status report for

A Sun Sparcstation-Based Speech Data Collection Platform

LDC Subagreement 5-24431-C
ISIP Project No. 02-95

for the period of October 1, 1995 to January 31, 1996

submitted to:

Linguistic Data Consortium

441 Williams Hall
University of Pennsylvania
Philadelphia, PA 19104-6305

submitted by:

Joseph Picone, Ph.D., Associate Professor

Institute for Signal and Information Processing
Department of Electrical and Computer Engineering
Mississippi State University
Box 9571
413 Simrall, Hardy Rd.
Mississippi State, Mississippi 39762
Tel: 601-325-3149
Fax: 601-325-3149
email: picone@isip.msstate.edu

EXECUTIVE SUMMARY

The third phase of this project resulted in a major milestone — Linkon finally delivered functional hardware. We received our first copy of the software in early November'95 (v5.0 Beta). The hardware followed several weeks later, arriving at the end of November. We lost one month debugging their installation script, and managed to have a basic demo of the analog telephone portion of the system working by the end of the year.

We completed verification of the installation of the system shortly after resolving the installation problems. Linkon uses a fairly inflexible installation procedure based on a new Solaris convention — pkgadd. Not only is this installation incompatible with most user's standard Unix research environments (utilizing distributed filesystems) but the pkgadd command turns out to be highly sensitive to certain standard Unix mechanisms, such as links. We resolved the installation problems by reconfiguring our machine slightly, and increasing our spool space. Linkon support was totally unhelpful during this process — at one point suggesting we wait until a CDROM release was available in February'96.

The next major obstacle we overcame was debugging Linkon's demo programs, which were obviously not tested or expected to work on standard analog telephone lines (these demos were expecting some ISDN signaling and a compression software license which we weren't given). It was at this point we observed a byte-swapping problem with 16-bit linear data involving long-word swapping. Again, Linkon support was useless (aside from eventually telling us how to "dejive" the data on an Intel machine). Though we are not happy with the current solution to this problem, we can at least make the board compatible with existing Unix I/O standards for 16-bit data. A firmware fix is promised in the next release of the software. The third major problem we faced was an imbalance in the play and record levels. We are still not sure exactly what the analog hardware uses for A/D, but we have a strong belief it is an 8-bit codec with some asymmetric μ-law to linear conversion software. The fourth problem we faced is utterance detection — a hook is not provided to pad the utterance with silence prior to the start of the utterance. This is a serious problem that will affect both analog and digital collection.

Despite these small problems, we have managed to complete a demonstration of POLYPHONE-style data collection (type I). This was made available on January 28 (call 601-325-2292). A demo of SWITCHBOARD-style data collection (type II) is near completion (we just fixed one of two remaining bugs today while I was writing this report). We expect this demo to be complete within the week. Given that we now appear to have stable hardware, we seem to be proceeding smoothly with construction of the data collection system. Poor turnaround with Linkon support continues to be a major frustration. However, we do seem to have the ear of the marketing people, who have helped resolve the support problems.

We expect at the completion of the next phase of the project, we will have flexible software in place to perform type I and type II data collection on analog hardware. This is an important milestone, because it will complete our certification of the product from a software interface standpoint. From here, we can proceed rapidly to deployment of the digital hardware and creation of a GUI for the data collection software. We expect the GUI development to extend beyond the current May 15 deadline. However, we have made provisions for the current staff to be available throughout summer in order that we can accomplish this task.

1. HARDWARE INSTALLATION

We received a pre-release of the Linkon software in early November (a Beta release of v5.0 with documentation dated October 14, 1995). The software came on a 1/4” cartridge tape — not a good sign. We had hoped this would give us a lead in the software design process. Unfortunately, the documentation recommended not loading the software without having the hardware installed, since both use a common installation driver (standard Unix installation procedures based on tar and install were not provided). The software installation attempts to do some kernel modifications and some hardware-dependent configuration. Further, the documentation did not describe the software in sufficient detail, and the installation did not provide any options to simply install the software, so we were left hanging.

The hardware arrived in late November, just prior to the Thanksgiving holidays. We immediately proceeded to install it — with no major problems. There was one major surprise — the system occupies TWO SBus slots. The FS-4000 is a dual SBus card systems that lays flat in the Sparc hardware and takes two SBus slots horizontally adjacent. This leaves one available slot on the same row (Sparc 5’s have three slots across and one slot underneath these three slots) to hold the T1 card (which is also supposed to be only one slot). Due to the thickness of these cards, I am not sure the fourth slot can be utilized. However, since we have three available slots due to the fact that we are using a Sparc 5 server model with no video cards, this should not pose any problems.

Each of these two cards contains one RJ-45 connector that multiplexes four analog phone lines into the card (the RJ-45 connector is an 8-wire connector). The system comes with a harmonica-style RJ-45 block that is used to collapse 8 telephone lines into 2 RJ-45 jacks. The unit we received had the analog phone lines incorrectly labeled, which cost us some time debugging demo software that, in the end, simply was addressing the wrong telephone port.

2. SOFTWARE INSTALLATION AND DESIGN

To quote from the Linkon manual, “to install your selected Linkon software product enter the following commands:”

```
pkgtrans /dev/rmt/0 spool LKONxvox
pkgadd LKONxvox
```

Needless to say, this didn’t work. This assumes you have all disk space local to the machine, have 100M available in your spool directory, and have 100M of space available in /opt. The machine hosting the hardware initially did not have these resources — like all our machines, it uses network-mounted file systems and operates as a data-less node.

Our standard procedures to work around such things are to use links and such to make the appropriate file systems appear as expected. The above commands did not work on our systems for various reasons (it seems pkgadd is very sensitive to links being used in place of local disks because it operates as root, and root on one machine is not root on another machine). After reconfiguring our machine to have local disk configured exactly the way Linkon wanted it, the installation procedure would still not work. At this point, Linkon told us to wait for a CDROM distribution of the software in February’96, acknowledging that several customers had problems with the software installation. This was obviously unacceptable. They refused to provide ftp access to the software from their machines, and would not help us debug the problems. It is our belief they don't understand these installation scripts. In any event, by studying the pkgadd software carefully, and using some arguments that control installation paths, we were able to get the software to load correctly.

The software is now installed in two places:

```
/usr/tvox
/usr/xvox
```

The first directory is required for backward compatibility with their old software from their previous generation product — the FS-3000. The second directory is where most of the new software is located. We aren't satisfied with the current organization of the code, and haven't finished relocating the code. It is our preference that Linkon's code be installed in /usr/local/linkon, which is a standard convention on most Unix systems, including ours. After we have completed the development of our first set of demos, we will revisit the installation problems and experiment with a reorganization of the code.

There is also a boot file in /etc/rc2.d/S27xvox that is created at the time of installation. The file somehow instructs the host to boot the Linkon board before the CPU actually boots. However, these is an additional configuration program that must be run at the end of the host boot sequence, named linkvox, which must be run by root. There is an option to automatically restart the system upon boot, but we haven't experimented with it yet. Such a capability is critical to having a robust data collection system. The default configuration requires the board to be manually initialized by root.

After installing the software, we found that most of their configuration and verification programs required minor modifications to work correctly. Unfortunately, their demo programs (the last step in verification of a successful installation according to the manual) didn't work. After consulting with Linkon, we ended up writing our own program to verify the installation. Fortunately, when all the dust settled, we were able to declare a successful installation. Since then, the hardware has functioned well, though the Linkon board has mysteriously locked up twice in the one month it has been on-line (we haven't finished robustness tests yet).

We have some long-term concerns about the quality of this software. It is clear Unix is not their main platform. The code is written in K&R C, not ANSI-C, which causes numerous compiler warnings when compiled with gcc in g++ mode. The vendor appears uninterested in supporting ANSI-C and gcc. Thus far, we haven't found any problems that could be attributed to gcc, so we are proceeding with our plans to use the GNU software tools.

3. DATA COLLECTION DEMONSTRATIONS

The next step in our verification of the system was to implement two basic forms of data collection: POLYPHONE-style and SWITCHBOARD-style data collection. A summary of the call flows are shown in Figs. 1 and 2. Type I data collection is fully functional and has been made available via the telephone number 601-325-2292. Type II data collection is in the final stages of debugging and should be available shortly.

```
a.  call 325-2292
b.  system answers and plays welcome message
c.  system says “please speak now” and follows with a beep
d.  system records my voice and stops approximately 3 secs after
    I stop talking (this requires setting the utterance detector)
e.  go to step c unless:
    I hangup - in which case you exit
    I hit a touchtone “*” - in which case I go to f
f.  play the “thank you message”
g.  exit and wait for next call
```

Figure 1. Type I data collection (POLYPHONE-style) consists of a prompt and record type scenario in which the system plays an audio prompts and records the following audio data.

```
a.        call 325-2292
b.        system answers and plays welcome message
c.  (1)   system asks for a pin number
    (2)   system says “please enter the telephone number of your party”
d.        user enters the telephone number using touchtones
          system verifies the phone number by speaking the digits
          “You have entered three two five six one three zero.”
          “Press the pound key to continue.”
          if no pound key is pressed after 5 secs go to h.
e.        System says “please wait while I connect you.”
          System dials the phone number.
          if:
              no answer: report back to user #1
                  “there was no answer
                  “please try again”
                  go to c(2)
              busy: report back to user #1
                  “the line was busy”
                  “please try again”
                  go to c(2)
              line was answered: go to f
f.        Announce to user #2: “I am connecting you to your party — please identify yourself”
          Connect them
g.        Begin recording both sides of the conversation.
          Turn off all utterance detection.
h.        When either party hangs up, terminate the call on both ends by playing the end message
          (if necessary), hang up, and wait for next call.
```

Figure 2. Type II data collection (SWITCHBOARD-style) consists of a teleconferencing scenario where two callers are connected and both sides of the conversation are recorded.

Thus far, we have been pleased with the software interface to the board. All of the necessary functions appear to be provided — though documentation is clearly lacking. Multiple processes (different data collection applications) can address the board simultaneously. There is even a low-level call for the teleconferencing function, which was a major technical issue in this project. We have yet to test this function, but it appears to do what we need for type II data collection.

An example of some of our prototype code is shown in Fig. 3. This is code for a function that records data. We have been successful thus far in abstracting the code to a reasonably well-structured user-friendly C++ API. The code shown in Fig. 3 is just prototype code that will be refined as we begin coding the production systems.

4. OUTSTANDING ISSUES WITH THE ANALOG SYSTEM

Thus far, we have encountered four major problems with the board. These are:

(1) byte-swapping (not a problem for μ-law data)

16-bit audio data needs to be long-word byte swapped:

Sun:	b0	b1	b2	b3
Linkon:	b2	b3	b0	b1

(Linkon seems to be heavily influenced by Intel platforms.)

(2) playback/record levels differ:

We believe the A/D converter is an 8-bit codec that is translated to a 13-bit linear signal when using the board in 16-bit mode. However, audio prompts can be played at 16-bit levels with no audible distortion, and must be played at that level to be easily heard. Prompts and recorded data have a noticeable difference in loudness. This seems to be a simple scaling problem. We were led to believe the audio quality would be 16-bit linear — not 8-bit μ-law. The recorded data does appear to be fairly clean (no noticeable 60 Hz hum or granular background noise).

(3) utterance detection does provide a pad time option for the onset of the utterance

The utterance detector does not support a parameter setting for the amount of silence that precedes a word. Trailing silence is controlled by a user-defined parameter, but this value also is used to determine the end of utterance. The net result is that recorded utterances begin within milliseconds of the beginning of the utterance, and have three seconds of silence at the end of the utterance. We can use our own utterance detector in real-time to segment data, but this will needlessly load the host CPU for a system processing 8 channels of data (one of the goals of the project). It is a function best done in the DSP hardware. Their utterance detector seems to work well — it just needs a few more parameters to control the definition of the start and stop times of the utterance it finds.

(4) ddi_dial() does not work (a function to dial a phone number)

Very simply a bug which we can work around for the moment with our own code. However, yet another indication of how little of this software has been tested.

We have believe all of the problems can be resolved with a minimum of work. However, getting Linkon’s attention has been a problem. Under duress, the support person finally admitted they rushed to get this board out the door, and haven’t had time to refine it. Fortunately, George Dinsdale, the lead salesperson, has been very helpful in getting the attention of Linkon’s

```
/* file: /isip/d00/linkon/lib/record.cc */

/*
 * Richard Duncan
 * The Institute for Signal and Information Processing
 * December 1995
 *
 *
 * This function reads in audio data from the f3kb board
 *
 */

#include <stdio.h>

#ifndef _RECORD_H
#define _RECORD_H
#include <record.h>
#endif

#ifndef _SYS_DDI_H
#define _SYS_DDI_H
#include <sys/ddi.h>
#endif
...

int record(char *speech_data_filename,
   int channel, char *dtmf,
   int *disconnect){

  VCB vcb;
  FILE *pout;
  int ret;
  int totlen;
  int wc;

  // initialize arguments passed to program
 //
  *dtmf=(char) NULL;
  *disconnect=0;

  // Print the log message
  fprintf(stdout, “record: recording %s on channel %d\n”,
  speech_data_filename,
  channel);
  fflush(stdout);

  //  open the output data file
  pout = fopen(speech_data_filename, “wb”);

  // Initialize a vcb to the channel
  ddi_ivcb(&vcb,channel);

 //  start recording
  ret = ddi_nb_record(&vcb);

  if(ret != VRC_SUCCESS){
    return(ret);
  }

  //  count bytes recorded
  totlen = 0;

  while (1) {
    // get next vcb
    ret = ddi_getvcb(&vcb, 0xf000);
    if(ret != VRC_SUCCESS){
      continue;
    }

    // record data received
    if(vcb.opcode == TOP_MDM_RECV){

      // swap every 4 bytes of data
     swap)4_bytes((short int *)
          &(vcb.vp.rmd.data),vcb.vp.rmd.words);
      wc = fwrite(vcb.vp.rmd.data, sizeof(U32), vcb.vp.rmd.words,
           pout);
      // test that the correct amount was read
      if (wc != vcb.vp.rmd.words){
         fprintf(stdout,”Record write failed\n”);
         fclose(pout);
         return(FAILURE);
      }
      totlen+=wc;
      continue;
    }

    if(vcb.opcode != TOP_TV_EVENT){
      continue;
    }

    if(vcb.result == CEC_DTMF){
         if (pout != (FILE *) NULL) fclose(pout);
        ret=vcb.errno;
       get_dtmf_code(ret,dtmf);
       fprintf(stdout,”DTMF detected >%c<\n”, *dtmf);
     ddi_clrabort(channel);
      ddi_offhook(&vcb, 0xffff);
      sleep(1);
      return(SUCCESS);
    }

    if (vcb.result == CEC_DISCONNECT){
      fprintf(stdout, “USER DISCONNECTED\n”);
      fclose(pout);
      ddi_clrabort(channel);
      ddi_offhook(&vcb, 0xffff);
      sleep(1);
      *disconnect=1;
      return(SUCCESS);
    }

          if ((vcb.result == CEC_RECC) || (vcb.result ==
CEC_RFAXC)){
      fprintf(stdout, “Record ended %d bytes\n”,vcb.errno);
      fclose(pout);
      ddi_clrabort(channel);
      ddi_offhook(&vcb, 0xffff);
      sleep(1);
      return(SUCCESS);
    }
  }
}
```

Figure 3. An example piece of code demonstrating how to record data from the Linkon board from a user-level C++ program. This code is used in both type I and type II data collection demos.

Engineering Department.

5. STAFFING AND NEAR-TERM PLANS

In addition to the delay in hardware delivery, we had a significant interruption in our staffing with the untimely death of the key engineer on this project in October'95. This has been a hard thing to overcome, especially since ISIP is not overflowing with programmers of his capability. Given that we were in the middle of the project, and it would have been hard to replace him with someone of his caliber on such short notice, we decided to continue executing the project with only one software engineer (who is my best programmer) and the PI. While it has slowed down the project, and will mean we will miss our May 15 deadline for completion of the project, I think this is the best thing overall to do to guarantee the highest quality output from this project.

We have also added a new member to our group — a sys admin with extensive experience in telecommunications and computer networking, including T1 data connections. (This is a senior person who has returned to school after running his own computer businesses in Jackson, MS for 10 years.) The department is paying his salary, and has assigned him to our group for sys admin training. He will be a valuable asset to us on this project — particularly in supervision of the T1 installation and certification of its integrity. He has reviewed the project documents and seems to understand the technical details. Feel free to call upon his expertise as you need it.

We should not miss our May 15 deadline by much. We expect to deliver software for types I and II data collection using the analog or digital system by May 15. This software will allow the Linkon system to be configured from parameter files (the standard approach we use in ISIP software) to perform a variety of general data collection tasks (we are following a model I previously developed at Texas Instruments). You might think of this as a scripting-level interface to the system.

In the summer months following this delivery, we plan to accomplish two things: prove the system with an actual corpus collection (performing whatever additional debugging is necessary), and develop a more user-friendly tcl-based GUI for system configuration. The remaining software engineer has committed to remaining here during the summer to work on the project, so there should be no further disruption in staffing. (Incidentally, this person has written 95% of the useful software for the JEIDA and T1 projects.)

If this is not acceptable, please let us know, and we will negotiate a more satisfactory arrangement.