ROYAL SIGNALS & RADAR ESTABLISHMENT

THE AXIS TEST BOX SOFTWARE REPORT

Author: A L Simcock

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
RSRE MALVERN,
WORCS.

UNLIMITED 83 10 05 159
ROYAL SIGNALS AND RADAR ESTABLISHMENT

Memorandum 3585

Title: THE AXIS TEST BOX SOFTWARE REPORT
Author: A L Simcock
Date: April 1983

SUMMARY

This memorandum describes in detail the software program which the M6809 microprocessor obeys to perform the functions of the AXIS Test Box.

Extensive use is made of Flow Diagrams where these either enhance or replace wordy descriptions.

This memorandum is for advance information. It is not necessarily to be regarded as a final or official statement by Procurement Executive, Ministry of Defence.

Copyright C
Controller HMSO London
1983
RSRE MEMORANDUM No 3585

THE AXIS TEST BOX SOFTWARE REPORT

A L Simcock

CONTENTS
1 INTRODUCTION
2 INITIALISATION
3 BASIC COMMAND MODE
4 ASSEMBLE MESSAGE
5 DISPLAY ON REQUEST
6 AUTO DISPLAY
7 REPEAT
8 DISPLAY CURRENT MENU
9 PREPARE DATA FOR TRANSMISSION
10 DATA MANIPULATION SUBROUTINES
11 BASIC INPUT SUBROUTINES
12 BASIC OUTPUT SUBROUTINES
13 DATA ENTRY SUBROUTINES
14 DISPLAY ON REQUEST SUBROUTINES
15 THE HARDWARE INTERFACES
16 HARDWARE HANDLERS
17 THE DATA STRUCTURES
18 TRANSMIT DATA FLOW
19 RECEIVE DATA FLOW

ANNEXES
A INTERNAL IDENTIFIER CROSS REFERENCE
B DRAWING SYMBOLS USED
C CROSS REFERENCE BY INTERNAL LABEL NAME

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification

By
Distribution/
Availability Codes

Dist | Special

A
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Diagram Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Basic Command Mode Routine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>2</td>
<td>Assemble Message Routine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>3</td>
<td>Display on Request Routine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>4</td>
<td>Auto Display Routine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>5</td>
<td>Repeat Routine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>6</td>
<td>Menu Display Routine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>7</td>
<td>Prepare Data for Transmission</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>8</td>
<td>Prepare for Hardware Handlers</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>9</td>
<td>Basic Input/Output Subroutines</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Input Message Type Subroutine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>11</td>
<td>Input Check Code Subroutine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>12</td>
<td>Input Register Address Subroutine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>13</td>
<td>Input Data Subroutines</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>14</td>
<td>Get Parity Subroutine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>15</td>
<td>Get Format/Structure Subroutine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>16</td>
<td>Get Transmission Option Subroutine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>17</td>
<td>Get Wait Conditions Subroutine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>18</td>
<td>Get Display Option Subroutine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>19</td>
<td>Input Segmentation Subroutine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>20</td>
<td>Print Block Subroutine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>21</td>
<td>Print Message Subroutine</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>22</td>
<td>PPI Port and Bit Allocations</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Polling Routines</td>
<td>Flow Diagram</td>
</tr>
<tr>
<td>24</td>
<td>Interrogation Routines</td>
<td>Flow Diagram</td>
</tr>
</tbody>
</table>
25 Internal Message Structure
26 TX Message Structure
27 Display Structure
28 Segmentation for Display Structure
29 Transmit Data Flow
30 Receive Data Flow

LIST OF TABLES
1 PPI 1 Notations
2 PPI 2 Notations
3 PPI 3 Notations
4 PPI 4 Notations

LIST OF ABBREVIATIONS
MMI Man Machine Interface
PPI Programmable Peripheral Interface
VDU Visual Display Unit
GTU Group Terminating Unit
o/p Output
i/p Input
Ø The symbol Ø is used to distinguish zero from the letter 0.
HEX Hexadecimal
TX Transmit
RX Receive
EOT End of Text
CC Check Code
RA Register Address
1 INTRODUCTION

This is the third in a series of 4 reports on the AXIS Test Box. The first report (Ref 1) is the Introduction to the Test Box and describes the role of the Test Box within the Axis experimental project. The second report (Ref 2) is the Operating Guide, and details the rationale for the nature of some of the commands/functions provided. The third report (Ref 3) is the Hardware report.

This software report describes briefly the program which controls the hardware and provides the MMI described in Ref 2.

This report is brief since it is felt that the commented program listing is almost self explanatory. The report mainly details program flow and structure and uses diagrams, wherever possible, so that program construction may easily be followed. Data and data flow are also described in Sections 17 to 19. Annex B is a list of the drawing symbols used.

The program listing is held in a file called FINALTB on floppy disc for the FUTUREDATA Microprocessor Development System in use in T24 (Ref 4).

It can be seen from the diagrams illustrating program flow that the body of each of the major routines consists mainly of a series of subroutine calls.

The body of each of the major routine will be described very briefly using the illustrations as a guide to the subroutines being used.

The subroutines themselves will be described in greater detail, since this is where the majority of the work of the program is carried out. The preparation of data for transmission, and the hardware handlers are also described separately.

The report is written in this way in order that the 'casual' reader should be able to obtain a superficial understanding of the program and the 'determined' reader should find the detail necessary for a full understanding. Ref 2 describes fully the functions provided by the Test Box, this report describes how those functions are provided.

2 INITIALISATION

The initialisation routine sets the Programmable Peripheral Interface devices (PPIs) to their initial state and programs them for input or output; some internal flags are also initialised.

3 BASIC COMMAND MODE

The Basic Command Mode routine allows the user to select any one of the 5 functional routines (see Fig 1).

4 ASSEMBLE MESSAGE

The Assemble Message routine takes the user step by step through the input of the parameters necessary to assemble a message ready for transmission (see Fig 2). Individual parameter entry is controlled by subroutines and the Assemble Message routine calls these subroutines in the correct order to enable a complete message to be assembled.
5 DISPLAY ON REQUEST

The Display on Request routine allows the user to display input messages which have been stored. The user should enter the segmentation format required for the display, then the user should enter the number of the block to be displayed. Fig 3 illustrates the flow of the program.

6 AUTO DISPLAY

The Automatic Input Display routine is very simple. The user is allowed to select the display options required for display of the input messages and is then able to jump to the hardware handlers to receive the input messages (see Fig 4).

7 REPEAT

The Repeat routine firstly causes the current Menu to be displayed on the VDU and then allows the user to select parts of the message to change, the user also has the option to display the Menu, Quit the routine or Go to the hardware handlers (see Fig 5).

8 DISPLAY CURRENT MENU

This routine is itself written as a subroutine which may be called from the Repeat routine. The program flow is illustrated in Fig 6.

9 PREPARE DATA FOR TRANSMISSION

The entry to this routine is either from the Assemble Message or Repeat routines.

Firstly, the hardware interfaces (PPIs) are set to the transmit mode. The Message Type is then checked. If the Type indicates an A message or T (for both Messages) then a pointer to the A message is passed to the FMFORP (Format for Parity) subroutine. If the Type is B then the pointer is set to the B message before calling the FMFORP subroutine. The Format of the message is then determined in order that the correct number of parity bits may be generated. The CALCP (Calculate Parity) subroutine is then called:–

a Once for a GTU message
b Twice for a MATRIX Command message
c Three times for a MATRIX Connection message

When the necessary parity bits have been generated the whole message with its newly generated parity bit(s) is transformed by the TFTOTX (Transfor to Transmit Format) subroutine into a format directly applicable to loading into the hardware output registers.

The type is checked again and if TYPE = T (for both messages) the whole procedure is repeated using a pointer to the B message. Figure 7 illustrates this flow in diagrammatical form.

9.1 Prepare for Hardware Handlers

This is a continuation of the above routine but is also the entry point from the Auto Receive Mode. Figure 8 illustrates the program flow in diagrammatical form.
Firstly the last valid input stores (for A and B message responses) are preset and then the start address of the input message store is initialised. At this point if this routine was entered from the Auto Receive Mode the Programmable Peripheral Interface (PPI) devices, which interface directly to the hardware input and output boards, are set to receive inputs and the program control transfers to the hardware handling routines. If entry was not via Auto Receive the output message(s) must be loaded into the hardware output register (S).

If the message Type is A or B then O/P register 1 is loaded with the A or B message and if more than one message is to be sent (ie Transmit Options Continuous or Multiple have been chosen) then O/P Register 2 is also loaded with the A or B message. If the Type is T (for both messages Together) then O/P register 1 is loaded with the A message and O/P register 2 is loaded with the B message.

The time interval between O/P transfers, from the hardware output transfer board, is determined either by a fixed hardware delay or by software. The hardware delay will only be activated if the input message response to each output is not to be interrogated, ie delay is only activated if:-

a) No Wait conditions are set
b) Display Option chosen is NONE

under these circumstances the number of output transfers is loaded into the hardware counters and program control transfers to the hardware handlers.

If the above conditions are not met a single output will be sent and the input response will be interrogated by the hardware handlers, corresponding action will be taken, and control returned to this routine in order that the next output may be sent. In this way the time interval between output transfers is determined by the software.

If the Transmission Option selected is Multiple the number of outputs required is copied from MOPTNUM into OPTXNUM and this decremented at each output in order to call the hardware handlers to send the correct number of output transfers.

DATA MANIPULATION SUBROUTINES

10.1 FMFORP ie Format for Parity Generation

This generation takes the data identified as Message 1 or Message 2 in the internal message structure (Fig 25) and re-organises the data in order into a 42 byte block of data, so that it is in a suitable format for parity generation. (Fig 26A).

This routine should be called as a subroutine and given the address of the check code of the message (ie 1 or 2) for transformation. This address parameter should be stored in TEMP.
10.2 CALCP ie Calculate Parity

Three types of message each requiring different parity calculations have been identified in the User guide.

These are:

- G = GTU
- M1 = Matrix Command
- M2 = Matrix Connection

The identification of the type of message is stored in AFORMAT for the A Message and BFORMAT for the B Message.

- Ø = GTU
- 1 = M1
- 2 = M2

The parity generating routine requires 3 parameters:

- a Start bit number for parity generation in STAP
- b End bit number for parity generation in ENDP
- c Bit number to store the generated parity in PUTP

The routine should be called as a subroutine with the above 3 parameters defined. Setting these parameters allows the CALCP routine to be used to calculate parity bits over any data area within a message. (See also Fig 26 A).

10.3 TFTOTX ie Transform to Transmit Format

This routine takes the information from the 42 byte data area used for parity generation and transforms this into a format which will make it immediately usable by the Tx hardware handlers, ie the format illustrated in the Tx message structure, Fig 26B.

This routine should be called as a subroutine and requires the address of the first byte of the first or second (whichever is required) Tx Message Structure (see Fig 26B). The address parameter should be stored in TEMP 2 before the routine is called.

10.4 LOAD HARDWARE REGISTERS

There are 5 subroutines under this heading:

- OPAMS1 Loads A message into O/P Register 1
- OPAMS2 Loads B message into O/P Register 2
OPBMS1 Loads B message into O/P Register 1
OPPMS2 Loads B message into O/P Register 2
PUTBYOU Generates the parallel load pulse and selects the shift register to be loaded.

The A or B message stored in the Transmit Message (Fig 26B) is loaded byte by byte into successive 8 bit shift registers which form the O/P registers. Six load sequences are necessary to complete the loading of each O/P register.

11 BASIC INPUT SUBROUTINES (See also Fig 9)

11.1 GETIP Get Input

This subroutine gets characters from the VDU, checks the character is not a back space and counts the number of input characters. It also checks that the number of input characters does not exceed the maximum required by the calling routine.

The calling routine should set the maximum number of characters required in the variable called MAXNUM and call GETIP as a subroutine.

The GETIP subroutine will return to the calling routine upon the entry of a Carriage Return on the VDU. The actual number of characters entered will be in the variable COUNT and the characters will be stored on the U stack. The PULU A instruction used by the calling routine will extract input character from the stack and put it in the A register. The first character available at the top of the stack is the first character entered on the VDU.

eg If the User entered A B C D, these characters would be stored on the U stack and successive PULU A instructions would retrieve A, B, C and D respectively.

11.2 GETHEXIP Get Hex Input

This subroutine gets Hex input from the VDU. It uses the GETIP subroutine previously defined. The GETHEXIP subroutine checks the validity of the input characters (to ensure hex input) and also counts the number of input characters and checks also that the number of input characters does not exceed the maximum required by the calling routine.

The calling routine should set the maximum number of characters required in the variable MAXNUM and call GETHEXIP as a subroutine.

The GETHEXIP subroutine will transform the input characters into HEX bytes, ie 2 input characters constitute one HEX byte. GETHEXIP assumes leading zeroes, ie if the calling routine set a maximum number of input characters as 4 requiring an input between 0000 and FFFF, and the User entered '1234' carry age return the GETHEXIP subroutine would transform this into 0123 0400.

This function as defined as NORMALISING the hex input since a normal hex byte contains 2 input characters.
Normalising is only carried out in the event of an odd number of characters being entered on the VDU,

ie if the User was to enter 23 or this would remain as 23.

The GETHEXIP subroutine returns to the calling routine when a carriage return is entered on the VDU. The actual number of input characters is returned in COUNT. The number of Hex bytes is in REVCO and the hex bytes are stored on the U stack.

Examples 1 and 2 below illustrate more precisely the values returned by GETHEXIP. In both examples the maximum number of allowable input characters set by the calling routine is 6.

Example 1

User enters 1 2 3 4 5 cr

GETHEXIP returns with
COUNT = 5
REVCO = 3
USTACK = 01, 23, 45

Example 2

User enters 1 2 3 4 cr

GETHEXIP returns with
COUNT = 4
REVCO = 2
USTACK = 12, 34

11.3 INDECU Input Decimal Number

This subroutine gets decimal characters from the VDU and transforms them into a hex value. The calling routine should set the maximum number of input characters in MAXNUM. The absolute maximum allowed is 5 decimal characters representing FFFF Hex or 65,535 decimal. INDECU checks the validity of the input characters, also checks that the count does not exceed the maximum number set by the calling routine and also checks that the input does not exceed 65,535.

The calling routine should set the maximum character count acceptable in MAXNUM and call INDECU as a subroutine. INDECU will return to the calling routine when a carriage return is entered on the VDU. The hex representation of the decimal input is in the D register.

11.4 INASCA

This subroutine in the ALS M6809 Monitor program is used by GETIP and INDECU (see Fig 9) to obtain characters from the VDU. It interacts directly with the VDU interface devices and returns the input character in the A register.
12 BASIC OUTPUT SUBROUTINES

12.1 OUTASC

This subroutine takes as its parameter the start address of an ASCII string and prints the string on the VDU. The string should be terminated by the End of Text (EOT) character (Hex 04).

12.2 OUTHEX

This subroutine takes as its parameter the address of an 8 bit byte which is printed on the VDU as a pair of Hex digits.

12.3 OUTDEC

This subroutine takes the address of a 16 bit number to be printed on the VDU as a decimal number.

The INASC, OUTASC, OUTHEX and OUTDEC subroutines are all part of the ALS M6809 Monitor program. For a full description of these subroutines see Ref 5.

13 DATA ENTRY SUBROUTINES

Data entry subroutines 13.2 to 13.6 require the address of either the A or B internal message structure (CCIMS or CC2MS in Fig 25) to be stored in a parameter passing location TEMP4.

13.1 Input Message Type GETTYPE (See also Fig 10)

Firstly the request to enter message type is output to the VDU. The type is entered via the GETIP subroutine. The input is validated and if invalid an error message is sent and the routine restarted. The internal identifier MSTYPE (See Fig 25) is loaded with 0, 1 or 2 corresponding respectively to A, B or T input. If T is input the Transmission Option field is interrogated and if a single transmission has been selected the number of transmission in MOPTNUM is increased from 1 to 2, in order that a single A plus B Message may be output. Program control is then returned to the calling routine.

13.2 Input Check Code INPCC (See also Fig 11)

This subroutine firstly prints a request to enter the check code on the VDU. The check code input is then collected via the GETHEXIP routine and then validated. ie 0 < CC < F. If the check code is invalid the 'Invalid Input' error message is printed and the input request repeated. If the check code is valid the input is stored in the check code field of the correct message (See Fig 25).

13.3 Input Register Address INPRA (See also Fig 12)

Firstly this subroutine prints a request to enter the register address. The register address is entered via the GETHEXIP subroutine and then validated ie 0 < RA < 7. If invalid the 'Invalid Input' error message is printed and the subroutine restarted. If the RA is valid it is stored in the RA field of the correct message (See Fig 25).
13.4 Input Data GETDAT (See also Fig 13)

The enter data prompt is output to the VDU. Up to 9 hex digits of data may be input (via the GETHEXIP subroutine), this corresponds to up to 5 bytes of data. The number of bytes of data input is in the counter scratchpad REVCO. Data is removed from the U stack and stored in the correct data structure with the least significant byte being stored in either D11MS or D21MS (See Fig 25) (depending on whether data is for an A or B message respectively). When all the data has been pulled from the stack, the data structure up to and including the last byte of the data field D15MS or D25MS is filled with zeroes. Program control is then returned to the calling routine.

13.5 Input Parity GETPART (See also Fig 14)

Firstly the Enter Parity prompt is issued. The user response to the prompt "Parity True Y or N?" should be N to ensure false parity, any other character entered gives true parity. True parity is therefore the default value. The subroutine checks the response obtained via GETIP and sets the A or B message parity field (see Fig 25) to be either true or false.

13.6 Input Message Format/Structure MSSTR (See also Fig 15)

The request for Format prompt is first output to the VDU. The input is then obtained via the GETIP subroutine. The response is checked and if G the Message Structure field (see Fig 25) (of the correct message) is set to Ø ie GTU. If the input is neither G nor M an error message is printed and the subroutine re-started. If the input is M a check is made to see if a second parameter has been entered. If no second parameter has been entered it is requested and obtained via GETIP. This second parameter is checked and if neither 1 or 2 an error message is printed and the subroutine re-started. If the second parameter is 1 (ie Matrix Command) the Message Structure field is set to 1. If the second parameter is 2 (ie Matrix Connection) the Message Structure field is set to 2.

After the setting of the Message Structure field, program control is returned to the calling routine.

13.7 Get Transmission Option GETTXOPT (See also Fig 16)

The prompt for input is displayed on the VDU. The input obtained via GETIP is validated. If not S, C or M an error message is printed and the subroutine re-started. If the input is S for single, TXOPT is loaded with 0 and the message type checked. If the type is A or B then the number of outputs (MOPTNUM) is loaded with 1. If type is T then MOPTNUM is loaded with 2. If the input is C (for continuous) then TXOPT is loaded with 1 and MOPTNUM is loaded with 1. Then the ON/OFF line variable ONOFFLI is interrogated. If ON line then Comfort Display information is requested (see Section 13.9) - control is returned to the calling routine after this information has been input - if OFF line control is returned to the calling routine. If the input is M for Multiple TXOPT is loaded with 2 and then the number of output transfers (in Decimal) must be input via INDECMNU. This number if not zero is stored in MOPTNUM. If the number
is greater than 1000 then ONOFFLI is interrogated and, if ON line, comfort display information requested as above. If the number is less than 1000 or the display is OFF line then control is returned to the calling routine.

13.8 Get Wait Conditions GETWAIT (See also Fig 17)

The request for input is displayed on the VDU. The input is collected via the GETIP subroutine. If the input is neither Y or N then an error message is printed and the subroutine re-started. If the input is N for NO then WAITY is set to zero and program control is immediately returned to the calling routine.

If the input is Y for YES then WAITY is set to 1 and the Wait Register address is collected via the GETEXIP subroutine. The input is validated i.e. it should be greater than or equal to zero and less than 8. If invalid an error message is printed and the Wait Register Address should be re-entered. If valid the wait register address is stored in WAITRA (see Figure 25) and program control returned to the calling routine.

13.9 Get Display Option GETDISOP (See also Fig 18)

The Display Option prompt is issued. The input is collected via the GETIP subroutine. If the input is not N, A or C then an error message is printed and the subroutine re-started.

If the input is N for None then DISOPT is set to zero and control returned to the calling routine.

If the input is A for All or C for Changes DISOPT is set to either 1 or 2 respectively. ON/OFF line selection is then required. If Off line display is selected ONOFFLI is set to zero and control returned to the calling routine. If ON line display is selected Segmentation entry is required (see Section 14.1) then DISOPT is again inspected and if ALL display has been selected control will return to the calling routine. If the user has selected to display Changes of Message TXOPT is inspected and if Single transmission has been selected control will immediately return to the calling program. If Multiple outputs have been selected the number of outputs (stored in MDPTNUM) must be greater than 1000 or the calling routine will be re-entered. At this point the user has the option of displaying 'comfort' messages every 1000th input. Selecting to display 'comfort' messages will cause COMFSTOR to be set to 1 (otherwise it will be cleared) and program control will then return to the calling routine.

14 DISPLAY ON REQUEST SUBROUTINES

14.1 Input Segmentation SEGMENT (See also Fig 19 and Para 17.2)

The prompt requesting Segmentation information is printed on the VDU. (For full description of Segmentation see Ref 2). The input, obtained via GETIP, is then checked. If the segmentation chosen is 1, 2 or 3 (corresponding to GTU, MATRIX Command and MATRIX Status respectively) then the correct pre-set segmentation values are copied into the segmentation working stack and control returned to the calling routine.

If the segmentation requested is 4, i.e. Input Dependent, a flag is set (which will be used in the Print Message Subroutine see para 14.3) and control is immediately returned to the calling program. If the Segmentation is not 4 the flag is cleared.
If the segmentation selected is $\emptyset (\text{SEGO})$ ie User Defined then a second segmentation prompt is displayed on the VDU. The user should then decide whether or not to change the existing segmentation $\emptyset$ specification. If not the existing specification is copied into the segmentation working stack and control returned to the calling program. If the user elects to change the previously defined segmentation $\emptyset$ specification then segmentation specification input information is displayed on the VDU and user input enabled. (Up to 7 user defined segmentation spaces may be entered, input should be terminated by entering the letter 'E').

As each input is entered it is validated to ensure that it is either the letter 'E' or a decimal number (parts of INDECU, see section 11.3, are used to validate the decimal number). If the input is a valid decimal number it is stored in the User Segmentation stack and the next input requested. If the input is invalid the whole Segmentation $\emptyset$ section is restarted, this also happens if the user tries too many segmentation spaces. If the input is the termination character 'E' the segmentation specification built up in that part of memory allocated to user defined segmentation is copied in the segmentation working stack and program control returned to the calling routine.

14.2 Print Block PRINBLK (See also Fig 20 and Para 17.2)

The Print Block subroutine calculates the start address of each message (the datum being the start address of the Block) and calls the Print Message (PRINMES) subroutine (with the message start as the parameter) to print the message. It then calculates the start address of the next message and continues printing messages until every message in the block has been printed. Control is then returned to the calling routine.

14.3 Print Message PRINMES (See also Fig 21).

The automatic segmentation flag is checked and if set the Message Type (MSTYPE) is checked, if the Type is T (for both) then the input message number is checked. An odd input number will correspond to a response from an A message, similarly an even input number to a B message. The Format field of the structure (See Fig 25)of the correct message is then interrogated. If the format field is set to $\emptyset$ then segmentation 1 is used, otherwise the Register Address parameters are extracted from the input message. If the input Register Address is equal to 2 (ie a MATRIX Status input) then segmentation 2 is selected otherwise segmentation 3 is selected. The selected segmentation specification is then copied into the segmentation working stack. At this point either the automatic segmentation flag was clear, and the segmentation $\emptyset$, 1, 2 or 3 identified or the automatic segmentation has been determined as above.

The message start address parameter passed by the calling routine is then collected. The first two bytes contain the message number. This is printed on the VDU in decimal format using OUTDEC.

The remaining 6 bytes of the message contain the 42 bits of the input message. Each bit is collected in turn and stored in ASCII format (ASCII $\emptyset$ being used for a bit equal to $\emptyset$ and ASCII being used for bit equal to 1) in a vertical stack ASMSST. As each bit is stored the bit number is checked against the segmentation specification and if the bit number corresponds to a segmentation space an ASCII space character is inserted into ASMSST before the actual bit.
When all 42 bits of the message have been processed ASMSST contains a series of ASCII 0s, 1s and spaces. This whole stack is printed on the VDU using the OUTSAC subroutine. Control is then returned to the calling routine.

THE HARDWARE INTERFACES

15.1 Programming the PPIs

The Hardware Handlers control the input and output peripheral circuits via four Programmable Peripheral Interface devices (PPIs). The input/output functions to these devices are programmed initially in the Initialisation routine which is entered after the system is reset.

Tables 1 to 4 list the PPI addresses and their internal identifiers and Figure 22 illustrates the nature (ie input or output) of their individual programming and details the function of the bits of each of the PPI ports.

<table>
<thead>
<tr>
<th>HEX ADDRESS</th>
<th>PORT</th>
<th>FUNCTION</th>
<th>IDENTIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>8080</td>
<td>A</td>
<td>OUTPUT</td>
<td>POPDAT</td>
</tr>
<tr>
<td>8081</td>
<td>B</td>
<td>OOUTPUT</td>
<td>POPADD</td>
</tr>
<tr>
<td>8083</td>
<td>CONTROL</td>
<td>CONTROL</td>
<td>POPCON1</td>
</tr>
</tbody>
</table>

CONTROL BYTE HEX 89

Table 1 PPI 1 Notations

Loading the Control byte (Hex 89) into the control port POPCON1 programs Ports A and B to be outputs.

<table>
<thead>
<tr>
<th>HEX ADDRESS</th>
<th>PORT</th>
<th>FUNCTION</th>
<th>IDENTIFIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>8084</td>
<td>A</td>
<td>OUTPUT</td>
<td>POPCOM</td>
</tr>
<tr>
<td>8085</td>
<td>B</td>
<td>INPUT</td>
<td>POPSTA</td>
</tr>
<tr>
<td>8087</td>
<td>CONTROL</td>
<td>CONTROL</td>
<td>POPCON2</td>
</tr>
</tbody>
</table>

CONTROL BYTE HEX 8B

Table 2 PPI 2 Notations

Loading the control byte (Hex 8B) into the control port POPCON2 programs Port A to be an output and Port B to be an input.

PPIs 1 and 2 are used to interface to the Axis Test Box peripheral card controlling OUTPUT TRANSFERS.
HEX ADDRESS | PORT | FUNCTION | IDENTIFIER
-------------|------|----------|-------------
8088         | A    | INPUT    | PIPDAT      
8089         | B    | OUTPUT   | PIPADD      
808B         | CONTROL | CONTROL | PIPCON1     

CONTROL BYTE HEX 99

Table 3  PPI 3 Notations

Loading the control byte (Hex 99) into the control port PIPCON1 programs Port A to be an input and Port B to be an output.

HEX ADDRESS | PORT | FUNCTION | IDENTIFIER
-------------|------|----------|-------------
808C         | A    | OUTPUT   | PIPCOM      
808D         | B    | INPUT    | PIPSTA      
808F         | CONTROL | CONTROL | PIPCON2     

CONTROL BYTE HEX 8B

Table 4  PPI 4 Notations

Loading the control byte (Hex 8B) into the control port PIPCON 2 programs Port A to be an output and Port B to be an input.

PPIs 3 and 4 are used to interface to the peripheral card controlling INPUT TRANSFERS.

15.2 Hardware/Software Boundary

When considering the boundary between the hardware and software in the AXIS Test Box it is important to consider the functions to be performed by the hardware. In the most basic terms these functions are:

a To accept data from the microprocessor.

b To transmit this data in the form of an output transfer to the system under test.

c To receive input transfers from the system under test.

d To present the received data to the microprocessor for investigation.
Since the AXIS Test Box is used in place of the System 250 (see Ref 1) it is necessary that the interface between the Test Box and the system under test should be identical to that which the S250 would present. Of the 4 functions listed above items c) and d) must, therefore, be performed in real time, i.e. using the same transmission rate as the S250. Reference 5 gives full details of the interface requirements.

The time interval between transfers is not defined therefore the processing necessary to perform functions a) and d) above may be performed in non-real time, i.e. at a rate determined only by the instruction speed of the M6809 microprocessor controlling the Test Box.

15.3 Outputting Data to the Hardware Registers

Once data has been prepared for transmission, as described in Sections 9 and 10, and is in the format shown in Figure 26B it must be loaded into the hardware registers. There are two hardware registers A and B. Each is 42 bits long and each is made up of five shift registers of 8 bits and one shift register of 2 bits. Each of the individual shift registers has been allocated a specific address (see Annex D). When a hardware register address is output via POPADD (Fig 22) the hardware decodes the address and generates a parallel load pulse for the selected shift register. Each byte of the data is output to the hardware via POPDAT (see Fig 22). The hardware register address is then output via POPADD this then causes a hardware parallel load pulse to be generated for the correct hardware register and the POPDAT data is latched into the hardware register (see para 10.4).

When all the hardware data registers have been loaded the number of transmissions is loaded into the hardware counter in a similar manner.

Bit A₂ in POPCOM is set to Transmit, (i.e. logic 0) and bit A₁ set to logic 1 if the Continuous option was selected. Bit A₃ is cleared if not in Auto Receive Mode, and finally a GO pulse sent on bit A₀. Once the GO pulse has been sent the hardware assumes control and the data is transmitted in real time to the system under test.

15.4 Reading Data from the Hardware Registers

The hardware input buffer is 42 bits in length, and is made up of five shift registers of 8 bits and one of 2 bits. Data is accessed via six tri-state buffers. (one for each shift register of the input buffer). Annex E lists the addresses of these tri-state buffers. When an input transfer has been received in real time from the system under test the polling routines described in para 16.1.2 inspect the input board status inputs (PIPS/T in Fig 22). If the Ready to Read bit (bit B₀) is set then the address of each byte of the hardware input buffer is loaded into PIPADD and that data byte read into the microprocessor via PIPDAT. This process is repeated until each byte of the hardware input buffer has been read into micro-processor store. The interrogation routines (see para 16.2) then assume control. The transfer of data from the hardware to the microprocessor is not carried out in real time.

16 HARDWARE HANDLERS

The Hardware Handling code can be divided into two sections:
a Handling hardware go/stop, and polling for hardware or VDU responses. These are the POLLING routines.

b Interrogation of inputs, storing, displaying etc. These are the INTERROGATION routines.

16.1 Polling Routines (See also Figs 22 and 23)

16.1.1 Starting Transfers

For an output transfer the hardware counter is loaded with the numbers of transfers to be sent. RX/TX is set to TX and a logic pulse generated on the GO line commands the output board to proceed with the transfer.

For an input, transfer circuitry on the output transfer board generates the timing necessary for inputs to be received. RX/TX is set to RX and a GO pulse generated.

16.1.2 Polling for Responses

Responses may arise from any of the following three areas:

a User response from VDU ie Stop command.

b Output transfers.

c Input requests, received etc.

Responses from a above obviously come via the VDU interface, the Stop command has been set to be a specific sequence of the letter S followed by carriage return. Upon receipt of the letter S half the stop command has been received and the HALFSTOP flag set. A user input of carriage return will then cause a FULL-STOP, one more output and input sequence will then be accepted in order to synchronise the transmissions. An information message informing of user intervention will be printed on the VDU and control returned to the Basic Command Mode. (In the case of Automatic Input Display the next input will synchronise the termination).

If the character entered after the letter S is not a Carriage Return the HALFSTOP flag will be cleared and the whole Stop sequence must be re-entered before it will be accepted. This is done in order that the act of Stopping transfers should be a deliberate action and the recognition erroneous Stop commands minimised. Termination may also be caused by filling the input message store.

Once an output transfer has started it will continue until completed. Whenever the number of output transfers programmed into the hardware counter has been sent the Transmitting/Stopped bit in POPSTA will be raised to indicate completion of output sequence. When this has happened one more input will be accepted in order to synchronise with the system under test. If the Transmit Option selected is Continuous then the Continuous bit (see Figure 22) will be set thus disabling the hardware counters. In this case the user must intervene to stop outputs, or the message store must be filled.
When an output has been activated inputs should be expected and the polling routine will interrogate the status responses from the Peripheral Input Transfer card. These status responses (see Figure 22) are:

a. Ready to read - this indicates that data has been received and is in the input buffer shift register awaiting interrogation. (See also para 15.4)

b. Request i/p Transfer (TFR) - this indicates that the system under test has requested an input transfer. This request is granted by putting the RX/TX bit to RX and generating a GO pulse.

c. End i/p transfer - this indicates the end of the input transfer. If this bit is set and the Ready to Read bit is clear it indicates that the system under test has failed to return the standard 42 bits of data. (The Matrix returns only 36 bits of data). A circuit on the Input peripheral card which generates 6 extra pulses, for the input buffer, is then activated. This is done so that standard interrogation routines described in section 16.2 may be used for both GTU and Matrix messages.

d. Input timeout - this indicates that an output has been sent but the system under test has failed to respond within a specified period. After each Output transfer (or, in the case of Auto Display Mode, Input transfer) the timeout counter is reset. In order to avoid a possible 'Lock up' situation arising, if the system under test fails to respond, a timeout on Inputs is enabled. When the timeout status bit is set the peripheral hardware cards are re-set to Transmit mode and another Output sent.

16.2 Interrogation Routines (See also Fig 24)

The Interrogation Routines are entered when the Ready to Read Status bit is set. If the Display Option chosen is N for None and no Wait Conditions have been specified (ie the hardware delay is enabled) HALFSTOP is interrogated to check if this message should terminate the transfer sequence. If termination is indicated (ie HALFSTOP = 2) then the termination message is printed and control returned to the Basic Command Mode. If termination is not indicated (ie HALFSTOP ≠ 2) the hardware is re-set to Transmit, and the Polling Routines re-entered.

If the above conditions are not met the data in the input buffer is read and the message together with its message number is stored in a temporary storage area THISIP.

If a Wait Condition has been set the input Message Register Address is extracted and compared with the Wait Register Address. If a match is not achieved the termination check described is carried out and if no termination is indicated the hardware is re-set to Transmit and the Polling Routines re-entered.
If the Wait Register Address match is found, or no Wait conditions have been set, the Display Option chosen is inspected.

If All inputs are to be stored for display then the input message with its internally generated message number is copied from THISIP to the next message space available in the input message store. If ON Line display has been selected the message will then be displayed on the VDU using the Print Message subroutine.

The total number of input messages will then be checked and if the input message store is now full the input/output sequence will be terminated, otherwise the hardware will be re-set to Transmit and the polling routines re-entered.

If Changes of message are to be stored for display then this input message must be checked against the last message of the same type. If the Type is A then against the last A message preserved in LASTAIP. If the Type is B then the check is made against the last B message to be received and preserved in LASTBIP. If the type is T (for Both) then the input message number is inspected. If the input message number is ODD then the check is made against LASTAIP and if EVEN the check is made against LASTBIP.

If differences were found then the message in THISIP together with its message number is copied into the next space in the input message store and the relevant LAST A or B IP is overwritten with the new data only (excluding message number). If ON line display has been selected the message and message number will be printed on the VDU using the PRINMES subroutine; the termination checks will then be carried out and if no termination is indicated the hardware is re-set to Transmit and the Polling Routines re-entered.

If no differences were found then the 'comfort' message flag COMFSTOR is checked and if set the message number inspected. If this is the 1000th message received since the last display it will be displayed on the VDU using the PRINMES subroutine.

The termination checks are then carried out and in the absence of termination indication the hardware is again re-set to Transmit and the Polling Routines re-entered.

17 THE DATA STRUCTURES

17.1 Message Assembly

When the Message Assembly has been completed the INTERNAL MESSAGE STRUCTURE (Figure 25) will be filled with the correct information.

Information is stored in this structure in a form which makes changing individual parameters relatively simple since each parameter is stored independently. This structure is not, however, the ideal for operations such as parity calculations and therefore, when it becomes necessary to calculate parity bits, the message information stored in the A or B Message Fields is transformed by the FMFORP (Format for Parity) subroutine into the structure shown in Figure 26A. In this structure each bit of the 42 bit message occupies a whole byte with 00 representing a bit 0 and 01 representing a bit 1. With the information stored in this format
the CALCP subroutine may be called to generate parity over any data area within the 42 bit message and store the parity bit in whichever bit position is selected.

Once the parity bit(s) has been generated the information stored in the structure in Figure 26A is transformed by the TFTOTX subroutine into a format ideal for interfacing to the hardware by the hardware handlers. This format is illustrated in Figure 26B.

This is done in order that the user may select to transmit A, B or both when using the Repeat Command, and may also change individual elements of each message independently when using the Repeat Command.

17.2 Message Display

When input messages are received they are stored (in accordance with the display option selected) in the form illustrated in Figure 27A. Each input message is given a message number which is stored with the message.

Messages are stored in blocks of 16 with space available for up to 40 blocks. When the Display on request is selected the total number of messages stored is displayed and the user allowed to select the segmentation format required. This segmentation allows fields within the 42 bit message to be isolated for easier inspection. Three pre-set segmentations are available plus one which is user programmable. Figure 28 illustrates the segmentation storage area. The example shown in Figure 28 for segment 1 means that a space will be inserted in the bit pattern displayed after the 4th, 7th and 9th and 41st bits. This could isolate for example the check code, register address, data and parity fields of each message displayed.

In order to prepare each message for display and to include the spaces for segmentation the message is transformed from the format illustrated in Figure 27A to that illustrated in Figure 27B. In Figure 27B each bit of the 42 bit message occupies a byte, a bit 0 being represented by Hex 30 (ie ASCII 0) and a bit 1 being represented by HEX 31 (ie ASCII 1). The segmentation spaces (Hex 20) are inserted in the correct bit position thus when the message is output the message will appear as a series of 0s and 1s with spaces providing the segmentation isolation.

18 TRANSMIT DATA FLOW (See also Fig 29)

All input information is collected via the Data Entry Subroutines described in Section 13. At this point the information is stored in the internal message structure Fig 25. The Format for Parity subroutine (para 10.1) translates the Data, Check Code and Register Address into a format which makes parity calculation easier (Fig 26A). When the parity bit(s) have been inserted the information is then transformed (see para 10.3) into a format suitable for transmission (Fig 26B). This information is then loaded into the hardware registers as described in para 15.3.

19 RECEIVE DATA FLOW (See also Fig 30)

The hardware input buffer register is read (see para 15.4) and the information stored in the format shown in Fig 27A. If the input transfer is to be displayed on the VDU, using the Print Message subroutine (see para 14.3) it is firstly transformed into the format illustrated in Fig 27B. Segmentation spaces are inserted and each data bit translated into its ASCII representation ready for output to the VDU. Reference 2 describes and illustrates quite clearly the type of display to be expected using different segmentation specifications.
### INTERNAL IDENTIFIER CROSS REFERENCE

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>INTERNAL LABEL</th>
<th>EXTERNAL VALUE</th>
<th>INTERNAL VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESSAGE TYPE</td>
<td>MSTYPE</td>
<td>A - A MESSAGE</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B - B MESSAGE</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T - BOTH</td>
<td>2</td>
</tr>
<tr>
<td>A PARITY</td>
<td>APARTRU</td>
<td>TRUE Y - YES</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N - NO</td>
<td>1</td>
</tr>
<tr>
<td>B PARITY</td>
<td>BPARTRU</td>
<td>TRUE Y - YES</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N - NO</td>
<td>1</td>
</tr>
<tr>
<td>A FORMAT</td>
<td>AFORMAT</td>
<td>G - GTU</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M1 - MATRIX COMMAND</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M2 - MATRIX CONNECTION</td>
<td>2</td>
</tr>
<tr>
<td>B FORMAT</td>
<td>BFORMAT</td>
<td>G - GTU</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M1 - MATRIX COMMAND</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M2 - MATRIX CONNECTION</td>
<td>2</td>
</tr>
<tr>
<td>TRANSMISSION OPTION</td>
<td>TXOPT</td>
<td>S - SINGLE</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C - CONTINUOUS</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M - MULTIPLE</td>
<td>2</td>
</tr>
<tr>
<td>WAIT CONDITION</td>
<td>WAITY</td>
<td>Y - YES</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N - NO</td>
<td>1</td>
</tr>
<tr>
<td>DISPLAY OPTION</td>
<td>DISOPT</td>
<td>N - NONE</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A - ALL</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C - CHANGES</td>
<td>2</td>
</tr>
<tr>
<td>ON OR OFF LINE DISPLAY</td>
<td>ONOFFLI</td>
<td>ON LINE Y - YES</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N - NO</td>
<td>1</td>
</tr>
<tr>
<td>RECEIVE OR TRANSMIT</td>
<td>RXTX</td>
<td>TRANSMIT</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RECEIVE</td>
<td>1</td>
</tr>
<tr>
<td>COMFORT MESSAGES</td>
<td>COMFSTOR</td>
<td>COMFORT MESSAGES Y - YES</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N - NO</td>
<td>0</td>
</tr>
</tbody>
</table>

**ANNEX A**
ANNEX B

DRAWING SYMBOLS USED

**INCREMENT**

- **Activity**: Code performs this task

**POINTER**

**GETIF**

- **Subroutine**

**Decison Box**

**GOTO** eg GOTO A

**Entry point**: GOTO A would arrive at this point

**Continuation**

Flow Diagram continuing at Fig 8.

**Continued**

**FIG 8**

**FORMED FOR PARITY**

**FIG 26A**

**Data Structure**
### ANNEX C

**CROSS REFERENCE BY INTERNAL LABEL NAME**

<table>
<thead>
<tr>
<th>LABEL</th>
<th>CODE FUNCTION</th>
<th>PARAGRAPH</th>
<th>FIGURE NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASHMES</td>
<td>Message Assembly Routine</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>AUTODIS</td>
<td>Auto Display Routine</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>CALCP</td>
<td>Calculate Parity Subroutine</td>
<td>19.2</td>
<td>-</td>
</tr>
<tr>
<td>COMMODE</td>
<td>Basic Command Mode Routine</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>DISOREQ</td>
<td>Display on Request Routine</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>ENDOFSUB</td>
<td>Prepare for Hardware Handlers Routine</td>
<td>9.1</td>
<td>8</td>
</tr>
<tr>
<td>FMFDRP</td>
<td>Format for Parity Generation Subroutine</td>
<td>19.1</td>
<td>-</td>
</tr>
<tr>
<td>GETDAT</td>
<td>Input Data Subroutine</td>
<td>13.4</td>
<td>13</td>
</tr>
<tr>
<td>GETDISOP</td>
<td>Get Display Options Subroutine</td>
<td>13.9</td>
<td>18</td>
</tr>
<tr>
<td>GETHEXIP</td>
<td>Get Hexadecimal Input Subroutine</td>
<td>11.2</td>
<td>9</td>
</tr>
<tr>
<td>GETIP</td>
<td>Get Input Character Subroutine</td>
<td>11.1</td>
<td>9</td>
</tr>
<tr>
<td>GETPART</td>
<td>Get Parity Subroutine</td>
<td>13.5</td>
<td>14</td>
</tr>
<tr>
<td>GETTXOPT</td>
<td>Input Transmission Options Subroutine</td>
<td>13.7</td>
<td>16</td>
</tr>
<tr>
<td>GETTYPE</td>
<td>Input Message Type Subroutine</td>
<td>13.1</td>
<td>10</td>
</tr>
<tr>
<td>GETWAIT</td>
<td>Input Wait Conditions Subroutine</td>
<td>13.8</td>
<td>17</td>
</tr>
<tr>
<td>GOANDTX</td>
<td>Prepare Data for Transmission Routine</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>INASC</td>
<td>Get ASCII Character from Keyboard Subroutine</td>
<td>11.4</td>
<td>9</td>
</tr>
<tr>
<td>INDECNU</td>
<td>Input Decimal Number Subroutine</td>
<td>11.3</td>
<td>9</td>
</tr>
<tr>
<td>INITIAL</td>
<td>Initialisation Routine</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>INPCC</td>
<td>Input Check Code Subroutine</td>
<td>13.2</td>
<td>11</td>
</tr>
<tr>
<td>INPRA</td>
<td>Input Register Address Subroutine</td>
<td>13.3</td>
<td>12</td>
</tr>
<tr>
<td>LABEL</td>
<td>CODE FUNCTION</td>
<td>PARAGRAPH</td>
<td>FIGURE NUMBER</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>MENUDIS</td>
<td>Display Current Menu Subroutine</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>MSSTR</td>
<td>Input Message Format/Structure Subroutine</td>
<td>13.6</td>
<td>15</td>
</tr>
<tr>
<td>OUTASC</td>
<td>Output ASCII character to keyboard Subroutine</td>
<td>12.1</td>
<td>9</td>
</tr>
<tr>
<td>OUTDEC</td>
<td>Output Decimal Subroutine</td>
<td>12.3</td>
<td>9</td>
</tr>
<tr>
<td>OUTHEX</td>
<td>Output Hexadecimal Subroutine</td>
<td>12.2</td>
<td>9</td>
</tr>
<tr>
<td>POLLING</td>
<td>Polling Routines</td>
<td>16.1.2</td>
<td>23</td>
</tr>
<tr>
<td>PRINBLK</td>
<td>Print Block Subroutine</td>
<td>14.2</td>
<td>20</td>
</tr>
<tr>
<td>PRINMES</td>
<td>Print Message Subroutine</td>
<td>14.3</td>
<td>21</td>
</tr>
<tr>
<td>REAIPDAT</td>
<td>Interrogation Routines</td>
<td>16.2</td>
<td>24</td>
</tr>
<tr>
<td>REPEATX</td>
<td>Repeat Routine</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>SEGMENT</td>
<td>Input Segmentation Subroutine</td>
<td>14.1</td>
<td>19</td>
</tr>
<tr>
<td>TFTOTX</td>
<td>Transform to Transmit Format Subroutine</td>
<td>16.3</td>
<td>-</td>
</tr>
</tbody>
</table>
## ANNEX D

### OUTPUT BOARD HARDWARE REGISTER ADDRESSES

<table>
<thead>
<tr>
<th>ADDRESS (IN HEXADECIMAL)</th>
<th>REGISTER SELECTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NONE - THIS IS THE QUIESCENT STATE</td>
</tr>
<tr>
<td>1</td>
<td>Register A Shift Register 1</td>
</tr>
<tr>
<td>2</td>
<td>Register A Shift Register 2</td>
</tr>
<tr>
<td>3</td>
<td>Register A Shift Register 3</td>
</tr>
<tr>
<td>4</td>
<td>Register A Shift Register 4</td>
</tr>
<tr>
<td>5</td>
<td>Register A Shift Register 5</td>
</tr>
<tr>
<td>6</td>
<td>Register A Shift Register 6</td>
</tr>
<tr>
<td>7</td>
<td>Register B Shift Register 1</td>
</tr>
<tr>
<td>8</td>
<td>Register B Shift Register 2</td>
</tr>
<tr>
<td>9</td>
<td>Register B Shift Register 3</td>
</tr>
<tr>
<td>A</td>
<td>Register B Shift Register 4</td>
</tr>
<tr>
<td>B</td>
<td>Register B Shift Register 5</td>
</tr>
<tr>
<td>C</td>
<td>Register B Shift Register 6</td>
</tr>
<tr>
<td>D</td>
<td>Hardware Counter Shift Register 1</td>
</tr>
<tr>
<td>E</td>
<td>Hardware Counter Shift Register 2</td>
</tr>
<tr>
<td>F</td>
<td>UNUSED</td>
</tr>
</tbody>
</table>
### ANNEX E

**INPUT BOARD HARDWARE REGISTER ADDRESSES**

<table>
<thead>
<tr>
<th>ADDRESS (IN HEXADECIMAL)</th>
<th>REGISTER SELECTED</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NONE - THIS IS THE QUIESCENT STATE</td>
</tr>
<tr>
<td>1</td>
<td>Input Buffer Register 1</td>
</tr>
<tr>
<td>2</td>
<td>Input Buffer Register 2</td>
</tr>
<tr>
<td>3</td>
<td>Input Buffer Register 3</td>
</tr>
<tr>
<td>4</td>
<td>Input Buffer Register 4</td>
</tr>
<tr>
<td>5</td>
<td>Input Buffer Register 5</td>
</tr>
<tr>
<td>6</td>
<td>Input Buffer Register 6</td>
</tr>
<tr>
<td>7 → F</td>
<td>UNUSED</td>
</tr>
</tbody>
</table>
Enter from Basic Command Mode

1. Get type
   - A
   - B
   - Both

2. Input check code

3. Input register address

4. Input data

5. Input parity

6. Input format

7. Is type ≠ A?
   - Yes
   - No

8. Was last message B?
   - Yes
   - No

9. Get TX option S, C, or M
   - Is it M?
     - No
     - Yes
     - Input decimal number

10. Get display option N, A, C
    - Is it N?
      - Yes
      - Go to Basic Command Mode
      - No
      - On/off line display

11. Ready to TX?
    - Yes
    - No
    - Prepare data for transmission

12. Hardware handlers

Fig. 2: Assemble Message Routine Flow Diagram
ENTER FROM BASIC COMMAND MODE

GET DISPLAY OPTION
M, A, C etc.

READY TO RECEIVE
NO
GO TO BASIC COMMAND MODE
YES

GO TO HARDWARE HANDLERS

FIG 4 AUTO DISPLAY ROUTINE FLOW DIAGRAM
Enter from Basic Command or Repeat Modes

Display Type

Is Type A

Is Type B

Display Content of A Message

Yes: Is Type A

No: Is Type B

Yes: Display Content of B Message

No: Type Must Be Y for Both

Display Content of A and B Messages

Display Transmission Options

Display Wait Conditions

Display Display Options

Return to Calling Routine

Fig 6 Menu Display Routine Flow Diagram
FIG 7 PREPARE DATA FOR TRANSMISSION FLOW DIAGRAM
ENTER FROM FIG 7 OR FROM AUTO RECEIVE MODE

PRESET LAST INPUT MESSAGE STRUCTURES

SET START OF INPUT MESSAGE STORE

ARE WE IN AUTO RX MODE?

Y

SET HARDWARE (PPIs) FOR RECEIVE

N

GO TO HARDWARE HANDLERS

LOAD O/P REG 1 WITH A

Y

IS TYPE T?

N

LOAD O/P REG 1 WITH A

LOAD O/P REG 2 WITH B

LOAD O/P REG 2 WITH B

IS TYPE T?

Y

LOAD O/P REG 1 WITH A

N

LOAD O/P REG 2 WITH A

IS TYPE A?

Y

LOAD O/P REG 1 WITH A

N

LOAD O/P REG 2 WITH A

IS TYPE B?

Y

LOAD O/P REG 1 WITH A

N

LOAD O/P REG 2 WITH A

TX OPT SINGLE

YES

LOAD O/P REG 2 WITH A

NO

LOAD O/P REG 2 WITH A

IS THERE A WAIT SET?

YES

LOAD O/P REG 2 WITH A

NO

HAVE TO DISPLAY?

YES

LOAD H/W COUNTER WITH NUMBER

NO

IS O/P MULTIPLE?

YES

PREPARE FOR MULTIPLE COUNTDOWN

NO

HARDWARE HANDLERS

FIG 8 PREPARE FOR HARDWARE HANDLERS FLOW DIAGRAM
INPUT
11.1 GETIP
11.2 GETHEXIP
11.3 INDECHU
11.4 INASC

OUTPUT
12.1 OUTASC
12.2 OUTHEX
12.3 OUTDEC

A L S M6809 MONITOR SUBROUTINES

FIG 9 BASIC INPUT/OUTPUT SUBROUTINES
ENTER

OUTPUT TYPE REQUEST

GET IP

LOAD MSTYPE WITH 0

IS TYPE A

NO

LOAD MSTYPE WITH 1

IS TYPE B

NO

LOAD MSTYPE WITH 2

IS TYPE T

NO

NO SINGLE

YES

LOAD MNOT NUM WITH 2

RETURN TO CALLING ROUTINE

PRINT ERROR MESSAGE

FIG 10 INPUT MESSAGE TYPE SUBROUTINE FLOW DIAGRAM
GETHEXIP

ENTER

PRINT INPUT CC REQUEST

IS INPUT VALID
O &CC &P

STORE CC IN CC FIELD OF
CORRECT MESSAGE

RETURN TO CALLING ROUTINE

PRINT ERROR

FIG 11 INPUT CHECK CODE SUBROUTINE FLOW DIAGRAM
FIG 12 INPUT REGISTER ADDRESS SUBROUTINE FLOW DIAGRAM

ENTER

PRINT INPUT RA REQUEST

PRINT ERROR

HEXHEXIP

IS INPUT VALID
\( \theta \leq RA < 7 \)

YES

NO

STORE RA IN RA FIELD OF CORRECT MESSAGE

RETURN TO CALLING ROUTINE
ENTER

PRINT DATA REQUEST

GETHEXIP

GET POINTER TO DATA STRUCTURE

PULL BYTE OFF STACK

PUT BYTE INTO DATA STRUCTURE

INCREMENT POINTER

DECREMENT REVCO

IF REVCO 0

YES

PUT ZERO INTO DATA BYTE

ARE WE AT DIAM OR D25ME

NO

YES

INCREMENT POINTER

RETURN TO CALLING ROUTINE
FIG 14  GET PARITY SUBROUTINE  FLOW DIAGRAM
FIG 15 INPUT MESSAGE FORMAT/STRUCTURE SUBROUTINE FLOW DIAGRAM

ENTER

ISSUE STRUCTURE/FORMAT PROMPT

GETIP

IS IT G

NO

PRINT ERROR

YES

SET STRUCTURE FIELD TO 0

IS IT G

NO

IS IT M

YES

SET STRUCTURE FIELD TO 1

NO

IS THERE A 2nd PARAM

YES

SET STRUCTURE FIELD TO 2

NO

IS IT 2

YES

RETURN TO CALLING ROUTINE
FIG 16 GET TRANSMISSION OPTION SUBROUTINE FLOW DIAGRAM
FIG 22 PPI PORT AND BIT ALLOCATIONS
<table>
<thead>
<tr>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON LINE/OFF LINE</td>
<td>ONOFFLI</td>
</tr>
<tr>
<td>DISPLAY OPTION</td>
<td>DISOPT</td>
</tr>
<tr>
<td>WAIT REGISTER ADDRESS</td>
<td>WAITRA</td>
</tr>
<tr>
<td>WAIT Y OR D</td>
<td>WAITY</td>
</tr>
<tr>
<td>NUMBER</td>
<td>OPTXNUM1</td>
</tr>
<tr>
<td>NUMBER</td>
<td>OPTXNUM</td>
</tr>
<tr>
<td>NUMBER</td>
<td>MOPTNUM1</td>
</tr>
<tr>
<td>NUMBER</td>
<td>MOPTNUM</td>
</tr>
<tr>
<td>TRANSMISSION OPTION</td>
<td>TXOPT</td>
</tr>
<tr>
<td>'B' MESSAGE FORMAT/STRUCTURE</td>
<td>B FORMAT</td>
</tr>
<tr>
<td>'B' PARITY</td>
<td>BPARTRU</td>
</tr>
<tr>
<td>'B' DATA</td>
<td>D25MS</td>
</tr>
<tr>
<td>'B' DATA</td>
<td>D24MS</td>
</tr>
<tr>
<td>'B' DATA</td>
<td>D23MS</td>
</tr>
<tr>
<td>'B' DATA</td>
<td>D22MS</td>
</tr>
<tr>
<td>'B' DATA</td>
<td>D21MS</td>
</tr>
<tr>
<td>'B' REGISTER ADDRESS</td>
<td>RA2MS</td>
</tr>
<tr>
<td>'B' CHECK CODE</td>
<td>CC2MS</td>
</tr>
<tr>
<td>'A' MESSAGE FORMAT/STRUCTURE</td>
<td>A FORMAT</td>
</tr>
<tr>
<td>'A' PARITY</td>
<td>APARTRU</td>
</tr>
<tr>
<td>'A' DATA</td>
<td>D15MS</td>
</tr>
<tr>
<td>'A' DATA</td>
<td>D14MS</td>
</tr>
<tr>
<td>'A' DATA</td>
<td>D13MS</td>
</tr>
<tr>
<td>'A' DATA</td>
<td>D12MS</td>
</tr>
<tr>
<td>'A' DATA</td>
<td>D11MS</td>
</tr>
<tr>
<td>'A' REGISTER ADDRESS</td>
<td>RA1MS</td>
</tr>
<tr>
<td>'A' CHECK CODE</td>
<td>CC1MS</td>
</tr>
<tr>
<td>MESSAGE TYPE</td>
<td>MSTYPE</td>
</tr>
</tbody>
</table>

**Figure 25 Internal Message Structure**
DATA LSB

DATA MSB

VRISTK

PARITY

3 BYTES - P0, P1, P2  M1
2 BYTES - P0, P4  M2
1 BYTE - P0  GTU

DATA 32 BYTES  M1
31 BYTES  M2
30 BYTES  GTU

RA = 3 BYTES

CC = 4 BYTES

6 BYTES PER MESSAGE

A MESSAGE

M1 = MATRIX CONNECTION
M2 = MATRIX COMMAND

FIGURE 26A FORMED FOR PARITY

FIGURE 26B READY FOR TRANSMISSION

FIG 26 TX MESSAGE STRUCTURE
FIG 27A  INPUT MESSAGE STORE FORMAT

FIG 27B  READY TO DISPLAY
NUMBER OF DISPLAY BIT AFTER WHICH, SPACE INSERTED IN DISPLAY

NUMBER (42

USER DEFINED SEGMENT

8 SEGMENTS ALLOWED IN DISPLAY

SEG 1

SEG 2

SEG 3

FIRSEG

SECSEG

THISEG

SEGMENTATION NUMBERS IN ASCENDING ORDER
DATA ENTRY SUBROUTINES (Section 13) → INTERNAL MESSAGE STRUCTURE (Fig. 25)

FORMAT FOR PARITY (Para. 16.1) → FORMED FOR PARITY (Fig. 26A)

TRANSFORM TO TRANSMIT FORMAT (Para. 18.3) → READY FOR TRANSMISSION (Fig. 26B)

LOAD INTO HARDWARE REGISTERS (Para. 15.3) → HARDWARE REGISTERS
FIG 30 RECEIVE DATA FLOW

1. READ INPUT BUFFER REGISTERS (Para. 15.4)
2. INPUT MESSAGE STORE FORMAT (Fig. 27a)
3. PRINT MESSAGE (Para. 14.3)
4. READY TO DISPLAY (Fig. 27b)
5. OUTPUT MESSAGE TO VDU
6. MESSAGE DISPLAYED ON VDU SCREEN (See Ref. 2)
DOCUMENT CONTROL SHEET

Overall security classification of sheet ................................................. UNCLASSIFIED

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S) )

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORANDUM 3585</td>
<td></td>
<td></td>
<td>Unclassified</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5. Originator's Code (if known)</th>
<th>6. Originator (Corporate Author) Name and Location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Royal Signals and Radar Establishment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5a. Sponsoring Agency's Code (if known)</th>
<th>6a. Sponsoring Agency (Contract Authority) Name and Location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. Title</th>
<th>7a. Title in Foreign Language (In the case of translations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Axis Test Box Software Report</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7b. Presented at (for conference papers) Title, place and date of conference</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>8. Author 1 Surname, Initials</th>
<th>9(a) Author 2</th>
<th>9(b) Authors 3, 4...</th>
<th>10. Date</th>
<th>pp. ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simcock, A L</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 15. Distribution statement | |
|---------------------------|-
| Unlimited                 | |

Abstract

This memorandum describes in detail the software program which the M6809 microprocessor obeys to perform the functions of the AXIS Test Box.

Extensive use is made of Flow Diagram where these either enhance or replace wordy descriptions.