Inter-Office Memorandum

To	Taft, Boggs, Hunt Strollo, Stewart	Date	March 30, 1978
From	Larry Stewart	Location	Palo Alto
Subject	Alto-1822 Interface Microcode & Emulator Interface	Organization	SSL

XEROX

Filed on: [Maxc1]<LStewart>AISwSpec.press

Abstract

The Alto-1822 interface is now substantially working. This memo describes what is hoped to be the final interface specification. The bit patterns used to control it are *unlikely* to change again. Naturally, the emulator interface depends on the 1822 microcode. Included here is the microcode written for the 1822 test program, which may provide a plausible base for other situations.

General Notes

Hardware:

The Alto-1822 interface is a processor bus device requiring a single Task. It is designed to conform to the requirements of BBN Report 1822. As such it is suitable for Arpanet or Packet Radio Net use. The interface is a full duplex device with no buffering beyond 16 bit input and output shift registers. 1822 specifies a bit-by-bit handshaking protocol intended to allow interconnection of machines with different word lengths - data transfer may pause for essentially arbitrary intervals (i.e. seconds) between any two bits.

Performance:

The input and output data transfer sections of the interface are controlled by PROM based finite state machines clocked by the Alto at 170 ns intervals. With zero delays at the IMP end the input and output sections can each transfer a bit every six cycles - just under 1 Megabit per second. Actual transfer rates will be somewhat less. In loopback mode the throughput is about 680 Kb/s.

Hardware - Microcode Interface

Control of the hardware is accomplished through the emulator SIO instruction and various Task Specific F1's and F2's. The SIO/wakeup/branch logic is controlled by a PROM based finite state machine. Included here is a rather low level description of what the various functions and SIO's do. It is recommended that the reader look at the next section, describing the emulator interface of the test microcode before starting off to write his own!

SIO functions:

The interface task wakeup logic responds to bits 5 and 6 of the Bus during execution of an SIO (Task 0 STARTF):

SIO #3000 sets wakeup and arranges for the IBRNCH function to put ’01’ on NEXT[6-7]. In the attached microcode, this operation is used to start the ’Control’ microcode. This wakeup is cleared by ISWAKC

SIO #2000 sets wakeup and arranges for the IBRNCH function to put ’00’ on NEXT[6-7]. In the attached microcode, this operation is used to start the ’Start Input’ microcode. This wakeup is cleared by ISWAKC

SIO #1000 enables the output hardware in such a way that wakeup is set and the IBRNCH function will put ’10’ on NEXT[6-7]. In the attached microcode, this operation is used to start an output data transfer. This wakeup is cleared by IOCLR.

Task Specific Functions:

IBRNCH (F2-13):

Gates two bits from the Control PROM to NEXT[6-7].
’00’ - caused by SIO #2000 ’Input Start’
’01’ - caused by SIO #3000 ’Control’
’10’ - caused by output hardware data request and (indirectly) by SIO #1000 ’Output Start’
’11’ - caused by input hardware data available

ISWAKC (F2-14):

Clears wakeups generated by SIO #2000 and SIO #3000 *only*

IPOSTF (F1-14):

Gates hardware status to the Bus.
Bit 15: IMP Relay WAS off, it may still be off.
Bit 16: IMP Ready Relay is OPEN (off)
Bit 17: Host Ready flipflop is OFF (and so is the relay)
Other bits are left as 1’s.

ISETCS (F2-11):

Decodes the Bus to set and clear various controls in the interface.
#000000 - Does nothing
#000001 - Master Reset. Clears all wakeups, turns off all data transfers, but does not affect the relay.
#000002 - Set Last Word. Used after an IWRITE, this will set the ’LastWord’ flipflop, causing ’LastHostBit’ to be transmitted at the end of the current data word.
#000003 - Try Clearing ImpWasDown. If the Imp is now READY, this will clear the ’Imp Was Down’ status bit. Otherwise it won’t.
#000004 - Turn on Hardware LoopBack
#000005 - Turn off Hardware LoopBack
#000006 - Turn on Host Ready Relay
#000007 - Turn off Host Ready Relay
#002000 - Turn on Discard Flipflop (see IPTMOD)
#001000 - Turn off Discard Flipflop

Other bit patterns are undefined.

IPTMOD (F2-10):

Gates the ’Discard’ flipflop to NEXT[7]. In the attached microcode this operation is used during Packet Discard Mode to dispatch to a different section of input data transfer microcode.

IOCLR (F1-15):

Resets the output finite state machine to StateIdle. Commonly used to clear output data wakeup.

IWRITE (F1-16):

Loads the output data shift register from the Bus and starts an output data transfer.

IIENBL (F2-12):

Turns on the input hardware; requesting a single 16 bit word be read. After 16 bits have been read, or LastImpBit received, an input data wakeup will occur.

IREAD (F1-17):

Gates the input shift register to the Bus. If LastImpBit did *not* arrive, the data wakeup is cleared. If LastImpBit *did* arrive, a '1' is gated to NEXT[7] and *the wakeup is not cleared*. In this case, a second IREAD will clear the wakeup. The second IREAD will never disturb NEXT.

Emulator - Task interface

This section describes the emulator interface implemented by 'AIuCode.mu', attached to this memo.

An emulator program can wake up the IMP task by executing an SIO instruction with either or both of AC0 bits 5 and 6 set. Possible functions are start receiver, start transmitter, and read status/set control register. AC1 is must point to a command block during the execution of the *Read Status / Set Control Register* SIO. The task saves the control block pointer in an S register for later use. This means that a read status call must be made *before* trying to do a data transfer!

In order to hold the use of S registers to a minimum, both input and output buffer pointers are kept in main memory. This means that input/output operations require three main memory cycles per word. The single S register contains a pointer to the 12 word control block. The single control block contains entries for input, output, and control functions.

Control Block Structure

The command block must be on an even word (for Alto I compatibility).

```
CmdBlock:   struct
            [
                controlWord             //command
                blank
                InputPointer            //next word to be used
                InputBufferEnd          //first word not in buffer
                OutputPointer           //next word to be used
                OutputBufferEnd         //first word not in buffer
                ControlPost             //control post location
                ControlIntBits          //control interrupt channels
                InputPost               //input post location
                InputIntBits            //input interrupt channels
                OutputPost              //output post location
                OutputIntBits           //output interrupt channels
            ]
```

Start receiver

The receiver will collect a packet from the IMP. The final buffer location used (+1) is returned in inputPointer. Returns error if the buffer size is zero on entry, or if the buffer overflowed.

```
LDA     AC0,#002000
SIO
```

Start transmitter

The transmitter will send a packet (buffer) to the IMP. If the buffer length is zero, no bits will be sent, and the returned status will indicate sucessful completion.

```
LDA     AC0,#001000
SIO
```

Read Status / Set Control Register

This command allows an emulator program to observe the interface without disturbing it. It is also used to tell the microcode where the command block is. The command word is used to set and clear some control flip-flops in the hardware.

```
LDA     AC0,#003000
LDA     AC1,CmdBlock  //(pointer to)
SIO
```

Post Data

On completion of a command, the microcode status occupies the left byte of the Post location and the hardware status occupies the right byte.

Possible microcode status bytes:

```
00000001  -  all ok
00000010  -  buffer overflow (input only)
00000011  -  input buffer length was zero on entry
```

Hardware status bits:

```
struct
    [
    unused          bit  5;  //These bits come back as 1's
    IMPWasDown      bit  1;
    IMPnotReady     bit  1;
    HostnotReady    bit  1;
    ]
```

Command Word

The command word is used to set and clear some control flipflops in the hardware. Except for SetLastWord, these functions are never used by the microcode.

Possible command words:

0 does nothing
1 hardware master reset
2 set last word - causes *Last Host Bit* to appear eventually
(intended for use by microcode only)
3 try clearing *IMP Was Down* flop
(will not work if the IMP is still down)
4 turn on test mode - hardware loop back
(beware: this mode holds the Host ready line off)
5 turn off test mode

6	turn on *Host Ready* relay
7	turn off *Host Ready* relay
2000	turn on packet throwaway mode
1000	turn off packet throwaway mode

Packet Throwaway Mode:

By executing an SIO-set control function, the user may set the receiver hardware into "Packet Throwaway Mode". On each SIO-start receiver call (until the throwaway mode is cleared), the microcode will read in a packet (i.e. until the next *Last Imp Bit*) and throw it away. Although it is not used, the input buffer must have some space in it.

Zero buffer length:

On output, there is no longer any initialization microcode - the SIO reaches down into the output finite state machine and makes it think it was running already. The FSM then requests a normal output data wakeup to get the first word. Thus if the buffer size is zero, on that first wakeup the microcode will conclude that transmission is done and return normal end status - without having done anything. On input, while a zero length buffer probably represents a programming error, the microcode will return a bad status. If the receiver were actually started with a zero size buffer, the condition would not be noticed until the first input data wakeup. At that time, the microcode would have no choice but to throw away the input word and return a buffer overflow error - the entire packet would be messed up. With the current microcode, you can request input into a zero length buffer, get an error status back, and restart the operation with a non-zero buffer *without* losing data.

Microcode timing

A copy of the newest microcode is attached. The code uses one S register. I include here a table of timing information. A '+' indicates a TASK.

Operation		Microcycles	
Emulator control/status call		15	
Emulator input request	(Buffer length zero)	17	
Emulator input request	(Buffer length non-zero)	12	
Emulator output request	0		
Input wakeup	(Buffer Overflow)	20	
Input wakeup	(Normal)	18	
Input wakeup	(Last word)	18+10	(wierd kludge)
Input wakeup	(Normal throwaway)	13	
Input wakeup	(Last throwaway)	13+10	
Output wakeup	(Normal)	18	
Output wakeup	(Last word)	18	
Output wakeup	(Normal end)	19	

Microcode

```
; Larry Stewart March 25, 1978  11:37 PM

#AltoConsts23.mu;

; Address definition for Emulator main loop in ROM

$START $L004020,000000,000000 ; Start of emulator main loop

; Task Specific Function Definitions

$IREAD $L000000,070017,000100 ; F1-17     Input data
$IWRITE$L020016,000000,124000 ; F1-16     Output data
$IOCLR $L016015,000000,000000 ; F1-15     Clear hardware output wakeup
$IPOSTF$L016014,066014,000100 ; F1-14     Post (gate status to bus)
$ISWAKC$L024014,000000,000000 ; F2-14     Clear SIO generated wakeup
$IBRNCH$L024013,000000,000000 ; F2-13     4-way branch on wakeup
$IIENBL$L024012,000000,000000 ; F2-12     Start read (turn on RFNIB)
$ISETCS$L024011,000000,000000 ; F2-11     Set control functions from bus
$IPTMOD$L024010,000000,000000 ; F2-10     2-way branch on throwaway mode

;R (S) registers

$ICBPTR$R76;                              Control Block Pointer
$MTEMP $R25;                              Temporary storage
$AC1   $R2;                               Emulator register
$NWW   $R4;                               Interrupt system reg

; Task constants

$ISDON $777;                              done
$ISOVF $1377;                             buffer overflow (input only)
$ISIBLZ$1777;                             block length zero (input only)

; Initialization for putting this code in the RAM, including
    ; 'silent boot' code
!17,20,LOC0,IMLOOP,,,,,,,,,,,,;

;Silent Boot code, branches to ROM immediately
LOC0:   SWMODE;
        :START;

;Main loop.  Task waits here when not processing anything.

;IBRNCH gates two bits onto NEXT6 and NEXT7
    ;    00 - Start input  01 - Set Control
    ;    10 - Output data wakup  10 - Input data wakeup

; 4-way branch using NEXT6,NEXT7 (Caused by IBRNCH)
%14,14,0,IISTRT,ICTST,IODATA,IIDATA;

IMLOOP:  T_ ICBPTR,IBRNCH;                test wakeup conditions
         L_ ISDON,:IISTRT;                [IISTRT,ICTST,IODATA,IIDATA]

;Common Post routine
    ; Expects offset of post location in T
```

```
    ;   and task status in M

;IPOSTF gates hardware status bits to the bus

; 2-way branch using NEXT9 (Caused by SH=0)
!1,2,IIBLOK,IPOST;

IPOST:  MAR_ ICBPTR+T;                  Start double reference
        T_ NWW;
        MD_ M,IPOSTF;                   Bus AND hardware status
        L_ MD OR T,TASK;                NWW OR interrupt bits
INXT:   NWW_ L,:IMLOOP;

;Read status and set control register

;ISWAKC clears a wakeup caused by an SIO instruction
;ISETCS loads hardware control flops from the bus

ICTST:  MAR_ L_ AC1;                    Start fetch of args
        ICBPTR_ L;                      Save ctl block pointer
        T_ 6,ISWAKC;                    post location offset
        SINK_ MD,ISETCS;                Control function
INEND:  L_ ISDON,:IPOST;                Set control flops

;Input initialization

;IIENBL enables the hardware to receive a 16 bit word (turns it on)

IISTRT: MAR_ 2+T;
        ISWAKC;                         clear wakeup
        T_ MD;                          read data pointer
        L_ MD-T;
        L_ ISIBLZ,SH=0;
        T_10,:IIBLOK;                   [IIBLOK,IPOST]
IIBLOK: L_ NWW,IIENBL,TASK,:INXT;       length ok,start reader

;Input Main loop

;IPTMOD gates the state of the Throwaway mode flop to NEXT7
    ;   The branch is taken if the flop was set (by an ISETCS)

;IREAD gates the receiver shift register to the bus
    ;  and the PAD flop to NEXT7
    ;  If the branch is taken (which happens when the last word of
    ;  a packet is read), the wakeup will not be cleared, otherwise
    ;  the wakeup will be cleared

; 2-way branch using NEXT9 (Caused by SH=0)
!1,2,IIDMOR,IIDFUL;

; 2-way branch using NEXT7 (Caused by IPTMOD)
%4,4,0,IIACPT,IIDISC;

; 2-way branch using NEXT7 (Caused by IREAD)
%4,4,0,IIDCON,IIDLST;

; another 2-way branch using NEXT7 (Caused by IREAD)
```

```
%4,4,0,IIFINS,IIDPST;

IIDATA:  MAR_ L_ 2+T;                    Start fetch
         MTEMP_ L,IPTMOD;                save cb ptr,test mode
         T_ MD;                          get pointer [IIACPT,IIDISC]
IIACPT:  L_ MD-T;                        past end of buffer?
         MAR_ T,SH=0;                    start data fetch
         L_ ONE+T,:IIDMOR;               [IIDMOR,IIDFUL]

IIDMOR:  MD_ IREAD;                      Read and branch on last word.
    ;    Except on the last word, this
    ;    clears the wakeup
IICLNU:  MAR_ MTEMP,:IIDCON;             [IIDCON,IIDLST]

IIDCON:  IIENBL,TASK;                    enable receiver
IDCON:   MD_ M,:IMLOOP;                  update ptr,restart

IIDLST:  TASK;                           this TASK only works
    ;    because the hardware doesn't really clear the
    ;    wakeup until the next IREAD in this case

         MD_ M;                          update pointer
         L_ ISDON;
IIDPST:  SINK_ IREAD;                    clear wakeup (again)
IIFINS:  T_ 10,:IPOST;

IIDISC:  L_ T, T_ IREAD,:IICLNU;         throwaway word,branch if last

IIDFUL:  SINK_ IREAD;                    overflow status
         L_ ISOVF,:IIFINS;               [IIFINS,IIDPST]

;Main output loop

;IWRITE loads the output shift register from the bus,
    ;  clears the (hardware generated) wakeup if there was one,
    ;  and starts the output hardware

;IOCLR resets the output hardware.  This is how you clear the
    ;   wakeups without restarting the output hardware

; 2-way branches using NEXT9 (both caused by SH=0)
!1,2,IODMOR,IODEND;
!1,2,IONLST,IOLST;

IODATA:  MAR_ L_ 4+T;                    start pointer fetch
         MTEMP_ L;                       save cb ptr
         T_ MD,IOCLR;                    get pointer, clear wakeup
         L_ MD-T;                        past end of buffer?
         MAR_ T,SH=0;                    start data fetch
         L_ M-1,:IODMOR;                 [IODMOR,IODEND]

IODMOR:  IWRITE_ MD,SH=0;                send data,last?
         MAR_ MTEMP,:IONLST;             start update [IONLST,IOLST]
IOLST:   SINK_ 2,ISETCS;                 set last word function
IONLST:  L_ ONE+T,TASK,:IDCON;           finish

IODEND:  T_ 11+1,:INEND;                 offset of post loc
```