HEWLETT-PACKARD CO.

NOTE: This page provides a running history of changes for a multi-page drawing which cannot conveniently be re-issued completely after each change. When making a change, list for each page all before-and-after numbers (within reason; use judgement, and use "extensive" revision note if loss of past history is tolerable, or retype complete page) and associate with each a symbol made up of the change letter and a serial subscript to appear here and on the page involved (there enclosed in a circle, triangle, or other attention-getting outline).

Ltr	REVISIONS	DATE	INITIALS
B	RETYPED. PCO 40-12534.	8-3-82	B. Shelton

Model No.	Stock No.
98629 IRS	

Description	Date 7-21-82
By ALAN NELSON	Sheet No. 1 of 27
Supersedes	Drawing No. A-98629-90302-1

98629-90302-1 Firmware Internal Reference Specification

PURPOSE

This document is a guide to the design and implementation of the firmware on the 98629A interface card, known as "THEODORE".

SCOPE

This document is not intended to be a complete description of the firmware on the 98629A. It is a AID to understanding the workings of the firmware. The code listing contains many comments and notes which explain some details about the firmware.

TABLE OF CONTENTS

98629A IRS -- POWERUP/RESET

Powerup code:

1. ROM verification
   a. calculate logical redundancy checksum (LRC); try to generate error if bad

2. RAM verification
   a. do checkerboard test; generate error if failure

      failure if:

      -- memory doesn't contain what is written

      -- memory doesn't reside at correct addresses

3. CPU initialization
   a. Initialize stack pointer
   b. Set the interrupt vector register
   c. Set the interrupt mode

4. Read default switches and set modem drivers
   a. read node address from switches and store in RAM
   b. clear RESET latch
   c. select external clock

5. Peripheral verification and initialization

98629A IRS -- POWERUP/RESET

a. reset CTC channels and set up CTC interrupt vector

b. set CTC channel to interrupt, and verify that it interrupts after minimum time

c. reset SIO channels

d. set up SIO channels

e. verify all status registers in SIO

f. test semaphore for functionality then leave it set.

Common powerup/reset code:

6. Test hooks

   a. allow SA test interrupt

   b. allow execution of downloaded code

7. Set up data structures

   a. clear RAM

   b. initialize buffer pointers

   c. set up the data structures descriptor

   d. initialize alternate registers/SIO for data reception

   e. initialize COMMAND, ERROR_CODE, DSDP, etc.

98629A IRS -- POWERUP/RESET

8. Signal test complete to mainframe

   a. if powerup, give error interrupt with ERROR_CODE=0 if no error, with ERROR_CODE=6 if self-test error

   b. clear semaphore

98629A IRS -- IDLE LOOP

Idle loop

Functions performed in idle loop:

-- check response queue (RESP_QUEUE) for any packet responses (UA, RR, RCR) to be transmitted

-- check control queue (CTRL_QUEUE) for any connection request packets (SABMs) to be transmitted

-- check the COMMAND register for control/status commands

-- check Rx buffer for amount of room available for packet reception; update RX_ROOM and alternate registers

-- if data has been received, update the Rx data and control fill pointers (FPs); interrupt mainframe if enabled

-- monitor Tx data and control FPs for appearance of data and/or control blocks

IDLE:

```
VAR  CTL_BLOCK: BOOLEAN; (* indicates presence of a TX ctrl block*)
     RX_ROOM: 0 .. 2047; (* count of available bytes in Rx buffer *)
     RX_SIZE: 2048 (* size of Rx data buffer *)
     RX_BYTE_COUNT: 1 .. 256; (* B' register in CPU *)
     RX_BLOCK_COUNT: 0 .. 7; (* number of 256 byte blocks avail in Rx
          buffer *)

begin (* idle loop *)

CTL_BLOCK:=FALSE;
RX_ROOM:=RAMSIZE-1;
RX_BYTE_COUNT:= RX_ROOM MOD 256;
RX_BLOCK_COUNT:=RX_ROOM DIV 256;
```

LTR	PC NO	APPROVED	DATE
	SEE PAGE 1 FOR REV.		
	REVISIONS		

MODEL: 98629 IRS	STK NO:
BY: Alan Nelson	DATE: 7-21-82
APPD: B Shafton	SHEET NO: 6 OF: 27
SUPERSEDES:	DWG NO: A-98629-90302-1

98629A IRS -- IDLE LOOP

```
while true do
     begin
          CHECK_RESP_QUEUE (* check for responses to be transmitted *)

          if CTRL_QUEUE_FILL<>CTRL_QUEUE_EMPT then
               begin
                    SABM_DEST:=CTRL_QUEUE_EMPT^;
                    CTRL_QUEUE_FILL:=CTRL_QUEUE_EMPT+1;
                    if CTRL_QUEUE_EMPT=CTRL_QUEUE_END then (*do wrap*)
                         CTRL_QUEUE_EMPT:=CTRL_QUEUE
                    if SEQ_TAB(SABM_DEST) then
                         TX_SABM(SABM_DEST)  (* Tx a SABM *)
               end (* CHECK_CTRL_QUEUE *)

          CHECK_COMMAND (* check for control/status commands *)

          if RX_EP<>RXDATABUFF_EMPT then     (* Rx EP has moved *)
               begin
                    RX_EP:=RXDATABUFF_EMPT
                    if RXDATABUFF_EMPT<>RXDATABUFF_FILL then
                         INT_DRIVERS (READ_AVAIL_INT)  (* intr. MF *)
               end;

          RX_ROOM:=(RX_SIZE+RXDATABUFF_EMPT-1+RX_FP_INV) MOD RX_SIZE;
               (* space avail in RXDATABUFF is RX_ROOM; RX_FP_INV is
                  2's complement of working Rx FP (RX_FP) *)
          RX_BYTE_COUNT := RX_ROOM MOD 256;  (* init reg. B' *)
          RX_BLOCK_COUNT:= RX_ROOM DIV 256;  (* init reg. D' *)
          if RX_BYTE_COUNT := 0 then
               RX_BLOCK_COUNT := RX_BLOCK_COUNT -1; (* B'=0 means
                    RX_BYTE_COUNT=256, so block count is decremented *)

          if RX_AVAIL then     (* Rx data is available *)
                         begin
                              RX_AVAIL := false
                              RXDATABUFF_FILL:=RX_FP;
                              RXCTRLBUFF_FILL:=RX_CTRL_FP;
                              INT_DRIVERS (READ_AVAIL_INT)
                         end;
```

			MODEL	STK NO	
			98629 IRS		
			BY Alan Nelson	DATE 7-2-182	
	SEE PAGE 1 FOR REV.		APPD B. Hanton	SHEET NO 7	OF 27
LTR	PC NO	APPROVED DATE	SUPERSEDES	DWG NO A-98629-90302-1	
	REVISIONS				

98629A IRS -- IDLE LOOP

```
      CTRL_BLOCK_IN_TX; (* check for Tx control blocks *)
      if CTL_BLOCK then
           begin
                GETCTRLBLK   (* consume the Tx control block *)
                if COMMAND= 101 then  (* is a CLEAR command *)
                     SEND_ERROR (REG_ADDR_BAD); (* give error;
                               CLEAR command must be queued *)
                else
                     EXEC_CTRL_BLOCK
           end (* Tx ctrl block executer *)
      else
           TEMP:=TXDATABUFF_FILL-TXDATABUFF_EMPT (* # bytes in Tx
                     buffer; may be negative if wrapped *)
           if TEMP =< TXDATABUFF_EMPT then (* buffer is wrapped *)
                TEMP:=TXDATABUFF_FILL+TXDATASIZE
           if TEMP >=TXDATASIZE-2 then (*too many chars w/o term*)
                begin
                     TX_DATABUFF_EMPT:=TX_DATABUFF_FILL; (*empty
                        the Tx buffer*);
                     SEND_ERROR (NO_TERMINATOR)
                end;
      end (* Tx ctrl block handler *)
end; (* idle loop *)
```

			MODEL	STK NO	
			98629 IRS		
			BY Alan Nelson	DATE 7-21-82	
SEE PAGE 1 FOR REV.			APPD B. Keaton	SHEET NO 8	OF 27
LTR	PC NO	APPROVED	DATE	SUPERSEDES	DWG NO A-98629-90302-1
REVISIONS					

98629A IRS -- RECEIVE

Receive

When data is received by SIO Channel A, and the destination address matches my node address, the SIO will interrupt and generate a vector which branches to a routine called RECEIVE. The interrupt cause is "Rx character available". It's interrupt vector is seventh in the vector table.

RECEIVE has a few things to do before it gets going. These tasks are as follows:

1. Save CPU status (flags and accumulator).

2. Read the first byte from the SIO (which is the destination address). This is done quickly, and buys time before SIO RX FIFO overrun.

3. Get the parameters in the alternate register set for the block move. These parameters include byte count, source port number, and destination address.

4. Enable a watchdog timer on the CTC which will pull the CPU out of its wait state if the receive clock is lost during a data reception block move. Its timeout value is 65536 CPU clock cycles (17.778 milliseconds, or 1778 byte times).

5. Read 1 more byte from the SIO quickly; it is the first byte put into the Rx data buffer.

6. Give a RETI command to the SIO, and enable interrupts so that reception of the end of message (EOM), which is a Special Receive condition, will interrupt the CPU from its block move instruction.

7. Finally, the CPU can begin its block move instructions.

RECEIVE:

98629A IRS -- RECEIVE

```
begin
     GET_RX_PARAMS;          (* EXX *)
     SAVE_CPU_STATUS;        (* EX AF,AF' *)
     READ_DEST_ADDR;         (* IN A,[SIOA_DATA] *)
     ENABLE_WD_TIMER;        (* write a control byte to ch. #0 in CTC *)
     READ_ONE_BYTE;          (* INI, ensure no Rx overrun *)
     WRITE_RETI;             (* write a 00111000B to SIO ch. A reg #0 *)
     ENABLE_INTR;            (* EI *)
     if RX_BYTE_COUNT = 0 then
          RX_BLOCK_COUNT:=RX_BLOCK_COUNT-1; (* if byte count is now
               zero, block count must be adjusted *)
     repeat
         BLOCK_MOVE;         (* INIR *)
         RX_BLOCK_COUNT:= RX_BLOCK_COUNT -1;
     until RX_BLOCK_COUNT< 0

     RX_OVERFLOW:= true;
     THROW_DATA_AWAY; (* wait for SIO interrupt *)
end; (* RECEIVE *)
```

The SIO must be enabled to interrupt on Receive Character Available, and on Special Receive Condition before data is received.

The registers will contain the following when the Special Receive Condition interrupt occurs:

B': byte count for the first block move (RX_BYTE_COUNT)
C': SIOA_DATA register pointer
D': block count for the number of blocks that can be written to the Rx data buffer (RX_BLOCK_COUNT)
HL': pointer to the Rx data buffer

```
RECEIVE:
          EX   AF,AF'                4T (* save CPU status, A reg *)
          IN   A,[SIOA_DATA]        11T (* read 1st byte in Rx FIFO
                                    15T before 1st port read *)
```

			MODEL	STK NO	
			98629 IRS		
			BY Alan Nelson	DATE 7-21-82	
	SEE PAGE 1 FOR REV.		APPD B. Knutson	SHEET NO 10	OF 27
LTR	PC NO	APPROVED DATE	SUPERSEDES	DWG NO A-98629-90302-1	
	REVISIONS				

98629A IRS -- RECEIVE

```
                EXX                          4T (* get alt. register set *)
                XOR   A                      4T (* A=watchdog time const=0 *)
                OUT   [CTC_0],A             11T (* start watchdog timer *)
                INI   (read 1 byte)         13T (* until SIO port is read *)

                                            32T  between 1st read & 2nd read

                                             3T + shared memory wait (SMW)
                                                  to finish INI
                LD    A,RETI_CMD             7T (* get 38H to write to SIO *)
                OUT   [SIOA_CSR],A          11T (* write RETI to SIO *)
                EI                           4T (* enable interrupt from SIO *)
                JP    Z,B_IS_ZERO           10T (* if B=0, an extra block will
                                                   be read if we don't set
                                                   D:=D-1*)
MORE_RX_ROOM:   INIR                        13T (* until SIO port is read *)

                                            45T normally between 2nd & 3rd read

B_IS_ZERO:      DEC   D                      4T decrement RX_BLOCK_COUNT
                JP    P,MORE_RX_ROOM        10T loop if more room

                                            59T between 2nd & 3rd reads if the
                                                "B_IS_ZERO" jump was taken

                LD    H,0                       HL points to ROM
NO_RX_ROOM:     IN    A,[C]                     throw data away
                INC   L                         update the LSB of the pointer
                JR    NO_RX_ROOM                wait for SIO interrupt
```

Extensive testing indicates that if interrupts are disabled in the background routine for 51T + 4 SMW (which have an indeterminate probability of occurrence) there are 24T left for reading the 1st byte from the SIO before Rx overrun occurs. Because of the unknown probability of occurrence (and the unknown distribution of lengths of shared memory waits), Rx overrun time is not known exactly. However since it takes 19T + 2 SMW to get to the ISR after the interrupt is sampled true by the CPU, Rx overrun occurs in > 51T+19T+24T or 94T. This should be regarded as a minimum. We know that Rx overrun occurs sometime after 94T, but how much is not known. It is likely that

actual overrun occurs somewhere around 95T to 98T. Consequently, interrupts in the background should not be disabled for more than 51T + (24T-15T) or 60T + 4 SMW. If the background disable time or the receive data routine is ever changed, it should be tested extensively for Rx overruns.

It is also known that there is a finite probability of Rx overrun if the "B_IS_ZERO" jump is taken. It will occur if the first byte is received by the SIO near the beginning of the 51T + 4 SMW disable time (calculating RX_ROOM in the background), and if the B' register has been set to 1 before data is received. This last condition will only occur if the RXDATABUFF is not empty, and RX_ROOM mod 256 = 1. Even if this occurs, the Link Access Protocol will assure that the packet will be re-transmitted 20 times, and one of these re-transmitted packets should be received properly because the RXDATABUFF will be empty by then.

98629A IRS -- SIO INTERRUPTS

There are two interrupt conditions from the SIO that are of interest. These are:

1) Channel A Rx character avail (already discussed under RECEIVE)

2) Channel A special receive condition

-- Rx overrun

-- received incorrect CRC

-- Rx end of frame

The other six interrupting conditions should not occur so they will not be discussed here. The tasks that must be performed for the implemented interrupts are these:

1) Channel A Rx Character Available

The interrupt service routine for this interrupt is called RECEIVE, and has already been discussed.

2) Channel A Special Receive Condition

The conditions that can generate this interrupt are these:

-- Rx end of frame

-- Rx overrun

-- Received incorrect CRC

These three conditions all occur at the end of a reception of a message packet. The interface will rely on this interrupt to terminate the block move instruction being executed within RECEIVE when the end of frame occurs. The cause for this interrupt can be determined by reading SIO status register 1, where each of the above causes has its own status bit. The first actions, regardless of the cause for

the interrupt, will be to disable the watchdog timer and to throw away the return address so that the return from interrupt will return to the background routine and not to RECEIVE. The other actions are determined by the cause of the interrupt:

a. Rx overrun

An interrupt for this reason indicates a failure of the CPU to keep up with the data being received. Location RX_OVR_COUNT is incremented. Any data received prior to the overrun is ignored. It should be noted that the Link Access Protocol's retry mechanism will recover from this error if it occurs.

b. Received incorrect CRC

An interrupt for this reason indicates that data has been corrupted in transmission. The particular location and extent of the bad data is unknown, and since the destination address could be in error, any data received with an incorrect CRC is ignored. Location CRC_ERR_COUNT is incremented. It should be noted that the Link Access Protocol's retry mechanism will recover from this error if it occurs.

c. Rx end of frame

This condition is the normal termination of a received data packet (this is fortunate since the other two ignore the data !). When this interrupt cause occurs, the processor will do the following things:

-- throw away the return address so that the RETI ending the ISR will not cause the CPU to return to an INIR instruction

-- check for RXDATABUFF overflow (if there wasn't enough room to hold the whole Rx message RX_OVF_COUNT is incremented and message is ignored. The Link Access Protocol's retry mechanism will recover from this.)

98629A IRS -- SIO INTERRUPTS

-- turn off the watchdog timer

-- back up the buffer pointer past the CRC bytes in the buffer

-- if the received message is an information packet (I), roll call response (RCR), or an unnumbered information (UI, which are broadcast packets), the message is forwarded to the mainframe and an appropriate response may be queued. If the received message is a response (UA, RCR, or RR), the occurrence is noted and the message is thrown away. If the received message is a connection request (SABM), a response is queued and the message is discarded.

-- If the message is queued to the mainframe:

- update RX_FP, the working Rx data buffer fill pointer
- generate the appropriate control block in the Rx control buffer
- update RX_CTRL_FP, the working Rx ctrl buffer fill pointer
- set RX_AVAIL true
- update RX_ROOM & RX_FP_INV

-- set up interrupt registers for the next reception

-- restore CPU status as it was before the Rx Character Available Interrupt

-- enable interrupts

-- return from interrupt

98629A IRS -- SIO INTERRUPTS

The algorithm to perform these tasks is this:

```
SPEC_RX_INT (BUF_PTR);
begin
     STACK_POINTER:=STACK_POINTER+2; (* throw away the return *)
     if RX_OVERFLOW = true then
          RX_OVF_COUNT := RX_OVF_COUNT +1;
     else
          begin
               case SIO_RR#1 of

                    RX_OVERRUN: RX_OVR_COUNT:=RX_OVR_COUNT+1;

                    RX_CRC_ERROR: CRC_ERR_COUNT:=CRC_ERR_COUNT+1;

                    RESIDUE_ERROR: CRC_ERR_COUNT:=CRC_ERR_COUNT+1;

               case else  (* Rx data correctly *)
                    begin
                         INIT_CTC;
                         BUF_PTR:=BUF_PTR-2;
                         if BUF_PTR < RXDATABUFF_ADDR then
                              BUF_PTR:=BUF_PTR+RX_SIZE
                         RX_COUNT:= BUF_PTR - RX_FP
                         case PACKET_TYPE of

                         UA_TYPE, RR_TYPE:
                              begin
                                   ACK_MBOX_TYPE:=PACKET_TYPE;
                                   ACK_MBOX_NODE:=SOURCE_ADDR;
                                   ACK_MBOX_VALID:=true
                              end (* UA_TYPE & RR_TYPE *)

                         RC_TYPE:
                              WRITE_RESP_QUEUE (SOURCE_ADDR, RCR_TYPE,
                                   USER_SEQ_LSB)

                         SABM_TYPE:
                              WRITE_RESP_QUEUE (SOURCE_ADDR, UA_TYPE)
                              SEQ_TAB.CONNECT [SOURCE_ADDR]:=true;
```

				MODEL 98629 IRS	STK NO
	SEE PAGE 1 FOR REV.			BY Alan Nelson	DATE 7-21-82
LTR	PC NO	APPROVED	DATE	APPD B. Nelson	SHEET NO 16 OF 27
	REVISIONS			SUPERSEDES	DWG NO A-98629-90302-1

98629A IRS -- SIO INTERRUPTS

```
            RCR_TYPE, UI_TYPE:
                WRITE_CTRL_BLOCK; (* generate the pointe
                    term, and mode in a Rx ctrl block*)

            I_TYPE:
              begin
                if not SEQ_TAB.CONNECT[SOURCE_ADDR] then
                  begin
                    CTRL_QUEUE_FILL^:=SOURCE_ADDR;
                    CTRL_QUEUE_FILL:=CTRL_QUEUE_FILL+1;
                    if CTRL_QUEUE_FILL=CTRL_QUEUE_END
                       then CTRL_QUEUE_FILL:=CTRL_QUEUE;
                    if CTRL_QUEUE_FILL=CTRL_QUEUE_EMPT
                       then RX_OVF_COUNT:=RX_OVF_COUNT+1
                  end (* not connected with node *)
                else
                  begin
                    Ns:=(TYPE_FIELD mod 15)*16;
                    Vr:=SEQ_TAB.Vr[SOURCE_ADDR];
                    if Vr=Ns then
                      begin
                        WRITE_CTRL_BLOCK;
                        Vr:=Vr+1;
                        SEQ_TAB.Vr[SOURCE_ADDR]:=Vr;
                        EXP_RESP_TYPE:=RR_TYPE+Vr;
                        WRITE_RESP_QUEUE (EXP_RESP_TYPE,
                           SOURCE_ADDR)
                      end (* I packet in sequence *)
                     else
                      begin
                        EXP_RESP_TYPE:=RR_TYPE+Vr;
                        WRITE_RESP_QUEUE (EXP_RESP_TYPE,
                           SOURCE_ADDR)
                        BAD_SEQ_NODE:=SOURCE_ADDR;
                      end (* I packet out of sequence *)
                  end (* connected with node *)
              end (* I packet case *)
          end (* Rx data correctly case *)
INIT_ALT_REGS;   (* get prepared to receive data again *)
RESTORE_STATUS;
```

				MODEL	STK NO
				98629 IRS	
				BY Alan Nelson	DATE 7-21-82
	SEE PAGE 1 FOR REV.			APPD B. Hanton	SHEET NO 17 OF 27
LTR	PC NO	APPROVED	DATE	SUPERSEDES	DWG NO A-98629-90302-1
	REVISIONS				

98629A IRS -- SIO INTERRUPTS

```
     ENABLE_INTR;
     RETURN_FROM_INTERRUPT
end; (* Special Receive Interrupt *)


WRITE_CTRL_BLOCK (SOURCE_ADDR_PTR, BUF_PTR, RX_CTRL_FP, RX_CTRL_EP,
                  RX_COUNT);

begin
     RX_CTRL_FP^:=(RX_PTR div 256) mod 128; (*MSB of RX_PTR w/ most
                                    significant bit cleared for M/F*)
     RX_CTRL_FP:=RX_CTRL_FP+1;
     RX_CTRL_FP^:=(RX_PTR mod 256); (* LSB of RX_PTR *)
     RX_CTRL_FP:=RX_CTRL_FP+1;
     RX_CTRL_FP^:=END_DATA_TERM;
     RX_CTRL_FP:=RX_CTRL_FP+1;
     RX_CTRL_FP^:=END_DATA_MODE;
     RX_CTRL_FP:=RX_CTRL_FP+1;
     if RX_CTRL_FP=RXCTRLBUFF_END then
          RX_CTRL_FP:=RXCTRLBUFF    (* do wrap *)
     if not RX_CTRL_FP:=RX_CTRL_EP then  (* not overflowed the Rx
                                            ctrl buffer *)
          begin
               TYPE_FIELD:=0; (* clear command/type field for M/F *)
               RX_FP:= RX_PTR; (* set working FP *)
               RX_FP_INV:=-RX_FP; (* get 2's complement RX_FP *)
               RX_AVAIL:= true;
               RX_ROOM:=RX_ROOM-RX_COUNT;
          end (* Rx ctrl buffer not overflowed *)
end; (* WRITE_CTRL_BUFFER *)
```

LTR	PC NO	APPROVED	DATE
	SEE PAGE 1 FOR REV.		
	REVISIONS		

MODEL	STK NO
98629 IRS	
BY Alan Nelson	DATE 7-21-82
APPD B. Newton	SHEET NO 18 OF 27
SUPERSEDES	DWG NO A-98629-90302-1

98629A IRS -- TRANSMIT

Transmit

Data is transmitted by the interface to another GANGLINK node whenever a data packet is given to it by the mainframe. When the control block is generated by the mainframe, and the Tx data buffer fill pointer is moved, the interface will send the data over the link. The tasks performed by the interface are these:

-- fill in the source address, packet length, & control fields

-- get a pointer to the first byte in the message packet

-- enable the transmitter to send flags

-- monitor CS and do a timeout on it until it goes true

-- when CS goes true, disable the receiver

-- calculate the number of bytes to be transmitted, taking wraparound into account

-- enable the watchdog timer

-- call the appropriate transmit routine

-- disable the watchdog timer

-- enable the receiver if room in the receive buffer

-- wait the basic waiting time (to enable the receiver(s) to prepare for another reception)

-- disable the transmitter

98629A IRS -- TRANSMIT

This also is described algorithmically thus:

```
PROCEDURE TRANSMIT (TX_FP; TX_EP; TX_DATA_BUFFER);

begin
     if TX_FP > TX_EP then  (* no buffer wraparound *)
          PACKET_SIZE := TX_FP - TX_EP -1;
          TX_WRAPPED:=false;
     else
          begin
               TX_WRAPPED:=true;
               PACKET_SIZE:= TX_FP +TXDATABUFF_SIZE -TX_EP -1;
          end; (* wrapped *)
     if (PACKET_SIZE > 767) or (PACKET_SIZE < 5) then
          SEND_ERROR (NO_TERMINATION);
     PACKET_SIZE:=PACKET_SIZE+2; (* add 2 for CRC *)
     WRITE_INTO_TX; (* write source address, packet length *)
     COMMAND_FIELD:=I_TYPE;
     if TX_LEVEL=2 then  (* roll call type *)
          COMMAND_FIELD:=RC_TYPE;
     if DEST_ADDR:= 255 then (* broadcast unnumbered info type *)
          COMMAND_FIELD:=UI_TYPE;
     if COMMAND_FIELD<>I_TYPE then
          begin
               TX_PACKET;  (* transmit the packet *)
               if GOT_MISSING_CLK then
                    SEND_ERROR (MISSING_CLK)
          end (* UI & RC type *)
     else (* Tx I packet *)
          begin
               if not SEQ_TAB.CONNECTED[SOURCE_ADDR] then
                    TX_SABM (* transmit an SABM, require UA response*)
               Vr:=SEQ_TAB.Vr[DEST_ADDR];
               Ns:=SEQ_TAB.Vs[DEST_ADDR];
               Vs:=Ns+1;
               SEQ_TAB.Vs[DEST_ADDR]:=Vs;
               COMMAND_BYTE:=Vr*32+Ns+16; (* COMMAND BYTE is I type *)
               TXDATABUFF.COMMAND_BYTE:=COMMAND_BYTE; (* put in packet
               ACK_MBOX_VALID:=false;
```

98629A IRS -- TRANSMIT

```
                repeat
                     begin
                          TX_RETRY_COUNT:=MAX_RETRIES;
                          TX_EP:=TXDATABUFF_EMPT;
                          TX_PACKET; (* transmit the data *)
                          EXP_TYPE:=SEQ_TAB.Vs[DEST_ADDR]*16+RR_TYPE;
                          WAIT_FOR_RESP(DEST_ADDR,EXP_TYPE,RX_TYPE);
                          if RX_TYPE=UA_TO_DEST then (*reset Vs if he
                                                      hasn't connected to us *)
                               SEQ_TAB.Vs[DEST_ADDR]:=0;
                          if RX_TYPE=EXP_TYPE then (* Rx correct RR *)
                               CORRECT_RESPONSE:=true
                          else
                               TX_RETRY_COUNT:=TX_RETRY_COUNT-1;
                      end (* Tx retry loop *)
                until CORRECT_RESPONSE or TX_RETRY_COUNT=0;
                if not CORRECT_RESPONSE then (* retry count exhausted*)
                     if GOT_MISSING_CLK then (* failed by msg. clock *)
                          SEND_ERROR(MISSING_CLK);
                     else
                          SEND_ERROR(LINK_ERR_NUM); (*data link failure
                     TX_EP:=TX_FP; (* consume the Tx packet *)
                     SEQ_TAB.CONNECT[DEST_ADDR]:=false; (* disconnect*)
                TXDATABUFF_EMPT:=TX_EP mod 32768 (* update EP *)
           end (* Tx I packet *)
end (* TRANSMIT *)

TX_PACKET(TX_COUNT,TX_WRAPPED_FLAG,TX_EP,TX_FP);
begin
     GET_CS_TRUE(SUCCESS);
     if not SUCCESS then
          SEND_ERROR(CTS_FALSE); (* CTS false too long *)
     else
          if not TX_WRAPPED_FLAG then (* Tx not wrapped *)
               begin
                    BLOCK_COUNT := TX_COUNT div 256;
                    BYTE_COUNT:=TX_COUNT mod 256;
                    if BYTE_COUNT = 0 then
                         BLOCK_COUNT := BLOCK_COUNT - 1;
                    ENABLE_WD; (* enable the watchdog timer in the CTC
```

98629A IRS -- TRANSMIT

```
                case TX_COUNT of

                <257: TX_FIRST_BYTE;
                      RESET_TX_EOM; (* enable intr. at Tx EOM *)
                      TX_BLOCK; (* Tx up to 255 bytes *)

                >256&<513: TX_FIRST_BYTE;
                           RESET_TX_EOM;
                           TX_BLOCK;
                           TX_BLOCK;

                >512: TX_FIRST_BYTE;
                      RESET_TX_EOM;
                      TX_BLOCK;
                      TX_BLOCK;
                      TX_BLOCK;

                end (* case *)
               INIT_CTC;
               if BUFFER_PTR=TXDATABUFFEND then
                    BUFFER_PTR=TXDATABUFF   (* do wrap *)
               TX_EP:=BUFFER_PTR;
          end (* Tx not wrapped *)
     else
          begin (* do wraparound *)
               ENABLE_WD; (* turn on the watchdog timer *)
               TX_EP' := TX_BUF_ADDR;
               PACKET_SIZE' := TX_FP - TX_EP';
               BLOCK_COUNT' := PACKET_SIZE' DIV 256;
               BYTE_COUNT' := PACKET_SIZE' MOD 256;
               if BYTE_COUNT' := 0 then
                    BLOCK_COUNT' := BLOCK_COUNT' -1;
               PACKET_SIZE := TX_BUF_END - TX_EP -1;
               BLOCK_COUNT := PACKET_SIZE DIV 256;
               BYTE_COUNT := PACKET_SIZE MOD 256;
               if BYTE_COUNT = 0 then
                    BLOCK_COUNT := BLOCK_COUNT - 1;
```

98629A IRS -- TRANSMIT

```
                    (* There is a lengthy case statement here in the
                       actual code which predetermines the transmit
                       routine to use based on BLOCK_COUNT,
                       BLOCK_COUNT', BYTE_COUNT, & BYTE_COUNT.
                       See the source code for details. *)

                    TX_ONE_BYTE;
                    ENABLE_TX_EOM;
                    TX_DATA_BEFORE_WRAP; (* see source code for details
                    TX_DATA_AFTER_WRAP;  (* see source code for details
                    TX_EP:= BUFFER_PTR;
                    INIT_CTC;
               end; (* wraparound *)
          WAIT_AWHILE; (* wait 800 uS for trailing flags to be sent *)
          ENABLE_RX; (* enable the SIO to receive data again *)
          DISABLE_TX (* disable the SIO from transmitting flags *)
end; (* TX_DATA *)
```

98629A IRS -- EXEC COMMAND

When the COMMAND register is found to be non-zero, the mainframe is requesting action of some kind from the interface. This action is to be a status response or a control command. Also, when a control block is written, it has the same effect as a control command. The following events take place:

1) Read the command from COMMAND

2) Execute the command, which may entail the following:

   a. If the command is a write to a control register (bit 7= 1) the register number is encoded in the least significant bits of the command, and the data to be written is in the DATA_REG. Thus, the processor reads DATA_REG and writes its contents into the control "register". Note that this register may not exist as an actual 8-bit memory location; it's bits may be mapped into different locations. Therefore, the control registers (and status registers) are LOGICAL registers which are managed by the processor on the card.

   b. If the command is a read from a status register (bit 7= 0) the register number is again encoded in the command. The processor will read the command, get the data requested, and write that data into DATA_REG.

3) At the end of the execution of either a status or control command, the processor will clear COMMAND as a signal of completion. If the mainframe expects to read data from DATA_REG, this is also a signal that the data is in DATA_REG.

This can be expressed in this manner:

```
PROCEDURE EXEC_CTRL_BLOCK (TERM; MODE);

begin
     case TERM of:
```

98629A IRS -- EXEC COMMAND

```
0,2,4,6-11,13-100,102..110,112..119,125..127:

            begin
                 SEND_ERROR (REG_ADDR_ERR)
            end;

      3: begin (* set protocol ID *)
            if MODE <> PROTOCOL_ID (which is 3) then
                 SEND_ERROR (REG_VALUE)
         end;

      5: begin (* transmit terminator *)
            TRANSMIT;
         end;

     12: begin (* set maximum # of Tx retries *)
            MAXRETRIES:= MODE
         end;

    101: begin (* CLEAR *)
          GET_ACCESS; (* get access to buffer pointers *)
          RX_DATA_FP:= RX_DATA_EP;
          RX_CTRL_FP:= RX_CTRL_EP;
          TX_DATA_FP:= TX_DATA_EP;
          TX_CTRL_FP:= TX_CTRL_EP;
          RELEASE_ACCESS (*release access to pointers *)
          INIT_SIOA (* initialize channel A of SIO *)
         end;

    111: begin (* ignore command *)
         end;

    120: begin (* JUMP TO RAM *)
           goto location pointed to by MODE
         end;

    121: begin (* INT_COND MASK *)
          INT_COND_MASK:= MODE
         end;
```

			MODEL		STK NO
			98629 IRS		
			BY Alan Nelson	DATE 7-21-82	
	SEE PAGE 1 FOR REV		APPD B. Hatton	SHEET NO 25	OF 27
LTR	PC NO	APPROVED DATE	SUPERSEDES	DWG NO A-98629-90302-1	
	REVISIONS				

98629A IRS -- EXEC COMMAND

```
            123: begin (* set magic poke/peek pointer *)
                  POKE_PEEK_PTR:= MODE
                 end;

            124: begin (* magic poke *)
                  POKE_PEEK_PTR^:=MODE
                 end;
end (* CONTROL command *)


EXEC_STATUS ;(* STATUS command *)

 REGISTER#:= COMMAND; case REGISTER of

              0..2,4,9-11,13-120,122,123,125-127:

               begin
                    SEND_ERROR (REG_RANGE)
               end;

              3: begin (* protocol ID *)
                    DATA_REG := PROTOCOL_ID
                 end;

              6: begin (* Node address returned *)
                    DATA_REG := NODE_ADDR *)
                 end;

              7: begin (* CRC error count *)
                    DATA_REG := CRC_ERR_COUNT
                 end;

              8: begin (* Rx overflow count *)
                    DATA_REG := RX_OVF_COUNT
                 end;

             12: begin (* Tx retry counter *)
                    DATA_REG := RETRY_COUNTER
                 end;
```

LTR	PC NO	APPROVED	DATE	REVISIONS
	SEE PAGE 1 FOR REV.			

MODEL	STK NO
98629 IRS	
BY Alan Nelson	DATE 7-21-82
APPD B. Hayton	SHEET NO 26 OF 27
SUPERSEDES	DWG NO A-98629-90302-1

98629A IRS -- EXEC COMMAND

```
            121: begin (* return INT_COND mask *)
                    DATA_REG := INT_COND_MASK
                 end;

            124: begin (* magic peek *)
                    case POKE_PTR of

                    SIOA_DATA, SIOA_CSR, SIOB_DATA, SIOB_CSR,
                     CTC_0, CTC_1, CTC_2, CTC_3,
                     SWITCH_BLK_A, SWITCH_BLK_B:

                           DATA_REG := POKE_PEEK_PTR^;

                    case else
                           SEND_ERROR (REG_ADDR_ERR);
                 end;
         end
end; (* EXEC_STATUS *)
```

				MODEL	STK NO
				98629 IRS	
				BY Alan Nelson	DATE 7-21-82
SEE PAGE 1 FOR REV.				APPD B. Knutson	SHEET NO 27 OF 27
LTR	PC NO	APPROVED	DATE	SUPERSEDES	DWG NO A-98629-90302-1
REVISIONS					