98638A DIO 8-PORT MUX

BEBOP/CONCERTO (DIO-8 MUX)

INTERNAL MAINTENANCE SPECIFICATIONS

February 1990

HEWLETT PACKARD

GRENOBLE NETWORKS DIVISION

REVISION HISTORY

BUG FIX MAY-1988 by Sylvie MOULIN

Due to a design "feature" of the CTC chip, a spurious interrupt is generated when Z80 performs a write to the chip to enable interrupts and when down-counter reaches zero value at the same time.
The fix affects CTC initialization for SIO/PIO loopback test in MX4ST. Only the first two lines of each CTC are used as baud rate generator. So the initialization is restricted to these four lines.

CHANGES : in MX4ST see labels MSIO_20 and MSIO_30
locations 2FC(H) and 30B(H)
LD B,4 has been replaced by LD B,2.

BUG FIX FEB-1990 by Sylvie MOULIN

This fix corrects forgettings of changes proposed in CARMEN (HP 98638A) IMS, paragraph "9.2 a) * SIO initialization : a)".
Each time WR0 is changed, bit 4 should be set to 1.
In the "original" firmware, changes (with regard to FORDYCE) have been made in MX4ST and MX4IN.

CHANGES : in MXPT0 see locations 1E(H), 30(H) and 45(H).
in MXPT1 see locations 1F(H), 31(H) and 46(H).
in MXPT2 see locations 20(H), 32(H) and 47(H).
in MXPT3 see locations 21(H), 33(H) and 48(H).
in MXSBR see locations 35(H) and 42(H).

BUG FIX FEB-1990 by Sylvie MOULIN

This fix reflects a FORDYCE (HP 98642A) fix written by Randy STOUT in August 1989.
The fix sets up RTSB line before initializing the first SIO port. This was accomplished by reversing the order of initialization of the four SIO ports. So now port 3 (SIO 1 ch B) is set up first, then port 2 (SIO 1 ch A), port 1 (SIO 0 ch B) and port 0 (SIO 0 ch A).
This fix avoids receiving garbage on a port before being in internal loopback mode.

CHANGES : in MX4ST
- see labels MSIO_120 and MSIO_170
  (six lines changed)
- see label ROM_SIO
  (order of the table reversed).

RS422 MODIFICATION FEB-1990 by Sylvie MOULIN

CONCERTO uses an RS422 ADP. On this RS422 ADP all the modem signals are hardware-loopbacked to keep the compatibility with the RS232 ADP (product number 40299-60002).
During Self Test there is a test to determine whether there is a loopback hood on each port. To detect that loopback hood data is looping through the following line combination :

out 3 (SR) ---> in 3 (IC).

With the RS422 ADP this test always detects a hood on all the ports. And then when the external loopback test starts, it fails because TX and RX are not loopbacked too ! (when there is no loopback hood on each port of course...)

CHANGES : in MX4ST
all the tests about external loopback have been deleted.

CHAPTER 1
PRODUCT IDENTIFICATION AND OVERVIEW

1.1 IDENTIFICATION

This document describes the internal structure of the firmware implemented for the HP-DIO II EIGHT MODEM PORT MULTIPLEXER CARD : HP 98638A.

This product will be referred to as CARMEN throughout this document.

Here is the list of software project members :

* Sylvie MOULIN in GND for the firmware,
* Perry SCOTT in FSD for the interface between firmware and driver.

Throughout this document HP 98642A product will be referred to as FORDYCE.

NOTE : This document assumes the reader has the full understanding of all the information given in the firmware External Reference Specifications (ERS).

1.2 OVERVIEW

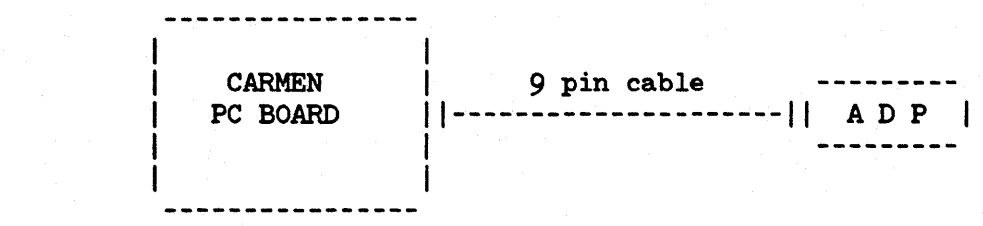
BEBOP is a multiplexor assembly designed for HP9000 serie 3X0 system. It provides the full modem connection of up to 8 asynchronous workstations to the system in a point-to-point configuration.

BEBOP is the project that releases CARMEN SPECIAL as an ING product.
BEBOP/CARMEN is the same product. CARMEN name will be used hereunder.

CONCERTO is an addendum to CARMEN. CONCERTO provides the RS422 connection to the DIO-8 Multiplexer.

CARMEN will be used starting with the #6.2 HPUNIX version and the following ones.

CARMEN is basically composed of 3 parts : a PC board assy, a cable, an Active Distribution Panel (ADP).



CARMEN is leveraged from the today 98642A four port Mux and re-use cable and connection box (ADP) from 40299A NIO Mux. It implements with CREM chip the SESAME architecture designed in Grenoble Network Division.

The PC Board is basically leveraged from the 98642A four port Mux : roughly 2 sets of this electronic are implemented on the board to provide 8 channels. Some electronic has been added to supply modem connections on all ports. All data and modem signals are multiplexed inside a chip coded name "CREM". Information is transmitted on a serial link cable to the Active Distribution Panel (ADP). Inside the ADP is also a "CREM" chip to demultiplexed data and modem signals.

As described above, the PC board is mainly composed of 2 four port Mux linked together from one side to DIO-II P2 connector and other side to CREM chip. These 2 sets of four ports work exactly in the same way.

CHAPTER 2
REFERENCE

2.1 RELEVANT DOCUMENTS

* HP-DIO I four channel terminal multiplexer
  firmware
  External Reference Specifications (ERS)
  by Elizabeth POTEET.

* HP-DIO I four channel terminal multiplexer
  firmware
  Internal Maintenance Specifications (IMS)
  by Elizabeth POTEET.

* HP 98642A four channel asynchronous multiplexer
  installation manual (no 98642-90001).

* ZILOG Z80 CPU (Central Processing Unit)
  Product specification.

* ZILOG Z80 CTC (Counter/Timer Circuit)
  Product specification.

* ZILOG Z80 SIO (Serial Input/Output controller)
  Product specification.

* ZILOG Z80 PIO (Parallel Input/Output controller)
  Product specification.

* MICROPROCESSOR APPLICATIONS REFERENCE BOOK (Volume 1)
  Using the Z80 SIO in asynchronous communications
  Application note.

2.2 GLOSSARY

The following is a list of the abreviations used in this document :

- RX : Receive, most commonly used to describe the characters which must be sent to the host from one of the ports.

- TX : Transmit, most commonly used to describe the characters which are sent by the host to one of the ports.

- ISR : Interrupt Service Routine.

CHAPTER 3
DESIGN OVERVIEW

3.1 DESIGN APPROACH

Because of the CARMEN board structure, there are two microprocessors, two RAMs and two ROMs. So there are also two CARMEN firmware : each one controls four modem ports to reach the number of eight for the CARMEN card.

It has been decided to reuse the FORDYCE firmware and to extend it to full modem ports.

3.2 OVERVIEW OF OPERATION

The purpose of this paragraph is to give an overview of the basic structure of the CARMEN firmware.

Except for the self test and initialization routine, all of the firmware on the card is completely interrupt driven.

The CARMEN firmware can be accessed in three ways:

A) system power up

B) soft reset

C) Z80 interrupts :
- SIO :
  * receive interrupt routines
  * receive error routines
  * transmit interrupt routines
  * external status interrupt routines
- CTC :
  * host interrupt service routine
  * timer interrupt service routine
  * modem timer interrupt service routine.

3.2.1 SYSTEM POWER UP

This causes a card reset and a jump to location 0h in the ROM where is MX4ST file.

This file consists in a self test which is followed in all cases by the MX4IN file (the initialization routine).

The end of initialization routine is an idle loop that is in essence the main routine of the firmware : it is called the "do nothing loop".

3.2.2 SOFT RESET

A soft reset causes a NMI interrupt to the Z-80 causing a jump to location 66h in the ROM.

This location is a routine which contains a call to the MX4IN file. So the initialization routine (ending with the "do nothing loop") is executed.

3.2.3 Z80 INTERRUPTS

The Z80 may be interrupted by either the SIOs or the CTCs.

3.2.3.1 INTERRUPTS COMING FROM THE SIOs

For each port, there are four different types of interrupts.

The interrupts vector associated with the SIOs looks like :

@ 1FC0	TX_1	transmit ISR	(port 1)
@ 1FC2	EX_1	external status ISR	(port 1)
@ 1FC4	REC_1	receive ISR	(port 1)
@ 1FC6	RX_ERR1	receive error ISR	(port 1)
@ 1FC8	TX_0	transmit ISR	(port 0)
@ 1FCA	EX_0	external status ISR	(port 0)
@ 1FCC	REC_0	receive ISR	(port 0)
@ 1FCE	RX_ERR0	receive error ISR	(port 0)
@ 1FD0	TX_3	transmit ISR	(port 3)
@ 1FD2	EX_3	external status ISR	(port 3)
@ 1FD4	REC_3	receive ISR	(port 3)
@ 1FD6	RX_ERR3	receive error ISR	(port 3)
@ 1FD8	TX_2	transmit ISR	(port 2)
@ 1FDA	EX_2	external status ISR	(port 2)
@ 1FDC	REC_2	receive ISR	(port 2)
@ 1FDE	RX_ERR2	receive error ISR	(port 2)

An interrupt coming from the SIOs, invokes one of the following files :

MX4TX (for the TX_i ISRs),
MXEXT (for the EX_i ISRs),
MX4RX (for the REC_i ISRs),
RXERR (for the RX_ERRi ISRs).

3.2.3.2 INTERRUPTS COMING FROM THE CTCs

The interrupts vector associated with the CTCs looks like :

@ 1FE4	HSTINT	(host interrupt)
@ 1FF0	TMR_ISR	(timer interrupt)
@ 1FF6	MDM_SUB	(modem interrupt)

Host interrupt:
Whenever the host writes a value to the COM_REG, an interrupt is generated to the Z80 via the CTC 0 channel 2.
The ISR invoked by the interrupts vector is HSTINT defined in the MXHST file.
The purpose of this routine is to determine the type of host interrupt called.

Timer interrupt :
The CTC 1 channel 2 is a 16 millisecond timer. When it times out, the Z80 is interrupted.
The ISR invoked by the interrupts vector is TMR_ISR defined in the MXTMR file.
The purpose of this routine is to send an interrupt to the host to inform it to check the Rx registers.

Modem interrupt :
The CTC 1 channel 3 is a timer which interrupts the Z80. The ISR invoked by the interrupts vector is MDM_SUB defined in the MXMDM file.
The purpose of this routine is to check all the modem input lines.

3.3 DESIGN CONVENTIONS & STANDARDS

Because of complexity and real-time delays of the HP-UX MUX driver, the driver writer didn't want to grab the SEM_REG (the semaphore register) any longer than necessary. So the read/write of some registers have been made outside the grab-ungrab frame (registers like MODM_OUT_i, MODM_IN_i, RFIFO, XFIFO).

CHAPTER 4
HARDWARE CONSIDERATIONS

4.1 Dual Inline Package (DIP) SWITCHES

There are 8 DIP switches on the CARMEN board :

1 indicates the system console connection
2...3 indicates the card interrupt priority
4...8 indicates the card select code.

WARNING : Number 8 (of select code) will always be set to zero by hardware. So you have no action on the least significant bit of the select code.

NOTE : Because of CARMEN board structure, the CARMEN card will be seen in two addresses. As there is only one DIP switches on this board, the hardware adds 1 to the select code (on the DIP switches) to obtain a second select code. So the CARMEN card will be in two consecutive addresses.
PORTS 0 to 3 of the ADP will be addressed by the LOWER select code (which ends by 0) and PORTS 4 to 7 of the ADP by the HIGHER select code (which ends by 1).

4.2 CTCs

There are two CTCs for a Z80 microprocessor.

Each CTC has four counter/timer channels for a total of 8 available in the CARMEN firmware.

Four of these are used as baud rate generators (one for each port).

One is used for the interrupts coming from the host and two are used as timers.

The last one is unused.

CTC 0 channel 0 : baud rate generator for port 0
CTC 0 channel 1 : baud rate generator for port 1
CTC 0 channel 2 : host interrupt line
CTC 0 channel 3 : unused

CTC 1 channel 0 : baud rate generator for port 2
CTC 1 channel 1 : baud rate generator for port 3
CTC 1 channel 2 : timer for interface registers
CTC 1 channel 3 : modem timer for input lines

WARNING : make sure that there isn't CTC with the following date codes, 8727 or 8722, on the board.

4.3 SIOs

There are two SIOs for a Z80 microprocessor.

Each SIO has two channels. Each channel represents one port for the TX and RX lines.

SIO 0 channel A : port 0
SIO 0 channel B : port 1
SIO 1 channel A : port 2
SIO 1 channel B : port 3.

WARNING : the RTS signal of SIO 1 channel B is used to select the internal loopback on the CARMEN card, ie the RTS signal enables the frontplane RS232 buffer ICs.

4.4 PIOs

There are two PIOs for a Z80 microprocessor.

Each PIO has two channels. Each channel represents one port for the modem lines (CS, DM, RR, IC, SR, TR, RS).

PIO 0 channel A : port 0
PIO 0 channel B : port 1
PIO 1 channel A : port 2
PIO 1 channel B : port 3.

WARNING : make sure that there isn't PIO 8551 B version on the board.

4.5 FIRMWARE PRIORITY SCHEME

All firmware events will be interrupt driven.

When the Z-80 is executing an Interrupt Service Routine, interrupts will be disabled to prevent another interrupt from preempting the current routine. Therefore, the priority of the interrupts is dependent upon the priority of the SIO and CTC channels and their placement on the interrupt daisy chain. The following is a list of the firmware events in order of their priority (high to low) :

1. RECEIVE DATA - PORT 0
2. TRANSMIT DATA - PORT 0
3. RECEIVE DATA - PORT 1
4. TRANSMIT DATA - PORT 1
5. RECEIVE DATA - PORT 2
6. TRANSMIT DATA - PORT 2
7. RECEIVE DATA - PORT 3
8. TRANSMIT DATA - PORT 3
9. TIMER INTERRUPTS
10. HOST INTERRUPTS
11. MODEM TIMER INTERRUPTS

CHAPTER 5
DEFAULT SETTINGS

5.1 DEFAULT Dual Inline Package (DIP) SWITCHES

1	(console connection)	set to "0" (i.e. no)
2...3	(card interrupt priority)	set to "3" (i.e. highest)
4...8	(card select code)	set to "28" (in decimal)

5.2 DEFAULT LINE CHARACTERISTICS AND FORMAT

When the card powers up, it will set up the SIOs with the default line characteristics. The host will be able to change these after self test and initialization routine.

The following is a list of each line characteristic and its default value. The default line characteristics will be the same for each port.

* SPEED	set to	"9600 BAUD"
* NUMBER OF STOP BITS	set to	"1"
* PARITY	set to	"NONE"
* NUMBER OF BITS PER CHARACTER	set to	"8"

5.3 DEFAULT BIT MAP

After the initialization routine has been executed, the Bit Map will be cleared (i.e. all locations = 0).

In other words, the card will not be set to recognize any character.

5.4 DEFAULT TIMERS SETTING

The 16 millisecond timer will be off after power up and the initialization routine. The host is responsible for enabling the timer.

The modem timer will be on after power up and the initialization routine. It will cause an input modem lines check which has no effect toward the host until this one decides to start "work" (i.e. when MODM-MASK-i are different from zero).

CHAPTER 6
MODULE INTERFACE SPECIFICATIONS

6.1 FILES LIST

- MX4ST : self-test
- MX4IN : initialization routine
- MX4RX : receive ISR's for all four ports
- RXERR : receive error ISR's for all four ports
- MX4TX : transmit ISR's for all four ports
- MXTMR : 16 ms timer ISR
- MXHST : host ISR
- MXEXT : external status ISR's for all four ports
- MXPT0 : port specific interrupts for port 0
- MXPT1 : port specific interrupts for port 1
- MXPT2 : port specific interrupts for port 2
- MXPT3 : port specific interrupts for port 3
- MXSBR : subroutines for configuration change interrupt and for send break interrupt
- MXMOD : modem output lines change ISR
- EXTMR : 16 ms timer on/off ISR
- MXMDM : modem input lines change ISR
- MX_VA : variable labels
- MX4EQ : system equates

6.2 TOP-DOWN DIAGRAM

The following is an outline of the relationships of the firmware modules to each other and the source of the interrupt that starts off a particular chain of events.

The labels in parenthesis are those which are used in the code. The preceeding file names are more general because they referred to the UNIX files.

The file names that are indented are those routines that are called by the preceeding file name. For example, MX4ST calls MX4IN which in turn calls MXMDM.

6.2.1 WITHOUT INTERRUPT

```
MX4ST
  |-->> MX4IN
          |-->> MXMDM
```

6.2.2 WITHIN INTERRUPT

```
IT SIO --> MX4TX
           (TX_0
            TX_1
            TX_2
            TX_3)

IT SIO --> MX4RX
           (REC_0
            REC_1
            REC_2
            REC_3)

IT SIO --> RXERR
           (RX_ERR0
            RX_ERR1
            RX_ERR2
            RX_ERR3)

IT SIO --> MXEXT
           (EX_0
            EX_1
            EX_2
            EX_3)
```

```
IT CTC --> MXTMR
           (TMR_ISR)

IT CTC --> MXMDM
           (MDM_SUB
            MDM_COM)

IT CTC --> MXHST
           (HSTINT)
             |-->> MX4ST
                   (MX4ST)
                     |-->> MX4IN
                           (INIT)
             |-->> EXTMR
                   (TMROFF)
             |-->> MXMOD
                   (MODOUT)
             |-->> MXPT0
                   (ISRPT0)
                     |-->> MXSBR
                           (HSTCON
                            SNDBRK)
             |-->> MXPT1
                   (ISRPT1)
                     |-->> MXSBR
                           (HSTCON
                            SNDBRK)
             |-->> MXPT2
                   (ISRPT2)
                     |-->> MXSBR
                           (HSTCON
                            SNDBRK)
             |-->> MXPT3
                   (ISRPT3)
                     |-->> MXSBR
                           (HSTCON
                            SNDBRK)
```

6.2.3 VARIABLES AND EQUATES

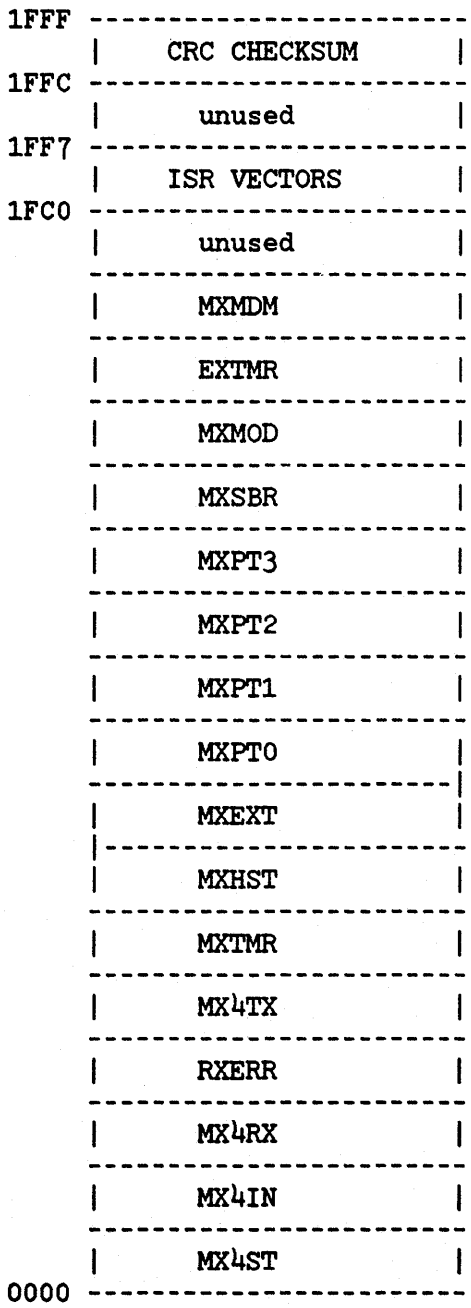
MX_VA

MX4EQ

6.3 ROM MAP

The following is an illustration of ROM showing the files position.

The value at the high byte of ROM is a CRC checksum value which is used to test ROM in the Self Test.



6.4 DETAILED DESCRIPTION OF FIRMWARE MODULES

This paragraph is devoted to a detailed description of each of the firmware modules. A firmware module is rather loosely defined as a piece of code with one entry point and one exit point which performs one basic function.

This paragraph identifies each of these modules by entry point name and shows the source of interrupt which causes the execution of the routine.

Included in the description is the following information : all labels which are either used or defined in the file which have impact on other files (simple in-file jump labels are not included), variables used in the file (all variables in the firmware are defined in the file MX-VA), and all macros called in the file (all macros used in the firmware are contained in the file MX4EQ).

The term "Global Labels" will be used to denote those labels which are defined in the file being described but used in other files.

The term "External Labels" will be used to denote the opposite : those labels which are defined in other files and used in the currently described file.

The term "Variables" will be used to describe those labels which are used to define a portion of RAM address space. As mentioned above, all of the variables in the firmware are defined in the file MX-VA and are therefore external to all of the other files.

6.4.1 MX4ST - SELF TEST

This file contains the entire Self Test.

```
Global Labels   : MX4ST, CTC-ERR0
External Labels : INIT
Variables       : ST-COND,TEST,PORT
Macros          : none
Include         : none
```

For more details, see chapter VII "SELF TEST".

6.4.2 MX4IN - INITIALIZATION ROUTINE

The initialization code is contained in this file. At the end of the initialization, the file also contains the "do nothing" loop that occupies the card while waiting for interrupts.

```
Global Labels    : INIT, BD-TAB
External Labels  : TX-0,TX-1,TX-2,TX-3,
                   REC-0,REC-1,REC-2,REC-3,
                   RX-ERR0,RX-ERR1,RX-ERR2,RX-ERR3,
                   EX-0,EX-1,EX-2,EX_3,
                   CONFG-0,CONFG-1,CONFG-2,CONFG-3,
                   BD-0,BD-1,BD-2,BD-3,
                   TMR-ISR,
                   CTC-ERR0,
                   HSTINT,
                   MDM-SUB
Variables        : THEAD-0,THEAD-1,THEAD-2,THEAD-3,
                   WR3-0,WR3-1,WR3-2,WR3-3,
                   TTAIL-0,TTAIL-1,TTAIL-2,TTAIL-3,
                   BITS-0,BITS-1,BITS-2,BITS-3,
                   WR4-0,WR4-1,WR4-2,WR4-3,
                   WR5-0,WR5-1,WR5-2,WR5-3,
                   TMRFLG,
                   C-MSTAT-REG,
                   PORT
Macros           : none
Include          : MX4EQ
```

EXTERNAL DESCRIPTION : INIT

This routine is divided in two parts. The first one is used by the self test to test NMI. The second part is the initialization code it-self. All the RAM is set to zero except for the ST-COND register which indicates the result of self test.

INTERNAL DESCRIPTION : INIT

```
Clear reset register
Reset cleared interrupt mode
Set initial stack address
Reset all SIOs
Load CTCs with baud rate and time constant value
Configure interrupt vector addresses in the SIOs
Program all SIO channel
Initialize RAM to 0
Initialize bits masks
Initialize the SIO write register variables
Initialize configuration and baud rate registers
Initialize all PIOs
Check the input modem lines
Release semaphore register
Initialize CTC timers
Mainline idle loop
```

6.4.3 MX4RX - RECEIVE ISR's

This file contains the Receive ISR's for all four ports. These routines are expanded macros.

```
Global Labels   : REC-0,REC-1,REC-2,REC-3
External Labels : none
Variables       : STAT-0,STAT-1,STAT-2,STAT-3,
                  RHEAD-0,RHEAD-1,RHEAD-2,RHEAD-3,
                  RTAIL-0,RTAIL-1,RTAIL-2,RTAIL-3,
                  BIT-MAP,
                  ICR-TAB,
                  BITS-0,BITS-1,BITS-2,BITS-3
Macros          : RECISR
Include         : MX4EQ
```

EXTERNAL DESCRIPTION : REC-0,REC-1,REC-2,REC-3

The four routines, REC-0, REC-1, REC-2 and REC-3 will be described together as they are virtually the same routine. The code for all four is defined in the macro RECISR.

The Receive routines are called when the SIO has received a character at one of the ports. The Z-80 accesses the correct vector location for the interrupt and causes a jump to the correct Receive routine.

The Basic purpose of the Receive routine is to retrieve the character from the SIO and place it in the correct Receive buffer in RAM along with an accompanying status byte. The character is placed in RAM, and then the Bit Map location for the character is checked to see if it is a special character, i.e., the host wants to know of it presence immediately. If the correct bit for the character and the port is set, a Special Character interrupt is sent to the host. It is the responsibility of the host to determine which character is special because the Special Character interrupt only notifies the host that such a character has been received. It doesn't specify which character it is and where in the buffer it has been placed.

```
INTERNAL DESCRIPTION : REC-0,REC-1,REC-2,REC-3

Retrieve head pointer index for Receive buffer
Retrieve tail pointer index for Receive buffer
Tail pointer = Tail pointer + 2
If Head = Tail then                          ;no more room in buffer
   Retrieve character and discard
   Set 'buffer overflow' bit in Status byte
   Go to exit
else                                         ;available buffer space
   Retrieve character from SIO
   Mask off any parity bits
   Put character into buffer
   Increment buffer address
   Put status byte into buffer
   Clear status byte register
   Tail pointer index = tail pointer index + 1
   Check correct Bit Map location
   If Bit Map position for port set then     ;special character
     Grab semaphore
     Set bit in ICR-TAB for port-specific interrupt
     Set bit in INT-COND register
     Clear semaphore
   Effective tail pointer = base + tail pointer index

UPON ENTRY      : No relevant values in any registers.

UPON EXIT       : No relevant values in any registers.

CALLED BY       : SIO - Receive character interrupt

CALLED ROUTINES : none
```

6.4.4 RXERR - RECEIVE ERROR ISR's

This file contains the Receive Error interrupt service routines for all four ports. These routines are expanded macros.

```
Global Labels   : RX-ERR0,RX-ERR1,RX-ERR2,RX-ERR3
External Labels : None
Variables       : STAT-0,STAT-1,STAT-2,STAT-3,
                  RHEAD-0,RHEAD-1,RHEAD-2,RHEAD-3,
                  RTAIL-0,RTAIL-1,RTAIL-2,RTAIL-3,
                  BIT-MAP,
                  ICR-TAB,
                  BITS-0,BITS-1,BITS-2,BITS-3
Macros          : SPEC-RX
                  RECISR
Include         : MX4EQ
```

EXTERNAL DESCRIPTION : RX-ERR0,RX-ERR1,RX-ERR2,RX-ERR3

As with the Receive routines, these four routines , RXERR0, RXERR1, RXERR2, RXERR3, will be described together as they too are virtually identical except for the port references. In addition, except for the addition of code to decipher the type of error, these routines are the same as the Receive routines. As a matter of fact, the same macro is called. Therefore, only the first portion of these routines will be described. At the end of that, each Receive error routine is identical to the regular Receive routine for that port.

The Receive Error routines are called when the SIO detects either a parity, framing, or SIO overflow error on the received character. The error type is denoted in the status byte and the Receive error then proceeds as the regular receive routine.

INTERNAL DESCRIPTION : RX-ERR0,RX-ERR1,RX-ERR2,RX-ERR3

```
Retrieve contents of SIO Read Register 1
Shift 1 bit to left                      ;so aligns with status byte
Mask off all but bits 7,6, & 5
Retrieve status byte register
Write new value to status byte
Reset SIO error latches
```

*The macro RECISR is now called - the routine proceeds exactly as a Receive routine.

UPON ENTRY : no relevant register values

UPON EXIT : no relevant register values

CALLED BY : SIO - RECEIVE ERROR INTERRUPT

CALLED ROUTINES : none

6.4.5 MX4TX - TRANSMIT ISR's

This file contains the Transmit interrupt service routines for all four ports. These routines are expanded macros.

```
Global Labels   : TX-0,TX-1,TX-2,TX-3
External Labels : none
Variables       : THEAD-0,THEAD-1,THEAD-2,THEAD-3,
                  TTAIL-0,TTAIL-1,TTAIL-2,TTAIL-3,
                  TON0,TON1,TON2,TON3,
                  ICR-TAB,
                  BITS-0,BITS-1,BITS-2,BITS-3
Macros          : TX-ISR
Include         : MX4EQ
```

EXTERNAL DESCRIPTION : TX-0,TX-1,TX-2,TX-3

As with the Receive and the Receive Error routines, these four routines, TX-0, TX-1, TX-2, TX-3, are also functionally identical, i.e. all four call the same macro. A Transmit interrupt is generated by the SIO as the SIO transmit buffer goes empty. In other words, the SIO interrupts the Z-80 when it is ready for another character to transmit.

The Transmit interrupt routine is responsible for retrieving a character from the appropriate Transmit buffer and sending it to the SIO. The Head and Tail index pointers for the Transmit buffer are first checked to see if the buffer is empty and the card sends the host a TX Buffer Empty interrupt. If it is, a value is sent to the SIO to turn off TX interrupts. If there are characters in the buffer, the next character is retrieved and sent to the SIO and the Head index is updated.

INTERNAL DESCRIPTION : TX-0,TX-1,TX-2,TX-3

```
Retrieve Head pointer index for Transmit buffer
Retrieve Tail pointer index for Transmit buffer
If Head = Tail then                              ;buffer is empty
  Turn off Transmitter interrupts from SIO
  Clear Transmitter on/off flag
  Grab Semaphore register
  Send host a TX Buffer Empty interrupt
  Release Semaphore register
else
  Effective Head pointer address = Head index + Base
  Retrieve character from TX buffer
  Send character to SIO
  Increment Head index
```

UPON ENTRY : no relevant register values

UPON EXIT : no relevant register values

CALLED BY : SIO - TRANSMIT BUFFER EMPTY INTERRUPT

CALLED ROUTINES : none

6.4.6 MXTMR - 16 MILLSEC. TIMER INTERRUPT

This file contains the CTC interrupt service routine which sends a Timer interrupt to the host.

```
Global Labels    : TMR-ISR
External Labels  : none
Variables        : none
Macros           : none
Include          : MX4EQ
```

EXTERNAL DESCRIPTION : TMR-ISR

This routine, TMR-ISR, is called every time the CTC timer associated with the routine is to send a Timer interrupt to the host to inform it to check the Receive buffers for characters.

INTERNAL DESCRIPTION : TMR-ISR

```
Grab Semaphore register
Send Timer interrupt to host          ;set bit in INT-COND register
Release Semaphore
```

UPON ENTRY : no relevant register values

UPON EXIT : no relevant register values

CALLED BY : CTC - TIME OUT INTERRUPT

CALLED ROUTINES : none

6.4.7 MXHST - HOST ISR

This file contains the beginning of the interrupt service routine which is invoked by CTC 0,CH 2 when the host puts a value in the COM-REG register. This file contains the portion of the host ISR which decodes the COM-REG register to decipher the reason for the interrupt.

```
Global Labels    : HSTINT,
                   EEE2,EEE3,EEE4,EEE5,EEE6,EEE7
External Labels  : ISRPT0,ISRPT1,ISRPT2,ISRPT3,
                   MODOUT,
                   TMROFF,
                   MX4ST
Variables        : TMPTAB,
                   CMND-TAB,
                   MINT-REG,
                   C-MINT_REG,
                   MODM-OUT-0,MODM-OUT-1,MODM-OUT-2,MODM-OUT-3
Macros           : none
Include          : MX4EQ
```

EXTERNAL DESCRIPTION : HSTINT

This routine, HSTINT, is called when the host writes a value to the COM-REG register, i.e. sends an interrupt to the card. This routine empties the contents of the CMND-TAB and COM-REG registers and begins checking the bits in both to determine what type of host interrupt was requested. When the interrupt has been interpreted the correct service routine is called. Once the interrupt has been completely serviced, control will return to this routine and a jump will be made to the beginning of the routine again to see if the host has sent another interrupt during the course of servicing the current one. This cycle will continue until the COM-REG register is empty.

INTERNAL DESCRIPTION : HSTINT

```
Grab Semaphore register
Retrieve value in COM-REG register
If COM-REG register = 0 then goto exit
else
 Retrieve value in CMND-TAB
 Clear COM-REG and CMND-TAB registers
 Release Semaphore register
 Check each bit in COM-REG reg. and jump to appropriate routine if set
 Go to beginning of routine
```

UPON ENTRY : no relevant register (Z-80) values

UPON EXIT : E register contains the remaining bits to be checked from the original value in the COM-REG register.

CALLED BY : CTC - HOST INTERRUPT

CALLED ROUTINES : ISRPT0,ISRPT1,ISRPT2,ISRPT3,MODOUT,TMROFF,MX4ST

6.4.8 MXEXT - EXTERNAL STATUS ISR's

This file contains the SIO External Status interrupt service routines for all four ports. An external status interrupt occurs when a Break has been received.

```
Global Labels    : EX-0,EX-1,EX-2,EX-3
External Labels  : none
Variables        : RBRK-0,RBRK-1,RBRK-2,RBRK-3,
                   STAT-0,STAT-1,STAT-2,STAT-3
Macros           : none
Include          : MX4EQ

EXTERNAL DESCRIPTION : EX-0,EX-1,EX-2,EX-3
```

As with the Receive, Receive Error, and Transmit routines, these routines, EX-0, EX-1, EX-2, EX-3, will be described together in this paragraph. These interrupt service routines are called when one of the SIO channels has a transition on the Break input. A TX underrun will also cause this interrupt although these routines will not take any action if that is what has triggered the ISR.

Break (BRK-SUB) is a subroutine which is called by all four routines. It will be described later in the current paragraph.

```
INTERNAL DESCRIPTION : EX-0,EX-1,EX-2,EX-3

Load parameters for Break subroutine
    Call BRK-SUB

UPON ENTRY       : no relevant registers

UPON EXIT        : Before calling BRK-SUB -
                   C reg = SIO control address for port
                   HL reg = Address of Break on/off flag for port
                   DE reg = Address of Status byte for port

CALLED BY        : SIO EXTERNAL STATUS INTERRUPT

CALLED ROUTINES  : BRK-SUB
```

EXTERNAL DESCRIPTION: BRK-SUB

BRK-SUB is a subroutine which is called by EX-0, EX-1, EX-2, and EX-3, the External Status interrupt service routines for ports 0 through 3. The purpose of this subroutine is to detect both the beginning of an incoming Break and the end of an incoming Break in the SIO. (See the Zilog Z80-SIO Product Specification details on how a Break is detected by the SIO).

INTERNAL DESCRIPTION: BRK-SUB

```
If Start-of-Break then              (BRK flag=0 and Break bit in SIO=1)
  Break Flag:=1
  Turn off RX interrupt             (To prevent interrupt for null char)
else
  If End-of-Break then              (BRK flag=1 and Break bit in SIO=0)
    Break Flag=0
    Error reset the port            (In case SIO is programmed for odd
                                    parity - null causes parity error)
    Set Break bit in status word    (Will get RX interrupt for the null
                                    char. when reinable)
    Reinable RX interrupt
```

```
UPON ENTRY      : C reg - SIO control address for port
                  HL reg - Address of Break on/off flag for port
                  DE reg - Address of Status byte for port

UPON EXIT       : B reg - Contains contents of SIO Read register  0

CALLED BY       : EX-0, EX-1, EX-2, EX-3

CALLED ROUTINES : none
```

6.4.9 MXPT0 - MXPT1 - MXPT2 - MXPT3 - PORT SPECIFIC ISR's

MXPT0

This file contains part of the Interrupt Service routine for a host interrupt. In particular it contains the routine for a port specific interrupt for port 0.

```
Global Labels   : ISRPT0
External Labels : BD-TAB,
                  EEE2,
                  SNDBRK,
                  HSTCON
Variables       : TMPTAB,
                  CONFG-0,WR3-0,WR4-0,WR5-0,BD-0,TTAIL-0,THEAD-0,
                  TON0,BITS-0
Macros          : HOSTTX
Include         : MX4EQ
```

MXPT1

This file contains part of the Interrupt Service routine for a host interrupt. In particular it contains the routine for a port specific interrupt for port 1.

```
Global Labels   : ISRPT1
External Labels : BD-TAB,
                  EEE3,
                  SNDBRK,
                  HSTCON
Variables       : TMPTAB,
                  CONFG-1,WR3-1,WR4-1,WR5-1,BD-1,TTAIL-1,THEAD-1,
                  TON1,BITS-1
Macros          : HOSTTX
Include         : MX4EQ
```

MXPT2

This file contains part of the Interrupt Service routine for a host interrupt. In particular it contains the routine for a port specific interrupt for port 2.

```
Global Labels   : ISRPT2
External Labels : BD-TAB,
                  EEE4,
                  SNDBRK,
                  HSTCON
Variables       : TMPTAB,
                  CONFG-2,WR3-2,WR4-2,WR5-2,BD-2,TTAIL-2,THEAD-2,
                  TON2,BITS-2
Macros          : HOSTTX
Include         : MX4EQ
```

MXPT3

This file contains part of the Interrupt Service routine for a host interrupt. In particular it contains the routine for a port specific interrupt for port 3.

```
Global Labels   : ISRPT3
External Labels : BD-TAB,
                  EEE5,
                  SNDBRK,
                  HSTCON
Variables       : TMPTAB,
                  CONFG-3,WR3-3,WR4-3,WR5-3,BD-3,TTAIL-3,THEAD-3,
                  TON3,BITS-3
Macros          : HOSTTX
Include         : MX4EQ
```

EXTERNAL DESCRIPTION : ISRPT0,ISRPT1,ISRPT2,ISRPT3

These four routines, ISRPT0, ISRPT1, ISRPT2, ISRPT3, will be documented together as they are virtually identical except for variable names. These four routines (one for each of the four ports) identify which port specific interrupt the host is sending from the bits in ICR-TAB. The interrupt can be either a Configuration Change interrupt, a TX Buffer Not Empty interrupt, or a Send Break interrupt (or any combination of the three).

The purpose of the Configuration Change interrupt is to reconfigure the line characteristics of the SIO and change the baud rate as desired by the host. The CONFG register contains the parity type, the number of stop bits, and the number of bits per character. This register is set by the host and accessed in this routine by the card. The BD register is the index to the BD table which contain the CTC Channel Control Word and prescale value for the baud rate requested.

The Send Break interrupt is fully contained in a subroutine called SNDBRK which will be described in its own section later in the current document.

```
INTERNAL DESCRIPTION : ISRPT0,ISRPT1,ISRPT2,ISRPT3

Retrieve bit 0 from TMP-TAB              (bit determined Confg. interrupt)
If bit 0 = 1 then
  Call HSTCON                            (routine does 1st part of Confg.)
  Load SIO Write Reg. 4 with new value
  Load SIO Write Reg. 5 with new value
  Load SIO Write Reg. 3 with new value
  Get contents of BD register
  Multiply by 2
  Add to BD Table base
  Retrieve CTC Channel Control Word from BD-TAB
  Send to CTC
  Inc pointer
  Retrieve CTC Time Constant value
  Send to CTC
Retrieve remaining bits from CMND-TAB
If bit 1 = 1 then                        (bit for TX Buffer Not Empty ISR)
  If Transmitter flag off then
    Retrieve Head Pointer Index for TX Buffer
    Retrieve Tail Pointer Index for TX Buffer
    If Head <> Tail then
      Obtain effective TX Buffer address
      Retrieve character
      Send character to UART (SIO)
      Increment index
      Turn on Transmitter flag
If bit 2 = 1 then                        (bit for Send Break interrupt)
  Call SEND BREAK routine
Return to calling routine                (calling routine is HSTINT)

UPON ENTRY      : REGISTER D - contains the TMP-TAB bits which were
                               retrieved from CMND-TAB
                  REGISTER E - DO NOT USE! The HSTCON routine uses this
                               register to hold the contents of the
                               COM-REG register. Remember, there can be
                               more than one interrupt at a time sent.

UPON EXIT       : REGISTER E - Unchanged

CALLED BY       : HSTINT

CALLS ROUTINES : HSTCON, SNDBRK
```

6.4.10 MXSBR - SUBROUTINES FOR PORT SPECIFIC ISR's

This file contains two subroutines which are part of the host interrupt service routine. These subroutines are called by ISRPT0, ISRPT1, ISRPT2 and ISRPT3. The first subroutine is part of a Configuration Change interrupt from the host. The second subroutine is part of the Send Break interrupt from the host.

```
Global Labels   : HSTCON,
                  SNDBRK
External Labels : none
Variables       : BITS-MSK
Macros          : none
Include         : MX4EQ
```

EXTERNAL DESCRIPTION : HSTCON

This subroutine, HSTCON, is the first part of the processing of a port specific Configuration Data Change Interrupt from the host. This routine basically changes the order of the bits read from the CONFG register to the corresponding bit patterns needed to program the SIO write registers. The basic algorithm of this routine is to start with the value of CONFG and change first the parity bits, then the stop bits, then the bits per character, to match the corresponding patterns needed to program the SIO write registers correctly. At the end of this routine, the A register will contain the three pieces of information in from the CONFG register with the bits changed so they match the bit patterns needed by the SIO write registers to make the actual configuration changes. However, this routine does not include actually programming the SIO write registers. That is done in the calling routine (as explained in the section on ISRPT0, ISRPT1, ISRPT2, ISRPT3). This routine does include programming the mask value in the BITS-i register which will be used to strip parity bits off of Receive characters. This mask is based on the number of Receive bits per character requested by the change. (This algorithm is described in the paragraph 6.5 "Common Algorithms").

INTERNAL DESCRIPTION : HSTCON

```
If bit 1 in CONFG=0 then bit 0=1 (even parity; set parity enable/WR4)
Rotate 2 bits right              (stop bits pattern same if add 1)
Increment A register             (contains original value of CONFG)
Rotate back 2 bits left
Swap bits 4 & 5                  (now matches bits per char in WR3 & 5)
If 8 bits per character
 BITS-i=FF
If 7 bits per character
 BITS-i=7F
If 6 bits per character
 BITS-i=3F
If 5 bits per character
 BITS-i=1F
```

```
UPON ENTRY      : A Reg - contains the CONFG register value
                  D reg - used by calling routine - DO NOT USE
                  E reg - used by calling routine - DO NOT USE

UPON EXIT       : A Reg - contains the altered value of CONFG reg.
                  D reg - unaltered
                  E reg - unaltered

CALLED BY       : ISRPT0, ISRPT1, ISRPT2, ISRPT3

CALLS ROUTINES  : none
```

EXTERNAL DESCRIPTION: SNDBRK

This subroutine, SNDBRK, is used when the host sends the card a Send Break interrupt. A break interrupt can be notifying the card to either begin or end a break. The card determines which by checking the Break bit in WR5. If the Break bit (bit 4)=0 then this is a start of break. If bit 4=1 then this is a signal to end a break.

INTERNAL DESCRIPTION : SNDBRK

```
If Break bit = 0 then
  Set WR5 bit 4 in WR5 variable
  Send new WR5 value to SIO
Else
  Reset WR5 bit 4 in WR5 variable
  Send new WR5 value to SIO

UPON ENTRY      : D reg - Used in the calling routine - DO NOT ALTER
                  E reg - Used in the calling routine - DO NOT ALTER

UPON EXIT       : D reg unaltered
                  E reg unaltered

CALLED BY       : ISRPT0, ISRPT1, ISRPT2, ISRPT3

CALLS ROUTINES  : none
```

6.4.11 MXMOD - MODEM OUTPUT LINE CHANGE ROUTINE

This file contains the part of the host interrupt service routine which is responsible for handling a Modem Output Line Change interrupt.

```
Global Labels   : MODOUT
External Labels : EEE6
Variables       : MODM-OUT-0,MODM-OUT-1,MODM-OUT-2,MODM-OUT-3,
                  MINT-REG,
                  C-MINT-REG,
Macros          : none
Include         : MX4EQ
```

EXTERNAL DESCRIPTION : MODOUT

This routine, MODOUT, is basically a subroutine called by the Host interrupt routine when a Modem Ouput Change is sent by the host. The purpose of this routine is to set the modem output lines to match the bit pattern in the MODM-OUT-i register. Bit 0 represents the RS line. Bit 1 represents the TR line and bit 2 represents the SR line. As there is no record of which line is different, this routine sets all the lines as indicated by the MODM-OUT-i register.

INTERNAL DESCRIPTION : MODOUT

```
For each port
    if there is a change on modem output lines
      then write new value on PIO
Return to the calling routine
```

UPON ENTRY : E Reg - used in the calling routine - DO NOT ALTER

UPON EXIT : E Reg unaltered

CALLED BY : HSTINT

CALLS ROUTINES : none although returns to HSTINT by a jump

6.4.12 EXTMR - TIMER ON/OFF ROUTINE

This file contains the part of the host interrupt service routine which is responsible for handling a Timer On/Off interrupt.

```
Global Labels   : TMROFF
External Labels : EEE7
Variables       : TMRFLG
Macros          : none
Include         : none
```

EXTERNAL DESCRIPTION : TMROFF

This routine, TMROFF, is part of the Host interrupt Timer On/Off interrupt service routine. The purpose of this routine is to either turn the 16 millisecond Receive buffer timer on or off. A flag is used to determine whether it is already on or off. If the flag is off, this routine turns the timer on and if the flag is on this routine turns the timer off. The flag is changed accordingly at the end of the routine.

INTERNAL DESCRIPTION : TMROFF

```
Retrieve the Timer flag
If Timer flag=1              (timer is already on)
 Turn off CTC timer
 Timer flag=0                (update timer flag)
If Timer flag=0              (timer is off)
 Retrieve CTC Channel Control Word
 Send to CTC
 Retrieve Time Constant Register value
 Send to CTC                 (restarts timer)
 Timer flag=1                (update timer flag)
Return to caller
```

UPON ENTRY : E Reg - used in the calling routine - DO NOT ALTER

UPON EXIT : E Reg Unaltered

CALLED BY : HSTINT

CALLS ROUTINES : none although returns to HSTINT by jump

6.4.13 MXMDM - MODEM INPUT LINE CHANGE ROUTINE

This file contains the CTC interrupt service routine and the subroutine itself which check the modem input lines.

```
Global Labels   : MDM-SUB,MDM-COM
External Labels : none
Variables       : MODM-IN-0,MODM-IN-1,MODM-IN-2,MODM-IN-3,
                  MODM-MASK-0,MODM-MASK-1,MODM-MASK-2,MODM-MASK-3,
                  MSTAT-REG,
                  C-MSTAT-REG,
                  MSTAT-FLAG,
                  PIO-WM
Macros          : none
Include         : MX4EQ
```

EXTERNAL DESCRIPTION : MDM-SUB

This subroutine is the ISR invoked by CTC 1 CH 3.

INTERNAL DESCRIPTION : MDM-SUB

```
Disable interrupts
Call MDM-COM
Enable interrupts
Return from interrupt
```

EXTERNAL DESCRIPTION : MDM-COM

The purpose of this subroutine is to check the status of the four modem input lines and see whether or not there has been a change in the lines. If there has been a change, this routine then reflects that change in the MODM-IN-i register and checks the MODM-MASK-i register to see if the host wants to be interrupted. If the bit in MODM-MASK-i representing the changed line is on, the card will then send an INPUT MODEM LINE CHANGE INTERRUPT to the host. The four lines that this routine deals with are the modem lines RR, CS, and DM.

INTERNAL DESCRIPTION : MDM-COM

```
For each port
  Read PIO
  If bits read = RR, CS, and DM bits in MODM-IN-i then Exit
  else
    MODM-IN-i := PIO read bits
    If MODM-IN-i.AND.MODM-MSK-i > 0 then
      grab semaphore
      send Input Modem Line Change Interrupt to host
      release semaphore
```

UPON ENTRY : none

UPON EXIT : no relevant register values

CALLED BY : INIT,MDM-SUB

CALLS ROUTINES : none

6.4.14 MX-VAR - VARIABLES

This file contains all of the variable labels which are used in the firmware. The file is divided into two segments, a data segment and an absolute segment. The variables defined in the data segment (DATA) are those used only by the firmware and the variables defined in the absolute segment (ORG) are used by both the card and the host.

Global Labels	: every label defined in the file is public. All of the labels listed as "Variables" in the other file descriptions are Global Labels in this file
External Labels	: none
Variables	: not applicable
Macros	: not applicable
Include	: not applicable

6.4.15 MX4EQ - EQUATES

This file is not part of the object code. It is a sort of service file which contains all of the equates used in the firmware and defines all of the macros. This file is copied to almost every other file in the firmware with the exception of MX4ST and MX-VA .

The names of the macros contained in this file are:

RECISR - Used in MX4RX
SPEC-RX - Used in RXERR
TX-ISR - Used in MX4TX
HOSTTX - Used in MXPT0, MXPT1, MXPT2, MXPT3

6.5 COMMON ALGORITHMS

6.5.1 RECEIVE BUFFER, EMPTY/FULL DECISION

The Receive buffers are handled as circular FIFO data structures with an associated head and tail index for each. The algorithm used here never lets the buffer get completely full, so when then head and tail indexes are equal it means that the buffer is empty, not full. The method for making sure the buffer never gets completely full is to add 2 to the tail index and check for equality with the head index before a Receive character is placed into the buffer. If they are equal, the buffer is assumed full and the character received is discarded. In essence this means that there is really only room for 127 characters per Receive buffer instead of 128.

6.5.2 RECEIVE HEAD & TAIL POINTER HANDLING

The head and tail pointers for the Receive buffers consist of Head and Tail index pointers and a base pointer address. The base pointer is the upper byte of the Receive buffer address and the head and tail index pointers are the lower bytes. The effective address then for any address in a Receive buffer is the concatenation of the base and the head or tail index. As the Receive buffers are only 256 bytes, buffer wrapparound is automatically taken care of as the head and tail indexes are 8 bit quantities.

6.5.3 BIT MAP CHECK

As explained in the Firmware ERS, the Bit Map is a 256 byte table with each byte representing a character. In other words, the character whose value is 56 is associated with the byte in the Bit Map whose relative placement in the table is 56 from the beginning of the table. The first four bits in each Bit Map location represent the four ports. When a character is received, it is concatenated with the Bit Map Base value to form the effective address of the Bit Map location associated with the character. Once the byte is retrieved, the bit representing the port the character was received at is checked. If the bit is on, the character is a "special" character and a Special Character interrupt is sent to the host. If the bit is off, no interrupt is sent.

6.5.4 STRIPPING PARITY BITS

After each received character is retrieved from the SIO, a logical AND is performed with it and the contents of a location called BITS-MSK. This location contains a mask designed to strip off any possible parity bits that might be at- tached to the character. The value of BITS-MSK is based upon the number of bits per character the card is configured to. If the card is configured to 7 bits per character, BITS-MSK will contain a "1" in the first 7 bit locations and a "0" in the 8th bit. If the card is set to 6 bits per character, BITS-MSK is 00111111 or 3F hex. The same idea holds for other bits per character settings. BITS-MSK is updated every time the card is reconfigured.

6.5.5 SENDING AN INTERRUPT TO THE HOST - USE OF SEMAPHORE REGISTER

As described in the ERS, when the card wants to send an interrupt to the host it writes a value to the INT-COND register. However, before writing to either the INT-COND register or the ICR-TAB, the card will first "grab" the Semaphore register. In other words, the card will check the Semaphore register to see if it is free. If not, the card will sit and cycle, continually checking the Semaphore register until the host releases it. The basic protocol is the same for the card and the host. Both grab the Semaphore before accessing either the COM-REG register, the INT-COND register, the CMND-TAB registers, or the ICR-TAB registers.

6.5.6 STATUS BYTE

There is a location reserved for the status byte which is initially set to zero in the initialization routine. This byte is retrieved and written to the appropriate Receive buffer as each character is placed in the buffer. If there is no room in the buffer and the receive character is discarded, the buffer overflow bit is set in this byte. The next character that is placed in the Receive buffer will also have the status byte with the overflow bit set, notifying the host that there are missing characters between the last one picked up and the current character. The Receive error routine also can alter the status byte to display error conditions associated with an incoming character. However, once a character is placed in the buffer with the status byte, the status byte register is cleared for the next character.

6.5.7 TRANSMIT BUFFER, EMPTY/FULL DECISION

As with the Receive buffers, the Transmit buffers are also handled as circular FIFO buffers. Also, the Transmit algorithm which resolves empty or full buffer arbitration is the same for Transmit Buffers as it is for Receive buffers. In the case of the Transmit buffers, the host never lets the buffer get completely full, so when the Head and Tail pointer indexes are equal, the buffer is empty.

6.5.8 TRANSMIT HEAD & TAIL POINTER HANDLING

As with the Receive buffer pointers, the head and tail pointers are actually a concatenation of head and tail pointer indexes and a Transmit buffer base address (which represents the upper byte of the actual Transmit buffer address). However, unlike the Receive buffer pointers, the Transmit head index actually consists of two values, the base lower byte and the head pointer index. Buffer wrapparound is handled by incrementing the head pointer index, and masking off the top nibble. When the actual pointer address is needed, the head index is added to the base lower byte and the result is concatenated with the base upper byte.

6.5.9 REASON FOR CYCLING IN HSTINT ROUTINE

As with all of the interrupt service routines, the HSTINT is non-interruptable. In other words, interrupts are disabled at the start of the routine and reinabled at the end of the routine. Consequently, during the course of this routine, if the host sends another interrupt it will be lost because the CTC can't buffer interrupts. Therefore, this routine will keep checking for and servicing interrupts until the COM-REG register is empty.

6.5.10 DECIPHERING THE TYPE OF INTERRUPT

The E register is used to hold the contents of the COM-REG register as it is being deciphered. Each bit position in the COM-REG register represents a particular interrupt (with the exception of bit 7). Therefore, the interrupts are deciphered by putting the value in the COM-REG register into the E register and rotating each bit to the right one by one testing the carry bit each time. If a bit is on, this routine jumps to the subroutine responsible for handling that particular interrupt. It is possible for there to be more than one interrupt set in the COM-REG register. When program control returns from a subroutine, this routine resumes checking the rest of the bits.

6.5.11 CHANGING THE SIO WRITE REGISTERS TO NEW CONFIGURATION

The subroutine, HSTCON, is responsible for changing the bit pattern in the CONFG register to match the format in the SIO Write registers (This is explained in more detail in the section on HSTCON.) Upon return from HSTCON the B register contains the changed bit pattern. Write Register 4 is updated by clearing out the old lower byte and ANDing it with the lower byte of the B register value which contains the bits representing the new parity and stop bits information. The new value in Write register 4 is then written to the SIO. SIO Write register 5 is updated next. Bits 5 and 6 in the B register value represent the new TX bits-per-character information. Bits 5 and 6 in the old Write register 5 are cleared and the replaced with those in the B register. This is then written to the SIO. Finally, bits 6 & 7 in the B register are substituted for bits 6 and 7 in Write register 3. These bits represent the RX bits-per-character information. The new copy of Write register is then written to the SIO.

6.5.12 CHANGING THE BAUD RATE

The BD register contains a number which represents an index into the BD-TAB, the table which contains the CTC Channel Control Words and the CTC Time Constant values which determine a specific baud rate. The value in the BD register is multiplied by 2 (since each baud rate has the two associated CTC values) and added to the base BD-TAB address to form the effective address. The correct Channel Control Word and the Time Constant value are then sent to the CTC.

6.5.13 MUTUAL MANAGEMENT FOR MODEM LINES

The management of MSTAT-REG and MINT-REG is the responsability of both the card and the host.

For MSTAT-REG the card writes it and the host , after reading it, clears it. For MINT-REG the host writes it and the card, after reading it, clears it.

CHAPTER 7
SELF TEST

The purpose of this chapter is to give a detailed explanation of the Self Test firmware. Self Test is that portion of code which attempts to functionally test all accessible hardware on the board. It includes a ROM test, a RAM test, a CTC test, a SIO test and a PIO test. Self Test resides in the EPROM beginning at address 0H.

There are two ways that Self Test can be invoked: it is automatically invoked during power up when the Auto Reset line on the Z-80 is pulled and it may be invoked by a Self Test interrupt from the host.

The following paragraphs will contain a detailed explanation of each of the component tests in the Self Test. The term "component" is rather loosely here to refer piece of hardware that can be separately tested.

7.1 SELF TEST INITIALIZATION

The following tasks are done before any of the hardware component tests:

* Interrupts are disabled.

* The stack pointer is initialized.

* The reset bit in the Reset/I.D. register is cleared (the reason is that the state of this bit is indeterminate after power up and must be cleared for a Soft Reset to be issued).

* The CTC, SIO and PIO channels are all reset. They should all have been reset automatically during the power up.

* The IX register which is used to identify the type of failure (in case Self Test fails) is set to zero. As each test is performed, the IX register is incremented.

* The COM-REG register is first set to zero, then read back into the A register. This is to insure that it is both cleared and that the interrupt line is reset (remember that a write from the host to the COM-REG register sends an interrupt to the card and a read from the card to the COM-REG register clears it. Also, the state of the COM-REG register is indeterminate after power up so this puts it in a known state).

NOTE : the COM-REG register cannot really be tested in Self Test because, as explained above, the host must write to it to generate an interrupt.

* At the end of this self-test, when it is successful, all the modem lines (on PIOs) are inactives (i.e. set to "0").

WARNING : FOR SIOs (see the application note "using the Z80 SIO ...") :

- "Up to two transitions can be remenbered by the internal logic of the SIO. Therefore, it is desirable to do at least two different Reset External/Status Interrupt commands as late as possible in the initialization".
  "Since it doesn't hurt, these commands are given each time WR0 is changed to point to another register. This is an easy way to code the initialization to ensure that the appropriate resets occur".

- "The SIO contains a three-character input buffer for each channel". During power up (or just after), the CREM (a circuit on the board) sometimes sends garbage to the SIO. In that case the input buffer has to be emptied (more details in the SIO-TEST subroutine).

7.2 RESERVED ADDRESSES

There are a few reserved locations in the first part of the Self Test. These are addresses that have (or might have) fixed meaning in certain circumstances.

There are only two that are fixed : addresses 038H and 066H. They have been chosen because the Self Test jumps around this section of addresses (35H to 6DH).

The interrupt vectors for the CTC test are placed in between (addresses 48H to 57H) simply because there was room there.

7.2.1 ADDRESS 38H

This address is the one triggered if the Z-80 ever gets the value FFH as an operation code.

If the program ever jumps outside the legal address space, ie physical memory, the value the Z-80 gets will most likely be FFH (tri-state line assuming high) in which case this will be the address which is jumped to.

The routine at this address adds 100H to whatever is in the IX register to identify the error and then jumps to the Self Test failure section of code.

7.2.2 ADDRESS 66H

This address is the one triggered when the NMI (Non Maskable Interrupt) line on the Z-80 is pulled.

Setting bit 7 in the Reset/I.D. register causes an NMI.

The code at this location disables interrupts, calls the initialization routine, reinables interrupts and returns from the NMI.

RECALL THAT : setting bit 7 in the Reset/I.D. register is called a Soft Reset.

When a Soft Reset is issued by the host, a jump is made to the initialization routine and THERE IS NO RETURN FROM THE NMI. The initialization routine jumps over the return in this circumstance.

However, to test the NMI, the Self Test also issues a Soft Reset. In this case, due to a value set in a test variable, the initialization routine returns control back to the calling routine and the RETN instruction at location 6BH is executed.

7.3 INT_COND AND INTERRUPT REGISTER TEST

The INT_COND and INTERRUPT registers are tested together because the function of INT_COND register impacts the INTERRUPT register. In other words, they are intertwined in some respects.

The INTERRUPT test is split into two parts in the firmware separated by the NMI and RESET/I.D. register test.

The reason for this is that the first part of the test writes to the INT_COND register which causes the IRQ (bit 6) bit to be set in the INTERRUPT register. Keeping in mind that we do not want to send an interrupt to the host, the only way to clear this is either a read from the host or a reset. Since the NMI test causes a Soft Reset, this bit is cleared.

Then the second part of the INTERRUPT register test is performed.

TEST OUTLINE :

Increment IX
Clear bit 7 (IEN) of the INTERRUPT register
Read INTERRUPT register
If bit 7 <> 0 then jump to Self Test error routine

Write to the INT_COND register (should set the IRQ bit in INTERRUPT reg)
Read the INTERRUPT register
Should be IEN=0 (bit 7) and IRQ=1 (bit 6). If not, jump to the Self Test error routine

NOTE: AT THIS POINT THE TEST IS SEPARATED BY THE NMI & RESET/I.D. TEST

Set bit 7 of the INTERRUPT register
Read INTERRUPT register
Should be IEN=1, IRQ=0. If not, jump to the Self Test error routine

Clear the INTERRUPT register
Read the INTERRUPT register
Should be IEN=0, IRQ=0

7.4 NMI AND RESET/I.D. TEST

The first portion of this test masks off the upper 3 bits of the Reset/I.D. register and tests the remaining 5 bits for correct card I.D. The correct card I.D. for the CARMEN card is 5.

The second portion of this code causes a Soft Reset to the Z-80 and tests whether a NMI is actually generated. A value (SVAL) is written to a test register before the NMI is executed. If the NMI executes correctly, control will be passed to the initialization routine where the value in the test register will be changed to match EVAL.

The SVAL and EVAL matching algorithm works in the following manner: if the initialization routine finds the value SVAL in the test register, it changes the test register value to EVAL and returns to the calling routine. When the NMI test has regained control, it verifies that the routine actually executed the NMI by identifying the value EVAL in the test register which was set by the initialization routine.

If the initialization routine is called and the test register does not have the value, SVAL in the test register, the initialization routine executes the rest of its routine and does not return to the caller.

TEST OUTLINE :

Increment IX
Retrieve value in RESET/I.D. register
Mask off bit 7 (bits 5 and 6 are hardwired to 0)
If lower bits <> 5 then jump to Self Test error routine

Load Test with Sval
Cause a Z-80 reset by setting bit 7 in the RESET/I.D. register
Wait for return from interrupt
Retrieve value in Test
Compare with EVAL. If different, jump to Self Test error routine
Clear Test register

7.5 SEMAPHORE REGISTER TEST

There are basically three parts to the Semaphore register test.

The first part puts the register into a known state by writing to it and testing that the write set it to zero.

The second part of the test checks the Semaphore register again to see if the read (which was performed to check the write results) set it.

The third part of the test is another write to the Semaphore register and a check to see if bit 7 went from a 1 to 0 correctly.

The Semaphore register is left set (bit 7=1) by the last read. It will be cleared in the initialization routine. The reason for this is added protection against the host attempting to send or receive an interrupt before the card has completed its Self Test and initialization routine.

NOTE : Remember that if the Semaphore register is initially reset (bit 7=0), reading it will return the value with bit 7=0, but the act of reading the Semaphore register will have set bit 7 to 1. A second read would confirm this.

TEST OUTLINE :

```
Increment IX
Write to the Semaphore register
Read the Semaphore register
If bit 7=1 then jump to Self Test error routine

Read the Semaphore register
If bit 7=0 then jump to Self Test error routine

Write to the Semaphore register
Read the Semaphore register
If bit 7=1 then jump to Self Test error routine
```

7.6 ROM TEST

This test performs a CRC using the polynomial X**16+X**2+X+1 and checks it against a previously calculated CRC already stored in the upper two bytes of ROM.

The check character is stored with the low order byte first.

7.7 RAM TEST

This performs a test of the static RAM for stuck-at-0 and stuck-at-1 faults and address decoder failures.

The test basically consists of four stages.

In the first, a pattern (55H) is written to every location in RAM.

The second pass consists of reading each location and checking the value read against the pattern written.

An alternate pattern (AAH) is then written to every location.

The final is pass is a second read of each location, checking each value read against the second pattern written.

```
TEST OUTLINE :

Increment IX
Write 01010101 to each RAM location
I=0
While (I=I+1) <= end of RAM Do
  Begin
    If RAM(I) <> 01010101 then jump to Self Test error routine
    RAM(I)=10101010
  End

I=Index of last RAM location
While (I=I-1) >= Beginning of RAM Do
  Begin
    If RAM(I) <> 10101010 then jump to Self Test error routine
    RAM(I) = 01010101
 End
Reset Stack Pointer with Stack Address
```

7.8 CTC TEST

There are basically two CTC tests, both of which are executed on all four channels of both CTC's.

The first test checks the downcounting ability of the CTC channels by setting the CTC time constant to a known value, then checking whether it downcounts correctly within a known time period.

The second test checks the timer ability and interrupt priority each of the CTC channels by setting them up in sequence and letting them downcount to zero, cause an interrupt, and jump to the test interrupt vector, altering the vector for the next interrupt expected.

WARNING : This is a very tricky test with strict timing constraints. For further explanation see below.

The two tests are performed on the two CTC's in the following order: Algorithm 1 (the first test) is performed on both CTC first, then Algorithm 2. Each test is done on one CTC at a time.

TEST OUTLINE FOR ALGORITHM 1 :

Increment IX
Reset all CTC channels
Set up interrupt vectors in RAM with the data in the ROM test interrupt vectors for the CTC - 8 locations with a jump instruction to a CTC error routine (in the first test, if the downcounter counts to zero, its an error and the CTC error routine will call the Self Test error routine).
Set the Interrupt mode to 2
Enable interrupts

NOTE : For the first part of this test, all channels must be read before 256 T states have elapsed from the time each channel is started - i.e. before the time constant register downcounts.

For each channel . . .
    Load Channel Control Word (interrupt enabled, timer mode, prescale value = 256)
    Load Time Constant Register = 0

For each channel . . .
    Read Time Constant Register
    If <> 0 then jump to Self Test error

NOTE : For the second part of this test, all channels must be read after 256 T states but before 512 T states from the time the channel is started. Remember that the time constant register was originally set to zero, so when it downcounts (after 256T states) it will be 255.

For each channel . . .
    Read Time Constant Register
    If <> 255 then jump to Self Test error

Execute this test a second time (without reseting the timing)

TEST OUTLINE FOR ALGORITHM 2 :

NOTE: Remember that Algorithm 1 set the jump-to-error-routine instructions in RAM locations and passed the RAM address to the CTC as interrupt vector addresses. In the following test, each CTC channel is set to interrupt in a controlled sequence. The sequence should be channel 2, followed by channel 0, then channel 1, finally channel 3 (interrupt priority also determines sequence). The sequence is verified by giving a different interrupt vector address to the channel which is expected to interrupt. This new vector address points to a routine which changes the interrupt vector address of the second channel which is supposed to interrupt. In other words, if the channels interrupt in the right sequence the interrupt does not jump them to the error routine originally pointed to. Also, each alternate interrupt routine sets a bit in the B register which verifies that the routines were actually executed.

Increment IX
Load all CTC channels - interrupt enabled, prescale value = 16, timer mode
Load address of second interrupt routine into RAM CTC vector location
Load Time Constant for each channel (these are carefully calculated as is the order that the channels are triggered to insure the correct interrupt sequence.
Enable interrupts
Wait (2 NOP instructions)
Disable interrupts
If B register <> 0F0H then call Self Test error routines

SECOND INTERRUPT ROUTINE EXPLANATION

Reload vector address for this channel with old error routine address
Reset channel
Set bit in B register
Load RAM vector address for next expected channel with address of the second interrupt routine for that channel
Enable interrupts
Return from interrupt

7.9 SIO TEST

This test basically performs loopbacks on the transmit and receive lines of all of the SIO ports.

The test will perform an "internal" loopback which disables the frontplane buffer ICs and sends data out the SIO and back again.

NOTE : Internal loopback is determined by the status of the RTS line on channel B in SIO 1. If RTS=1, the frontplane RS232 buffer ICs are disabled and the TX lines loop back into the RX lines.

Available bit in Read Register 0 of the SIO. A deadman timer of approx. 16 milliseconds is set initially to insure that the code does not loop forever in case there is an error.

TEST OUTLINE :

```
Reset all CTC channels and program for 19.2K baud
Set up interrupt vector for deadman timer on CTC  1, CH. 2
Program all SIO channels - 8 bit, 1 stop bit, no parity

Internal loopback
   Start deadman timer
   Enable interrupts
   For I=1 to 8 do
     Wait until TX buffer empty
     If RX Character Available bit set (Read Reg 0)      * because of the
     then empty the FIFO                                 * three-char
          If RX Character Available bit set (Read Reg 0) * input buffer
          then empty the FIFO                            * of SIOs
               If RX Character Available bit set (Read Reg 0)
               then jump to Self Test error
     Send 0AAH on TX line
     Wait until RX Character Available bit set
     Read in character
     If <> character sent go to Self Test error routine
     Complement test character (now 055H)
   Stop deadman timer
```

NOTE: The subroutine used to loop back data is SIO-TEST.

7.10 SELF TEST END

7.10.1 UPON SUCCESSFUL COMPLETION

* Self test will put the PASS variable (value E0H) into the ST_COND register.

* After saving the PASS variable in the ST_COND register, a self test done interrupt is sent to the host.

* Then the self test executes the initialization routine.

7.10.2 UPON UNSUCCESSFUL COMPLETION

* Self test will put the value of the IX register into the ST_COND register. The value in the IX register indicates where the self test failed (see the IX values below for their interpretation).

* After saving the IX register in the ST_COND register, a self test done interrupt is sent to the host.

* Then the self test executes (or attempts to execute) the initialization routine.

7.10.3 SELF TEST RESULTS

IN OTHER WORDS, the card is left in basically the same state upon self test failure that it is upon a successful completion of self test.

When booting, the system console will display a message identifying the card by :

* ID number
* select code

When a failure happens, in addition there will be :

* the word "failed"
* a number which indicates the type of failure (the value of ST_COND register).

VALUE OF IX/ST-COND REGISTER UPON SELF TEST FAILURE :

```
IX = 1    : INT_COND and INTERRUPT registers
IX = 2    : NMI and RESET / ID register
IX = 3    : SEMAPHORE register
IX = 4    : ROM
IX = 5    : RAM
IX = 6    : CTC 0 - algorithm 1
IX = 7    : CTC 0 - algorithm 2
IX = 8    : CTC 1 - algorithm 1
IX = 9    : CTC 1 - algorithm 2
IX = 10   : internal loopback on port 0 (SIO 0 channel A)
IX = 11   : internal loopback on port 1 (SIO 0 channel B)
IX = 12   : internal loopback on port 2 (SIO 1 channel A)
IX = 13   : internal loopback on port 3 (SIO 1 channel B)
IX = 1xx  : jumped outside of address space
```

CHAPTER 8
DATA STRUCTURE IMPLEMENTATION

This chapter defines all the symbols which are not used as a label or subprogram name. All equates and variables used in the firmware are contained in two files: MX-VA and MX4EQ.

The labels defined in MX-VA are all of the variables used in the firmware. They will be defined in two section; those that are accessed by both the card and the host and those that are only accessed by the card.

The labels defined in MX4EQ are equates used throughout the firmware. This file is copied to almost every other file. The labels defined in MX4EQ are cross referenced by the files which use each in the individual file descriptions.

This chapter will merely give a description of the usage of each without specifying which firmware module uses them.

8.1 SHARED VARIABLES (by host and card) IN MX-VA

```
BD-0 : This contains the baud rate value for port 0.
BD-1 : This contains the baud rate value for port 1.
BD-2 : This contains the baud rate value for port 2.
BD-3 : This contains the baud rate value for port 3.

BIT-MAP : This defines the starting address for the Bit Map table

CMND-TAB : This defines the starting address of the COM-REG
           register port specific interrupt table.

CONFG-0 :  Contains the current configuration data code for port 0
CONFG-1 :  Contains the current configuration data code for port 1
CONFG-2 :  Contains the current configuration data code for port 2
CONFG-3 :  Contains the current configuration data code for port 3

ICR-TAB : This defines the starting address of the INT-COND
          register port specific interrupt table.

MINT-REG : contains the information designating on which port(s), output
           modem lines have to be changed

MODM-IN-0 : Contains current status of the input modem lines for port 0
MODM-IN-1 : Contains current status of the input modem lines for port 1
MODM-IN-2 : Contains current status of the input modem lines for port 2
MODM-IN-3 : Contains current status of the input modem lines for port 3

MODM-MASK-0 : Contains the information designating which modem input
              lines the host wants to be notified of in the event
              of a change, for port 0
MODM-MASK-1 : ... for port 1
MODM-MASK-2 : ... for port 2
MODM-MASK-3 : ... for port 3

MODM-OUT-0 : Contains current status of the output modem lines for port 0
MODM-OUT-1 : Contains current status of the output modem lines for port 1
MODM-OUT-2 : Contains current status of the output modem lines for port 2
MODM-OUT-3 : Contains current status of the output modem lines for port 3

MSTAT-REG : contains the information designating on which port(s), input
            modem lines have changed (according to MODM-MASK-i)

RHEAD-0 : The head pointer index for the Receive FIFO for port 0
RHEAD-1 : The head pointer index for the Receive FIFO for port 1
RHEAD-2 : The head pointer index for the Receive FIFO for port 2
RHEAD-3 : The head pointer index for the Receive FIFO for port 3

RTAIL-0 : The tail pointer index for the Receive FIFO for port 0
RTAIL-1 : The tail pointer index for the Receive FIFO for port 1
RTAIL-2 : The tail pointer index for the Receive FIFO for port 2
RTAIL-3 : The tail pointer index for the Receive FIFO for port 3
```

ST-COND : Contains the result of Self Test

THEAD-0 : The head pointer index for the Transmit FIFO for port 0
THEAD-1 : The head pointer index for the Transmit FIFO for port 1
THEAD-2 : The head pointer index for the Transmit FIFO for port 2
THEAD-3 : The head pointer index for the Transmit FIFO for port 3

TTAIL-0 : The tail pointer index for the Transmit FIFO for port 0
TTAIL-1 : The tail pointer index for the Transmit FIFO for port 1
TTAIL-2 : The tail pointer index for the Transmit FIFO for port 2
TTAIL-3 : The tail pointer index for the Transmit FIFO for port 3

8.2 UNSHARED VARIABLES (by card only) IN MX-VA

BITS-0 : Contains mask to strip off parity bits on RX chars for port 0
BITS-1 : Contains mask to strip off parity bits on RX chars for port 1
BITS-2 : Contains mask to strip off parity bits on RX chars for port 2
BITS-3 : Contains mask to strip off parity bits on RX chars for port 3

C-MINT-REG : copy of MINT-REG

C-MSTAT-REG : work area for MSTAT-REG

MSTAT-FLAG : used to indicate which port is scanned

PIO-WM : work area for PIO-i-xx

PORT : store area for IX register and then work area designating one port

RBRK-0 : This is the end-of-break-detected flag for port 0
RBRK-1 : This is the end-of-break-detected flag for port 1
RBRK-2 : This is the end-of-break-detected flag for port 2
RBRK-3 : This is the end-of-break-detected flag for port 3

STAT-0 : Contains the bit pattern for the status register - port 0
STAT-1 : Contains the bit pattern for the status register - port 1
STAT-2 : Contains the bit pattern for the status register - port 2
STAT-3 : Contains the bit pattern for the status register - port 3

TEST : This is a general purpose location used in Self Test

TMPTAB : The starting addr. of the temporary table for CMND-TAB data

TMRFLG : The flag which indicates whether the timer is off or on

TON0 : Transmitter on/off flag for port 0
TON1 : Transmitter on/off flag for port 1
TON2 : Transmitter on/off flag for port 2
TON3 : Transmitter on/off flag for port 3

WR3-0 : Contains the current value in SIO write register 3 for port 0
WR4-0 : Contains the current value in SIO write register 4 for port 0
WR5-0 : Contains the current value in SIO write register 5 for port 0

WR3-1 : Contains the current value in SIO write register 3 for port 1
WR4-1 : Contains the current value in SIO write register 4 for port 1
WR5-1 : Contains the current value in SIO write register 5 for port 1

WR3-2 : Contains the current value in SIO write register 3 for port 2
WR4-2 : Contains the current value in SIO write register 4 for port 2
WR5-2 : Contains the current value in SIO write register 5 for port 2

WR3-3 : Contains the current value in SIO write register 3 for port 3
WR4-3 : Contains the current value in SIO write register 4 for port 3
WR5-3 : Contains the current value in SIO write register 5 for port 3

8.3 EQUATES IN MX4EQ

BEG-BD : Initial value for BD registers - 9600 baud

BEG-CONFG : Initial value for CONFG registers

BREAK : Contains bit position value for the status byte break bit

CTC-0-C0 : CTC 0 Channel 0 address
CTC-0-C1 : CTC 0 Channel 1 address
CTC-0-C2 : CTC 0 Channel 2 address
CTC-0-C3 : CTC 0 Channel 3 address

CTC-1-C0 : CTC 1 Channel 0 address
CTC-1-C1 : CTC 1 Channel 1 address
CTC-1-C2 : CTC 1 Channel 2 address
CTC-1-C3 : CTC 1 Channel 3 address

COM-REG : Address of commamd register

CTC-V0 : Beginning CTC 0 vector in RAM for Self Test CTC tests

CTC-V1 : Beginning CTC 1 vector in RAM for Self Test CTC tests

CTCWRD : CTC Channel Control word value for 16 millsec. timer

ERR-MSK : Mask used to isolate status byte bits in RX Error ISR

EVAL : Test value in Self Test - NMI test

FRAME : Contains bit position value for Framing error in Status byte

IC-BIT : Bit position in MODM-MASK and MODM-IN reg. for IC bit

INT-CODE : INT-COND register value of Self Test Done interrupt

INT-COND : Address of INT-COND register

INT-REG : Address of Hardware status register - INT-REG

MOD-INT : INT-COND bit for Input Modem Line Change interrupt

OVRRUN : Contains bit position value for Overrun error in Status byte

PARITY : Contains bit position value for Parity error in Status byte

PASS : Value of ST-COND register when Self Test passes

```
PIO-0-AD : PIO  0 Channel A data address
PIO-0-AC : PIO  0 Channel A control address
PIO-0-BD : PIO  0 Channel B data address
PIO-0-BC : PIO  0 Channel B control address

PIO-1-AD : PIO  1 Channel A data address
PIO-1-AC : PIO  1 Channel A control address
PIO-1-BD : PIO  1 Channel B data address
PIO-1-BC : PIO  1 Channel B control address

PORT0 : Bit position for port specific int. in INT-COND reg. - port 0
PORT1 : Bit position for port specific int. in INT-COND reg. - port 1
PORT2 : Bit position for port specific int. in INT-COND reg. - port 2
PORT3 : Bit position for port specific int. in INT-COND reg. - port 3

RAM-BEG : Address of beginning of RAM

RAM-SEG : Number of 256 byte segments in RAM - used in Self Test

RAM-SIZ : Number of bytes in RAM

RESET : Address of RESET/ID register

ROM-BEG : Address of beginning of ROM

ROM-END : Address of last byte of ROM

ROM-SEG : Number of 4K segments of ROM - used in Self Test

RX-BASE0 : High byte of RX FIFO tail pointer indexes - port 0
RX-BASE1 : High byte of RX FIFO tail pointer indexes - port 1
RX-BASE2 : High byte of RX FIFO tail pointer indexes - port 2
RX-BASE3 : High byte of RX FIFO tail pointer indexes - port 3

SEM-REG : Address of Semaphore register

SIO-0-AD : SIO  0 Channel A data address
SIO-0-AC : SIO  0 Channel A control address
SIO-0-BD : SIO  0 Channel B data address
SIO-0-BC : SIO  0 Channel B control address

SIO-1-AD : SIO  1 Channel A data address
SIO-1-AC : SIO  1 Channel A control address
SIO-1-BD : SIO  1 Channel B data address
SIO-1-BC : SIO  1 Channel B control address

SPEC-ICR : ICR-TAB bit position value for Special Character interrupt

ST-COND : Address of ST-COND register (this is also defined in &MX-VAR
          Its a case of overkill but was done before this was written)

ST-INT : INT-COND bit position value for Self Test interrupt
```

STK-ADDR : stack address

SVAL : Test value used in Self Test - NMI test

TEST : Address of general purpose test location (this is also defined
in &MX-VAR as is ST-COND - another case of overkill)

TFIFO-0 : Low byte base for TX head pointer index for port 0
TFIFO-1 : Low byte base for TX head pointer index for port 1
TFIFO-2 : Low byte base for TX head pointer index for port 2
TFIFO-3 : Low byte base for TX head pointer index for port 3

TME-INT : INT-COND bit position value for Time-Out Timer interrupt

TMSK : Mask to isolate the low nibble of the TX head pointer

TMRPRE : CTC prescale value for the 16 millisecond timer

TX-BASE : Contains the high byte value for all of the TX head pointers

TX-ICR : ICR-TAB bit for Tx buffer empty interrupt

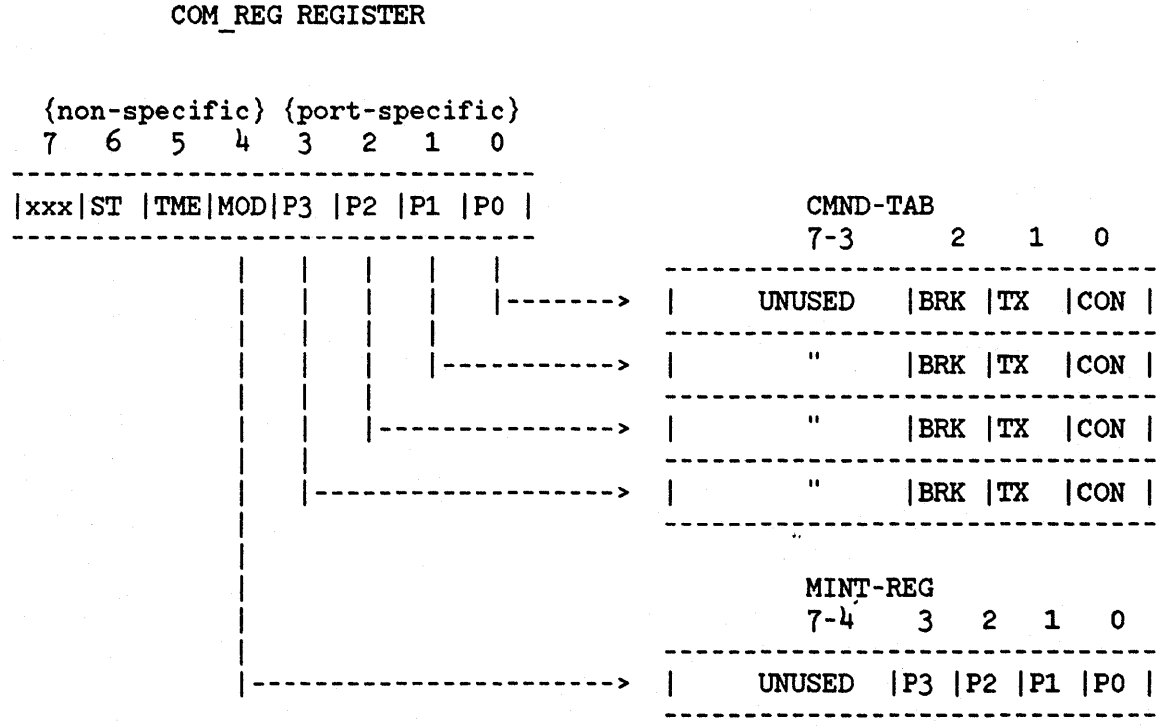
VEC : The beginning address of the interrupt vectors in ROM

CHAPTER 9
PRODUCT EVOLUTION

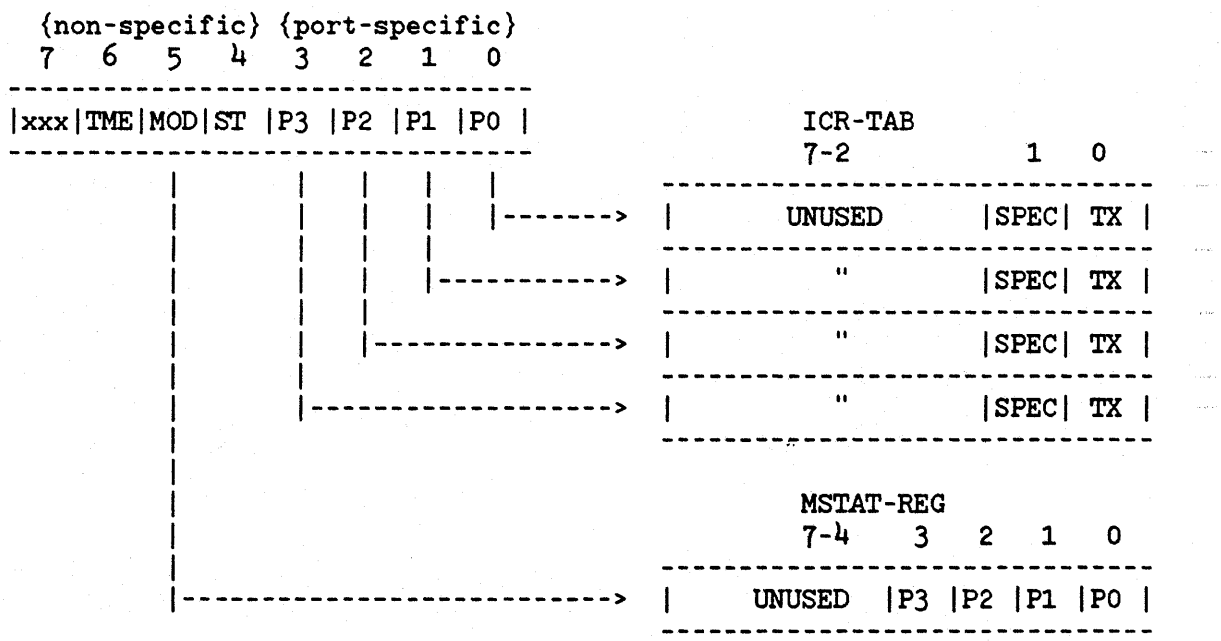
9.1 CARMEN

To implement the changes for the modem lines, a choice has been made. I will first explain this solution. Then I will list the second choice which fits better the FORDYCE firmware structure.

a) When a change occurs on a modem line, it has been choosen to keep the FORDYCE manner to indicate that change to the other part (host or card).



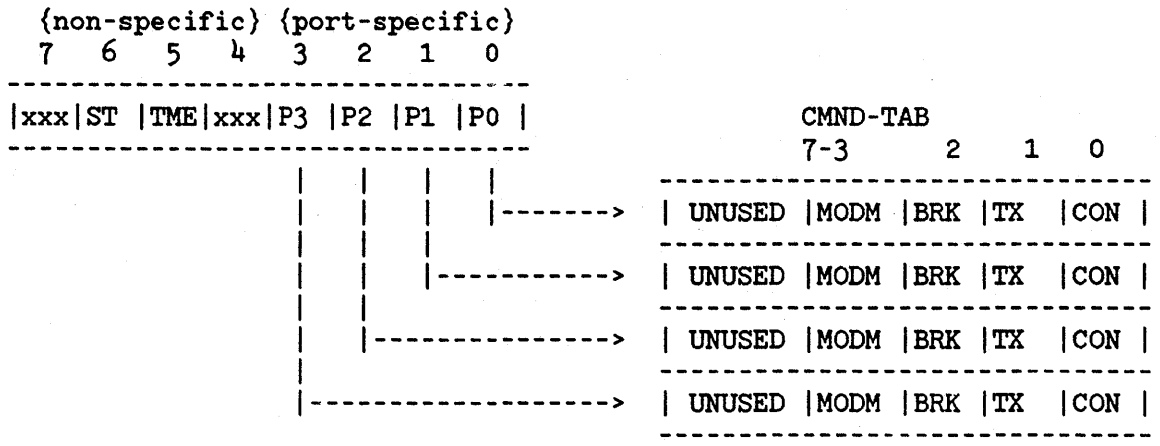
INT-COND REGISTER



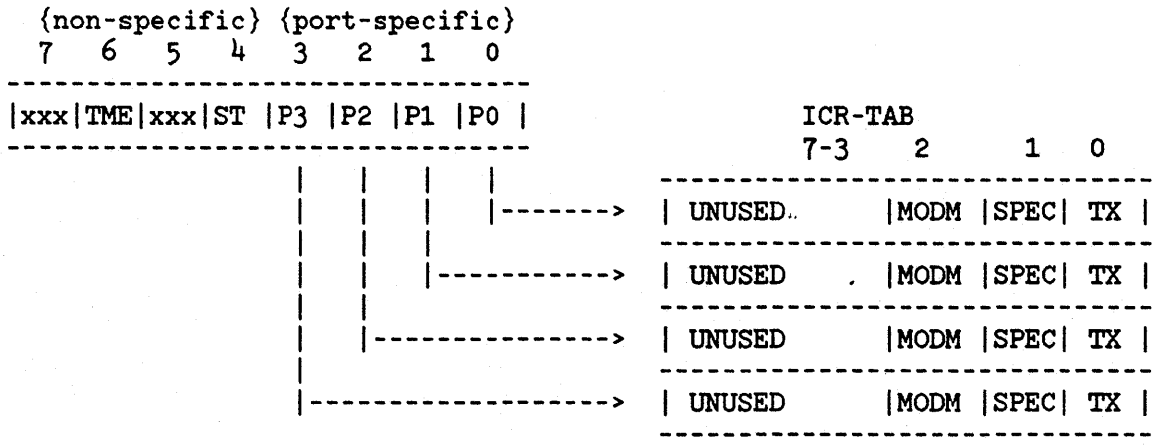
So MODM bit in COM-REG and INT-COND registers is set to "1". To know on which port the change occured, the MINT-REG and MSTAT-REG have to be read.

b) An other solution could have used the FORDYCE structure for the port-specific interrupts and the unused bits of CMND-TAB and ICR-TAB.

COM_REG REGISTER



INT-COND REGISTER



So if a change occured on a modem line for a particular port, the corresponding bit (from 0 to 3) should be set to "1" in the CMND-TAB or ICR-TAB to indicate on which port it occured. Then an unused bit of CMND-TAB or ICR-TAB should be used as a MODM bit.

9.2 FORDYCE

Here are some remarks about the FORDYCE firmware. If a PCO was decided for an other reason, it would be good to make these changes too.

In CARMEN firmware these changes are already done (for the lines of code kept from FORDYCE to CARMEN firmware of course).

NOTE : The line numbers refer to the assembled source of FORDYCE.

a) Changes to be done in FORDYCE firmware (and already done in CARMEN firmware because these lines of code were kept) :

* NMI and RESET/I.D. register :

  Before line 345 an instruction should be included.
  It should look like :

```
XOR  A
LD   [INT-REG],A
LD   A,[INT-REG]
AND  0C0H
CP   0
JP   NZ,ST-ERR
```

  So, when the AND instruction is executed, the logical operation is done between the value C0 and the A register which has been previously updated with the INT-REG value (without the new line, the A register was not updated with the INT-REG value, and was always equal to zero for the execution of the AND instruction).

* Subroutine SIO-TEST :

  The deadman timer should be stopped at the end of this subroutine. Two lines should be added before line 1311.
  It should look like :

```
DI
LD  A,33H
OUT [CTC-1-C2],A
RET
```

* Set the BAUD RATE GENERATION CLOCKS in CTCs :
  (in asynchronous SIO test)

  The four channels of each CTC are all initialized with the baud rate generation clock. But only the two first channels are used as baud rate generation clock. To prevent a design flaw of CTCs, the two other initializations have to be removed. So line 928 (before label MSIO-20) and line 938 (before label MSIO-30) should be changed in the following manner :

```
LD  B,2
```

* SIO and CTC addresses :

These addresses have 8 bits : B7...B0.

Bits 7 to 4 are used in the following manner :
0111 ==> SIO-0
1011 ==> SIO-1
1101 ==> CTC-0
1110 ==> CTC-1
So a "0" in a bit selects a particular circuit.

Bits 1 and 0 are used to select the channel within a particular circuit (there are two different channels for a circuit).

Bits 3 and 2 are unused : they are set to "0" but it would be better if they were set to "1" because a "0" means a selection...

* SIO initialization :

a) The ZILOG application note called "using the Z80 SIO in asynchronous communications", recommends to give the Reset External/Status Interrupt command each time WR0 is changed to point to another register...
This means that each time WR0 is changed, bit 4 should be set to "1".

b) Each SIO channel has a three-character input buffer. During power up, this buffer may get full of garbage. So it would be good to empty it during the self-test routine.

FOR MORE INFORMATION, see paragraph 7.1 SELF TEST INITIALIZATION.

* RECISR macro :

When a special character is received, the interrupt is first sent to the host and then the character is put in the FIFO. It would be better to put the character in the FIFO before sending the interrupt to the host.

b) Changes to be done in FORDYCE firmware (but not done in CARMEN firmware because these lines of code were not kept) :

* Subroutine LOOP-TEST :

The deadman timer is useless in this subroutine. Lines 1213 thru 1217 and line 1247 should be removed. These lines are the following ones :

```
LD  A,0B7H
OUT [CTC-1-C2],A
XOR A
OUT [CTC-1-C2],A
EI

DI
```

* About SIO reconfiguration for 8 bit TX and TX enable :

Before label PORT2, lines 1147 thru 1150 are useless because SIO 0 channel A had been configurated and never used inbetween. These four lines should be removed. They look like :

```
LD  A,5
OUT [SIO-0-AC],A
LD  A,68H
OUT [SIO-0-AC],A
```

9.3 FORDYCE TO CARMEN

a) ALL the FORDYCE files have been modified to convert the FORDYCE firmware in an HP9000 format.

NOTE : this HP9000 version of FORDYCE firmware is available.

b) Some FORDYCE files have been modified to become CARMEN files. There are the following ones :

- MX4ST,
- MX4IN,
- MXHST,
- MXEXT,
- MXMOD,
- MX-VA,
- MX4EQ.

c) A file has been added for the CARMEN firmware :

- MXMDM.

Table of contents

CHAPTER 1
PRODUCT IDENTIFICATION AND OVERVIEW 6

CHAPTER 2
REFERENCE 8

2.1 RELEVANT DOCUMENTS 8
2.2 GLOSSARY 8

CHAPTER 3
DESIGN OVERVIEW 9

3.1 DESIGN APPROACH 9
3.2 OVERVIEW OF OPERATION 9
3.2.1 SYSTEM POWER UP.....9
3.2.2 SOFT RESET.....10
3.2.3 Z80 INTERRUPTS.....10
3.2.3.1 INTERRUPTS COMING FROM THE SIOs.....10
3.2.3.2 INTERRUPTS COMING FROM THE CTCs.....11
3.3 DESIGN CONVENTIONS & STANDARDS 11

CHAPTER 4
HARDWARE CONSIDERATIONS 12

4.1 Dual Inline Package (DIP) SWITCHES 12
4.2 CTCs 12
4.3 SIOs 13
4.4 PIOs 13
4.5 FIRMWARE PRIORITY SCHEME 13

CHAPTER 5
DEFAULT SETTINGS 14

5.1 DEFAULT Dual Inline Package (DIP) SWITCHES 14
5.2 DEFAULT LINE CHARACTERISTICS AND FORMAT 14
5.3 DEFAULT BIT MAP 14
5.4 DEFAULT TIMERS SETTING 14

CHAPTER 6
MODULE INTERFACE SPECIFICATIONS 15

6.1 FILES LIST 15
6.2 TOP-DOWN DIAGRAM 16
6.2.1 WITHOUT INTERRUPT.....16
6.2.2 WITHIN INTERRUPT.....16
6.2.3 VARIABLES AND EQUATES.....17
6.3 ROM MAP 18
6.4 DETAILED DESCRIPTION OF FIRMWARE MODULES 19

6.4.1 MX4ST - SELF TEST....19
6.4.2 MX4IN - INITIALIZATION ROUTINE....20
6.4.3 MX4RX - RECEIVE ISR's....21
6.4.4 RXERR - RECEIVE ERROR ISR's....23
6.4.5 MX4TX - TRANSMIT ISR's....24
6.4.6 MXTMR - 16 MILLSEC. TIMER INTERRUPT....25
6.4.7 MXHST - HOST ISR....26
6.4.8 MXEXT - EXTERNAL STATUS ISR's....27
6.4.9 MXPT0 - MXPT1 - MXPT2 - MXPT3 - PORT SPECIFIC ISR's....29
6.4.10 MXSBR - SUBROUTINES FOR PORT SPECIFIC ISR's....32
6.4.11 MXMOD - MODEM OUTPUT LINE CHANGE ROUTINE....34
6.4.12 EXTMR - TIMER ON/OFF ROUTINE....35
6.4.13 MXMDM - MODEM INPUT LINE CHANGE ROUTINE....36
6.4.14 MX-VAR - VARIABLES....37
6.4.15 MX4EQ - EQUATES....38
6.5 COMMON ALGORITHMS....39
6.5.1 RECEIVE BUFFER, EMPTY/FULL DECISION....39
6.5.2 RECEIVE HEAD & TAIL POINTER HANDLING....39
6.5.3 BIT MAP CHECK....39
6.5.4 STRIPPING PARITY BITS....39
6.5.5 SENDING AN INTERRUPT TO THE HOST - USE OF SEMAPHORE REGISTER....40
6.5.6 STATUS BYTE....40
6.5.7 TRANSMIT BUFFER, EMPTY/FULL DECISION....40
6.5.8 TRANSMIT HEAD & TAIL POINTER HANDLING....40
6.5.9 REASON FOR CYCLING IN HSTINT ROUTINE....41
6.5.10 DECIPHERING THE TYPE OF INTERRUPT....41
6.5.11 CHANGING THE SIO WRITE REGISTERS TO NEW CONFIGURATION....41
6.5.12 CHANGING THE BAUD RATE....41
6.5.13 MUTUAL MANAGEMENT FOR MODEM LINES....42

CHAPTER 7
SELF TEST....43

7.1 SELF TEST INITIALIZATION....43
7.2 RESERVED ADDRESSES....45
7.2.1 ADDRESS 38H....45
7.2.2 ADDRESS 66H....45
7.3 INT_COND AND INTERRUPT REGISTER TEST....46
7.4 NMI AND RESET/I.D. TEST....47
7.5 SEMAPHORE REGISTER TEST....48
7.6 ROM TEST....49
7.7 RAM TEST....49
7.8 CTC TEST....50
7.9 SIO TEST....52
7.10 SELF TEST END....53
7.10.1 UPON SUCCESSFUL COMPLETION....53
7.10.2 UPON UNSUCCESSFUL COMPLETION....53
7.10.3 SELF TEST RESULTS....53

CHAPTER 8
DATA STRUCTURE IMPLEMENTATION....55

8.1 SHARED VARIABLES (by host and card) IN MX-VA....56
8.2 UNSHARED VARIABLES (by card only) IN MX-VA....58
8.3 EQUATES IN MX4EQ....59

CHAPTER 9
PRODUCT EVOLUTION 62

9.1 CARMEN 62
9.2 FORDYCE 65
9.3 FORDYCE TO CARMEN 67