that is available only at the kernel level. Two methodologies for programming the 968 chip are available: addressing the I/O address space of the Intel x86, and mapping the registers of the 968 into the normal address space of the computer. The second method is used, as it does not require special I/O commands and is faster and more efficient. The 968, in turn, provides access to the programming registers of the TVP3026. However, both methods require that a calling process run with super-user permissions in order to modify the relevant memory locations.

The display card programming is similar to that for video modes in X Windows,

except that special provisions are made for the display of continuous video. Because the RAMDAC is almost always accessing the current image in memory, merely overwriting the image with the next video frame would likely introduce errors into the display as incomplete images are read for output. Therefore, two buffers are reserved for video frames in the Diamond Stealth card's memory. While one buffer is being used for the output, the other is being filled with the next frame. After the frame transfer is complete, the roles of the two buffers are switched, with the recently filled buffer now used for output and the other for the transfer.

The switch between the two buffers is made by reprogramming three Start Address registers [24, pp 14(17), 17(16)], whose contents are concatenated to form the starting address in the VRAM from which the image should be read. But because all three cannot be modified in one atomic operation, and the registers are read only as the RAMDAC cycles to the beginning of the image, an incorrect address could be read and used for an entire image. Therefore, the beginning of the buffers in VRAM must be aligned so that only one Start Address register need be changed to switch between buffers. In this way, the change is effectively atomic.

The RAMDAC is configured for a refresh rate of 30 frames per second, so that the rate at which images are written to VRAM is the same as the rate at which they are read. Otherwise, the length of time for which a frame would be displayed could vary.

4.2 968.c

All calls to functions controlling the display board include a `video_mode` structure as an argument. The elements of this structure, initialized by the user before the call to `setup_video_display()`, determine the resolution of the new video mode and its refresh rate.

```
int setup_video_display(struct video_mode *mode);
```

Put the display card into the video mode described in `mode`.

```
int clean_video_display(struct video_mode *mode);
```

Restore the display board to its original state and free all resources.

```
int use_screen(struct video_mode *mode, int screen);
```

Select the buffer in the display board's memory from which the display image should be read. screen is an integer from 0 to n, with the upper limit determined by the amount of memory and the video mode.

```
int draw_screen(struct video_mode *mode, void *image, int screen);
```

Copy image to the buffer in the display board's memory selected by screen.

4.3 ti3026.c

These functions control the TVP3026 RAMDAC, which is programmed indirectly through registers on the 968. The TVP3026 provides the clocking signals for the 968's timing of memory access, pixel processing, and data transfer. It also reads pixel data from the frame buffer and converts it to an analog format for display. These functions are only called by the 968.c software module.

```
int ti_save_registers(void);
```

Save the current state of the programming registers.

```
int ti_restore_registers(void);
```

Restore the saved state of the programming registers.

```
int ti_setup_clock(struct video_mode *mode);
```

Program the internal phase lock loop so that the correct clocking signal is sent to the 968.

```
int ti_setup_pixels(struct video_mode *mode);
```

Setup the output circuitry for the appropriate conversion of pixels to analog levels.

4.4 NTSC Conversion

The scan converter from AITech International that is used to transform the VGA format output of the Diamond card into NTSC format video requires that the VGA signal correspond to a 640 by 480 pixel image refreshed at 30 frames per second, consistent with the video mode of the Diamond Stealth display card. Thus, no adjustments to the video display mode were necessary.

Chapter 5

Control and Overlay

A user program is responsible for the overall control of the video display system for the Automatic Cue Generator. It manages the hardware drivers, communicates with the Automatic Speech Recognition computer, and overlays the cues on the video images. The display system's intended operation is somewhat unique for Linux, as the user program has hard real time operating requirements without a very supportive framework in the operating system. The ways in which the resultant difficulties were addressed will be discussed, as well as the general structure of the control program.

5.1 Real Time System Control

The control of the video system posed many challenges because of the need to respond in real time to a variety of events and to do so in coordination with two other computers distributed across a network. No approach completely resolved all of the issues, but the one that was adopted did provide a framework in which errors were either tolerable or correctable.

5.1.1 Synchronization

The audio and video signals provided by the camera imaging the speaker are captured and processed by two different computer systems. In order for the two systems to

be able to communicate with each other about their captured data, they must have a common frame of reference for the timing of the camera data. For example, the audio system must be able to inform the video system as to when in the video stream a particular cue should be overlaid. Unfortunately, the two systems share no clock that can assure proper synchronization of the audio and video data. Instead, during system initialization, the ASR computer issues a message on the network to the video and audio computers for each to start its timer. Typical network latencies for data transmission are on the order of 10 milliseconds, which is sufficiently low that noticeable timer skew errors are not introduced.

5.1.2 Fault Tolerance

Response to the interrupts generated by the frame grabber board is asynchronous with respect to the rest of the system, and usually occurs immediately. In contrast, the controlling user program that processes and displays the video data is subject to the scheduling policies of the operating system. Linux does allow for a form of real time scheduling, in which a user process has a higher priority than any other, but the high priority user process must share CPU time with essential system processes. For this reason, the control program may at times respond late to the input of video data as it waits for its CPU time slice.

In addition, when the program does respond, it may find that the video data was corrupted due to PCI bus conflict. The PCI bus and memory arbitration protocols have no provisions for real time data transfer, and so competition with the display card, network card, hard drive, and CPU may sometimes force the frame grabber to lose data.

The control program must be prepared to detect and respond to both of these conditions so that a cue is applied to the proper, error free, video frame.

5.2 Cue Overlay

The Automatic Speech Recognition computer processes the audio data to determine the cues associated with the speech. The ASR transmits the information relevant to the cues over the network so that the video display system can then overlay the cues on the video stream. The cue message specifies which hand shape is to be displayed, the position at which it is to be placed relative to the image of the speaker's face, and the time in the video stream at which it should first be displayed.

The start time for the cue is with reference to the synchronization timer that each computer started simultaneously, as described above. This time can then be matched up with those times that are assigned to the video frames as they are digitized and stored in the delay buffer. Since the time for a particular video frame is assigned at the end of its capture, the cue overlaid on it is the one most recently received whose start time does not exceed the frame time.

The position at which the cue is placed relative to the speaker's face is dictated by the rules of Manual Cued Speech. However, the overlay routine must still locate the position of the speaker's face within the video frame. To ease this process, the location of the cue positions are marked on the video image by the user at the beginning of a session, using a movable cursor, and the speaker is then requested to limit the movements of his head to maintain the accuracy of the positioning.

To better simulate manual cueing, the overlay routine attempts to reproduce the coarticulation present in the transition of the hand from cue position to cue position. Previous experiments involving simulations of the present system [4] displayed the cueing hand in a fixed position for the entire duration of the cue, with a discrete change to the next position. To present a more realistic transition of the hand from one cue position to the next, both the current and the next cue are considered in the placement of the hand. After the hand has been displayed in the current cue position for several frames, it follows a linearly interpolated path to the next cue position, arriving at the start time for the next cue. The hand shape changes from the current to the next shape halfway along the path between the two cue position.

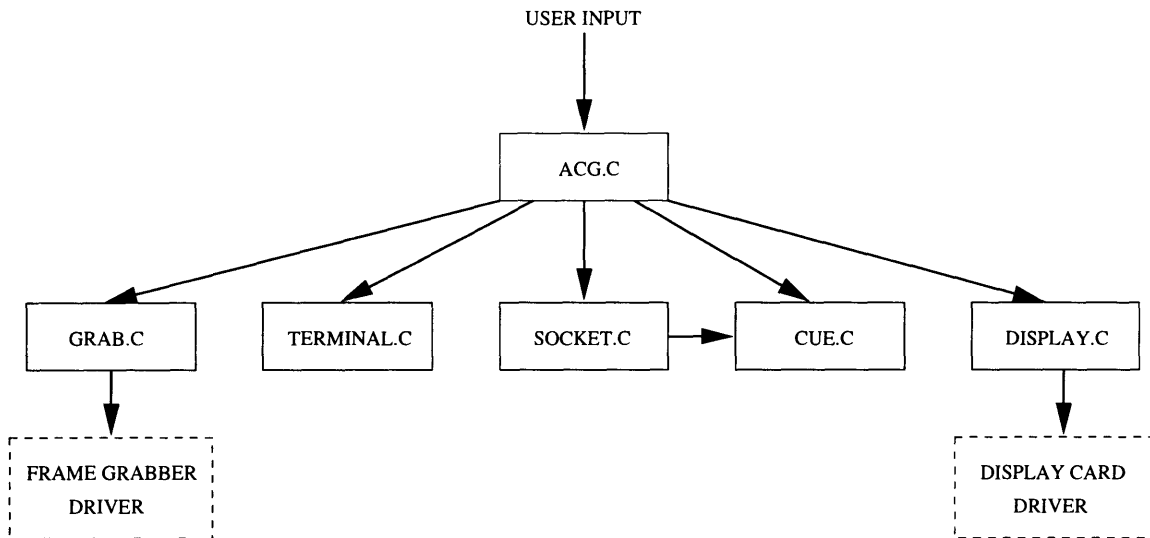


Figure 5-1: Display System Software Module Block Diagram

In this manner, the cue receiver can obtain information about the next cue before it actually begins, as in manual cueing.

5.3 Software Modules

The source code of the control program for the video display system consists of several software modules of delimited functionality. As shown in Figure 5-1, one module, `acg.c` interacts with the user in order to direct the operation of the other modules.

5.3.1 `acg.c`

The `acg.c` module first processes any argument flags on the command line, which modify the behavior of the program. These options include:

- `--help` List the command line options.
- `--delay DELAY` Set the length of the delay buffer to be `DELAY` frames.
- `--server SERVER` Specify the name of the ASR computer to be `SERVER`.

`--composite` Use a composite video source.

`--yc` Use an S-Video source.

`--hres HRES` Specify the horizontal resolution of the captured image to be HRES pixels.

`--vres VRES` Specify the vertical resolution of the captured image to be VRES pixels.

`--cue` Overlay cues on the video image.

`--locate` Manual specify the cue positions in the video image.

`--directory DIRECTORY` Specify the directory in which the cue images are located as DIRECTORY.

Initialization

Once the command line options have been parsed, a system call is made to request real time scheduling priority. The initialization routines of each of the other modules are called to set up the frame grabber, the display card, and the network interface. At this point, the module prepares its own arrays for holding frame timing and error data, and issues the call to begin the digitization of video.

Control Loop

The control module enters an infinite loop that handles all of the normal activities of the display system; an abbreviated version of the loop is listed in Figure 5-2. At the head of the loop is a call to `grab_wait_for_frame()`, which regulates the timing of the event loop. Ultimately, the call is processed by the frame grabber device driver, which puts the process to sleep until another video frame has been captured. When the frame grabber receives an interrupt signaling the end of the frame, it awakens the process and calls the scheduler to run it. Normally, when the process restarts, it overlays a cue on the oldest video frame in the buffer, transfers it to the video card for display, and jumps to the beginning of the loop to wait for another cycle.

The loop must check for two error conditions. First, `grab_wait_index()` returns an error if the captured frame was corrupted while being transferred on the PCI bus. If so, the frame is marked as having an error, so that it will not be displayed after

```

for (;;) {
    val = grab_wait_for_frame(&index_in);
    index_out = (index_in + 2) % delay;

    if (val < 0) {
        grab_error[index_in] = TRUE;
    } else {
        grab_error[index_in] = FALSE;
    }

    times[index_in] = timer_get_time();

    index_prev = (index_prev + 1) % delay;
    while (index_prev != index_in) {
        grab_error[index_prev] = TRUE;
        index_prev = (index_prev + 1) % delay;
    }

    if (grab_error[index_out]) {
        continue;
    }
    grab_get_frame(index_out, &frame);

    if (times[index_out])
        cue_draw_cue(frame, times[index_out], cue_positions);

    display_draw_screen(frame);
}

```

Figure 5-2: Main Control Loop

the delay period. Second, the process may not have been able to attend to each video frame that has been captured since the last time through the loop. Either the process may not have been scheduled in time after the end of capture, or the last cycle through the loop may have taken more time than the capture of a video frame (due to another process being scheduled during the loop). In either case, the `grab_wait_for_frame()` returns information as to which buffer element was just stored, so that a comparison can be made with its previous value to determine if the processing of any elements was skipped. If so, all those skipped elements are marked as containing errors, since

their frames may have been corrupted in capture.

If the oldest frame is marked as corrupted, `continue` directs execution back to the top of the loop; the frame currently displayed will remain for another cycle. Otherwise, a pointer to the output buffer element is obtained from `grab_get_frame()` and passed on to the cue overlay routine `cue_draw_cue()`. Also passed to the overlay routine is the time stamp that was assigned to that video frame by `timer_get_time()` after it was captured. However, the call is not made if that time is 0, which means that the frame was captured before the system clocks were synchronized. After this step, the frame is sent for display by `display_draw_screen()`.

Exiting

The control program exits upon an unanticipated error condition, or on a signal from the user. It first stops the capture of video frames, restores the normal computer display, and then exits.

5.3.2 `cue.c`

The routines in this module store the cue specification received over the network and overlay the images of cues on the video stream. Because the overlay is done by the CPU, an algorithm was developed to reduce computation and memory access cycles. In the conventional overlay of one image over another, a background color distinguishes which pixels of the first image should be copied and which should not; only pixels that are not of the background color are copied. While this method is appropriate for arbitrary images, it is very inefficient when the images to be copied can be processed ahead of time, as is the case for the cues.

The hand shape images are stored as graphics whose dimensions are the same as the images that will be captured, with a black color denoting the background. When a shape is first loaded, the image is scanned pixel by pixel in a manner similar to run length encoding. Counters record the length of background and hand pixel runs, so that only pixels corresponding to the hand need be stored in memory. When

the image is overlaid, the routine can use these counters to place the hand pixels in the destination image without the need to filter out the background pixels. Using this technique, the overlay requires the processing of up to twenty times fewer pixels, resulting in significant time savings.

```
int cue_init(int h, int v, int p, char *dir);
```

Read the cue shape images from disk and prepare the linked list that holds the cue information.

```
int cue_add_cue(struct cue_datum *cue);
```

Add the cue information in the struct cue to the list of cues sent from the ASR system.

```
int cue_draw_cue(void *frame, int time, struct coordinate cue_positions[]);
```

Overlay the appropriate cue on the video frame frame, whose capture end at time. The array cue_positions[] specifies the cue positions in the video image.

5.3.3 timer.c

The timer is used to synchronize data exchanged between the ASR computer and the video display system. The timer increment is one one-hundredth of a second, the fundamental length of time for the speech recognition system.

```
void timer_set_zero_time(void);
```

Set the origin of the timer to be the current time.

```
int timer_get_time(void);
```

Return the time in hundredths of a second since the call to `timer_set_zero_time()` or 0 if the call has not yet been made.

5.3.4 display.c

The `display.c` module manages interaction with the device driver for the display card and the flipping of display pages for a non-flickering video display.

```
int display_init(int hres, int vres, int pixel);
```

Put the display card in the graphics mode of hres by vres pixels, with a color depth specified by pixel.

```
int display_draw_screen(void *buffer);
```

Use video frame buffer as the next for display.

```
void display_end(void);
```

Restore the original state of the display.

5.3.5 grab.c

The grab.c module interfaces with the frame grabber device driver to control the capture of video and to help manage the video delay buffer.

```
int grab_init(int hres, int vres, int pixel_size, char *camera, int del);
```

Initialize the frame grabber to capture images of hres by vres pixels at a color depth of pixel_size from camera. The delay buffer is del frames long.

```
int grab_start_grab(void);
```

Begin the transfer of captured video into the delay buffer.

```
int grab_stop_grab(void);
```

Stop the transfer of captured video into the delay buffer.

```
int grab_wait_for_frame(int *index);
```

Sleep until the frame currently being captured has been fully transferred into memory; return its buffer index in index. Return EIO if the frame was corrupted.

```
int grab_get_frame(int index, void **frame);
```

Return in frame the address in memory of the delay buffer element with index index.

5.3.6 `socket.c`

During its initialization, `socket.c` sets up an asynchronous signal handler that is called whenever the process receives data over the network. The signal handler parses the message from the ASR computer and acts according to its command field. If it is a `GO` command, the synchronization timer is started, and a reply is made with a confirmation message. For a `STOP` message, the video system is shut down. A message with a `V_DATA` command contains cue data which is passed on to the cue software module.

```
int socket_tcp_init(char *server)
```

Setup the TCP socket for communicating with the ASR computer, and set the process to receive the `SIGIO` signal from the socket.

```
int socket_tcp_ready(void);
```

Send a message to the ASR computer that the video system is prepared to receive the `GO` message.

5.3.7 `terminal.c`

The video card used for the cueing display is also the one that is used for the display of system graphics and text. `terminal.c` acquires control of the display from the system and prevents any other process from manipulating the board and causing errors. The module also reads the user input to position a cursor marking the cue positions.

```
int terminal_init(void);
```

Change to the first unused virtual terminal and lock it. Put `STDIN` in non-canonical mode.

```
int terminal_end(void);
```

Release the display to the system and restore the original state of `STDIN`.

```
char terminal_get_char(void);
```

Return the most recent character from `STDIN`.

Chapter 6

Discussion

As has been mentioned in previous chapters, several aspects of the video display system can still be improved. Problems specific the base system implementing the video control and delay and to the cue overlay system are discussed below. Where possible, solutions and paths of investigation are proposed.

6.1 Base System Improvements

In general, the video acquisition, display, and delay hardware met the minimal requirements of the system. The modifications described below would serve to extend the video system's abilities and to increase its operating robustness.

6.1.1 Video Hardware

As mentioned in Subsection 2.1.7, the display card was the hardware device with the slowest and most primitive memory interface. If it had incorporated DMA controllers, video images with greater color depth could be captured and transferred from system memory to the card, without intervention by the CPU. Unfortunately, such a modification is wholly dependent on the future availability of appropriate display cards.

The memory address restriction of the frame grabber's DMA controller, discussed

in Subsection 3.3.1, requires an awkward and wasteful allocation of memory. This feature can be disabled, but at the risk of corrupting other areas of system memory if the DMA controllers are misprogrammed. The trade-offs between memory use and reliability should be examined. In addition, the buffer allocation and management routines presently in the frame grabber device driver should probably be moved into a separate module, so that a more abstract interaction between the two can be realized.

6.1.2 Computer Synchronization

The method used to synchronize the start of the timers for the audio and video data is susceptible to errors introduced by network delays. Any delay in the transmission of the GO message from the ASR computer to the display computer increases the skew between the timers. If the skew is a significant fraction of the time for which a video frame is displayed, cues will be overlaid late in the video stream, providing incomplete and possibly erroneous information to the cue receiver.

To correct the problem using the existing hardware setup (i.e. without introducing a common clock source), the ASR computer should examine the round trip time for sending the GO message and receiving a reply. If the total time is greater than a threshold value, the synchronization process should be restarted. By repeating this step as necessary, the clock skew can be guaranteed to be less than a specified value.

6.1.3 Real Time Scheduling

Although the control process is marked for real time execution by the scheduler, it could be subject to delays incurred by page faults if pages of its memory are swapped out while it is not running. System calls exist for locking pages of a process into memory so that they will not be swapped out, which may speed the program's response. Whether such a measure is necessary depends on the loading of the system while the cue display task is being performed.

The control process currently is put to sleep after it finishes sending a frame for display and is ready for the next incoming frame. Although the delay associated with

awakening the process is probably not significant, it should be examined in more detail. The delay in fact may already be nearly irreducible. Even if the process were to be modified so that it continuously runs and just polls the device driver, it would still incur the overhead of the two context switches necessary for the system call. The current scheme incurs the delay of only one context switch with the awakening of the process.

6.1.4 Graphical Interface

The display system would be much more portable and user friendly if it could make use of the X Window system. X Windows is a multi-platform graphical environment that provides an abstract interface to the underlying display hardware. Using X would make the cue display system less dependent on its maintainer's ability to program a particular video card. Unfortunately, the conventional X system has neither an efficient means of transferring large amounts of image data nor an acceptable way of dealing with the display of video. If these difficulties could be overcome, the system would be greatly improved.

6.2 Cueing Improvements

The complete cueing system was tested in real time with speakers and cue receivers on three different occasions. Although the system did provide measurable benefit to the cue receivers as compared to speechreading alone, its performance was severely limited by the accuracy of the ASR system. In addition, the overlaid cues were distracting to some of the subjects. Cue receivers were disturbed by the appearance of the hands and the nature of the transitions between cue positions. These issues as well as others relating to improvements in the cue display are discussed below.

6.2.1 Cue Presentation

There were several problems with the both the appearance and the movement of the synthetic cues. First, the overlaid hand shapes did not always match well in skin tone and size with the speaker's face. While they probably did not affect the cue receiver's ability to understand the speaker, these discrepancies did reduce their comfort with the system. Second, no provision was made for the "flick" of cued speech. To signify consecutive identical cues in the side position of the manual system, the hand is flicked forward by the speaker. But it is not clear how to simulate this movement in a 2-D display. Third, the timing of the cueing for diphthongs was chosen on an ad hoc basis and should be refined to more closely match human actions. Finally, the hand did not transition smoothly from one cue shape to another. To model such movements accurately would require either very sophisticated graphics rendering algorithms or a comparatively large image data base. The extent to which such modeling would benefit the cue receiver is unknown.

6.2.2 Automatic Cue Positioning

The current system requires that an operator manually specify the five cue positions in the video image. While this procedure is acceptable for situations in which the speaker's movements are restricted, it is clearly unsuitable for many settings. A system for automatically tracking the location of the speaker would greatly augment the utility of the Automatic Cue Generator.

Systems have already been implemented that track the location of face within an image and follow an individual's movements throughout a room [30]. The main research effort would be in developing an algorithm that can extract the exact location of the cue positions based on a general positioning of the speaker's head. To ease the addition of such algorithms to the control program, procedure hooks have been included in the source code. They now call empty functions, but can be modified as necessary.

Bibliography

- [1] AITech International. *PC/TV-1108 Digital Scan Converter User's Guide*, 1995.
- [2] Ariel Corporation. *User's Manual for the DSP-96 DSP96002 Floating-Point Attach Processor Board with Dual-Channel Analog I/O*, April 1995.
- [3] Michael Beck et al. *Linux Kernel Internals*. Addison-Wesley, 1996.
- [4] Maroula Sarah Bratakos. The effect of imperfect cues on the reception of cued speech. Master's thesis, Massachusetts Institute of Technology, 1995.
- [5] Brooktree Corporation. *Bt819 EVM*.
- [6] Brooktree Corporation. *VideoStream Decoders*.
- [7] Coreco, Inc. *User's Manual for Oculus TCi*, 1995.
- [8] R. Orin Cornett and Mary Elsie Daisey. *The Cued Speech Resource Book: For Parents of Deaf Children*. National Cued Speech Association, 1992.
- [9] Crimson Tech Reference Catalog. *Reference Catalog*, 1994.
- [10] Cypress Semiconductor Corporation. *ICD2061A: Dual Programmable Graphics Clock Generator*.
- [11] Diamond Multimedia Systems, Inc. *Stealth 64 Video VRAM User's Guide*.
- [12] Edward Dolnick. Deafness as culture. *The Atlantic Monthly*, September 1993.
- [13] Entropic Research Laboratory, Inc. *HTK: Hidden Markov Model Toolkit V1.5*.

- [14] Richard F. Ferraro. *Programmer's Guide to the EGA, VGA, and Super VGA Cards*. Addison-Wesley, 1994.
- [15] Vincent Himpe. I2C bus FAQ. Available at <http://www.ping.be/ping0751>.
- [16] Pamela J. Jackson. The theoretical minimal unit for visual speech perception: Visemes and coarticulation. *The Volta Review*, September 1988.
- [17] Hongjiu Lu. Elf: From the programmer's perspective. 1995.
- [18] David S. Lum. Lab guide to using the DSP-96 board under linux. 1996.
- [19] Hans-Peter Messmer. *The Indispensable PC Hardware Book*. Addison-Wesley, 1995.
- [20] NCSA. Guidelines on the mechanics of cueing. *Cued Speech Journal*, V, 1994.
- [21] Philips Semiconductors. *SAA7116: Digital Video to PCI Interface*.
- [22] Philips Semiconductors. *SAA7192A: Digital Color Space Converter*.
- [23] Prime Image. *Audio Video Delay*.
- [24] S3 Incorporated. *Vision964 Graphics Accelerator*.
- [25] S3 Incorporated. *Vision968 Multimedia Accelerator*.
- [26] Matthew G. Sexton. C language development on the DSP-96 DSP96002 floating-point attached processor board. 1996.
- [27] Sony Corporation. *Digital Video Camera Recorder*, 1995.
- [28] Texas Instruments. *TVP3026 Video Interface Palette: Data Manual*, 1994.
- [29] Rosalie M. Uchanski et al. Automatic speech recognition to aid the hearing impaired: Prospects for the automatic generation of cued speech. *Journal of Rehabilitation Research and Development*, 31(1), 1994.

- [30] Jie Yang and Alex Waibel. Tracking human faces in real-time. Technical Report CMU-CS-95-210, Carnegie Mellon University, Pittsburgh, PA 15213, November 1995.