Transmission Of Voice Signals Over The Ethernet Network

Michael Georgiopoulos

Nicos Christou

Yousuf Cheng Hung Ma

This Research Report is brought to you for free and open access by the Digital Collections at STARS. It has been accepted for inclusion in Institute for Simulation and Training by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

Recommended Citation
https://stars.library.ucf.edu/istlibrary/202
Transmission of Voice Signals Over the Ethernet Network

Michael Georgiopoulos
Nicos Christou
Yousuf C. H. Ma

May 7, 1991
Transmission of Voice Signals over the Ethernet Network

Michael Georgiopoulos
Department of Electrical Engineering
University of Central Florida, Orlando, FL 32816

Nicos Christou
Department of Electrical Engineering
University of Central Florida, Orlando, FL 32816

Yousuf C. H. Ma
Department of Computer Engineering
University of Central Florida, Orlando, FL 32816

May 7, 1991

Abstract

In this report, we consider the transmission of voice signals over an Ethernet network. The experimental set-up consists of two AT computers each one of which is equipped with a DSP56001 board, manufactured by Ariel Corporation, and a 3Com EtherlinkII adapter, manufactured by 3Com Corporation. It is worth mentioning that at the transmitter site (i.e., one of the AT computers) the voice signals generated, prior to their transmission over the Ethernet network, are sampled, quantized and organized into packets. At the receiver site (i.e., the other computer) the arriving packets are assembled together, transformed into analog signals and played out. One of the objectives of this experimental set-up is to examine the effect of network packet delay variability on the reconstructed speech signals at the receiver site.
1 Introduction

Future communication networks are expected to handle a variety of data traffic types, covering a range of applications as diverse as very low bit-rate control and alarm signals for the home and business, interactive information services, electronic mail, digital voice, facsimile, file transfers and wideband digital video services among many others. These networks have been termed *Integrated Services Digital Networks* (ISDNs).

Networks that interconnect simulation devices have to operate in an integrated services networking environment. As a result, networks interconnecting simulation devices must be integrated services digital networks. Currently, simulation devices require the transmission of both data traffic (state information) and voice traffic (FM radio). It is also suggested, that simulation networking should be capable of handling video traffic as well. This need will arise whenever a simulation device requests terrain data information.

In this report we focus on the integration of voice and data over an Ethernet local area network. In particular, we concentrate on the transmission of voice signals over the Ethernet, because the transmission of data signals is a much easier task. The experimental set-up used to perform our experiments consists of two AT computers each one of which is equipped with a DSP56001 board, manufactured by Ariel Corporation, and a 3Com Etherlink II adapter, manufactured by 3Com Corporation. Our purpose is to develop software that will allow us to transmit voice signals from one computer to another. The organization of our report is as follows: In Section 2, we discuss briefly the software development to achieve the aforementioned goal, in Section 3 we present a flow chart description of the software packages discussed in Section 2 and in the appendices we include supporting material (i.e., hardware speci-
fications and software packages). It is worth mentioning, that at the transmitter site (i.e., one of the AT computers) the voice signals generated, prior to their transmission over the Ethernet network, are sampled, quantized and organized into packets. At the receiver site (i.e., the other computer) the arriving packets are assembled together, transformed into analog signals and played out. One of the objectives of this experimental set-up is to examine the effect of network packet delay variability on the reconstructed speech signals at the receiver site.

2 Software Development

2.1 Preliminaries

The DSP56001 microprocessor is a product of Motorola. Software for the DSP56001 microprocessor can be developed in assembly, as well as in C language. Both C and assembly language compilers are available by Motorola. In order to generate software that runs on the DSP56001 microprocessor one has to be familiar with the DSP56001 microprocessor and assembler. For software developments that run on the DSP56001 microprocessor and communicate with external devices, such as, the Host (in our case AT computer) and the ADC/DAC (analog to digital converter/digital to analog converter), the programmer has to be familiar with the board on which the DSP56001 microprocessor resides. The DSP56001 board that we are using is manufactured by Ariel Corporation. A debugger (Bug-56) for program development on the DSP56001 board is available by Ariel Corporation. For detailed information about the DSP560001 microprocessor references [3],[4],[5] are recommended. For detailed information regarding the Bug-56 coprocessor board and the Bug-56 debugger references [1] and [2] are recommended.
2.2 Program Description

Three programs were produced. The purpose of Program 1 is to test the DSP Synchronous Serial Interface (SSI) and gain familiarity with it. The SSI is the interface through which the DSP56001 microprocessor communicates with the ADC/DAC. Program 1 consists of two processes (processes 1 and 2) and a main program running in parallel with each other. A block diagram of program 1 and its components is shown in Figure 1.

Process 1 reads data, sample by sample, from the ADC, packetizes them, and stores them in the DSP56001 X-memory. Process 2 reads data, sample by sample, from the DSP56001 Y-memory and sends them to the DAC. The main program initializes the SSI, sets up the buffers, and shifts packets from the DSP56001 X-memory to the DSP56001 Y-memory, whenever packets are available. Priorities are assigned to the main program and the two processes. The main program is assigned the lowest priority and the two processes are assigned equal priority. Each process can interrupt the main program but they can not interrupt each other. Process 1

---

Figure 1: Block Diagram of Program 1
is activated whenever a sample is generated by the ADC and process 2 is activated whenever a sample is required by the DAC.

Program 2 consists of five processes (processes 1, 2, 3, 4, 5) and the main program. The purpose of this program is to test the Host Interface (HI) and acquire familiarity with it. The HI is the interface through which the DSP56001 communicates with the Host (AT computer). A block diagram of program 2 and its components is exhibited in Figure 2. Processes 1 and 2 of this program are the same as the processes 1 and 2 in program 1. Process 3 reads packets, sample by sample, from the DSP56001 X-memory and sends them to the HI. This process is activated by the host. Process 4 reads packets from the HI, sample by sample, and stores them in the DSP56001 Y-memory. This process is also activated by the Host. Process 5 is a program running on the Host. This process reads packets, sample by sample, from the HI and stores them in the Host memory. Furthermore, process 5 retrieves packets, sample by sample, from the Host memory and sends them to the HI. The main program initializes the interfaces (HI and SSI interfaces), sets up the buffers, and notifies the Host for the availability of new packets.

In summary, process 1 notifies the main program whenever the generation of a new packet occurs. Then, the main program informs the Host through the HI for the arrival of the new packet. Then, the Host activates process 3. Once process 3 is activated it will start sending the packet to the HI. The Host will, in turn, read the packet from the HI, and it will store it in its memory (Process 5). Then, the Host activates process 4 and it subsequently retrieves the packet from its memory and sends it back to the HI (Process 5). Once the packet is read, the DSP56001 microprocessor activates process 4. Process 4 reads the packet from the HI and stores it in the DSP56001 Y-memory. Once the packet has been stored in the DSP56001 Y-memory, process 4 notifies the main program for the arrival of a new packet. The main program
turns on the DAC and process 2 reads the packet from the DSP56001 Y-memory and sends it to the DAC. The aforementioned procedure is repeated indefinitely. In this program, processes 1 and 2 are assigned the highest priority, processes 3 and 4 are assigned the next higher priority, and the main program is assigned the lowest priority. Processes 1 and 2 can interrupt process 3, 4 and the main program but they can not interrupt each other. Processes 3 and 4 can interrupt the main program but they can not interrupt each other or processes 1 and 2. Process 5 runs on the Host, and as a result, it can not be interrupted by the DSP56001 microprocessor.

**Program 3** is the final program that we produced and it is basically the same as program 2. A block diagram of program 3 and its major components are shown in Figure 3. The only difference between programs 2 and 3 is the code that runs on the Host (previously referred to as process 5). In program 3, the code that runs on the Host reads the packets, that are transferred in its memory from the DSP56001 through the HI, and sends them to the Ethernet Network Interface (3Com board). Then, the packets are transmitted through this interface to the Ethernet network.
Set up the Buffers: \( r_0, r_1, r_4, r_6 = 0, r_5 = 128 \)
\( m_0, m_1, = 1FF \)

Set up priorities for each process
Enable interrupts for process
(Turn on ADC).

Check for interrupt requests

Yes

\( R_6 = 2 \)

Shift packet from x-memory to y-memory.
\( \text{move } x: (r7), a0 \)
\( \text{move } a0, y: (r7) + \)

No

Decrement R6

Enable interrupts for process 2
(Enable DAC).

Check for interrupt requests

No

\( R_6 = 0 \)

Shift packet from x-memory to y-memory

Yes

Decrement R6

Figure 4: Flow chart for main program of program 1
As a result, the voice packets generated by one computer will be available to any other computer on the network. Furthermore, in program 3, the code that runs on the Host reads packets from the Ethernet Network Interface to the Host memory and then, sends these packets from the Host memory through the HI to the DSP56001.

3 Flow charts description

In this section a flow chart description of the programs mentioned in the previous section is presented. Figure 4 shows the flow chart of the main program of Program 1. At the beginning, all interrupts are disabled and the SSI interface is initialized. Then, the sampling rate is chosen, the interrupts are enabled, the ADC is activated, and the main program falls into a loop reading register R6. At this point, the main program will be interrupted by process 1 every time a sample is generated. Once the ADC has generated two packets of samples, process 1 sets register R6 to 2. At this point, the main program moves out of the loop, it shifts a packet from the DSP56001 X-memory to the DSP56001 Y-memory and it turns on the DAC. Following that, the
main program falls into another loop reading register R6. When the value of this register is different than 0 it means that a packet was generated by the ADC. When this happens the main program shifts the packet from the DSP56001 X-memory to the DSP56001 Y-memory, it decrements R6 and it continues to implement the loop. At this stage the main program can be interrupted by process 1 or 2 at any time.

The main programs for Programs 2 and 3 are the same and their flow chart is shown in Figure 5. Note that the part of the program which runs on the DSP56001 is downloaded from the Host. Then, the program is executed on the Host. From the flow chart, we can see that, at the beginning of the program, all interrupts are disabled, the HI is initialized and the program falls into a loop waiting for the Host to get ready. Then, the program sets a flag on the HI telling the DSP56001 that the Host is ready. Initially, the flag is set at 0. When the flag changes to 1 the program initializes the SSI and to sets up the sampling rate and the buffers. Then, the program sets up the priority levels for all the processes and enables the interrupts. From this point on, process 1 is activated every time a sample is generated from the ADC. After enabling the interrupts the main program falls into a loop reading R4. As soon as a packet is generated by the ADC, R4 is incremented by 1. When this happens, the main program moves out of the loop and sets flag 2 of the HI. This will be an indication to the Host that a packet is available for transmission. Subsequently, the program falls into a loop reading the register R6. When two packets are sent by the Host to the DSP, the register R6 will be set to 2. Once this occurs, the program moves out of the loop, enables the interrupts for the DAC and falls into another loop reading R4. Each time R4 has a value different than 0, flag 2 at the HI is set to notify the Host for the availability of packets. The program executes this loop indefinitely. At this point, process 1 through 4 are at an idle state and they become active and interrupt the main program whenever they need attention.
Set up the Buffers: r0, r1, r2, r3, r4, r6 = 0, r5 = 128
m0, m1, m2, m3 = 1FF

Set up priorities for each process
Enable interrupts for process 1, 3 and 4.
(adcin, dsphost, hostdsp)

Check for interrupt requests

R4 = 0

No

No

Notify host that at least
a packet is available
(set HF2 = 1).

Yes

Yes

Enable interrupts for process 2
(enable DAC).

R6 = 2

R4 = 0

No

No

Notify host that at least
a packet is available
(set HF2 = 1).

Yes

Check for interrupt requests

Yes

Figure 5: Flow chart for the main program of programs 2 and 3
The flow chart for process 1 is shown in Figure 6. This process is activated whenever a sample is generated by the ADC. The sampling rate of the ADC is 8Khz and there are two ADCs, that is there are two channels. This implies that process 1 is activated every 0.25 ms. Every time process 1 is activated any one of the other three processes or the main program might be in progress (this is shown by the four cycles at the top of the flow chart). If process 2 is in progress when process 1 asks for attention, process 1 will not interrupt process 2 but it will wait (program counter of process 1 is saved into the system stack) for process 2 to finish before it gets access to the Central Processing Unit (CPU). When process 2 finishes execution, the program counter of process 1 will be pulled out of the system stack and process 1 will start execution. If process 3, process 4 or the main program are in progress when process 1 asks for attention any one of these processes will be interrupted and process 1 will start execution. When process 1 starts execution, it checks to determine whether the sample is from channel A or channel B. Then, process 1 continues its execution and it sets the flag OFO of the HI to 0 if the sample is from channel A and to 1 if the sample is from channel B. Subsequently, the sample is moved from the receive register of the SSI to the X-memory of the DSP56001. Initially register R5 is set to the value 128 (the desirable number of samples contained in a packet). For every sample generated, process 1 decrements R5 and checks to determine if the value of R5 is zero. When the value of R5 reaches 0 the value of register R4 is incremented by one indicating the generation of a new packet. Once a new packet is generated, the value of R5 is set back to 128. At this time the program counter returns to the interrupted process. That is, the process that was going on before it was interrupted by process 1 will continue its execution. Note that both samples from channels A and B are shifted into the same packet. This mixing of samples within the packet will not create a problem when they are sent to the DAC since the DAC knows that
every other sample in the packet is from channel A or channel B.

The flow chart for process 2 is shown in Figure 7. The interrupt process works similarly as in the case of process 1. When process 2 gets access to the CPU it checks to determine which channel requested a sample. Then, process 2 continues its execution and shifts a sample from the DSP56001 Y-memory to the SSI transmit register. At this time, the CPU time will be assigned to the process that process 2 interrupted.

The flow chart for process 3 is shown in Figure 8. This process is activated from the Host. We observe from the flow chart that process 3 can interrupt only the main program since it has equal priority with process 4 and lower priority than processes 1 and 2. When process 3 gets access to the CPU, it checks to determine whether the HOST TRANSMIT REGISTER (HTX) is empty by examining the HOST TRANSMIT DATA EMPTY (HTDE) bit of the HOST STATUS REGISTER (HSR). When this bit is high, it indicates that the HTX register is full and the program counter enters a loop waiting for this bit to go low. When this bit is low, it means that the HTX register is empty and a sample will be shifted into the HTX. This operation sets the HTDE bit. The HTDE bit is reset when the Host reads the sample from the HTX register. This procedure continues until all 128 samples are transmitted. When this is accomplished, the value of the R4 register is decremented by one designating to the main program that a packet was transmitted. Furthermore, flag 2 of the HI is reset indicating to the program that runs on the Host that there are not any packets which require transmission to the Host. If there are any such packets the main program will set flag 2 of the HI again.

The flow chart for process 4 is shown in Figure 9. This process is activated by the Host whenever the Host wants to send a packet to the DSP56001. The DSP56001 reads the packet from the HI the same way that sends it. When the first two packets
Figure 6: Flow chart of process 1
Figure 7: Flow chart of process 2
Figure 8: Flow chart of process 3

- Process 1 in progress
- Process 2 in progress
- Process 3 in progress
- Process 4 in progress
- Main Program

- Request
  - Deny

- Process 3 interrupt request

- Go through loop 128 times
  - Start loop
    - HTDE=1
      - No
      - Move sample from x-memory to Host receive register
        (movep x:(r2)+, x: <= HTX)
        - End loop
        - decrement R4
        - Return to Main Program
        - RTI
      - Yes
        - Continue Loop
are received process 4 will notify the main program that two packets have arrived by making the value of register R6 equal to 2.

The flow chart of process 5 is shown in Figure 10. On initialization this process notifies the main program on the DSP56001 that the Host is reading by setting a flag. When this flag is set, it indicates that a packet is available on the DSP56001 to be transmitted to the Host. If the flag is set, the Host will activate process 3 on the DSP56001. The Host can access one at a time, up to 32 routines on the DSP56001 by shifting a number equivalent to half the starting address of the routine into the five less significant bits of the COMMAND VECTOR REGISTER (CVR). The starting address of those subroutines is located at the first 64 memory locations of the program memory of the DSP56001. In our case (i.e., activation of process 3) the number 12 is shifted into the CVR and the MSB of the CVR is set (the final value of the CVR will be 92h). After process 3 is activated, process 5 falls into a loop reading the RXDF bit. When process 3 sends a sample to the RECEIVE BYTE REGISTERS (RXH:RXM:RXL) the RECEIVE DATA REGISTER FULL (RXDF) bit of the INTERRUPT STATUS REGISTER (ISR) goes high. When this occurs, process 5 shifts the sample from the RXH:RXM:RXL registers byte by byte into the Host memory and then, process 5 goes back to check bit RXDF and waits for another sample to arrive. Bit RXDF is cleared when data is read from the RXL register. Bit RXDF is set when data are written into the RXH:RXM:RXL registers. When all 128 samples are moved into the Host memory, the contents of the register SI is incremented indicating that one more packet has arrived in the Host memory. Process 5 continues to check the register SI. If the content of the register SI is not zero, it means that a packet is available on the Host memory to be transmitted to the DSP56001. The packet is transmitted in a similar way that it was received. When this is done, the contents of the register SI is decremented by one, indicating that a
Figure 9: Flow chart of process 4
packet was transmitted. Process 5 continues by checking bit HF2. If it is low it will check register SI. If the content of register SI is equal to zero process 5 will recheck bit HF2 and so on.
Figure 10: Flow chart of process 5

- Initialize HI
- 
  - HF2 = 1
    - Yes
      - Activate process 3 on the DSP
        (mov dx, 342h / move al, 92h / out dx, al)
        - Go through loop 128 times
      - No
        - SI = 1
          - Yes
            - Activate process 4 on the DSP
              (mov dx, 342h / move al, 94h / out dx, al)
              - Go through loop 128 times
            - No
              - TXDE = 1
                - Yes
                  - Continue Loop
                - No
                  - Move sample from the HI to the PC memory
          - No
            - TXDE = 1
              - Yes
                - Continue Loop
              - No
                - Move sample from the PC memory to the HI
            - Continue Loop
    - No
      - SI = 1
        - Yes
          - Start loop
        - No
          - RXDF = 1
            - Yes
              - Continue Loop
            - No
              - End loop

- Decrement SI
4 Apendix A: Hardware Specifications.

The DSP-56 Coprocessor board manufactured by Ariel Inc. was used in this project.

The board carries the Motorola DSP56001 chip.

DSP56001 Specifications:

- Processor: 20.5 Mhz Motorola DSP56001.
- 97.5 nsec minimum instruction cycle.
- 24-bit word width (144 dB dynamic range) with twin 56-bit accumulators.
- Single-cycle 24 x 24-bit multiplier with 56-bit product and accumulation (336 dB total dynamic range).
- Parallel data/address movement on up to seven internal busses during execution of ALU/multiplier instructions.
- 8 addressing pointers. Programmable auto-indexing supported with 8 offset registers. Module and reverse-carry addressing supported with 8 module registers.
- 62 basic instruction; no overhead DO-loops and repeated instructions are directly supported in the hardware.
- Built-in 16 and 8 bit serial ports.
- 8-bit handshaking port interfaces directly to the Host.
- 512 words of internal program RAM.
- 512 words of internal data RAM.
• 512 words of internal sine and companding ROM data.

• 15-level hardware stack.

DSP-56 specifications:

The DSP-56 board augments the function of the DSP56001 chip providing these additional capabilities:

• From 32K to a maximum of 192K words of zero wait state external memory.

• Two channels of sixteen bit analog I/O, including high performance input output stages and anti-aliasing filters.

• An industry standard SCSI disk drive interface.

• DSPnet, a versatile, multimaster 24 bit wide expansion bus for interconnecting DSP boards.

• Single bit auxiliary I/O through rear panel.

For more information about the DSP56001, references [3] and [4] are recommended. For more information about the DSP-56 board, reference [2] is recommended.
5 Appendix B: Software Packages
Program # 1: This program receives data from the ADC and stores them in the dsp x-memory. From the dsp x-memory it shifts them, packet by packet, to the dsp y-memory, and from the dsp y-memory to the DAC.

include 'ioequ.asm'

ORG P:0
jmp progrl

nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop

jsr <adcinc
nop

jsr <adcinc
nop

jsr <dacout
nop

jsr <dacout
nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop
nop
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

progr1

ORI  #$03,MR
movep  #$245E,X:<<M_BCR
ORI  #$80,OMR

; Initialize host interface

MOVEP  #1,X:<<M_PBC
MOVEP  #$04,X:<<M_HCR

; initialize SSI interface

movep  #$4100,X:<<M_CRA
movep  #$AA04,X:<<M_CRB
movep  #$1F8,X:<<M_PCC
movep  #$3800,X:<<M_IPR

; Set up 8 KHz sample rate
move  #$20,a
    move  a,Y:$FF0
    bset  #0,x:<<M_CRB
    ANDI  #$FC,MR

; Set up buffers

    MOVE  #$0,R0
    MOVE  #$0,R1
    MOVE  #0,R2
    MOVE  #0,R3
    MOVE  #$1FF,M0
    MOVE  #$1FF,M1
    MOVE  #$1FF,M2
    MOVE  #$1FF,M3
    MOVE  #0,R4
    MOVE  #$1FF,M4
    MOVE  #128,R5
    MOVE  #$1FF,M5
    MOVE  #0,R7
    MOVE  #$1FF,M7

; Wait for a packet to arrive before enable the DAC.

LODAC
    CLR   A
    move  #0,X0
    MOVE  R6,X0
    CMP   X0,A
    JEQ   LODAC

; Shift packet

    DO    #128,SH11
        MOVE  X:(R7),A0
        MOVE  A0,Y:(R7)+
        NOP

SH11
    RND   B (R6)-

LODAC1
    CLR   A
    move  #0,x1
    move  r6,x1
    cmp   x1,a
    jeq   LODAC1
    do    #128,sh22
    move  X:(r7),a0
    move  a0,y:(r7)+
    nop

sh22
    RND   B (R6)-

; Enable the DAC.

    BSET  #$C,X:<<M_CRB
    BSET  #$E,X:<<M_CRB

LO7
    CLR   A
    MOVE  R6,X0
; PROCESS 2:
; Interrupt service routine to send data to the DAC from the DSP
; Y-memory

dacout
jset  #2,x:<<M_SR,channelA
nop
movep y:(r1)+,x:<<M_TX
nop
riti

; PROCESS 1:
; Interrupt service routine to receive data from the ADC and sent
; them to the DSP X-memory. Also this routine increment the number
; of packets available in the DSP X-memory.

adcin
jset  #3,x:<<M_SR,Chann_A
nop
bset  #0,x:<<M_CRB
nop
movep x:<<M_RX,x:(r0)+
RND  B  (R5)-
CLR  B
MOVE  R5,Y0
CMP  Y0,B
JNE  ad11
MOVE  #128,R5
RND  B  (R6)+
ad11  nop
riti

Chann_A
bclr  #0,X:<<M_CRB
nop
movep x:<<M_RX,x:(r0)+
nop
RND  B  (R5)-
CLR  B
MOVE  R5,Y1
CMP  Y1,B
JNE  ad22
MOVE  #128,R5
RND  B  (R6)+
ad22  nop
rti
END
Program # 2: This program receives data from the ADC and stores them in the DSP x-memory. From the DSP x-memory it shifts data, packet by packet, to the host interface. It also reads data from the host interface and stores them in the DSP y-memory. From the DSP y-memory it shifts data, byte by byte, to DAC. This program works in conjunction with process 5. Process 5 is used to read data from the host interface and store them in the host memory. Process 5 also reads data from the host memory and sends them to the host interface.

```
include 'ioequ.asm'
ORG P:0
jmp progrl

nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
jsr <adcin
nop
jsr <adcin
nop
jsr <dacout
nop
jsr <dacout
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
```
nop
nop
nop

jsr <dsphost ;($24)
nop

rti
nop

jsr <hostdsp ;($28)
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

rti
nop

progr1

ORI #$03,MR
movep #$245E,X:<<M_BCR
ORI #80,OMR

; Initialize host interface

MOVEP #1,X:<<M_PBC
MOVEP #$04,X:<<M_HCR

; Wait for the host to get reading

Noready NOP
JCLR #$4,X:<<M_HSR,Noready

; initialize SSI interface

movep #$4100,x:<<M_CRA
movep #$AA04,x:<<M_CRB
movep #$1F8,x:<<M_PCC
movep #$3800,x:<<M_IPR

; Set up 8 Khz sample rate

move #$20,a
move a,Y:$FF0
bset #0,x:<<M_CRB
ANDI #$FC,MR

; Set up buffers

MOVE #$0,R0
MOVE #$0,R1
MOVE #0,R2
MOVE #0,R3
MOVE #$1FF,M0
MOVE #$1FF,M1
MOVE #$1FF,M2
MOVE #$1FF,M3
MOVE #0,R4
MOVE #128,R5
MOVE #$1FF,M6
MOVE #0,R6

; Wait for a packet to arrive before enable the DAC.

LODAC

CLR A
MOVE R4,X0
CMP X0,A
NOP
JEQ L012
BSET #3,X:<<M_HCR

L012 NOP
RND B (R6)+
CLR A
move #2,a1
MOVE R6,X1
NOP
CMP X1,A
JNE LODAC

; Enable the DAC.

BSET #$C,X:<<M_CRB
BSET #$E,X:<<M_CRB

; Wait for interrupts. Also notify the host in case that a packet is available in the DSP X-memory.
; PROCESS 3:
; Interrupt service routine to send data to the host from the DSP
; X-memory. Also this routine decrement the number of packets
; available to the DSP X-memory.

dsphost
NOP
DO    #128,LO3
NOP
NOP

LO1
JCLR  #1,X:<M_HSR,LO1
NOP
MOVEP X:(R2)+,X:<M_HTX
NOP
NOP

LO3
BCLR  #3,X:M_HCR
RND   B (R4)-
RTI

; PROCESS 4:
; Interrupt service routine to receive a packet from the host
; and store it in the DSP Y-memory.

hostdsp
NOP
DO    #128,LO4
NOP
NOP

LO5
JCLR  #0,X:<M_HSR,LO5
NOP
MOVEP X:<M_HRX,Y:(R3)+
NOP
NOP

LO4
MOVE  R6,X1
CLR   A
MOVE   #2,A1
NOP
CMP   X1,A
JEQ   LOO
RND   B (R6)+

LOO
NOP
RTI

; PROCESS 2:
; Interrupt service routine to send data to the DAC from the DSP
; Y-memory

dacout
jset   #2,x:<M_SR.channelA
; PROCESS 1:
; Interrupt service routine to receive data from the ADC and sent
; them to the DSP X-memory. Also this routine increment the number
; of packets available in the DSP X-memory.

adcin  jset  #3,x:<<M_SR,chann_A
       nop
       bset  #0,x:<<M_CRB
       nop
       movep x:<<M_RX,x:(r0)+
       RND   B  (R5)-
       CLR   B
       MOVE  R5,Y1
       CMP   Y1,B
       JNE   ad11
       RND   B  (R4)+
       MOVE  #128,R5
       nop
       rti

chann_A
bclr  #0,x:<<M_CRB
       nop
       movep x:<<M_RX,x:(r0)+
       RND   B  (R5)-
       CLR   B
       MOVE  R5,Y1
       CMP   Y1,B
       JNE   ad22
       RND   B  (R4)+
       MOVE  #128,R5
       nop
       rti

END
PROCESS 5: It reads data from the host interface packet by packet and store them in the host memory. It also reads packets from the host memory and send them to the host interface.

include in1.asm

@kbdchk macro
mov ah,0bh
int 21h
endm

MODEL SMALL
STACK 100h

GROUP DATA, RCODE

DATA SEGMENT WORD PUBLIC
buffer DB 256 DUP(?)
DATA ENDS

RCODE SEGMENT WORD PUBLIC
assume cs:code, ds:code

start:
; Initialize Host Interface (PC Side)
mov al,10h ;Set flag HFI high to notify the dsp
mov dx,icr ;that the host is reading.
out dx,al ;Send 10h at port 340h.
mov al,1 ;Initialize the Command Vector Register.
mov dx,ivr ;
out dx,al ;Send 1 to port 341h
mov al,3 ;Initialize the Interrupt Vector Register.
mov dx,ivr ;
out dx,al ;Send 3 to port 233h
mov ax,cs ;
mov ds,ax ;Set the number of packets available on
mov si,0 ;the host memory to zero.

Repeat1:
chkpk3:
@kbdchk
or al,al
jz hea
jmp dos1

; Check to see if there is a packet waiting in the DSP RAM.

hea: mov dx,isr ;Read the Interrupt Status Register
in al,dx ;at port 342h.
test al,1000b ;If bit 4 of isr is set, the dsp has
jz continuel ;one or more packets for the host.

; If there is a packet available on the DSP memory,
; transfer the packet from the DSP RAM to the HOST RAM.
; First, access the Interrupt service routine (DSPHOST) on the DSP.

loll: mov dx,cvr ;Reset the CVR by sending zero to
mov al,0 ;the port 343h.
out dx,al
mov al,92h ;Access the DSPHOST routine by shifting
out dx,al ;12h in the Host Vector and at the same
mov cx,10 ;time sending the HC bit.
hold1: loop hold1 ;Wait for the DSP to accept the command.
in al,dx ;Check to see if the DSP has accepted the
test al,10000000b ;command. If it didn't try again.
jnz loll ;If HC bit is still high jump to loll.

; Read data sent by the DSPHOST routine.

mov di, OFFSET buffer ;Set the buffer OFFSET.

loop1: mov dx, isr ;Check the first bit of the ISR,
in al,dx ;if it is high the dsp has sent data
test al,1b ;to one or all ports 345h,346h,347h.
jz loop1 ;Otherwise, wait for the DSP to send data.
mov dx,rxh ;Read the contents of port 345h and shift it
in al,dx ;into the accumulator Al.
mov [di],al ;Shift the contents of al to the memory location
inc di ; pointed by di. Increment di.
mov dx,rxm ;Read the contents of port 346h
in al,dx ;and store it into memory location
mov [di],al ;pointed by di.
inc di ;Increment di.
mov dx,rxl ;Read the contents of the port 347h to let
in al,dx ;the dsp know that the data where red by

loop loop1 ;the host. Repeat the loop until the
inc si ;packet is transferred. Increment
;the number of packets available on the host

continuel:

hold: loop hold ;
; check to see if there are any available packets in the HOST RAM
; to be send to the DSP RAM.
cmp si,0 ;
JE Repeat1

; Access interrupt service routine (HOSTDSP) in the DSP.
lo22: mov dx, cvr ;Reset the CVR by sending zero to
mov al,0 ;the port 343h.
out dx,al
mov al,94h ;Access the HOSTDSP routine by shifting
out dx,al ;14h in the Host Vector and at the same
mov cx,10 ;time sending the HC bit.

hold2: loop hold2 ;Check to see if the DSP has accepted the
in al,dx ;command. If it didn't try again.
test al,10000000b ;If the HC bit is still high jump to lo22.
jnz lo22

; Set data to host transmit register.
mov di, OFFSET buffer ;Set the buffer OFFSET.

loop2: mov dx, isr ;Check to see if the host transmit register
in al,dx ;is empty. If it is not wait to get empty.
test al,10b
jz loop2
mov al,[di] ;Shift the contents of memory location
inc di ;pointed by di into the register al and inc
mov dx,txh ;di. Move the contents of al into the most
out dx,al ;significant byte of the host transmit
mov al,[di] ;register.
inc di
mov dx,txm ;Move the contents of al into the medium
out dx,al ;byte of the host transmit register.
mov al,0 ;move zero into al.
mov dx,tx1 ;move the contents of al into the less
out dx,al ;significant byte of the host transmit

loop
loop2 ;register.
dec si ;decrement si (Si indicates how many packet
jmp repeat1 ;are left in the host memory.
dos1: nop
mov ax,4C00h
int 21h
RCODE
ENDS
END start
#include <conio.h>
#include <stdio.h>

int ICR = 0x340;
int CVR = 0x341;
int ISR = 0x342;
int IVR = 0x343;
int RTXH = 0x345;
int RTXM = 0x346;
int RTXL = 0x347;

unsigned int dx = 0;
char *packet;
int OFFSET = 0,
PKTSIZE = 128;

main () {
    char al,byte;
    int count,i,j;
    int pktin;

    initdsp();
    pktin = 0;

    packet = (char *) malloc(PKTSIZE*2);

    while (!kbhit()) {
        dx = ISR;
        if (((al=inp(dx) & 0x08) == 1) {
            dx = CVR;
            do {
                outp(dx,0);
                outp(dx,0x92);
                for (count=0; count<10; count++);
            } while (((al=inp(dx) & 0x80) != 0);

            readsp();
            pktin = 1;
        }
        else if (pktin) {
            dx = CVR;
            do {
                outp(dx,0);
                outp(dx,0x94);
                for (count=0; count<10; count++);
            } while (((al=inp(dx) & 0x80) != 0);

            writedsp();
            pktin = 0;
        }
    }
}
free((char *) packet);
} /* main-rwport */

initdsp()
{
    dx = ICR;
    outp(dx,0x10);
    dx = CVR;
    outp(dx,1);
    dx = IVR;
    outp(dx,3);
} /* initdsp */

readsp()
{
    char    al;
    int     i,j;

    for (i=0; i<PKTSIZE*2; i+2) {
        while (al = inp(ISR) & 0x01 != 1);
        al = inp(RTXH);
        packet[i] = al;
        al = inp(RTXM);
        packet[i+1] = al;
        al = inp(RTXL);
    }
} /* readsp */

writedsp()
{
    char    al;
    int     i;

    for (i=0; i<PKTSIZE*2; i+2) {
        while (al = inp(ISR) & 0x02 != 2);
        al = packet[i];
        outp(RTXH,al);
        al = packet[i+1];
        outp(RTXM,al);
        al = 0;
        outp(RTXL,al);
    }
} /* writedsp */
; STDDSP.ASM - Program 3.
; A DSP program implements voice samples on the DSP board and communicates
; with Host computer.
; Program to read data form channel B(input) to the DSP memory and from the
; DSP memory to the host and vice versa to channel B(output).
; include '\dsp56\ioequ.asm'

ORG P:0
jmp progr1

nop
nop
nop
nop
nop
nop
nop
nop
nop
jsr <adcin
nop
jsr <adcin
nop
jsr <dacout
nop
jsr <dacout
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
jsr <dsphost ;(24)
nop
jsr <chgptlen ;(26)
nop
jsr <hostdsp ;(28)
nop
jsr <chgoffset ;(2A)
nop
rti
nop
jsr <passcount ;(2E)
nop
rti
nop
rti
nop
rti
nop
rti
nop
rti
nop
rti
nop
rti
nop
progrl
ORI #$03,MR
movep #$245E,X:<<M_BCR
ORI #80,OMR

; Initialize host interface

MOVEP #1,X:<<M_PBC
MOVEP #$04,X:<<M_HCR

; Wait for the host to get reading

Noready
NOP
JCLR #$4,X:<<M_HSR,Noready

; initialize SSI interface
movep  #$4100,x:<<M_CRA
movep  #$AA04,x:<<M_CRB
movep  #$1F8,x:<<M_PCC
movep  #$3800,x:<<M_IPR

; Set up 8 Khz sample rate
move  #$50,a
move  a,Y:$FF00
bset  #0,x:<<M_CRB
ANDI  #$FC,MR

; Set up buffers

MOVE  #$0,R0
MOVE  #$0,R1
MOVE  #0,R2
MOVE  #0,R3
MOVE  #$1FF,M0
MOVE  #$1FF,M1
MOVE  #$1FF,M2
MOVE  #$1FF,M3
MOVE  #0,R4
MOVE  #128,R5
MOVE  #0,R7
MOVE  #$1FF,M7
MOVE  #0,R6
CLR   A
MOVEM A1,P:counter
MOVEM A1,P:counter1
MOVEM A1,P:counter2
MOVEM A1,P:seqnum
MOVEM A1,P:sum
MOVE  #129,A1
MOVEM A1,P:pktlen
MOVEM P:pktlen,A1
MOVEM A1,P:adccount
MOVEM R0,P:safer0
BCLR  #4,X:<<M_HCR

; Wait for a packet to arrive before enable the DAC.

LODAC

CLR   A
MOVE  R4,X0
CMP   X0,A
JEQ   L012
BSET  #3,X:<<M_HCR
NOP
CLR   A
MOVE  R6,X0
CMP   X0,A
JEQ   LODAC

; Enable the DAC.

BSET  #$C,X:<<M_CRB
BSET  #$E,X:<<M_CRB

; Wait for interrupts. Also notify the host in case that a packet is
; available in the DSP X-memory.
;
LO7
    CLR A
    MOVEL P:counter2,R7 ;Ma
    MOVE R7,Y0 ;Ma
    CMP Y0,A ;Ma
    JNE noloadcnt ;Ma
    MOVEL P:counter1,R7 ;Ma
    MOVEL R7,P:counter2 ;Ma
;noloadcnt
    MOVEL P:counter,R7 ;Ma
    MOVE R7,Y0 ;Ma
    CMP Y0,A ;Ma
    JEQ BSET ;Ma
    MOVEM P:counter2,R7 ;Ma
    MOVEM P:counter1,R7 ;Ma
    JNE noloadcnt ;Ma

;waitHF2
    JCLR #3,X:<<M_HCR,LO6 ;Ma, wait for flag HF2 reset
    JMP waitHF2 ;Ma

LO6
    NOP
    BCLR #3,X:<<M_HCR ;Ma, reset flag HF2

; Interrupt service routine to send data to the host from the DSP
; X-memory. Also this routine decrement the number of packets
; available to the DSP X-memory.

dsp

    MOVEL A1,P:saveal ;Ma
    NOP
    MOVEL P:pktlen,A0 ;Ma
    DO A0,LO3 ;Ma
    DO #128,LO3
    NOP
    NOP

    L01
    JCLR #1,X:<<M_HSR,LO1
    nop
    MOVEM X:(R2)+,X:<<M_HTX
    NOP
    NOP
    NOP

    L03
    MOVEL P:counter,R4 ;Ma, load number packet
    NOP
    RND B (R4)- ;Ma
decrement # packet
    CLR B ;Ma
    MOVE R4,B1 ;Ma
    TST B ;Ma
    JES JES
    MOVEL R4,P:counter ;Ma, save # packet
    JMP noersetcnt0

;setcnt0
    MOVE #0,R4 ;Ma

;noersetcnt0
    BCLR #3,X:<<M_HCR
    NOP
    NOP
; Interrupt service routine to receive a packet from the host
; and store it in the DSP Y-memory.

dsp
  MOVEM A1,P:saveal  ;Ma
  NOP
  MOVEM P:pktlen,A0  ;Ma
  DO A0,LO4
  ; DO #128,LO4
  NOP
  NOP
  L05
  JCLR #0,X:<<<M_HSR,LO5
  NOP
  MOVEP X:<<<M_HRX,Y:(R3)+
  NOP
  NOP
  NOP
  L04
  MOVE #3,R6
  MOVEM P:saveal,A1  ;Ma
  RTI

; Interrupt service routine to receive the packet length from the host
; and store it into P:pktlen.

dl
  MOVEM A1,P:saveal  ;Ma
  JCLR #0,X:<<<M_HSR,LOc
  NOP
  MOVEP X:<<<M_HRX,A1
  MOVEM A1,P:pktlen  ;Ma

; compute bits of shifting
  MOVE #0,R4  ;Ma
  CLR A
  MOVEM P:pktlen,A1  ;Ma
  CLR B

loopsh
  CMP B,A  ;Ma
  JEQ ENDSHCNT
  NOP
  RND B (R4)+  ;Ma
  LSR A  ;Ma
  JMP LOOPSH

endshcnt
  NOP
  RND B (R4)-  ;Ma, decreament 1
  MOVEM R4,P:numshift
  MOVEM P:saveal,A1  ;Ma
  RTI

; Interrupt service routine to receive the noise level from the host
; and store it into P:offset.
chgoffset
LO8

MOVEM A1,P:saveal ;Ma
JCLR #0,X:<<M_HSR,LO8 ;Ma
NOP ;Ma
MOVEP X:<<M_HRX,A1 ;Ma
MOVEM A1,P:offset ;Ma
NOP ;Ma
NOP ;Ma
NOP ;Ma
MOVEM P:saveal,A1
RTI ;Ma

; Interrupt service routine to pass the number of voice packet to HOST
; and signal HOST to read a packet if counter is not 0

count

MOVEM A1,P:saveal ;Ma
JCLR #1,X:<<M_HSR,LOPASH ;Ma, wait for HTDE set
NOP ;Ma
MOVEM P:counter,X0 ;Ma
MOVEP X0,X:<<M_HTX ;Ma
NOP ;Ma
NOP ;Ma
NOP ;Ma
MOVEM P:saveal,A1
RTI ;Ma

; Interrupt service routine to send data to the DAC from the DSP
; Y-memory

dacout

BSET #4,X:<<M_HCR ;Ma, set flag HF3
MOVEM A1,P:saveal ;Ma
JCLR #2,x:<<M_SR,channelA
NOP
MOVEM P:saveal,A1
RTI ;Ma

channelA

BCLR #4,X:<<M_HCR ;Ma, reset flag HF3

; Interrupt service routine to receive data from the ADC and sent
; them to the DSP X-memory. Also this routine increment the number
; of packets available in the DSP X-memory.

dadin

BSET #4,X:<<M_HCR ;Ma, set flag HF3
MOVEM A1,P:saveal ;Ma
JSET #3,x:<<M_SR,channel_A
NOP

jset #0,x:<<M_SR,channel_A
nop
movep  x :< M_RX,x:(r0)+
MOVEM  P:adccount,R5
NOP
RND  B  (R5)-
CLR  B
MOVE  R5,Y0
CMP  Y0,B
JNE  ad11

MOVEM  P:counter,R4
NOP
RND  B  (R4)+
MOVEM  R4,P:counter
MOVEM  P:pktlen,R5

CLR  B
movep  x :< M_RX,B1
MOVE  B1,X:(R0)+
ABS  B
CLR  A
MOVEM  P:sum,A1
ADD  B,A
MOVEM  A1,P:sum
MOVEM  P:adccount,R5
NOP
RND  B  (R5)-
CLR  B
MOVE  #1,B1
MOVE  R5,X0
CMP  X0,B
JNE  ad11

MOVEM  P:sum,B1
REP  #8
LSL  B

MOVE  B1,X:(R0)+
CLR  B
MOVEM  B1,P:sum
MOVEM  P:numshift,R4
REP  R4
LSR  A
MOVEM  P:offset,B1
CMP  B,A
JCC  PKTOK

CLR  B
MOVEM  B1,P:seqnum
MOVEM  P:counter,R4
MOVEM  R4,P:counter1
MOVEM  B1,P:counter
MOVEM  P:saver0,R0
JMP  ad10

PKTOK

MOVEM  P:seqnum,R4
NOP
RND  B  (R4)+
MOVEM  R4,P:seqnum

;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
;Ma
; MOVE R4,B1
; REP #8
; LSL B
; MOVE B1,X:(R0)+
; MOVEM R0,P:saver0
; MOVEM P:counter,R4
; NOP
; RND B (R4)+
; MOVEM R4,P:counter
;ad10
; MOVEM P:pktlen,R5
ad11
nop
MOVEM R5,P:adccount
MOVEM P:saveal,A1
BCLR #4,X:<<M_HCR
rti
chann_A
movep x:<<M_RX,b0
nop
bset #0,X:<<M_CRB
MOVEM P:saveal,A1
BCLR #4,X:<<M_HCR
rti
; Data area in P-space
adccount DS 1 ;Ma, A/D input counter
counter DS 1 ;Ma, packet counter for +
counter1 DS 1 ;Ma, packet counter for +
counter2 DS 1 ;Ma, packet counter for cnt
counter3 DS 1 ;Ma, packet counter for cnt
offset DS 1 ;Ma, noise magnitude
pktlen DS 1 ;Ma, packet length
numshift DS 1 ;Ma, bits of shift
saveal DS 1 ;Ma, save al
saver0 DS 1 ;Ma, save R0'
segunum DS 1 ;Ma, packet sequent number
sum DS 1 ;Ma, summation of a packet's
END ;Ma, values
INTVOICE.C: Process 5, a mixed language program, C language part, transmits and receives voice packets.

Description: This file contains the code which calls the functions provide by the ist503c.lib to receive/transmit packets through 3COM EtherLinkII board. This program has two buffers to receive voice and data from the 3COM EtherLinkII board. This program integrates the DSP data and 3COM packet. It can transmit and receive voice packets to/from the network.

Packet header format:

1. Destination address - 6 bytes
2. Source address - 6 bytes
3. Packet data length - 2 bytes
4. Packet type - 1 byte

C include files

#include <conio.h>
#include <dos.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

Application include files

#include "lan_c.h"

3COM interface subroutines

extern cInitAdapters();
extern cInitParameters();
extern cResetAdapter();
extern cWhoAmI();
extern cRdRxFilter();
extern cWrRxFilter();
extern cPutTxData();
extern cGetRxData();
extern cSetLookAhead();
extern cXmit1();

extern cInitBufPtr();
extern cPassHead();
extern cVPtrarray();
extern cGetVStrtptr();
extern cGetVEndptr();
extern cDPtrarray();
extern cGetDStrtptr();
extern cGetDEndptr();
extern cGetOneVPkt();
extern cGetOneDPkt();
extern cResetVPtr();
extern cResetDPtr();
extern cGetPkttrxPtr();

/* Time stamping subroutines */
extern cGetTimeCount();
extern cGettimeptr();

/* DSP interface subroutines */
extern cinitdsp();
extern cgetvpktadd();
extern cpasvtrxadd();
extern csetvpktlen();
extern csetnoise();
extern csetvavenum();
extern cinitnoise();
extern int cvpktavailable();
extern int cwaitrdsp();
extern creadsp();
extern cDumpDsp();
extern cDumpSilent();

#define LINKSIZE 768
#define RANDMAX 32767
#define MAXDATLEN 980
#define SAMPLESIZE 2

int handler();

struct exception {
    int type;
    char *name;
    double arg1, arg2;
    double retval;
} *x;

int vpktsize;
char *errorptr;
char far *vrcvpktptr; /* DSP -> HOST */
unsigned long far *timeptr;

main()
{
    int i, j;
    int count;
    int Vindex, Dindex, numpkt, pktlen;
    int ttlpl, nb, flags, reqid, nreqid;
    int randelay, timetype, voicepkt;
    int notexit = 1, rc, rs = 0;
    int invcnt, indcnt, outvcnt, outdcnt;
    int vhdlen = 15;

    unsigned int newoffset;
    unsigned char hibyte, lobyte;
    char rcvtrrx;

    /* DSP voice variable */
int first,Spklen,lenspace;
int vpktin,vpktlen,vpktok,vsamples;
int vrcvseqnum,vrcvseqnew,vtrxseqnum,vavenum;
unsigned int vnoise,unsnoise;
unsigned long totvpkt;

/* Data & voice ratio variables */
float data_voice_part,rate,vlostrate,lostrate;
float DATARATE = 0;

/* Voice reconstruction variables */
char far *fillinptr;
int trxtype;
float maxtrxdelay;
float NTITIME,ITITIME,CTITIME;
unsigned long ititime,NTIdelay,trxdelay;
unsigned long trx1time,trx2time,rcv1time,rcv2time;

/* Set to interrupt calls " handler" */
signal(SIGFPE,handler);
vpktin = totvpkt = 0;
invcnt = indcnt = 0;
outvcnt = outdcnt = 0;
init_all();  /* initialize 3COM board */
cinitdsp();  /* initialize DSP board */
printf("Input the voice packet length: ");
scanf("%d",&vpktsize);
printf("\n");
vsamples = vpktsize/SAMPLESIZE;
csetvpktlen(&vsamples);
cgetvpktadd(&vrcvpktptr);
for (i=0; i<6; i++) {
  Pkttrx[i] = 255;
  vrcvpktptr[i] = 255;
}
for (i=0; i<6; i++) {
  Pkttrx[i+6] = Who->addr[i];
  vrcvpktptr[i+6] = Who->addr[i];
}
flags = 0x0060;
reqid = 0x0001;
rreqid = 0x0011;
first = 1;

while (notexit) {
  notexit = main_menu();
  if (notexit == 1) {
    rcvtrx = 't';
    printf("Input the voice packet averaging number: ");
    scanf("%d",&vavenum);
printf("\n");
csetvavenum(&vavenum);
rc = cinitnoise();
printf("Initnoise = %d\n",rc);
printf("Input the voice noise level: ");
scanf("%d",&vnoise);
printf("\n");
csetnoise(&vnoise);
}
if (notexit == 2) rcvtrx = 'r';

while (!kbhit() && notexit) {
    if (rcvtrx == 'r') {
        numpkt = VBuflinkptr[0];
        if (numpkt > 0) {
            invcnt++;
            cGetOneVPkt(&Pktrcv);
            /* New */
            hibyte = (unsigned char) Pktrcv[12];
            lobyte = (unsigned char) Pktrcv[13];
            pktlen = hibyte*256+lobyte;
            /* change voice length at receiver side */
            if (((pktlen-vhdlen)/SAMPLESIZE != vsamples && first) {
                vsamples = (pktlen-vhdlen)/SAMPLESIZE;
                csetvpktlen(&vsamples);
                first = 0;
            }
            /* Dump voice packet to DSP board */
            cpasvtrxadd(Pktrcv+vhdlen);
            cDumpDsp();
            } else {
            /* checking if there is a data packet */
            numpkt = DBuflinkptr[0];
            if (numpkt > 0) {
                indcnt++;
                cGetOneDPkt(&Pktrcv);
            }
            if ((numpkt = DBuflinkptr[0]) == 0) cResetDPtr();
        }
    }
    if (rcvtrx == 't') {
        vpktok = cvpktavailable();
        vpktok = (vpktok & 0x08 &! (vpktok & 0x10));
        data_voice_part = (float) rand()/RANDMAX;
        if (data_voice_part >= DATARATE || vpktok) {
            if (vpktok) {
                totvpkt++;
                while (rc=cwaitrdsp() != 1);
if (rc = creadsp() == 1) vpktin = 1;
else vpktin = 0;
}

if (vpktin) {
    vpktin = 0;
    pktlen = vpktsize+vhdlen;
    hibyte = pktlen/256;
    lobyte = pktlen-(int) hibyte*256;
    vrcvpktptr[12] = hibyte;
    vrcvpktptr[13] = lobyte;
    vrcvpktptr[14] = 0x0f;
    outvcnt++;
    rc=cXmitl(pktlen,pktlen,flags,reqid,vrcvpktptr,&nreqid);
}
else {
    rate = (float) rand()*MAXDATLEN/RANDMAX;
    pktlen = (int) rate+64+vhdlen;
    hibyte = pktlen/256;
    lobyte = pktlen-(int) hibyte*256;
    Pkttrx[12] = hibyte;
    Pkttrx[13] = lobyte;
    Pkttrx[14] = 0x0d;
    for (i=0; i<pktlen; i++)
        Pkttrxptr[i] = Pkttrx[i];
    outdcnt++;
    rc=cXmitl(pktlen,pktlen,flags,reqid,Pkttrxptr,&nreqid);
}
}
rc=cResetAdapter();
printf("cResetAdapter returns %d\n",rc);
printf("Total voice packet received: %d\n",invcnt);
printf("Total data packet received: %d\n",indcnt);
printf("Total packets received: %d\n",invcnt+indcnt);
printf("\n");
printf("Total voice packet transmitted: %d\n",outvcnt);
printf("Total voice packet read from DSP: %ld\n",totvpkt);
printf("Total data packet transmitted: %d\n",outdcnt);
printf("Total packets transmitted: %d\n",outvcnt+outdcnt);
} /* main-intvoice */

init_parameter()
{
    parmsdr->len=0x17;
    parmsdr->non1=0x00;
    parmsdr->non2=0x00;
    parmsdr->non3[0]=0x00;
    parmsdr->non3[1]=0x00;
    parmsdr->non4[0]=0x00;
    parmsdr->non4[1]=0x00;
parmsdr->non4[2]=0x00;
parmsdr->non4[3]=0x00;
parmsdr->non5[0]=0x00;
parmsdr->non5[1]=0x00;
parmsdr->non5[2]=0x00;
parmsdr->non5[3]=0x00;
parmsdr->non6=0x00;
parmsdr->cdend[0]=0x00;
parmsdr->cdend[1]=0x00;
parmsdr->cdend[2]=0x00;
parmsdr->cdend[3]=0x00;
/* parmsdr->argo = "c:\3com\ether503.sys /a:2e0/m:4/t:1/d:1/i:3\n"; */
parmsdr->argo = "c:\\3com\\ether503.sys /A:2e0 /D:1 /I:3\0x0a";
parmsdr->args=getds();
parmsdr->non7=0x00;
} /* init_parameter */

init_all()
{
    int rc, rxf=0x000c, rrrxf, Adapters=0;

    init_parameter();
    /* */
    rc=getds();
    printf("getds 0x%x\n",rc);
    /* */
    rc=cInitParameters(parmsdr);
    printf("cInitParameters returns %d\n",rc);
    rc=cInitAdapters(&Adapters);
    printf("cInitAdapters returns %d, Adp=%d\n",rc, Adapters);

    rc=cSetLookAhead(32);
    printf("cSetLookAhead returns %d\n",rc);

    rc=cWhoAmI(&Who);
    printf("cWhoAmI returns %d\n",rc);
    printf("addr = %02x %02x %02x", Who->addr[0],
             Who->addr[1], Who->addr[2]);
    printf(" %02x %02x %02x\n", Who->addr[3],
             Who->addr[4], Who->addr[5]);
    printf("ver major %02x ver minor %02x\n", Who->ver_major, Who->ver_minor);
    printf("transfer mode %x wait mode %x\n", Who->xfr_mode, Who->wait_mode);
    printf("ttl recp cnt %d (0x%4x)\n", Who->ttl_recp_cnt, Who->ttl_recp_cnt);

    rc=cWrRxFilter(rxf);
    printf("cWrRxFilter returns %d\n",rc);
    rc=cRdRxFilter(&rrxf);
    printf("cRdRxFilter returns %d, filter=%x\n",rc,rrxf);

    rc=cInitBufPtr();/* Ma */
    printf("cInitBufPtr returns %d\n",rc); /* Ma */

    rc = cPassHead(&Hdptr); /* Ma */
    printf("Header address is %04x\n",Hdptr->inh); /* Ma */

    cVPtrarray(&VBuflinkptr); /* Ma */
    cGetVStrtptr(&Vptrstrt); /* Ma */
    cGetVEndptr(&Vptrend); /* Ma */
    cDPtrarray(&DBuflinkptr); /* Ma */
    cGetDStrtptr(&Dptrstrt); /* Ma */
cGetDEndptr(&Dptrend);         /* Ma */
cGettimeptr(&timeptr);          /* Ma */
cGetPktrxPtr(&Pktrxptr);         /* Ma */
farvptr.lw.segoff = farvqptr.lw.segoff = (unsigned long int) VBuflinkptr+6;
fardptr.lw.segoff = fardqptr.lw.segoff = (unsigned long int) DBuflinkptr+6;
printf("Far pointer Vptrstrt = %lx\n",Vptrstrt);
printf("Far pointer Pkttrxptr = %lx\n",Pkttrxptr);
printf("Pointer Pkttrx = %x\n",Pkttrx);
farvqptr.lw.segoff = farvptr.lw.segoff;
}/* init_all */

int main_menu()
{
    int select;
    printf("\n\n Voice Reconstruction Experiments: \n\n");
    printf("0. Exit\n\n");
    printf("1. Transmission\n\n");
    printf("2. Reception\n\n");
    printf("\n Enter selection number: ");
    scanf("%d", &select);
    printf("\n");
    return(select);
} /* main_menu */

int handler()
{
    int rc;
    struct exception *M;
    printf("Math error - Divided by zero: %s\n", errorptr);
    rc = matherr(M);
    printf("The math routine is: %s %lf %lf\n", M->name, M->arg1, M->arg2);
    rc = cResetAdapter();
    abort();
} /* handler */

sysbeep()
{
    printf("\07");
} /* sysbeep */

int CheckHead()
{
    char rc;
    rc = Hdptr->inh[14];
    switch (rc) {
    case 0x00: /* voice packet type */
        rc = 1;
        break;
    case 0x0d: /* data packet type */
        rc = 2;
        break;
    default: /* unknown packet type */
        rc = 0;
        break;
    }
    return(rc);
} /* CheckHead */
;***********************************************************************
INTDSP.ASM: Process 5, a mixed language program, Microsoft assembler part,
which interfaces the DSP board and Host computer.

The DSP program is STDDSP.ASM and the Host computer program
is INTVOICE.C
;***********************************************************************

.286

;public _getds

public _cinitdsp
public _cgetvpktadd
public _cpasvtrxadd
public _csetvpktlen
public _csetvavenum
public _csetnoise
public _cvpktavailable
public _cinitnoise
public _cwaitrdsp
public _creadsp
public _cDumpDsp
public _cDumpSilent

public vtrxptr

extrn pklock :byte

include portadds.asm

vhdlen equ 21
PKTLENG equ 500
lf equ 0ah
cr equ 0dh
waitqty equ 20
waitqtyl equ 1

@print macro strloc          ;print string at strloc
local strloc
push cx
lea dx,strloc
mov ah,09h
int 21h
pop cx
endm

@kbdin macro                  ;get kbd char in al
mov ah,8
int 21h
endm

@kbdchk macro                 ;check for kbd char
mov ah,0bh
int 21h
endm

@prx macro                  ;print hex data in word dat, len = 1 to 4
mov ax,len
push ax

;don’t put data in ax
mov ax, dat
push ax
call prx
add sp, 4
endm

CODE GROUP _TEXT, DATA, ICODE

_TEXT segment byte public 'CODE'

DGROUP group _DATA, _BSS
assume cs:_TEXT, ds:DGROUP, ss:DGROUP

_TEXT ends

DATA segment word public 'CODE'
DATA ends

ICODE segment word public 'CODE'
ICODE ends

DATA segment
;save_cs dw ?
;save_ds dw ?
;save_es dw ?
;save_dx dw ?
vtrxptr dw ? ; contains Host->DSP voice packet address
vpktlen dw ?
spktlen dw ?
vnoise dw ?
vsun dw ?
vsequ dw ?
vnumave dw 2
vmpcount dw 0

WWmsg0 db "Starting request write routine." , cr, lf, '\$'
WWmsg1 db "Ending request write routine." , cr, lf, '\$'
WWmsg2 db "DX = ", ' $'
WWmsg3 db "CJ = ", ' $'
WWmsg00 db cr, lf, '\$'

vpkthd db vhdlen dup(0)
vpktbuf db PKTLENG-vhdlen dup(0)
d@ label byte
s@ label byte

_DATA segment word public 'DATA'
_d@ label byte
_DATA ends

_BSS segment word public 'BSS'
_b@ label byte
_BSS ends

_DATA segment word public 'DATA'
_s@ label byte
_DATA ends

_TEXT SEGMENT
ASSUME CS:_TEXT, DS:DGROUP, SS:DGROUP

;********************** getds *****************************
; _getds proc near
;    mov    ax, cs
;    mov    cs: save_cs, ax
;    mov    ax, ds
;    mov    cs: save_ds, ax
;    mov    ax, es
;    mov    cs: save_es, ax
;    mov    ax, l
;    ret
;_getds endp

; *******************************************************
;_cinitdsp proc near
;    push   bp
;    mov    bp, sp
;    push   dx
;    push   di
;    push   si
;    call   initdsp
;    mov    cs: vsequ, 0
;    mov    ax, 1
;    pop    si
;    pop    di
;    pop    dx
;    mov    sp, bp
;    pop    bp
;    ret
;_cinitdsp endp

; *******************************************************
;_cgetvpktadd: passes addresses of vpkbuf to C.
;Calling sequence:
;    ogetvpktadd(&vrcvpktptr,&vtrxpktptr);
;Return: NON
;********************************************************
;_cgetvpktadd proc near
;    push   bp
;    mov    bp, sp
;    push   bx
;    push   es
;    push   ds
;    mov    ax, cs
;    mov    ds, ax
;    pop    ds
;    mov    si, [bp+4]
;    mov    word ptr [si], offset cs: vpkthd
;    mov    word ptr [si+2], ax
;_cgetvpktadd endp
mov ax, 1
pop es
pop bx
pop si
pop bp
ret
_cgetvpktadd endp

;---------------------------------:_cpasvtrxadd: passes addresses of vtrxpktptr from C to vtrxptr.
;Calling sequence:
; cpasvtrxadd(vtrxpktptr);
;Return: NON
;---------------------------------:
_cpasvtrxadd proc near
push bp
mov bp, sp
push si
push bx
mov si, [bp + 4]
mov cs: vtrxptr, si
mov ax, 1
pop bx
pop si
pop bp
ret
_cpasvtrxadd endp

;******************************************************************************************
;_csetvpktlen: set up voice packet length
;Calling sequence:
;csetvpktlen(&vpksize);
;Return: NON
;******************************************************************************************
_csetvpktlen proc near
push bp
mov bp, sp
push cx
push dx
push si
mov si, [bp + 4]
mov ax, [si]
mov cs: vpktslen, ax

;signal the DSP change voice packet length subroutine interruption.
wintv:
    mov dx, DSPCVR
mov al,0
out dx,al
mov al,93h
out dx,al
mov cx,waitqty
delayv:
loop delayv
in al,dx
test al,80h
jnz wintv

; pass voice packet length to DSP board.
mov cx,cs:vpktlen
mov dx,DSPISR

waitvw:
in al,dx
test al,2d
jz waitvw
mov dx,DSPRTH
mov al,0
out dx,al
mov dx,DSPRTM
mov al,ch
out dx,al
mov dx,DSPRTL
mov al,cl
out dx,al
mov ax,1
pop si
pop dx
pop cx
mov sp,bp
pop bp
ret

_csetvpktlen endp

; ***************************************************
;_csetvavenum: set up voice packet averaging number
;Calling sequence:
; csetvavenum(&vavenum);
;Return: NON
;***************************************************

_csetvavenum proc near
push bp
mov bp,sp
push di
push si
mov si,[bp+4] ; receive averaging number
mov ax,[si] ; from C
mov cs:vnumave,ax

_csetvavenum endp

;***************************************************
mov     ax, 1
pop     si
pop     di
mov     sp, bp
pop     bp
ret

_csetvavenum endp

;**********************************************
;
;csetnoise:
;
;Calling sequence:
;
;    csetnoise(&vnoise);
;
;Return: NON
;
;------------------------------------------------

_csetnoise proc near
push    bp
mov     bp, sp
push    dx
push    cx
push    di
push    si

; getting voice noise offset from the C program.
mov     si, [bp+4] ; receive voice noise offset from C
mov     ax, [si]  ; from C
mov     cs:vnoise, ax

; signal the DSP change voice noise subroutine interruption.

wintvn:
mov     dx, DSPCVR
mov     al, 0
out     dx, al
mov     dx, 95h
out     dx, al
mov     cx, waitqty

delayvn:
loop    delayvn
in      al, dx
test    al, 80h
jnz     wintvn

; pass voice noise offset to DSP board.

mov     cx, cs:vnoise
mov     dx, DSPISR

waitvwn:
in      al, dx
test    al, 2d
jz      waitvwn
mov     dx, DSPRTH
mov     al, ch
out     dx, al
mov     dx,DSPRTM
mov     al,cl
out     dx,al
mov     dx,DSPRTL
mov     al,0
out     dx,al
mov     ax,1
jmp     exitoffset

; echo back voice noise offset to HOST.

mov     dx,DSPISR

waitvnwb:
in      al,dx
test    al,1d
jz      waitvnwb
mov     dx,DSPRTH
in      al,dx
mov     ch,al
mov     dx,DSPRTM
in      al,dx
mov     cl,al
mov     dx,DSPRTL
in      al,dx
mov     ax,cx

exitoffset:
pop     si
pop     di
pop     cx
pop     dx
mov     sp,bp
pop     bp
ret

_csetnoise endp

;---------------------------------------------------------------
;_cvpktavailable: check voice packet is available in DSP board
;
;Calling sequence:
;  vpktok = cvpktavailable();
;Return: ax = 0 - no voice packet
;        1 - voice packet
;---------------------------------------------------------------

_cvpktavailable proc near
mov     ax,0
mov     dx,DSPISR
in      al,dx
or      al,cs:pklock ;Ma, orring packet locking
ret

_cvpktavailable endp

;_cinitnoise: sum the first two voice packets, then get the
;average as the noise offset.
;Calling sequence:
; rc = cinitnoise();
;Return: Averaged noise
-------------------------------------------------------------------
_cinitnoise proc near
push dl
push dx
push cx
push bx

mov ax,0
mov cs:vsum,ax
mov di,cs:vnnumave

waitpkt:
mov dx,DSPISR
in al,dx
test al,08h
jz waitpkt ;test flag HF2 set

wintrin:
mov dx,DSPCVR
mov al,0
out dx,al
mov al,92h
out dx,al
mov cx,waitqty

delayrin:
loop delayrin
in al,dx
test al,80h
jnz wintrin

nextwrin:
mov cx,cs:vpktlen

waitwrin:
mov dx,DSPISR
in al,dx
test al,1d
jz waitwrin

mov dx,DSPRTH
in al,dx
mov ah,al

mov dx,DSPRTM
in al,dx
cmp ax,0
jg plusin
mov bx,0
sub bx,ax

plusin:
add cs:vsum,bx
mov dx,DSPRTL
in al,dx
loop nextwrin

_cinitnoise endp

mov  dx,0
mov  ax,cs:vsum
mov  bx,cs:vnumave
div  bx
mov  dx,0
mov  bx,cs:vpktlen
div  bx
mov  cs:vnoise,ax
dec  di
jnz  waitpkt
pop  bx
pop  cx
pop  dx
pop  di
ret

_cinitnoise endp

;****** cwaitrdsp - wait for DSP read interrupt subroutine ******

_cwaitrdsp proc near

push  bp
mov  bp,sp
push  cx
push  dx

wintr:
mov  dx,DSPCVR
mov  al,0
out  dx,al
mov  al,92h
out  dx,al
mov  cx,waitqty

delayr:
loop  delayr
in  al,dx
test  al,80h
jnz  wintr
mov  ah,0

mov  ax,1
pop  dx
pop  cx
mov  sp,bp
pop  bp
ret

_cwaitrdsp endp

;****** creadsp - wait for DSP read interrupt subroutine ******

_creadsp proc near

push  bp
mov  bp,sp
push  cx
push  dx
push  di
push  ds
mov  ax,cs
mov ds,ax
lea di,cs:vpktbuf
mov cx,cs:vpktlen
mov ax,0
mov cs:vsun,ax
mov cs:vsmplcount,ax

nextwr:
    mov bx,DX

waitwr:
in al,dx
    test al,16d
    jnz waitwr
    test al,1d
    jz nextwr

mov bx,DX
in al,dx
mov ax,al

mov bx,DX
in al,dx

mov [di],ah
inc di
mov [di],al
inc di

cmp ax,0
jg plus
mov bx,0
sub bx,ax
mov ax,bx

plus:
add cs:vsun,ax
inc cs:vsmplcount

mov bx,0
mov ax,cs:vsun
mov bx,cs:vpktlen
div bx
mov bx,cs:vpklen
    cmp ax,bx
    jg pktok
mov ax,0
mov cs:vsequ,0
jmp exitread

pktok:
inc cs:vsequ
mov bx,cs:vsequ

mov [di],bh
inc di
mov [di],bl
inc di
mov ax,1

exitread:
  pop ds
  pop di
  pop dx
  pop cx
  mov sp,bp
  pop bp
  ret

_creadsp endp

;***********************************************************************
;cDumpDsp: Dumping voice packet to DSP from C program
;Calling sequence:
;cDumpDsp();
;Return: Non
;***********************************************************************
_cDumpDsp proc near
wintxd:
  push bp
  mov bp,sp
  push cx
  push dx
  push di
  push ds
  mov ax,cs
  mov ds,ax
  mov bp,sp
  push cx
  push dx
  push di
  push ds
  mov ax,cs
  mov ds,ax
  mov bx,DSPCVR
  mov al,0
  out dx,al
  mov al,94h
  out dx,al
  mov cx,waitqty
  loop delayxd
  in al,dx
  test al,80h
  jnz wintxd
  mov di,cs:vtrxptr
  mov cx,cs:vpktlen
  mov dx,DSPIISR
  mov dx,DSPISR
  in al,dx
  test al,10h
  jnz waitwwd
  test al,2d
  jz waitwwd
  mov dx,DSPRTHR
  mov al,[di]
  out dx,al
  inc di
  mov dx,DSPRTM
  mov al,[di]
  out dx,al

nextwwd:
  mov dx,DSPRTHR
  mov al,[di]
  out dx,al
  inc di
  mov dx,DSPRTM
  mov al,[di]
  out dx,al

waitwwd:
  mov dx,DSPRTHR
  mov al,[di]
  out dx,al
  inc di
  mov dx,DSPRTM
  mov al,[di]
  out dx,al

;Ma
;wait for HF3 reset

endp

_cDumpDsp endp
inc      di
mov      dx, DSPRTL
mov      al, 0
out      dx, al
loop     nextwwd
mov      ax, 1
pop      ds
pop      di
pop      dx
pop      cx
mov      sp, bp
pop      bp
ret
_cDumpDsp  endp

;*****************************************************************************
; _cDumpSilent: Dumping previous voice packet to DSP.
; Calling sequence:
;   cDumpSilent();
; Return: None
;*****************************************************************************
_cDumpSilent  proc  near
push      bp
mov      bp, sp
push      cx
push      dx
push      di
push      ds
mov      ax, cs
mov      ds, ax
wintxi:
mov      dx, DSPCVR
mov      al, 0
out      dx, al
mov      al, 99h
out      dx, al
mov      cx, waitqty
delayxi:
loop     delayxi
in      al, dx
test     al, 80h
jnz      wintxi
mov      di, cs:vtrxptr
mov      cx, cs:spktlen
nextwwi:
mov      dx, DSPISR
waitwwi:
in      al, dx
;       or      al, cs:pklock
;       test    al, 20h
;       jnz     waitwwi
;       test    al, 2d
    jz      waitwwi
mov      dx, DSPRTH

mov     al, [di]
out     dx, al
inc     di

mov     dx, DSPRTM
mov     al, [di]
out     dx, al
inc     di

mov     dx, DSPRTL
mov     al, 0
out     dx, al
loop    nextwwi

mov     ax, 1

pop     ds
pop     di
pop     dx
pop     cx
mov     sp, bp
pop     bp
ret

_cdumpsilent endp

;***************************************************** initdsp *****************************************************

initdsp proc near

mov     dx, DSPICR
mov     al, 10h
out     dx, al
mov     dx, DSPCVR
mov     al, 01h
out     dx, al
mov     dx, DSPIVR
mov     al, 03h
out     dx, al

ret

initdsp endp

;-----------------------------------------------------------------------------------------------------------------

; prx - routine to print a hex value from binary data up to word length
; INPUTS:
; [bp+4] = binary data to convert
; [bp+6] = number of bytes to print (1 to 4)
;
;-----------------------------------------------------------------------------------------------------------------

prx    proc near

push    bp
mov     bp, sp
mov     bx, bp
sub     bx, 4       ;local space
mov     sp, bx

push    si
push    dx
push    cx

...
push ds
mov ax, ss ; make temp buf accessible
mov ds, ax
lea bx, [bp-4] ; temp buffer address
mov dx, [bp+4] ; data to cvrt
call wtoa
mov cx, [bp+6] ; char count to print
prxl:
mov dl, [bp+si-4] ; get a byte
mov ah, 2
int 21h ; print it
inc si
loop prxl
pop ds
pop cx
pop dx
pop si
mov sp, bp
pop bp
ret
prx endp

;---------------------------------------------------------------------
; CONVERT WORD TO ASCII HEX
; Calling sequence:
; mov dx, word ; word to convert
; mov bx, offset output ; where to put output
; call wtoa
;
; ds:bx needs 4 bytes for result
;---------------------------------------------------------------------

wtoa proc near
push ax
push bx
push cx
push dx
push si
mov si, 4 ; digits per word
wtoa01:
mov al, dl ; get a digit
mov cl, 4
shr dx, cl ; strip the digit
and al, 0fh ; keep low nibble
add al, 090h
daa
adc al, 040h
da
dec si ; count the digit
mov [bx+si], al ; store the digit
jnz wtoa01
pop si
pop dx
pop cx
pop bx
pop ax
ret
wtoa endp
wtoa endp

_TEXT ends
end
; include file for INTDSP.ASM

DSPICR equ 340h
DSPCVR equ 341h
DSPISR equ 342h
DSPIVR equ 343h
DSPRTH equ 345h
DSPRTM equ 346h
DSPRTL equ 347h
/*
This file contains the C routine which is needed by the cto3la.asm
*/

#include <stdio.h>

void myRxProcess(Status, PacketSize, RequestID, PacketHeader)
int Status, PacketSize, RequestID;
char far *PacketHeader;
{
    /* fprintf(stderr,"Called by ASM - myRxProcess\nNot implement yet\n"); 
    fprintf(stderr,"Status=%d, PacketSize=%d, RequestID=%d\n",Status,PacketSize
    RequestID); */
}

void myTxProcess(Status, RequestID)
int Status, RequestID;
{
    /* printf("Called by ASM - myTxProcess\nNot implement yet\n"); 
    printf("Status=%d, RequestID=%d\n",Status, RequestID); */
}

void myExitRcvInt()
{
    /* printf("Called by ASM - myExitRcvInt\nNot implement yet\n"); */
} /* myExitRcvInt */
title a3ltoc.asm

;***************************************************************************
; A3LTOC.ASM: Process 5, a mixed language program, Microsoft assembler part,
which interfaces the 3COM board and Host computer.

;Description: This file contains subroutines which provide the C program
with an interface to the 3L 1.0 routines.
The receiver voice & data buffer pointers array has format as:

> vrcvptrq+6  6  8  10  11
+--------------------------------------------------------+  
| packet address | packet length | next packet Q address |  
+--------------------------------------------------------+  

vrcvptrq  0-5  0  2  4  5
+--------------------------------------------------------+  
| number packets | start index | end index |  
+--------------------------------------------------------+  

;***************************************************************************

; Functions called by C
PUBLIC  _getds

PUBLIC  _cInitParameters
PUBLIC  _cInitAdapters
PUBLIC  _cResetAdapter
PUBLIC  _cWhoAmI
PUBLIC  _cRdRxFilter
PUBLIC  _cWrRxFilter
PUBLIC  _cPutTxData
PUBLIC  _cGetRxData
PUBLIC  _cSetLookAhead
PUBLIC  _etext

PUBLIC  _cXmitl

PUBLIC  _cInitBufPtr
PUBLIC  _cGetOneVPkt
PUBLIC  _cGetOneDPkt
PUBLIC  _cResetVPtr
PUBLIC  _cResetDPtr
PUBLIC  _cPassHead
PUBLIC  _cVPtrarray
PUBLIC  _cGetVStrtptr
PUBLIC  _cGetVEndptr
PUBLIC  _cDPtrarray
PUBLIC  _cGetDStrtptr
PUBLIC  _cGetDEndptr
PUBLIC  _cGetPktxptr

PUBLIC  _cGetTimeCount
PUBLIC  _cGettimeptr

;Need to be written in C
extrn  _myExitRcvInt  :near
extrn  _myRxProcess   :near
extrn  _myTxProcess   :near
extrn  _CheckHead     :near
; Functions provide by this file
PUBLIC  ExitRcvInt
PUBLIC  RxProcess
PUBLIC  TxProcess

PUBLIC  hdptr,pklock

; 3L functions
extrn  InitParameters  :near
extrn  InitAdapters    :near
extrn  WhoAmI         :near
extrn  ResetAdapter   :near
extrn  RdRxFilter     :near
extrn  WrRxFilter     :near
extrn  GetRxData      :near
extrn  SetLookAhead   :near
extrn  PutTxData      :near
extrn  v_hdr_size     :word
extrn  packet_hdr_addr :word

lf    equ      0ah
cr    equ      0dh
Hdlen equ      024h ;Ma
VLinklen equ     768d ;Ma
DLinklen equ      384d ;Ma
Vbuflen equ     15360d ;Ma
Dbuflen equ     5120d ;Ma
Vtype  equ      1d ;Ma
Dtype  equ      2d ;Ma

@dmprt macro  buf,adr,len ;hex dump a data area
mov  ax,len
push ax
mov  ax,adr
push ax
mov  ax,buf
push ax
call  dmprt
add  sp,6
endm

@print macro strloc ;print string at strloc
local    strloc
push ax
push cx
push ds
push dx
mov  dx,seg strloc
mov  ds,dx
mov  dx,offset strloc
mov  ah,09h
int  21h
pop  dx
pop  ds
pop  cx
pop  ax
endm
@kbdin macro
        mov ah, 8
        int 21h
        endm

@kbdchk macro
        mov ah, 0bh
        int 21h
        endm

@prx macro
        len, dat
        mov ax, len
        push ax
        mov ax, dat
        push ax
        call prx
        add sp, 4
        endm

CODE GROUP _TEXT, DATA, ICODE

_TEXT segment byte public 'CODE'.
DGROUP group _DATA, _BSS
assume cs:_TEXT, ds:DGROUP, ss:DGROUP
_TEXTEnds

DATA segment word public 'CODE'
DATA ends

ICODE segment word public 'CODE'
ICODE ends

DATA segment

his_ds dw ?
his_es dw ? ;Ma
int_ds dw ?
int_es dw ? ;Ma
int_di dw ?
int_si dw ?
int_cx dw ?
int_dx dw ? ;Ma
_ etext db ?

stkcheck dw 0ABCDh ; stack clobber check dw
topstack dw 512 dup(0) ; adapter 0 stack top (and stack in use flag)

vectsv dd 22h dup (0) ;save all vectors so we can cleanup
retsav dw ?
crlf db cr, lf, '$'
pklock db 0
pklen dw 0
pkerr dw 0
pkcnt dw 0
pkcount dw 0
trxbuf db 1500 dup(0)
pkthd  db  Hdlen dup(0)
vptkdat db  Vbuflen dup(0)
vrnvbend dw  $
vptkdat db  Dbuflen dup(0)
drcvbend dw  $
vrnvptrq dw  VLinklen dup(0) ;vrnvptrq should be >= INT(Vbuflenx2/78)+1
vptqend dw  $
drcvptrq dw  DLinklen dup(0) ;drcvptrq should be >= INT(Dbuflenx2/78)+1
strtvptr dw  0
endvptr dw  0
vbptr dw  0
vbptr dw  0
strtdptr dw  0
enddptr dw  0
vbufptr dw  0
dbufptr dw  0
temp_hi db  0
temp_lo db  0
temp_hi_bit db  0
timelo dw  0
timehi dw  0

DATA ends

_DATA segment word public 'DATA'
_d label byte
_DATA ends

_BSS segment word public 'BSS'
_b label byte
_BSS ends

_DATA segment word public 'DATA'
_s label byte
_DATA ends

_TEXT SEGMENT
ASSUME CS:_TEXT, DS:DGROUP, SS:DGROUP

_getds proc near
mov ax,ds
mov cs:his_ds,ax
mov ax,es
mov cs:his_es,ax
ret
_getds endp

;------------------------------------------------------------------------
;cGettimeptr : This subroutine returns the time pointer points at low
;word to C program.
;Calling sequence:
;cGettimeptr(&timeptr)
;Return: Non
;------------------------------------------------------------------------
cGettimeptr proc near
push bp
mov bp,sp
push si
**push ds**

**mov ax,cs**

**pop ds**

**mov si,[bp+4]**

**mov word ptr [si],offset cs:timelo**

**mov word ptr [si+2],ax**

**pop si**

**pop bp**

**ret**

_cGettimeptr endp

---

**cGetTimeCount**

This function returns a timestamp constructed of the Timer 0 value and the lowest word of the MS-DOS clock. The Timer 0 is a count-down timer, so it is converted to form a coherent timestamp value. The Timer value is returned in the AX register (low word) and the clock value is returned in the DX register (hi word).

**cGetTimeCount proc near**

**push ds** ;set segment pointer for clock read

**mov ax,0040h** ;

**mov ds,ax** ;

**mov al,0c2h** ;set up for count/status latch

**cli** ;no ints here

**out 043h,al** ;latch

**mov dx,ds:006ch** ;get clock lsw

**sti** ;restore ints

**mov cs:timehi,dx** ;store time high word

**in al,040h** ;get status

**and al,080h** ;get msbit

**mov cs:temp_hi_bit,al** ;store msbit

**in al,040h** ;get lsb of count

**mov cs:temp_lo,al** ;store lsb of count

**in ah,al** ;get count into ax reg

**mov al,cs:temp_lo** ;get back bit 16

**ror ax,1** ;change from count-down to count-up

**or ah,cs:temp_hi_bit**

**not ax**

**mov cs:timelo,ax** ;store time low word

**pop ds** ;restore segment pointer

**ret**

_cGetTimeCount endp

---

_cInitAdapters: This procedure provides the glue between a C program and the 3L 1.0 InitAdapters function.
Calling Sequence:
    int cInitAdapters(&nAdapters)

Input Parameters:
    None

Output Parameters:
    int nAdapters

Returns:
    The return value of the InitAdapters function

_cInitAdapters proc near
    push bp
    mov bp,sp
    push si
    push di
    push ds
    mov ax,cs
    mov ds,ax
    mov di,offset CODE:RxProcess
    call InitAdapters
    pop ds
    mov di,word ptr[bp+4]
    mov word ptr[di],cx
    pop di
    pop si
    pop bp
    ret
_cInitAdapters endp

_cInitParameters:  This procedure provides the glue between a C program and the 3L 1.0 InitAdapters function.

Calling Sequence:
    int cInitParameters(Parms)

Input Parameters:
    char *Parms - Pointer to a structure with overrides of default parameters.

Output Parameters:
    None

Returns:
    The return value of the InitParameters function

cInitParameters proc near
    push bp
    mov bp,sp
    push si

.cResetAdapter: This procedure provides the glue between a C
program and the 3L 1.0 ResetAdapters function.

Calling Sequence:
int cResetAdapter()

Input Parameters:
None

Output Parameters:
None

Returns:
The return value of the ResetAdapter function

cResetAdapter proc near
push bp
mov bp,sp
push si
push di
push ds
mov dx,0
mov ax,cs
mov ds,ax
mov dl,0
call ResetAdapter
call fixvecs
pop ds
pop di
pop si
pop bp
ret
cResetAdapter endp
_cWhoAmI:  This procedure provides the glue between a C program and the 3L 1.0 WhoAmI function.

Calling Sequence:
  int cWhoAmI(&WhoPtr)

Input Parameters:
  None

Output Parameters:
  struct WhoStruct far *WhoPtr - Far pointer to the WhoAmI structure

Returns:
  The return value of the WhoAmI function

_cWhoAmI proc near
  push bp
  mov bp,sp
  push si
  push di
  push ds
  mov dx,0
  mov ax,cs
  mov ds,ax
  call WhoAmI
  mov ds,ax
  mov si,[bp+4]
  mov Word ptr [si],di
  mov Word ptr [si+2],es
  pop di
  pop si
  pop bp
  ret
_cWhoAmI endp

_cRdRxFilter:  This procedure provides the glue between a C program and the 3L 1.0 RdRxFilter function.

Calling Sequence:
  int cRdRxFilter(&RxFilter)

Input Parameters:
  None

Output Parameters:
  int RxFilter - The receive filter value

Returns:
  The return value of the RdRxFilter function
.CRdRxFilter proc near
  push  bp
  mov   bp,sp
  push  si
  push  di
  push  ds

  mov   ax,cs
  mov   ds,ax

  mov   dx,0
  call  RdRxFilter

  pop   ds
  mov   di,[bp+4]
  mov   [di],bx

  pop   di
  pop   si
  pop   bp
  ret
.CRdRxFilter endp

_cWrRxFilter:   This procedure provides the glue between a C
program and the 3L 1.0 WrRxFilter function.

Calling Sequence:
  int cWrRxFilter(RxFilter)

Input Parameters:
  int RxFilter - The new receive filter value

Output Parameters:
  None

Returns:
  The return value of the WrRxFilter function

_cWrRxFilter proc near
  push  bp
  mov   bp,sp
  push  ds
  push  si
  push  di

  mov   ax,cs
  mov   ds,ax

  mov   dx,0
  mov   ax,[bp+4]
  call  WrRxFilter

  pop   di
  pop   si
  pop   ds
  pop   bp
  ret
.cWrRxFilter endp

_cSetLookAhead:  This procedure provides the glue between a C program and the 3L 1.0 SetLookAhead function.

Calling Sequence:
  int cSetLookAhead(NumBytes)

Input Parameters:
  int NumBytes - The number of bytes of look ahead data

Output Parameters:
  None

Returns:
  The return value of the SetLookAhead function

_cSetLookAhead proc near
  push bp
  mov bp,sp
  push si
  push di
  push ds
  mov ax,cs
  mov ds,ax
  mov dx,0
  mov ax,[bp+4]
  call SetLookAhead
  pop ds
  pop di
  pop si
  pop bp
  ret
_cSetLookAhead endp

_cPutTxData:  This procedure provides the glue between a C program and the 3L 1.0 PutTxData function.

Calling Sequence:
  int cPutTxData(TotalPacketLen, NumBytes, Flags, RequestID, PacketAddr, &NewRequestID)

Input Parameters:
  int TotalPacketLen - The total packet length (first call only)
  int NumBytes - The number of bytes to transfer this call
  int Flags - The DL flags
  int RequestID - Used if not the first call
  char far * PacketAddr - A far pointer to the packet

Output Parameters:
  int NewRequestID - Returned after first call
i

- RETURNS: The return value of the PutTxData function
i

\_cPutTxData proc near

push bp
mov bp,sp
push si
push di
push ds
mov ax,ds
mov es,ax
mov bx,[bp+4]
mov cx,[bp+6]
mov dl,byte ptr[bp+8]
mov dh,byte ptr[bp+10]
mov si,[bp+12]
mov di,offset CODE:TxProcess
mov di,0ffffh ; no TxProcess
call PutTxData

pop ds
xchg dh,dl
xor dh,dh
mov di,[bp+16]
mov [di],dx

pop di
pop si
pop bp
ret

\_cPutTxData endp

\_cGetRxData: This procedure provides the glue between a C program and the 3L 1.0 GetRxData function.

Calling Sequence:
int cGetRxData(&NumBytes, Flags, RequestID, PacketAddr)

Input Parameters:
int NumBytes - The number of bytes to transfer this call
int Flags - The DL flags
int RequestID - The request identifier
char far * PacketAddr - A far pointer to the packet to copy the data

Output Parameters:
int NumBytes - The actual number of bytes transferred

Returns:
The return value of the GetRxData function

\_cGetRxData proc near

push bp
mov bp, sp
push si
push di
push ds

mov di, [bp+4]
mov cx, ss: [di]
mov dl, byte ptr [bp+6]
mov dh, byte ptr [bp+8]
mov di, [bp+10]
mov es, [bp+12]
call GetRxData

pop ds
mov di, [bp+4]
mov ss: [di], cx

pop di
pop si
pop bp
ret

_cGetRxData endp

----------------------------------------------------------------------------

TxProcess: This procedure is the protocol-side routine which is called
when a packet has finished transmitting (see _cInitAdapters). It
provides the glue between the 3L 1.0 routines and C routine called
myTxProcess.

myTxProcess Calling Sequence:
void myTxProcess(Status, RequestID)

myTxProcess Input Parameters:
int Status - Receive status
int RequestID - The request identifier

myTxProcess Returns:
Nothing

----------------------------------------------------------------------------
call _myTxProcess
add sp,4
pop es
pop ds
pop di
pop si
pop bp
ret

ExitRcvnt: This procedure is the protocol-side routine which is called when the 3L has completed a receive interrupt.

ExitRcvnt proc near
iret
ExitRcvnt endp

_cPassHead: This subroutine should be called by 'C' program at least once after the call to '_cInitBufPtr' in order to pass the address of 'pkthd' to 'Hdptr->inh' in 'C'.
Calling sequence:
cPassHead(&Hdptr);
Return: NON

_cPassHead proc near
push bp
mov bp,sp
push si
push ds
mov ax,cs
pop ds
mov si,[bp+4]
mov word ptr [si],offset cs:pkthd+4
mov word ptr [si+2],ax
pop si
pop bp
ret
_cPassHead endp

_cVPtrarray: This subroutine returns the receiver voice buffer pointer array to the C program.
Calling sequence:
cVPtrarray(&VBuflinkptr)
Return: Non

_cVPtrarray proc near
push bp
mov bp,sp
_cGetVPtrarray endp

---------------------------------------------------
_cGetVStrtptr : This subroutine returns the receiver voice buffer starting
pointer to the C program.
Calling sequence:  
cGetVStrtptr(&Vptrstrt)
Return: Non
---------------------------------------------------
_cGetVStrtptr proc near
 push bp
 mov bp,sp
 push si
 push ds
 mov ax,cs
 pop ds
 mov si,[bp+4]
 mov word ptr [si],offset cs:vrcvptrq
 mov word ptr [si+2],ax
 pop si
 pop bp
 ret
_cGetVStrtptr endp

---------------------------------------------------
_cGetVEndptr : This subroutine returns the receiver voice buffer ending
pointer to the C program.
Calling sequence:  
cGetVEndptr(&Vptrend)
Return: Non
---------------------------------------------------
_cGetVEndptr proc near
 push bp
 mov bp,sp
 push si
 push ds
 mov ax,cs
 pop ds
 mov si,[bp+4]
 mov word ptr [si],offset cs:strtvptr
 mov word ptr [si+2],ax
 pop si
 pop bp
 ret
_cGetVEndptr endp

mov word ptr [si+2],ax
pop si
pop bp
ret
_cGetVEndptr endp

;-----------------------------------------------------
;_cDPtrarray : This subroutine returns the receiver data buffer pointer
to the C program.
;Calling sequence:
cDPtrarray(&DBuflinkptr)
;Return: Non
;-----------------------------------------------------
_cDPtrarray proc near
push bp
mov bp,sp
push si
push ds
mov ax,cs
pop ds
mov si,[bp+4]
mov word ptr [si],offset cs:drcvptrq
mov word ptr [si+2],ax
pop si
pop bp
ret
_cDPtrarray endp

;-----------------------------------------------------
;_cGetDStrtptr : This subroutine returns the receiver data buffer starting
pointer to the C program.
;Calling sequence:
cGetDStrtptr(&Dptrstrt)
;Return: Non
;-----------------------------------------------------
_cGetDStrtptr proc near
push bp
mov bp,sp
push si
push ds
mov ax,cs
pop ds
mov si,[bp+4]
mov word ptr [si],offset cs:strtdptr
mov word ptr [si+2],ax
pop si
pop bp
ret
_cGetDStrtptr endp
_cGetDEndptr : This subroutine returns the receiver data buffer ending pointer to the C program.
Calling sequence:
\texttt{cGetDEndptr(&Dptrend)}
Return: Non

```asm
_cGetDEndptr proc near
    push bp
    mov bp,sp
    push si
    push ds
    mov ax,cs
    pop ds
    mov si, [bp+4]
    mov word ptr [si], offset cs:enddptr
    mov word ptr [si+2], ax
    pop si
    pop bp
    ret
_cGetDEndptr endp
```

_cGetPkttrxPtr : This subroutine returns the transmitting buffer pointer to the C program.
Calling sequence:
\texttt{cGetPkttrxPtr(&Pkttrxptr)}
Return: Non

```asm
_cGetPkttrxPtr proc near
    push bp
    mov bp,sp
    push si
    push ds
    mov ax, cs
    pop ds
    mov si, [bp+4]
    mov word ptr [si], offset cs:trxbuf
    mov word ptr [si+2], ax
    pop si
    pop bp
    ret
_cGetPkttrxPtr endp
```

RxProcess: This procedure is the protocol-side routine which is called when a packet has been received (see _cInitAdapters). It provides the glue between the 3L 1.0 routines and C routine called myRxProcess.

myRxProcess Calling Sequence:
\texttt{void myRxProcess(Status, PacketSize, RequestID, PacketHeader)}
myRxProcess Input Parameters:
  int Status - Receive status
  int PacketSize - Size of the received packet
  int RequestID - The request identifier
  char far *PacketHeader - Address of the virtual packet header

myRxProcess Returns:
  Nothing

----------------------------------------------------------------------------

xProcess proc near
push bp
di si ds es bx
push push push push push
mov cs:pklock,20h ;Ma, lock packet
mov ax,cs
mov ds,ax
mov es,ax
mov cs:int_dx,dx ;Ma
mov cs:int_dx,dx
mov cs:int_dx,dx ;Ma
mov cs:int_ds,ds ;Ma
mov cs:int_es,es ;Ma
mov cs:int_di,di ;Ma
mov cs:int_si,si ;Ma
mov ds,cs:his_ds ;Ma
mov es,cs:his_es ;Ma
call _CheckHead ;Ma
mov ds,cs:int_ds ;Ma
mov es,cs:int_es ;Ma
mov di,cs:int_di ;Ma
mov si,cs:int_si ;Ma
mov cx,cs:int_cx ;Ma
mov dx,cs:int_dx ;Ma

; At this point we could check returned value ax to make some
; decision on packet disposition, reception of voice in voice
; buffer, or reception of data in data buffer.
cmp ax,0 ;Ma
je nolen ;Ma
inc cs:pkcount
cmp ax,Vtype ;Ma
jne chkdtype ;Ma
call Rcv_Voice ;Ma, receive a voice packet.
jmp nolen ;Ma, end of receiving a voice pkt.
call Rcv_Data ;Ma, receive a data packet.

nolen:
Rcv_Voice proc near
receive a voice packet.
-------------------------------------------------------------
:cvalアナログproc near
-------------------------------------------------------------
mov cs:pkerr,0
mov di,cs:Strtvptr
mov ax,[di]
mov di,cs:endvptr
mov bx,[di]
cmp ax,bx
jne chkvqptr
jmp vbufok
chkvqptr:
mov ax,cs:endvptr
add ax,6
jmp vptrok
mov ax,offset CODE:vrcvptrq+6
vptrok:
cmp ax,cs:Strtvptr
je jnovlen
jmp vbufok
chkforward:
mov ax,cs:vbufptr
add ax,cx
jmp chkforward
cmp ax,cs:vrcvbend
jng lea ax,cs:vpktdat
mov cs:vbufptr,ax
mov di,cs:Strtvptr
mov bx,[di]
cmp ax,bx
jne vbufok
jmp vbufok
chkforward:
mov ax,cs:vbufptr
mov di,cs:Strtvptr
mov bx,[di]
cmp ax,bx
jg vbufok
add ax,cx
jmp novlen
jmp vbufok
novlen:
jmp novlen
```
bufok:
    mov    di,offset CODE:pkthd
    ;buffer /Ma
    mov    di,cs:vbufptr
    ;Ma, load offset in the buffer
    or     dl,40h
    ; ********************
    call    GetRxData
    ; ********************
    jcxz   novlen
    mov    cs:pkerr,ax
    mov    cs:pklen,cx
    mov    di,cs:endvptr
    mov    ax,cs:vbufptr
    mov    [di],ax
    mov    di,cs:enpvptr
    mov    ax,cs:vbufptr
    mov    [di],ax
    mov    di,cs:endvptr
    mov    ax,cs:vbufptr
    mov    [di],ax
    mov    2[di],cx
    mov    cs:vbufptr,cx
    mov    ax,cs:vbufptr
    add    ax,40h
    mov    ax,cs:endvptr
    add    ax,6
    inc    word ptr cs:vrcvptrq
    inc    cs:vrcvptrq
    cmp    ax,cs:vptrqend
    jnz    vptrqok
    jng    rptrqok:
    ; Ma, prepare for next packet pointer
    mov    ax,cs:enpvptr
    add    ax,40h
    cmp    ax,cs:dptrqend
    jne    dptrok
    mov    ax,offset CODE:drcvptrq+6
    ; Ma, reset pointers
    mov    cs:enpvptr,ax
    mov    4[di],ax
    mov    word ptr cs:vrcvptrq+4,ax
    ; Ma, store next packet queue address
    ; Ma, store packet pointer into
    ; Ma, store packet length into vrcvptrq
    ; Ma, store packet pointer into
    ; Ma, store next packet address
    ; Ma, load present packet queue address
    ; Ma, prepare for next queue pointer
    ; Ma, prepare for next packet pointer
    ; Ma, increment received packet
    ; Ma, increment received packet
    ; Ma, check if pointer buffer full
    ; Ma, store next packet queue address
    ; Ma, store pointer index

nonlen:
    ret

%cv_Voice endp

-------------------------------------------------------------
: Rcv_Data proc near
receive a data packet.
-------------------------------------------------------------

%cv_Data proc near

mov    cs:pkerr,0
mov    di,cs:strtdptr
mov    ax,[di]
mov    di,cs:enddptr
mov    bx,[di]
cmp    ax,bx
jne    chkdqptr
jmp    dbufok

:chkdqptr:
    mov    ax,cs:enddptr
    add    ax,6
    cmp    ax,cs:dptrqend
    jne    dptrok
    jmp    dbufok

:dbufok:
    mov    ax,offset CODE:drcvptrq+6
    ; Ma, release buffer

:dbufok:
    mov    di,offset CODE:pkthd
    ;buffer /Ma
    mov    di,cs:vbufptr
    ;Ma, load offset in the buffer
    or     dl,40h
    ; ********************
    call    GetRxData
    ; ********************
    jcxz   novlen
    mov    cs:pkerr,ax
    mov    cs:pklen,cx
    mov    di,cs:endvptr
    mov    ax,cs:vbufptr
    mov    [di],ax
    mov    di,cs:enpvptr
    mov    ax,cs:vbufptr
    mov    [di],ax
    mov    di,cs:endvptr
    mov    ax,cs:vbufptr
    mov    [di],ax
    mov    2[di],cx
    mov    cs:vbufptr,cx
    mov    ax,cs:vbufptr
    add    ax,40h
    mov    ax,cs:endvptr
    add    ax,6
    inc    word ptr cs:vrcvptrq
    inc    cs:vrcvptrq
    cmp    ax,cs:vptrqend
    jnz    vptrqok
    jng    rptrqok:
    ; Ma, prepare for next packet pointer
    mov    ax,cs:enpvptr
    add    ax,40h
    cmp    ax,cs:dptrqend
    jne    dptrok
    mov    ax,offset CODE:drcvptrq+6
    ; Ma, reset pointers
    mov    cs:enpvptr,ax
    mov    4[di],ax
    mov    word ptr cs:vrcvptrq+4,ax
    ; Ma, store next packet queue address
    ; Ma, store packet pointer into
    ; Ma, store packet length into vrcvptrq
    ; Ma, store packet pointer into
    ; Ma, store next packet address
    ; Ma, load present packet queue address
    ; Ma, prepare for next queue pointer
    ; Ma, prepare for next packet pointer
    ; Ma, increment received packet
    ; Ma, increment received packet
    ; Ma, check if pointer buffer full
    ; Ma, store next packet queue address
    ; Ma, store pointer index

nonlen:
    ret

%cv_Voice endp
```
cmp ax,cs:strtddptr ;Ma
je jnodlen ;Ma

:hkdbuf:
mov ax,cs:dbufptr ;Ma, current buffer pointer
add ax, cx ;Ma, add packet length
cmp ax,cs:drcvbend ;Ma, check if buffer is short
jng chkdforward ;Ma, buffer is not short
lea ax,cs:dpktdata ;Ma, initialize bufptr
mov cs:dbufptr, ax

mov di,cs:strtddptr ;Ma
mov bx,[di] ;Ma, queue starting address
cmp ax, bx ;Ma
jne dbufok ;Ma
jmp jnodlen ;Ma

:chkfoward:
mov ax,cs:dbufptr ;Ma
mov di,cs:strtddptr ;Ma
mov bx,[di] ;Ma, queue starting address
cmp ax, bx ;Ma
jg dbufok ;Ma
add ax, cx ;Ma
cmp ax, bx ;Ma
jg jnodlen ;Ma
jmp dbufok ;Ma

:jnodlen:
jmp nodlen

:dbufok:
mov di,offset CODE:pkthd ;buffer /Ma
mov di,cs:dbufptr ;Ma, load offset in the buffer
or di,40h ; ******************
call GetRxData ; ******************
jcxz nodlen
mov cs:pkerr, ax
mov cs:pklen, cx

mov di,cs:enddptr ;Ma
mov ax,cs:dbufptr ;Ma
mov [di], ax ;Ma, store packet pointer into
mov ax, cs:dbufptr ;Ma, drcvptrq
mov 2[di], cx ;Ma, store packet length into drcvptrq

add cs:dbufptr,cx ;Ma, prepare for next packet pointer
mov ax,cs:dbufptr ;Ma, next packet address
mov 4[di], ax ;Ma, store next packet address
mov ax,cs:enddptr ;Ma, load present packet queue address
add ax,6 ;Ma, prepare for next queue pointer
inc word ptr cs:drcvptrq ;Ma, increment received packet
inc cs:drcvptrq ;Ma, increment received packet
cmp ax,cs:dptrqend ;Ma, check if pointer buffer full
jnz dptrqok ;Ma
jng dptrqok ;Ma
mov ax,offset CODE:drcvptrq+6 ;Ma, reset pointers

:dptrqok:
mov cs:enddptr,ax ;Ma
mov 4[di],ax ;Ma, store next packet queue address
mov word ptr cs:drcvptrq+4,ax ;Ma, store pointer index

:jnodlen:
_cXmit1 endp

_cXmit1 proc near

改变 packet

_cXmit1 proc near

改变

bp

mov bp,sp
push si
push di
push ds

mov ax,cs ;Ma
mov ds,ax ;Ma
mov ax,ds
mov es,ax

;setup for PutTxData
mov bx,[bp+4] ;set lengths
mov cx,[bp+6]
mov dl, byte ptr[bp+8]
mov dh, byte ptr[bp+10]
mov si,[bp+12] ;buffer address
mov si,offset cs:trxbuf
mov di,0ffffh ;no TxProcess

call PutTxData

pop ds
xchg dh,dl
xor dh,dh
mov di,[bp+16]
mov [di],dx

pop di
pop si
pop bp
ret

_cXmit1 endp

_cInitBufPtr

This subroutine initializes the receiving buffer pointers and counters.
Calling sequence:
        cInitBufPtr();

Return: NON

_cInitBufPtr proc near

push bp
mov bp,sp
push ds
push di
push bx

mov ax,cs
mov ds,ax
; initialize voice buffer pointers
mov word ptr cs:vrcvptrq,0 ; initialize counter
mov cs:strtvptr,offset CODE:vrcvptrq+6 ; initialize starting ptr
mov ax,cs:strtvptr
mov cs:endvptr,ax
mov word ptr cs:vrcvptrq+2,ax ; store pointer index
mov word ptr cs:vrcvptrq+4,ax ; store pointer index
lea ax,cs:vpktdat
mov di,cs:strtvptr
mov [di],ax
mov cs:vbufptr,ax

; initialize data buffer pointers
mov word ptr cs:drcvptrq,0 ; initialize counter
mov cs:strtdptr,offset CODE:drcvptrq+6 ; initialize starting ptr
mov ax,cs:strtdptr
mov cs:enddptr,ax
mov word ptr cs:drcvptrq+2,ax ; store pointer index
mov word ptr cs:drcvptrq+4,ax ; store pointer index
lea ax,cs:dpktdat
mov di,cs:strtdptr
mov [di],ax
mov cs:dbufptr,ax
mov cs:v_hdr_size,Hdlen ; set header size
mov ax,offset cs:pkthd ; load 'pkthd' address
mov cs:packet_hdr_addr,ax ; store 'pkthd' address

mov bx,offset cs:strtdptr
@prx 4,bx
@print crlf
mov di,cs:strtdptr
mov di,[di]
@prx 4,di
@print crlf
mov bx,offset cs:enddptr
@prx 4,bx
@print crlf
mov di,cs:enddptr
mov di,[di]
@prx 4,di
@print crlf
mov bx,offset cs:drcvptrq
@prx 4,bx
@print crlf

mov ax,0

pop bx
pop di
pop ds
pop bp
ret

.cInitBufPtr endp

#------------------------------------------------------------------
_cGetOneVPkt
This subroutine returns a voice pointer points to the packet address in the buffer 'vpktdat' from the pointer buffer 'vrcvptrq'. Then update the strtvptr pointer in vrcvptrq.

Calling sequence:

```
cGetOneVPkt(&Pktrcv);
```

Return: [si+2] - contains segment register
        [si] - contains packet offset address

```
.cGetOneVPkt proc near
    push bp
    mov bp,sp
    push si
    push di
    push bx
    push ds
    mov ax,cs
    mov ds,ax
    mov di,word ptr cs:vrcvptrq+2
    mov bx,[di]
    pop ds
    mov si,[bp+4]
    mov word ptr [si+2],ax
    mov word ptr [si],bx
    push ds
    mov ax,cs
    mov ds,ax
    dec word ptr cs:vrcvptrq
    mov di,word ptr cs:vrcvptrq+2
    add di,6
    mov cs:strtvptr,di
    mov word ptr cs:vrcvptrq+2,di
    pop ds
    pop bx
    pop di
    pop si
    pop bp
    ret
```

.cGetOneVPkt endp

-------------------------------------------------------------

This subroutine returns a data pointer points to the packet address in the buffer 'dpktdat' from the pointer buffer 'drcvptrq'. Then update the strtdptr pointer in drcvptrq.

Calling sequence:

```
cGetOneDPkt(&Pktrcv);
```

Return: [si+2] - contains segment register
        [si] - contains packet offset address

```
.cGetOneDPkt proc near
    push bp
    mov bp,sp
```
This subroutine reset voice pointers, strtvptr, endvptr, and vrcvptrq[0],[1],[2].

Calling sequence:
cResetVPtr();

Return: None
mov cs:vbufptr,ax
pop ds
pop di
ret
_cResetVPtr endp

;--------------------------------------------------------------
:_cResetDPtr

This subroutine reset data pointers, strtdptr, enddptr, and
vrcdptrq[0],[1],[2].

Calling sequence:
: _cResetDPtr();

Return: Non
;--------------------------------------------------------------

_cResetDPtr proc near
push di
push ds

mov ax,cs
mov ds,ax

; initialize data buffer pointers

mov word ptr cs:drcvptrq,0 ; initialize counter
mov cs:strtdptr,offset CODE:drcvptrq+6; initialize starting ptr
mov ax,cs:strtdptr
mov cs:enddptr,ax
mov word ptr cs:drcvptrq+2,ax ; store pointer index
mov word ptr cs:drcvptrq+4,ax ; store pointer index
lea ax,cs:dpktdat
mov di,cs:strtdptr
mov [di],ax
mov cs:dbufptr,ax

pop ds
pop di
ret
_cResetDPtr endp

;avvecs proc near
push ds
push es
push si
push di
push cx

mov ax,ds
mov es,ax
xor ax,ax
mov ds,ax
mov cx,22h*2 ;vectors 0 - 21h, 2 wds per
mov di,offset CODE:vectsv
xor si,si
cli

rep movs
sti ;save 'em all
lixvecs proc far
push bp
mov bp,sp
mov bx,bp
sub bx,0ch
mov sp,bx
push si
mov ax,[bp+8] ;len
005c: sub dx,dx
mov cx,10h
0061: div cx
mov [bp-4],ax ;lines

mov cx,ax
mov es,ax
mov cx,22h*2 ;vectors 0 - 21h, 2 wds per
mov si,offset CODE:vectsv
xor di,di
cli
rep movsw
;restore 'em all
sti

add di,di
add si,si
add cx,1
rep movsw
;restore 'em all
sti

pop ax
pop di
pop si
pop es
ret

ixvecs endp

ixvecs proc near
push es
push si
push cx
push ax
xor ax,ax
mov es,ax
mov cx,22h*2 ;vectors 0 - 21h, 2 wds per
mov si,offset CODE:vectsv
xor di,di
cli
rep movsw
;restore 'em all
sti

pop ax
pop cx
pop di
pop si
pop es
ret

ixvecs endp

---------------------------------------------------------------------

dmprt - produces dump listing, calling parameters are pushed on stack
(converted from a C routine)

INPUTS:
[bp+4] = data address
[bp+6] = starting address for line headers
[bp+8] = length of data to print

OUTPUT:
Dump listing to stdout device

---------------------------------------------------------------------

dmprt proc near
push bp
mov bp,sp
mov bx,bp
sub bx,0ch ;local var
mov sp,bx
push si
mov ax,[bp+8] ;len
005c: sub dx,dx
mov cx,10h
0061: div cx
mov [bp-4],ax ;lines
0063:    mov    [bp-6],dx    ;rem
0066:    mov    word ptr [bp-8],0    ;i
006b:    mov    word ptr [bp-0ah],0    ;line
0070:    jmp    d0158
0073:
push    dx
mov    dl,cr    ;000d
mov    ah,2
int    21h
mov    dl,lf    ;000A
mov    ah,2
int    21h
mov    dl,''
mov    ah,2
int    21h
mov    dl,''
mov    ah,2
int    21h
pop    dx
mov    ax,4
push    ax
mov    ax,[bp+6]    ;adr
add    ax,[bp-8]    ;i
push    ax
call    prx
add    sp,4    ;0004
push    dx
mov    dl,''
mov    ah,2
int    21h
mov    dl,''
mov    ah,2
int    21h
pop    dx
mov    word ptr [bp-0ch],0    ;j
00c5:    test    byte ptr [bp-0ch],3    ;j
jnz    d00d5
push    dx
mov    dl,''
mov    ah,2
int    21h
pop    dx
00d5:    mov    ax,2    ;0002
push    ax
mov    bx,[bp-8]    ;i
mov    si,[bp+4]    ;buf
mov    ah,[bx+si]    ;buf[i]
push    ax
call    prx
add     sp,4  ;0004
inc     word ptr [bp-8];i
inc     word ptr [bp-0ch] ;j

100f0: cmp     word ptr [bp-0ch],10h   ;j
          jb      d00c5
push     dx
mov      dl,''
mov      ah,2
int      21h
mov      dl,''
mov      ah,2
int      21h
pop      dx
sub      word ptr [bp-8],10h   ;i,0010
mov      word ptr [bp-0ch],0  ;j

;do ascii
10113: mov      bx,[bp-8]   ;i
mov      si,[bp+4]   ;buf
push     dx
mov      dl,[bx+si]   ;buf[i]
cmp      dl,''
jb      d013f
cmp      dl,7fh
jb      d0142
1013f: mov      dl,'.'    ;002e
10142:
mov      ah,2
int      21h
pop      dx
inc     word ptr [bp-8];i
inc     word ptr [bp-0ch] ;j
cmp     word ptr [bp-0ch],10h   ;0010
jb      d0113
inc     word ptr [bp-0ah]   ;line
10158: mov      ax,[bp-4]   ;lines
cmp      [bp-0ah],ax   ;line
jnb     d0163
jmp     d0073
10163: cmp     word ptr [bp-6],0   ;rem
jnz     d016c
jmp     d0272
1016c:
push    dx
mov      dl,cr   ;000d
mov      ah,2
push dx
mov dl,''
mov ah,2
int 21h
pop dx

101dd:
push dx
mov dl,''
mov ah,2
int 21h
mov dl,''
mov ah,2
int 21h
pop dx
inc word ptr [bp-0ch] ;j

101f4: cmp word ptr [bp-0ch],10h ;0010
jb d01cd
push dx
mov dl,''
mov ah,2
int 21h
mov dl,''
mov ah,2
int 21h
pop dx
mov ax,[bp-6] ;rem
sub [bp-8],ax ;i
mov word ptr [bp-0ch],0 ;j

;do ascii
10219: cmp ax,[bp-6] ;rem
cmp [bp-0ch],ax ;j
jnb d026c
mov bx,[bp-8] ;i
mov si,[bp+4] ;buf
push dx
mov dl,[bx+si] ;buf[i]
cmp dl,''
jb d024d
cmp dl,7fh
jb d0250

1024d: mov dl,'.' ;002e

10250: mov ah,2
int 21h
pop dx
inc word ptr [bp-8] ;i
inc word ptr [bp-0ch] ;j
jmp short d0219

1025f:
push dx
mov dl,.
push dx
inc word ptr [bp-0ch] ;j
1026c: cmp word ptr [bp-0ch],10h ;0010
     jb d025f
10272:
push dx
mov dl,cr ;000d
mov ah,2
int 21h
mov dl,lf ;000a
mov ah,2
int 21h
pop dx
pop si
mov sp,bp
pop bp
ret

prx - routine to print a hex value from binary data up to word length
INPUTS:
[bp+4] = binary data to convert
[bp+6] = number of bytes to print (1 to 4)

prx proc near

push bp
mov bp,sp
mov bx,bp
sub bx,4 ;local space
mov sp,bx
push si
push dx
push cx
push ds
mov ax,ss ;make temp buf accessible
mov ds,ax
lea bx,[bp-4] ;temp buffer address
mov dx,[bp+4] ;data to cvrt
call wtoa
mov cx,[bp+6] ;char count to print
xor si,si
prxl:
    mov dl,[bp+si-4] ;get a byte
    mov ah,2
    int 21h ;print it
    inc si
    loop prxl
pop ds
pop cx
pop dx
pop si
mov sp,bp
pop bp
ret

;------------------------------------------------------------
; CONVERT WORD TO ASCII HEX
; Calling sequence:
;    mov dx,word          ;word to convert
;    mov bx,offset out   ;where to put output
;    call wtoa
;
;    ds:bx needs 4 bytes for result
;------------------------------------------------------------

wtoa proc near
push ax
push bx
push cx
push dx
push si
mov si,4 ;digits per word
wtoa01:
mov al,d1 ;get a digit
mov cl,4
shr dx,cl ;strip the digit
and al,0fh ;keep low nibble
add al,090h
da
adc al,040h
da
dec si ;count the digit
mov [bx+si],al ;store the digit
jnz wtoa01
pop si
pop dx
pop cx
pop bx
pop ax
ret

wtoa endp

.TEXT ends
end
References

DSP boards. *Ariel Corporation*.


*Motorola Inc.*

