MC68HC11 Implementation of IEEE-488 Interface for DSP56000 Monitor

Prepared by: Richard Soja and Mark Maiolani, Motorola Semiconductors Ltd, East Kilbride, Scotland

This application note describes the implementation of an IEEE-488 (GPIB) interface to the Motorola DSP56000 Digital Signal Processor using an MC68HC11 single chip MCU. The original purpose of this interface was to permit the development of DSP56000 software and hardware on a Hewlett Packard HP9836 engineering workstation, which is furnished as standard with Hewlett Packard’s implementation of the IEEE-488 interface, called the HPIB.

In addition, the software was designed to permit GPIB access to the MCU’s Serial Communications Interface (SCI).

The hardware for the DSP to GPIB interface is composed of the MCU/Port replacement block, the DSP decoding logic and the GPIB handshake logic and I/O buffering. The block diagram of a typical system implementation is shown in figure 1.

To minimise the component count of the MC68HC11 circuit and, as the entire software occupies less than 2K bytes of memory, an MC68HC811A2 is used in the implementation. The result of this is to minimise external decoding and to remove the need for any external memory. (The MC68HC811A2 has 2K bytes of on-chip EEPROM plus 256 bytes of on-chip RAM for program execution and storage).

Figures 2 and 3 show the entire hardware circuit. The MC34064 LVI circuit shown in figure 2 ensures that the MCU is reset when the power supply voltage is below 4.5 Volts.

Fig. 1 Block Diagram of Typical System Interface

To simplify the hardware and software needed to access the DSP56000 host interface registers, the MC68HC811A2 is run in expanded mode, with code executed from internal EEPROM. The external bus is used to communicate with the DSP host port, which is configured by the DSP decoding logic to appear as a group of MC68HC11 external memory locations.

I/O ports B and C, which become the address and data bus of the MCU in expanded mode, are replicated by an MC68HC24 I/O port replacement device. These ports plus STRA and STRB are used to implement the GPIB interface. In particular, it is the powerful handshake features of PORTC, in conjunction with STRA and STRB which are extensively utilised in the implementation of the GPIB protocol.

Relocating and programming the MC68HC811A2’s internal memory.

Since the MC68HC811A2’s internal EEPROM can be remapped on any 4K byte boundary by modifying its CONFIG register, it is important to verify that the CONFIG register is set to $FF, to ensure correct mapping of the software. Details of how to do this can be found in the MC68HC811A2 data sheet and M68HC11EVM user manual. The software described in this application note does not include routines to reprogram the CONFIG register.

The EVM user manual also describes how to download software to the MC68HC811A2’s internal EEPROM.

GPIB Implementation.

An important point to note about this particular design is that, apart from the fully implemented handshake and 8 bit data bus, only two of the five GPIB control lines have been utilised. These are ATN and IFC. The remaining lines REN, EOI and SRQ are not needed and are therefore not implemented.

To simplify operation, the software was designed as a finite state machine, with the ATN and IFC lines providing interrupt sources to the MCU via two of its input capture pins - IC2 and IC3 on PORTA.

IFC was primarily included to assist in the debug of the development software. Assertion of this line causes the MCU to vector to the program segment <ABORT>, which forces the MCU’s GPIB interface to the idle state, i.e. all bus lines are released and the program re-initialises into a known state. Note the unusual method of clearing the IC2 flag, which avoids the use of an accumulator. (line 263 of the program listing)
Fig. 2 MCU/PRU Block and DSP Address Decoding.

Fig. 3 GPIB Handshake Logic and I/O Buffer.
Switch Representation

Fig 4 - Switch Representation

CMDSTAT Flags
(68HC11 address $00)

7 6 5 4 3 2 1 0
X X X X NSADS SPMS NTADS LADS

MCU States

SLACS SPAS TACS LACS

or STACS

Note:
Flag names correspond to states shown in Figures 8 and 9.
SLACS and STACS correspond to Secondary LACS and Secondary TACS.
The N preceding SADS and TADS indicates that the state is active when the bit is logic 0.

Fig 5 - CMDSTAT Flags and MCU States

GPIB Command processing
The logic level of the ATN control line dictates whether a command or data byte is available on the data bus during assertion of DAV. Changing the state of ATN always forces an interrupt to the MCU. This is serviced by routine <ATNSRV>, which polls ATN (PORTA,0) to determine whether commands or data bytes are to be processed.

While ATN is asserted, all bytes received are treated as commands, and processed by the program segment <CMDMODE>.

Once ATN is deasserted, the service routine <ATNSRV> is re-entered, and program flow is passed to the segment determined by the status of flags in variable CMDSTAT. Program vectoring is accomplished by first removing all registers stacked by the interrupt, then pushing the entry point of the required program segment back on the stack and then executing an RTS instruction.

Figure 5 shows the relationship between flags and program segments which would be entered on deassertion of ATN.

GPIB handshake
The most critical software and hardware feature of the MCU’s GPIB interface is the I/O switching and re-routing of the handshake lines.

In an attempt to optimise the hardware and software configuration of the GPIB interface, PORTC is used as the I/O data port, while STRA and STRB provide the appropriate handshake signals. However, as the GPIB is a 3-wire handshake, it is necessary to multiplex the GPIB’s RFD and DAC handshake lines on to either STRA (during source state), or on to STRB (during acceptor state).

The source state is used when the MCU is an active talker (i.e. outputting data), while the acceptor state is used when the MCU is an active listener or receiving commands (i.e. inputting data).

The remaining handshake line, DAV is also multiplexed between STRB (in source state) and STRA (in acceptor state). Figure 4 illustrates the basic operation of the handshake multiplexer, while figure 3 shows the hardware implementation, which was designed to guarantee that no transient states occur during multiplexing of the lines on the GPIB side.

Fig 6 - MC68HC11 Acceptor Handshake Sequence
Table 1 - State Mnemonics used in Application Note

<table>
<thead>
<tr>
<th>ACDS</th>
<th>LACS</th>
<th>LADS</th>
<th>LIDS</th>
<th>SIDS</th>
<th>SPAS</th>
<th>SPIS</th>
<th>SPMS</th>
<th>TACS</th>
<th>TADS</th>
<th>TIDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceptor Data State</td>
<td>Listener Active State</td>
<td>Listener Addressed State</td>
<td>Listener Idle State</td>
<td>Source Idle State</td>
<td>Serial Poll Active State</td>
<td>Serial Poll Idle State</td>
<td>Serial Poll Mode State</td>
<td>Talker Active State</td>
<td>Talker Addressed State</td>
<td>Talker Idle State</td>
</tr>
</tbody>
</table>
Figures 6 and 7 illustrate the relative timings of the handshake and control lines during source and acceptor states. Handshake states are controlled by PORTB,0 which also controls the direction of the 74LS245 I/O data buffer. PORTB,1 is used simply to disable the 74LS245 I/O data buffer when the interface is initialised. PORTB,3 is used to control which of RFD or DAC is connected to STRB in acceptor state, or STRA in source state.

Note that if the active listener state is entered from command state, there is no change in the handshake state. Acceptor to source handshake switch only occurs when changing from command state to talker state.

Listener and talker state diagrams are shown in figures 8 and 9.

Table 1 lists the meanings of the state mnemonics used in this application.

**Acceptor to source handshake state switch**

When switching from command state to primary or secondary TACS, the routine INITOP is first executed, on the deassertion of ATN. The secondary TACS state is referred to as STACS within this application note, though the mnemonic is not in the IEEE-488 specification.

The additional control line, provided by PORTB, 4 is used to deassert STRB by clamping STRA low, prior to switching the handshake line from acceptor to source state. This prevents DAV from being asserted when the handshake switch occurs. An unwanted side effect of this action is that the STAF flag in PIOC register is set by STRA changing state. To ensure that this flag is cleared before entering TACS or STACS, and to prevent STRB from being asserted (which would produce an erroneous assertion of RFD), the non-handshake mode for PORTC is enabled. The STAF clearing operation of reading the PIOC register followed by a dummy read from PORTCTL is then safely executed. (Lines 402-405 of the program listing).

For the duration of the above operations, the handshake lines are in the acceptor state. (i.e. RFD and DAC are outputs, DAV is an input). Once completed, and full handshake mode has been restored to PORTC (line 406), the handshake lines are switched to source state and the data buffer is enabled (line 407). The condition of the handshake lines is then: STRB connected to DAV; STRA connected to RFD. DAC is not needed at this stage, as it is the assertion of RFD which signals readiness of the active listener(s) to receive data from the MCU.

The STRA clamp is then released (line 410) and the data handshake sequence commences once the <SENDBYTE> routine is executed in TACS, STACS or SPAS.

A point to note within <SENDBYTE> is that interrupts are enabled to permit IFC or ATN to force the program to revert respectively to the idle state or acceptor state.

To initiate a valid data transfer, <SENDBYTE> will wait for assertion of RFD, which will set STAF in PIOC register (See Figure 7). Then, by writing to PORTCL, the data in ACCA is output on the data bus, with DAV asserted. The action of writing data to PORTCL (line 333) causes STAF to be cleared.

DAC (input) is then switched to STRA. Once it is asserted, STAF is again set and DAV is deasserted. STRA is then pulled low via PORTB,4 to ensure that subsequent assertion of RFD by the listener is not lost.

As previously described, STAF is then cleared without asserting STRB by selecting the non-handshake mode for PORTC. Before exiting the routine, full handshake mode is again selected and STRA is reconnected to RFD, and the PORTB,4 clamp is removed.

An important consequence of the above sequence of events is that, despite the apparent slowness of the software, the handshake protocol is always maintained correctly. For example, if RFD had been asserted prior to completion of the SENDBYTE routine, the assertion would be detected at the point that PORTB,4 clamp on STRA was removed. On later re-entry to <SENDBYTE>, a set STAF bit would be immediately detected, and normal execution would proceed as already described.

In this application <SENDBYTE> is only executed in TACS, STACS or SPAS.

**Source to acceptor handshake state switch**

This form of handshake switch will occur when TACS,STACS or SPAS is exited on the assertion of ATN, which in turn would occur when a change of MCU operating state is requested by the active controller (e.g. HP9836 work station). Before receiving and processing the command bytes necessary to incur the change in operating state, the MCU first switches the handshake lines from source to acceptor states and enables the I/O data buffer for input. This is performed by the routine INITOP, which also deasserts RFD by routing DAC to STRB. The hardware is designed to ensure that whichever of RFD or DAC is routed to STRB, the other line is held in the deasserted (low) state.

With RFD deasserted and the spuriously set STAF bit cleared by selecting non-handshake mode on PORTC, STRB is connected to RFD. The active states of STRA and STRB are then inverted, since the asserted states of RFD and DAC are the inverse of DAV.

The above sequence of events prepares the MCU's interface to accept data from the active controller, using the routine <READYBYTE>, where RFD is first asserted by performing a dummy read of PORTCL (See figure 6). The MCU now waits for DAV to be asserted by testing for a set STAF bit. When this occurs, RFD (STRB) is also deasserted by virtue of PORTCL handshake mode. The STRB line is now switched to DAC and STRA active edge sense is changed from negative to positive to permit detection of DAV deassertion. Following this action, STAF is cleared by a dummy read of PORTCL, which also causes DAC (STRB) to be re-asserted. This ultimately results in DAV being deasserted by the active talker, a condition which is detected by again testing STAF. In this case, it is the deassertion of DAV which causes deassertion of DAC (STRB) and the STAF bit to be set.

STRB is then clearly reconnected to RFD (line 325), STRA active edge is restored to negative sense, and the routine is exited with STAF still set. It is cleared by the next dummy load of PORTCL on re-entry to <READYBYTE>.

**GPIB-DSP Hardware Interface**

The DSP decoding logic performs the function of decoding the high address byte of the MC68HC11's external bus to enable up to four DSP host interfaces, by setting the corresponding Host Enable (HEN) lines low. By using only the high address byte, the need for demultiplexing the low byte, AD0-AD7, is avoided, thus
simplifying the decoding hardware. The hardware, shown in figure 2, consists of a 74LS138 three to eight line decoder, an 74LS05 inverter and an 74HC08 quad nand gate package. Address lines A11 - A15 are qualified with inverted E clock to provide the DSP enable signals HEN0 - HEN 3. Output O4 from the74LS138 provides a Multi DSP enable feature. When it is asserted, HEN0 - HEN3 are simultaneously asserted, thus providing synchronous access to all four DSPs. The outputs O0 - O3 are used to access each DSP independently.

Table 2 lists the valid addresses for all permutations of DSP enable.

The direction of data transfer between the interface and the DSP is controlled by the MC68HC11 R/W line, which directly drives the DSP host read/write (HR/W) line.

The DSP host interface is configured as a group of 8 x 8 bit registers. The MCU high address bits, A8-A10 are used to address these registers via the DSP’s HA0-HA2 lines. Table 3 summarises the functions of these registers.

The data to be transferred is available on the MC68HC11 multiplexed address/data lines AD0-AD7. No demultiplexing is necessary as the MCU's E clock is used by the DSP as a data valid signal.

**GPIB-DSP Software Interface**

The DSP software interface has been designed to support the development of up to four independent DSP56000s. In order to optimise data throughput, and to simplify the host computer's (e.g. HP9836) software drivers, the GPIB secondary addressing mode is used to select the required DSP.

**MCU High Address byte**  **DSP Enable**

<table>
<thead>
<tr>
<th>Address</th>
<th>Enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>01000XXX</td>
<td>HEN0</td>
</tr>
<tr>
<td>01001XXX</td>
<td>HEN1</td>
</tr>
<tr>
<td>01010XXX</td>
<td>HEN2</td>
</tr>
<tr>
<td>01011XXX</td>
<td>HEN3</td>
</tr>
<tr>
<td>01100XXX</td>
<td>HEN0-3</td>
</tr>
</tbody>
</table>

Bit 7 6 5 4 3 2 1 0

Bits0-2 specify the address of the required DSP host register.

**Table 2 - DSP Enable Addresses**

<table>
<thead>
<tr>
<th>HA2 HA1 H0</th>
<th>Register Function</th>
<th>Mnemonic</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>Interrupt Control Reg.</td>
<td>ICR</td>
<td>R/W</td>
</tr>
<tr>
<td>0 0 1</td>
<td>Command Vector Reg.</td>
<td>CVR</td>
<td>R/W</td>
</tr>
<tr>
<td>0 1 0</td>
<td>Interrupt Status Reg.</td>
<td>ISR</td>
<td>R only</td>
</tr>
<tr>
<td>0 1 1</td>
<td>Interrupt Vector Reg.</td>
<td>IVR</td>
<td>R/W</td>
</tr>
<tr>
<td>1 0 0</td>
<td>Not used</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>Receive data byte Regs.</td>
<td>RXH/TXH</td>
<td>R/W</td>
</tr>
<tr>
<td></td>
<td>(during Host reads)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>Transmit data byte Regs.</td>
<td>RXM/TXM</td>
<td>R/W</td>
</tr>
<tr>
<td></td>
<td>(during Host writes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td></td>
<td>RXL/TXL</td>
<td>R/W</td>
</tr>
</tbody>
</table>

**Table 3 - Summary of DSP Host Registers**
The interface’s GPIB primary address is not used in conjunction with the DSP interface, but instead is used to communicate with the MC68HC11’s SCI port. When writing to the DSP, the interface is in secondary master active state (SLACS), while data reads cause the secondary master active state (STACS) to be entered.

When making either data reads or writes, bits 3 and 4 of the 5-bit secondary address field are used to specify which DSP is selected. Bits 0-2 are only used during data reads to select which DSP register is addressed. Figure 10 shows the relationship between secondary addresses and DSP register selects.

An assembly listing of the MC68HC811A2 software is given at the end of this application note.

Writing to DSP host registers

When writing data to the DSP, the first data byte received from the host computer is the Mode Byte. It is used to determine the method of accessing the DSP’s registers. The Mode Byte is also used to select the Multi DSP mode, where all DSPs are enabled simultaneously.

The two write modes are: Normal - executed in program segment, 54, NORM
Auto - executed in program segment SLAUTO

In Normal mode, each DSP register is selected explicitly by an address byte which precedes the data for that register in the same way any number of registers can be addressed randomly. Note that, as the DSP’s TXDE flag is not tested before writing data in this mode, its primary use is to access the DSP’s control registers. Using Normal mode to write to the DSP’s data registers may result in an overrun condition.

The general format for GPIB data in Normal Write mode is:

Mode byte, regx addr, regx data, regy addr, regy data, etc.

The following example statement executed from the host computer, loads DSP 1, command vector register, with a value of hexadecimal 93.

OUTPUT 70008, CHR(11), CHR(170), CHR(3), CHR(11), CHR(147);

The AutoWrite mode is used to write data sequentially to register addresses 5, 6 and 7 of the selected DSP. Each consecutive 3 byte block received from the GPIB is repeatedly written to these 3 registers in turn until no more data is available. This mode is primarily used to provide a simple high speed download of program code from the host computer to the DSP. In this case the TXDE flag is tested before each data transfer to ensure no data overrun occurs.

To simplify the MC68HC11’s software, the Multi mode feature is not permitted in Auto Write mode.

The general format for GPIB data in Auto mode is:

Mode byte, regx data, regx data, regy data, regy data, etc.

The following host computer statement downloads hexadecimal A55AA55 to DSP number 2:

OUTPUT 70008,CHR(11),CHR(170),CHR(3),CHR(11),CHR(170),CHR(3),CHR(147);

The AutoWrite feature is particularly powerful when used with host computers which support array operators in output statements:

e.g. OUTPUT 70008 USING "W,C,CHR(11),Program18"
will download the entire contents of the data (e.g. user program) stored in string array <Program18>, to DSP number 1.
Fig 11 - <SLACS> Program Segment Flowchart

Reading from DSP host registers

GPIB secondary address bits 3 and 4 select the required DSP, while bits 0-2 select which register is to be read first.

Following reception of the secondary address and on entering STACS, each byte transferred from the selected DSP to the GPIB is read from consecutive DSP register addresses. After register 7 is read, subsequent reads continue repeatedly through registers 5,6 and 7, until the host computer’s input is satisfied. This is to simplify and speed up the process of reading blocks of words from the DSP’s data registers. The RXDF bit in the DSP status register is tested before reads of registers 5,6 and 7 to ensure the validity of data. Single byte reads can be used to test the other registers on the DSP interface. In this case, RXDF is not tested.

As the GPIB-DSP interface has been designed without an EOI control line or any explicit end of transfer character sequence, it is important that the host computer data read statement is formatted in some way to ensure proper termination of data transfer.

On the HP9836, this is achieved by incorporating the <USING> function within the ENTER statement. In addition, the variable into which data is read must be dimensioned appropriately.

Example:
DIM Program2(30)(3)
Reserve 30 x 3 byte words for data
ENTER 70005 USING #,A,AS

will upload 30 words of program code (or data) from DSP number 0, into array <Program2> on the host computer.

Figures 11 and 12 are flow charts of the DSP interface software segments. These correspond to secondary LACS and secondary TACS states of the GPIB.

Fig 12 - <STACS> Program Segment Flowchart

Reading from and writing to the MC68HC11’s SCI via the GPIB.

Typical syntax for reading from the SCI is:
ENTER 700 USING #,A,AS

Note that this statement will wait until data is received by the SCI. To abort the ENTER statement under program control, a timeout interrupt must be set up by the user.

Typical syntax for writing to the SCI is:
OUTPUT 700;"ABCDEF"

Note that in this implementation, there is no software or hardware handshake associated with the SCI, other than checking the SCI status register.
************ GPIBDSP.ASC  28/10/87 ************

* Interrupt driven GPIB Talker/Listener function

*******************************************************************************

* HARDWARE PROTOCOL : DSP ADDRESS DECODE HARDWARE

* The DSP interface uses the high address byte to enable the required
* DSP(s) and drive the DSP register select lines HA2, HA1 and HA0 as
* shown below:

* High address byte : 0,1,M,S1,S0,HA2,HA1,HA0

* M bit - Master Select, enables all DSPs simultaneously
* S1 + S0 - Select DSP to be enabled (overriden by M bit)
* HA2 + HA1 + HA0 - Select DSP internal address register to be accessed

* Note: The low address byte is not used by the interface and therefore
* does not need to be setup.

*******************************************************************************

22 A  0000  PORTA  EQU   0
23 A  0004  PORTB  EQU   4
24 A  0003  PORTC  EQU   3
25 A  0005  PORTCL EQU   5
26 A  0007  DDRC  EQU   7
27 A  0002  PIOC  EQU   2
28 A  000C  OC1M  EQU   $C
29 A  0021  TCTL2 EQU  $21
30 A  0022  TMSK1 EQU  $22
31 A  0023  TFLG1 EQU  $23
32 A  0024  TMSK2 EQU  $24
33 A  0028  SPCR  EQU  $28
34 A  0028  BAUD  EQU  $28
35 A  002C  SCCR1 EQU  $2C
36 A  002D  SCCR2 EQU  $2D
37 A  002E  SCSR  EQU  $2E
38 A  002F  SCDR  EQU  $2F
39 A  A200  DUART EQU $A200
40 A  *

41 A  0000  ORG   $0
42 A  0000  CMDSTAT RMB  1
43 A  0001  STATUS  RMB  1
44 A  0002  ADDR2  RMB  1
45 A  0003  CURRENT RMB  1
46 A  *

47 A  *

48 A  0002  TXDE  EQU  $02
49 A  0001  RXDF  EQU  $01
50 A  0018  SPE   EQU  $18
51 A  0019  SPD   EQU  $19
52 A  0020  LAG   EQU  $20
53 A  0040  TAG   EQU  $40
54 A  0060  SCG   EQU  $60
55 A  0020  MLA   EQU  LAG+0
56 A  0040  MTA   EQU  TAG+0
57 A  0060  MSAL  EQU  SCG+0

*******************************************************************************

DSP status bit mask (Transmit Data Empty)
  .. .. .. .. (Receive Data Full)
59 A 0067  NSAH  EQU  SCG+7
60 A 003F  UML  EQU  $3F
61 A 005F  UNT  EQU  $5F
62 A 0001  LADS  EQU  1
63 A 0002  NTADS  EQU  2
64 A 0004  SPMS  EQU  4
65 A 0008  NSADS  EQU  8
66 A 0008  NIFC  EQU  8
67 A 0004  PIFC  EQU  4
68 A 0001  PATN  EQU  1
69 A 0002  NATN  EQU  2
70 A 0000  EOS  EQU  0
71 A 00C4  JTIC3  EQU  SC4
72 A 00C7  JTIC2  EQU  SC7
73 A 00FF  USTACK  EQU  SFF
74 A 0080  TDRE  EQU  $80
75 A 0020  RDRF  EQU  $20
76 A *
77 A *
78 A *
79 A F800  ORG  $F800
80 A F800  MAIN  EQU  *
81 A F800  BEOFF  LDS  USTACK  Initialise stack
82 A F803  BD922  JSR  INIT
83 A F806  3E  IDLE  WAI  Wait for ATN or IFC.
84 A *
85 A F807  CHNMODE  EQU  *
86 A F807  BD967  JSR  READBYTE  Get command byte
87 A F80A  B47F  AMDA  #$7F  Remove MS Bit
88 A F80C  16  TAB
89 A F80D  C460  AMDB  #$60
90 A F80F  C160  CMPB  #$60  If SCG then
91 A F811  2742  BEQ  SECOND  process the address field
92 A F813  C140  CMPB  #$40  else
93 A F815  2611  BNE  CHD1  If talk address then
94 A F817  8140  CMPA  #$TA  If my talk address then
95 A F819  2608  BNE  OTA
96 A F81B  150003  BCLR  CHDSTAT,(LADS+NTADS)  idle listener (LADS=0)
97 A F81E  140008  BSET  CHDSTAT,(#NSADS)  enable talker (NTADS=0)
98 A F821  2064  BRA  CHNMODE  and clear sec. addr. mode
99 A F823  14000A  OTA  BSET  CHDSTAT,((NTADS+NSADS))  else idle talker
100 A F826  20DF  BRA  CHNMODE
101 A F828  8120  CHD1  CMPA  #$LA  else if my listen address then
102 A F82A  2605  BNE  CHD2  idle talker &
103 A F82C  14000B  BSET  CHDSTAT,(LADS+NTADS+NSADS)
104 A F82F  2066  BRA  CHNMODE  enable listener
105 A F831  813F  CMPA  #$ML  else if unlisten then
106 A F833  2605  BNE  CHD3
107 A F835  150001  BCLR  CHDSTAT,$LADS  idle listener
108 A F838  20CD  BRA  CHNMODE
109 A F83A  815F  CMPA  #$MT  else if Untalk then
110 A F83C  2605  BNE  CHD4  idle talker
111 A F83E  14000A  CHD3A  BSET  CHDSTAT,(#NTADS+NSADS)
112 A F841  20C4  BRA  CHNMODE
113 A F843  8118  CMPA  #$SP  idle talker
114 A F845  2605  BNE  CHD5
115 A F847  140004  BSET  CHDSTAT,#SPMS
116 A F84A  2088  BRA  CHNMODE
117 A F84C 8119  CMD5  CPMA  #SPD
118 A F84E 2603  BNE  CMD6
119 A F850 150004  BCLR  CMDSTAT,#SPWS
120 A F853 2082  CMD6  BRA  CMDMODE
121 A
122 A  * /SECCMD/ Prepares secondary command address to correct format
123 A  * and stores it for STACS or SLACS
124 A
125 A  F855
126 A F855 150008  BCLR  CMDSTAT,#NSADS  Enter Sec. address state
127 A F858 845F  ANDA  #85F  Mask off Master Enable bit
128 A F85A 9702  STAA  ADDR2  Store
129 A F85C 20A9  BRA  CMDMODE
130 A
131 A  F85E  LACS  EQU  *
132 A F86 BDF967  JSR  READBYTE
133 A F861 BDF953  JSR  OUTPUT
134 A F864 20F8  BRA  LACS
135 A
136 A  F866  TACS  EQU  *
137 A F86 BDF94A  JSR  INPUT
138 A F869 BDF984  JSR  SENDBYTE
139 A F86C 20F6  BRA  TACS

**************************************************************************************
140 A
141 A  * /STACS/ Data read from DSP and sent to GPIB
142 A  * On entry the secondary address mask is stored in ADDR2
143 A  * DSP ready testing is performed only if the register to be
144 A  * read is no. 5, 6 or 7
145 A  * Multiple bytes of data are read from successive registers until
146 A  * register 7 has been read when the next register becomes 5.
147 A
148 A  F86E  STACS  EQU  *
149 A F86E 9602  LDAA  ADDR2  Load in address mask
150 A F870 16  TAB
151 A F871 C407  ANDB  #807  Get register field
152 A F873 C105  CMPB  #805  Test B and branch if less than 5
153 A F875 250D  BLO  STNMISS  (i.e. 0, 1, 2, 3 or 4)
154 A F877 8458  ANDA  #858  Clear reg. address field
155 A F879 BAO2  ORAA  #802  Set address to DSP status register
156 A F87B 188F  XGDY  Load address into Y
157 A F87D 0E  STLOOP  CLI  Allow IFC and ATM interrupts
158 A F87E 181F0001FB  BRCLR  0,Y,#RXDF,  Wait till DSP ready (+delay)
159 A F883 0F  SEI
160 A F8B4 18DE02  STNMISS  LDY  ADDR2  Re-load original address into Y
161 A F887 18A600  LDDAA  ,Y  Read data from DSP
162 A F88A BDF984  JSR  SENDBYTE  Send to GPIB
163 A F8B0 9602  LDAA  ADDR2  Get current address mask
164 A F8BF 16  TAB
165 A F890 4C  INCA  + increment
166 A F891 9702  STAA  ADDR2  + store
167 A F893 8407  ANDA  #807  Get register no.
168 A F895 26D7  BNE  STACS  Loop if not last register
169 A F897 C458  ANDB  #858  Clear reg. field of address
170 A F899 CA05  ORAB  #805  and set to 5
171 A F89B D702  STAB  ADDR2  Update ADDR2
172 A F89D 20CF  BRA  STACS  ...and loop
173 A
174 A  * /SLACS/ Data read from GPIB and sent to DSP

**************************************************************************************

On entry the secondary address mask is stored in ADDR2

The operating modes is selected by the first byte sent from
the GPIB:

X X M X X X X A

M - Master select, when set all DSPs are written to simultaneously
A - Auto address, when set the GPIB data is automatically routed
to DSP registers $5,$6,$7,$5,$6,$7,$5,$5,$

When A is clear normal mode is selected, in which alternate bytes
read from the GPIB indicate the DSP target register address then
the data to be written.

SLACS EQU *

JSR READBYTE Get mode byte from GPIB
TAB
ANDB #$01 Test auto bit
BEQ SLNORM Go to normal mode if clear

 SLAUTO/ Auto address mode
 Multiple addressing inhibited
 DSP ready testing activated

LDAB ADDR2 Get address mask
ANDB #$58 Clear Reg. address field and M bit
STAB ADDR2 Replace
LDAA #$05
ORA ADDR2 Set first address to $05
XGDY ...

LDAA #$02 DSP 'ready' testing
LDAA ADDR2 Set address to DSP status register
XGDY Load address into Y

CLD Allow ATN and IFC ints.

JSR READBYTE Get a data byte from GPIB ...

PSHA ...

LDAA #$02 DSP 'ready' testing
LDAA ADDR2 Set address to DSP status register
XGDY Load address into Y

CLI Allow ATN and IFC ints.

BRCLR O,Y,#TXDE,* Wait till DSP ready (+delay)

SEI Retrieve data for DSP

PULY Retrieve DSP write address

STAA ,Y Write data to DSP

XGDY Put address into D for processing

INCA Set to next register

TAB Test for last register in cycle.

ANDB #$07 (mask off register field and set flags)

XGDY Address back in Y

Continue if not last reg.

SLA LAO Repeat whole cycle

 SLNORM/ - Normal data write mode
 Multiple addressing used if indicated in mode byte
 No DSP ready testing used

LDAB #$60 Prepare as if M bit is set
ANDA #$20 Test M bit of mode byte
BNE SLNORM Miss if M is set
LDAB ADDR2 Prepare for M bit clear
ANDB #$58 (Clear Reg. field and M bit)
233 A F8E2 D702 SLWMISS STAB ADDR2 Update ADDR2
234 A F8E4 BDF967 SLLOOP JSR READBYTE Get DSP reg. address field
235 A F8E7 8407 ANDA #807 Remove any extra bits
236 A F8E9 9A02 ORAA ADDR2 Incorporate into address mask
237 A F8EB 188F XGDY Put into Y
238 A F8ED BDF967 JSR READBYTE Get data
239 A F8F0 18A700 STAA ,Y Send to DSP
240 A F8F3 20EF BRA SLWLOOP Start again

241 A

242 A *

244 A F8F5 SPAS EQU *
245 A F8F5 9601 LDAA STATUS Bit 6 in STATUS must be set if service requested
246 A F8F7 BDF9B4 JSR SENDBYTE (=0A):ACCA is inverted by SENDBYTE
247 A F8FA 20F9 BRA SPAS

248 A *

249 A F8FC SENDSTRG EQU .*
250 A F8FC 18A600 SSTRG1 LDAA ,Y
251 A F8FF BDF9B4 JSR SENDBYTE
252 A F902 1808 INY
253 A F904 81F5 CMPA #$F5
254 A F906 26F4 BNE SSTRG1
255 A F908 39 RTS

256 A *

257 A F909 WAIT EQU *
258 A F909 1809 WAIT1 DEY
259 A F90B 26FC BNE WAIT1
260 A F90D 39 RTS

261 A *

262 A F90E 18CE000A ABORT LDY #10
tables 7FGL1,X,#$F5 Clear I/C 2 flag only
263 A F912 1D23FD BCLR TFLG1,X,#$F5
264 A F915 BDF2 BSR WAIT Wait to see if IFC glitch only
265 A F917 1F000206 BKLR PORTA,X,#$2,ABORTEX If IFC line still asserted then
266 A F918 BEO0FF LDS #STACK
267 A F91E 7EF800 JMP MAIN 1 reset stack &
268 A F921 38 ABORTEX RTI re-start,
269 A *

270 A F922 INIT EQU *
271 A F922 0F SEI Disable interrupts.
272 A F923 CE1000 LDX #$1000
273 A F926 860A LDAA #$ (NTADS+HSADS)
274 A F928 9700 STAA CHDSTAT Set talker & listener to idle states.
275 A F92A BDFA01 JSR IDLEBUS Release STRA,RFD & DAC (Also disables I/O buff)
276 A F92D A623 LDAA TFLG1,X
277 A F92F A723 STAA TFLG1,X Clear all flags and
278 A F931 8605 LDAA #$ (P1FC+PATH) enable interrupt on ATN assertion.
279 A F933 A721 STAA TCTL2,X
280 A F935 8603 LDAA #3
281 A F937 A722 STAA TNSK1,X
282 A F939 6F0C CLR OCM1,X
283 A F93B 6F24 CLR TNSK2,X
284 A F93D 6F28 CLR TNSK2,X
285 A F95F CC300C LDD #$300C
286 A F962 6FC2 CLR SCCR1,X Enable SCI Tx + Rx
287 A F964 E72D STAB SCCR2,X for 9600 baud
288 A F966 A728 STAA BAUD,X
289 A F968 0E CLI
290 A F969 39 RTS before returning.
291 A INPUT EQU *
292 A F94A OE CLI allow ATN and IFC interrupts
293 A F94A 0E BRCRLR SCSR,X,#RDFR,* Wait for data
294 A F94B 1F2E20FC SEI
295 A F94F 0F LDAA SDDR,X Read in char
296 A F950 62F RTS
297 A F952 39
298 A
299 A F953 OUTPUT EQU *
300 A F953 OE CLI enable IFC and ATN interrupts
301 A F954 1F2E80FC BRCRLR SCSR,X,#TDRE,*
302 A F958 0F SEI
303 A F959 A72F STA SDDR,X When ready, send char and return
304 A F958 39 RTS
305 A
306 A F95C OUTSTRG EQU *
307 A F95C A600 LDA X
308 A F95E 2706 BEQ STRGE
309 A F960 80F953 JSR OUTPUT
310 A F963 08 INX
311 A F964 20F6 BRA OUTSTRG
312 A F964 39 STRGE RTS
313 A
314 A F967 READBYTE EQU *
315 A F967 0E CLI Allow ATN, IFC or T/O interruptions.
316 A F968 01 NOP
317 A F969 E605 LDAB PORTC,X Make sure RFD is asserted.
318 A F968 1F0280FC NDAV BRCRLR PIOC,X,#80,NDAV Wait for DAV (when RFD de-asserted)
319 A F96F 1C0408 BSET PORTB,X,#8 Switch STRB (low) to DAC line.
320 A F972 1C0202 BSET PIOC,X,#2 Set up detection of DAV de-assertion.
321 A F975 0F SEI inhibit interruptions until data checked
322 A F976 A605 LDAA PORTC,X and invert data,
323 A F978 43 COMA and wait for DAV to go false.
324 A F979 1F0280FC DAV BRCRLR PIOC,X,#80,DAV Switch STRB (low) to RFD line
325 A F97D 100408 BCLR PORTB,X,#8 Reset DAV neg. detection, and return with data in ACCA.
326 A F980 100202 BCLR PIOC,X,#2
327 A F983 39 RTS
328 A
329 A F984 SENDBYTE EQU *
330 A F984 0E CLI
331 A F985 43 COMA
332 A F986 1F0280FC NRFD BRCRLR PIOC,X,#80,NRFD Wait for RFD assertion, then send data to listener, with DAV asserted.
333 A F986 A705 STA PORTC,X
334 A F98C 1C0408 BSET PORTB,X,#8 Switch DAC to STRA,
335 A F98F 1F0280FC NDAV BRCRLR PIOC,X,#80,NDAV and wait for DAC assertion.
336 A F993 1C0410 BSET PORTB,X,#8 Pull STRA low.
337 A F996 100210 BCLR PIOC,X,#10 Select non-handshake mode
338 A F996 E605 LDAB PORTC,X to clear STAF without re-asserting DAV
339 A F998 1C0210 BSET PIOC,X,#10 Re-select handshake mode.
340 A F999 1C0408 BCLR PORTB,X,#8 Reconnect STRA to RFD,
341 A F9A1 100410 BCLR PORTB,X,#10 and release STRA before returning with STAF set if RFD asserted
342 A F9A4 39 RTS
343 A
344 A F9A5 ATNSRV EQU *
345 A F9A5 1F000110 BRCRLR PORTA,X,#1,DATAMODE If ATN asserted then
346 A F9A7 060A LDAB #5A ! make sure RFD is de-asserted
347 A F9B4 E704 STAB PORTB,X ! until all registers are configured.
348 A F9AD 805C BSR INITIP ! Set up input handshake mode

MOTOROLA
ANE415
349 A F9A6 C606 LDAB #((PIFC+MATN) ! enable interrupt on
350 A F9B1 E721 STAB TCTL2,X ! deassertion of ATN,
351 A F9B3 18CEF807 LDY #CHMODXDE ! Get required return address
352 A F9B7 203E BRA SRVEXIT ! and go there.
353 A F9BA C605 DATAMODE LDAB #((PIFC+PATH) else enable interrupt on
354 A F9BB E721 STAB TCTL2,X ! assertion of ATN
355 A F9BD 13000110 BRCLR CMDSSTAT,##LADS,TALKER ! If listener then
356 A F9C1 13000806 BRCLR CMDSSTAT,##SADS,LISTEN2 ! If not sec. listener then
357 A F9C5 18CEF80E LDY #$LACS ! enter LACS state,
358 A F9C9 202C BRA SRVEXIT
359 A F9CB 18CEF8FF LISTEN2 LDY #$SLACS ! else enter SLACS state.
360 A F9CF 2026 BRA SRVEXIT
361 A
362 A F9D1 1200021C TALKER BRSET CMDSSTAT,##NTADS,NOADDR ! else if talker then
363 A F9D5 8D48 BSR INITOP ! enable buffer for output
364 A F9D7 12000410 BRSET CMDSSTAT,##SPMS,POLLO ! if not serial poll mode then
365 A F9D8 13000806 BRCLR CMDSSTAT,##SADS,TALKER2 ! if secondary talker not
366 A F9DF 18CEF866 LDY #$TACS ! addressed, then enter
367 A F9E3 2012 BRA SRVEXIT primary TACS
368 A F9E5 18CEF86E TALKER2 LDY #$STACS !
369 A F9E9 200C BRA SRVEXIT enter secondary TACS
370 A F9EB 18CEF8F5 SPOLL LDY #$SAS ! else activate serial poll
371 A F9F0 2006 BRA SRVEXIT
372 A F9F1 8D06 NOADDR BSR IDLEBUS !
373 A F9F3 18CEF806 LDY #$IDLE !
374 A F9F7 1023FE SRVEXIT BCLR TFLG1,X,#8FE Clear interrupt flag generated by ATN
375 A F9FA 8E00FF LDS $USTACK Reset user stack
376 A F9FD 183C PSHY
377 A F9FF 0E CLI
378 A FA00 39 RTS ! and return.
379 A
380 A FA01 IDLEBUS EQU * Select handshake mode, to prevent STRB pulses
381 A FA01 CC1211 LDD #$1211 when writing to PORTB. Select +ive STRB.
382 A FA04 E702 STAB PIOC,X Disable bus driver, and de-assert STRB.
383 A FA06 A704 STAA PORTB,X Now release RFD & DAC, by selecting talker mode
384 A FA08 6C04 INC PORTB,X
385 A FA0A 39 RTS
386 A
387 A FA0B INITIP EQU * Set PortC to input.
388 A FA0B 6F07 CLR DDRC,X Connect DAC to STRB,select input buffer mode.
389 A FA0D 10407 BCLR PORTB,X,#7 Select -ive STRA, -ive STRB,
390 A FA10 C610 LDAB #$10 full i/p handshake mode.
391 A FA12 E702 STAB PIOC,X
392 A FA14 E602 LDAB PIOC,X
393 A FA16 1C0410 BSET PORTB,X,#10 Make sure STRB is de-asserted
394 A FA19 10404 BCLR PORTB,X,#10 by toggling STRA
395 A FA1C E602 LDAB PIOC,X Do dummy read, to prepare for STAF clearing,
396 A FA1E 10408 BCLR PORTB,X,#8 before connecting RFD to STRB (low).
397 A FA21 39 RTS
398 A
399 A FA22 INITIP EQU * Entered with data & handshake lines in Listener mode
400 A FA22 1040A BCLR PORTB,X,#50A Route RFD to STRA.
401 A FA25 1C0410 BSET PORTB,X,#50A Pull STRA low, to deassert STRB.
402 A FA28 C608 LDAB #$50B Select non-handshake mode, +ive STRA, +ive STRB
403 A FA2A E702 STAB PIOC,X
404 A FA2C E602 LDAB PIOC,X then read PIOC & PORTC in turn,
405 A FA2E E605 LDAB PORTC,X to clear STAF without asserting STRB.
406 A FA30 1C0210 BSET PIOC,X,#10 Now select full output handshake mode.
407 A FA33 1DA405   BSET   PORTB,X,#5  Connect RFD to STRA,STRB to DAV, enable o/p buffer
408 A FA36 C6FF   LDAB   #$FF
409 A FA3E E707   STAB   DDRC,X
410 A FA3A 10410   BCLR   PORTB,X,#$10
411 A FA30 39   RTS
412 A *
413 A FA3E   NULLSRV EQU   *
414 A FA3E 38   RTI
415 A *
416 A FA3F   ILLOPSRV EQU   *
417 A FA3F CF   STOP
418 A FA60 7EFA3F   JMP   ILLOPSRV
419 A *
420 A *
421 A *
422 A FFEA   ORG   $FFEA
423 A FFEA F9A5   FDB   ATNSRV  I/P capture 3
424 A FFEF F90E   FDB   ABORT  I/P capture 2
425 A FFEF FA3E   FDB   NULLSRV  I/P capture 1
426 A FFF0 FA3E   FDB   NULLSRV  Real time interrupt
427 A FFF2 FA3E   FDB   NULLSRV  IRQ
428 A FFF4 FA3E   FDB   NULLSRV  XIRQ
429 A FFF6 FA3E   FDB   NULLSRV  SWI
430 A FFF8 FA3F   FDB   ILLOPSRV  Illegal opcode trap
431 A FFFA FA3E   FDB   NULLSRV  COP timeout
432 A FFFC FA3E   FDB   NULLSRV  Clock monitor failure
433 A FFFE F800   FDB   MAIN
434 A *
435 A    END

This service routine should never be executed, as all unused interrupts are disabled.
Stop processor if there’s a problem with code. Just in case. (Should never be executed).
<table>
<thead>
<tr>
<th>ABORT</th>
<th>F90E</th>
<th>PIFC</th>
<th>0004</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABORTEX</td>
<td>F921</td>
<td>PIOC</td>
<td>0002</td>
</tr>
<tr>
<td>ADDR2</td>
<td>0002</td>
<td>PORTA</td>
<td>0000</td>
</tr>
<tr>
<td>ATNSRV</td>
<td>F9A5</td>
<td>PORT8</td>
<td>0004</td>
</tr>
<tr>
<td>BAUD</td>
<td>0028</td>
<td>PORTC</td>
<td>0003</td>
</tr>
<tr>
<td>CMD1</td>
<td>F828</td>
<td>PORTCL</td>
<td>0005</td>
</tr>
<tr>
<td>CMD2</td>
<td>F831</td>
<td>RDRF</td>
<td>0020</td>
</tr>
<tr>
<td>CMD3</td>
<td>F83A</td>
<td>READBYTE</td>
<td>F967</td>
</tr>
<tr>
<td>CMD3A</td>
<td>F83E</td>
<td>RXDF</td>
<td>0001</td>
</tr>
<tr>
<td>CMD4</td>
<td>F843</td>
<td>SCCR1</td>
<td>002C</td>
</tr>
<tr>
<td>CMD5</td>
<td>F84C</td>
<td>SCCR2</td>
<td>002D</td>
</tr>
<tr>
<td>CMD6</td>
<td>F853</td>
<td>SCDR</td>
<td>002F</td>
</tr>
<tr>
<td>CMDMODE</td>
<td>F807</td>
<td>SCG</td>
<td>0060</td>
</tr>
<tr>
<td>CMDSTAT</td>
<td>0000</td>
<td>SCSR</td>
<td>002E</td>
</tr>
<tr>
<td>CURRENT</td>
<td>0003</td>
<td>SECMD</td>
<td>F855</td>
</tr>
<tr>
<td>DATAMODE</td>
<td>F9B9</td>
<td>SENDBYTE</td>
<td>F984</td>
</tr>
<tr>
<td>DAV</td>
<td>F979</td>
<td>SENDSTRG</td>
<td>F8FC</td>
</tr>
<tr>
<td>DORC</td>
<td>0007</td>
<td>SLACS+</td>
<td>F89F</td>
</tr>
<tr>
<td>DUART</td>
<td>A200</td>
<td>SLAUTO</td>
<td>F8AD</td>
</tr>
<tr>
<td>EOS</td>
<td>0000</td>
<td>SLOOP</td>
<td>F883</td>
</tr>
<tr>
<td>IDLE</td>
<td>F806</td>
<td>SLNLOOP</td>
<td>F8E4</td>
</tr>
<tr>
<td>IDLEBUS</td>
<td>FA01</td>
<td>SLNMISS</td>
<td>F8E2</td>
</tr>
<tr>
<td>ILLOPSRV</td>
<td>FA3F</td>
<td>SLNORM</td>
<td>F808</td>
</tr>
<tr>
<td>INIT</td>
<td>F922</td>
<td>SLTL</td>
<td>F88F</td>
</tr>
<tr>
<td>INITIP</td>
<td>FA08</td>
<td>SPAS</td>
<td>F8F5</td>
</tr>
<tr>
<td>INITOP</td>
<td>FA22</td>
<td>SPCR</td>
<td>0028</td>
</tr>
<tr>
<td>INPUT</td>
<td>F94A</td>
<td>SPD</td>
<td>0019</td>
</tr>
<tr>
<td>JTIC2</td>
<td>00C7</td>
<td>SPE</td>
<td>0018</td>
</tr>
<tr>
<td>JTIC3</td>
<td>00C4</td>
<td>SPMS</td>
<td>0004</td>
</tr>
<tr>
<td>LACS</td>
<td>F85E</td>
<td>SPOLL</td>
<td>F9EB</td>
</tr>
<tr>
<td>LADS</td>
<td>0001</td>
<td>SRVEXIT</td>
<td>F9F7</td>
</tr>
<tr>
<td>LAG</td>
<td>0020</td>
<td>SSTRG1</td>
<td>F8FC</td>
</tr>
<tr>
<td>LISTEN2</td>
<td>F9C8</td>
<td>STACS</td>
<td>F86E</td>
</tr>
<tr>
<td>MAIN</td>
<td>F800</td>
<td>STATUS</td>
<td>0001</td>
</tr>
<tr>
<td>MLA</td>
<td>0020</td>
<td>STLOOP</td>
<td>F87D</td>
</tr>
<tr>
<td>MSAH</td>
<td>0067</td>
<td>STMISS</td>
<td>F884</td>
</tr>
<tr>
<td>MSAL</td>
<td>0060</td>
<td>STRGEX</td>
<td>F966</td>
</tr>
<tr>
<td>MTA</td>
<td>0040</td>
<td>TACS</td>
<td>F866</td>
</tr>
<tr>
<td>NATN</td>
<td>0002</td>
<td>TAG</td>
<td>0040</td>
</tr>
<tr>
<td>NDAC</td>
<td>F9BF</td>
<td>TALKER</td>
<td>F9D1</td>
</tr>
<tr>
<td>NDAV</td>
<td>F968</td>
<td>TALKER2</td>
<td>F9E5</td>
</tr>
<tr>
<td>NIFC</td>
<td>0008</td>
<td>TCTL2</td>
<td>0021</td>
</tr>
<tr>
<td>NOADDRD</td>
<td>F9F1</td>
<td>TDRE</td>
<td>0080</td>
</tr>
<tr>
<td>NRFD</td>
<td>F9B6</td>
<td>TFLG1</td>
<td>0023</td>
</tr>
<tr>
<td>NSADS</td>
<td>0008</td>
<td>TMSK1</td>
<td>0022</td>
</tr>
<tr>
<td>NTADS</td>
<td>0002</td>
<td>TMSK2</td>
<td>0024</td>
</tr>
<tr>
<td>NULSRV</td>
<td>FA3E</td>
<td>TXDE</td>
<td>0002</td>
</tr>
<tr>
<td>OCIN</td>
<td>000C</td>
<td>UNIL</td>
<td>003F</td>
</tr>
<tr>
<td>OTA</td>
<td>F823</td>
<td>UNT</td>
<td>005F</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>F953</td>
<td>USTACK</td>
<td>00FF</td>
</tr>
<tr>
<td>OUTSTRG</td>
<td>F95C</td>
<td>WAIT</td>
<td>F909</td>
</tr>
<tr>
<td>PATN</td>
<td>0001</td>
<td>WAIT1</td>
<td>F909</td>
</tr>
</tbody>
</table>