

# 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.

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)

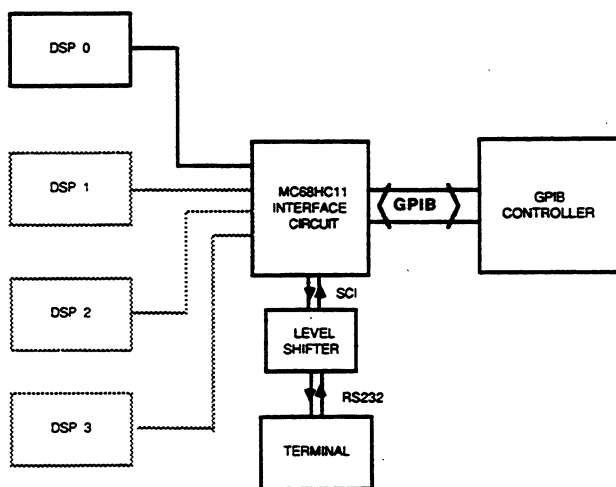


Fig. 1 Block Diagram of Typical System Interface

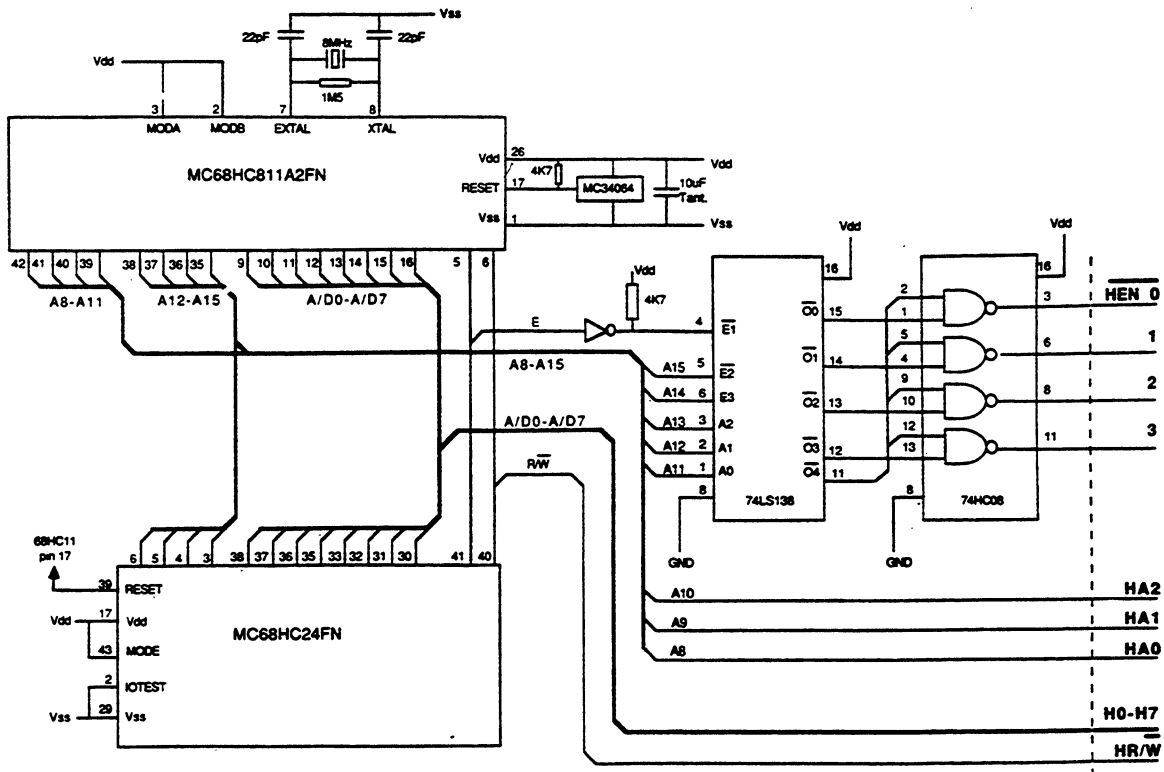


Fig. 2 MCU/PRU Block and DSP Address Decoding.

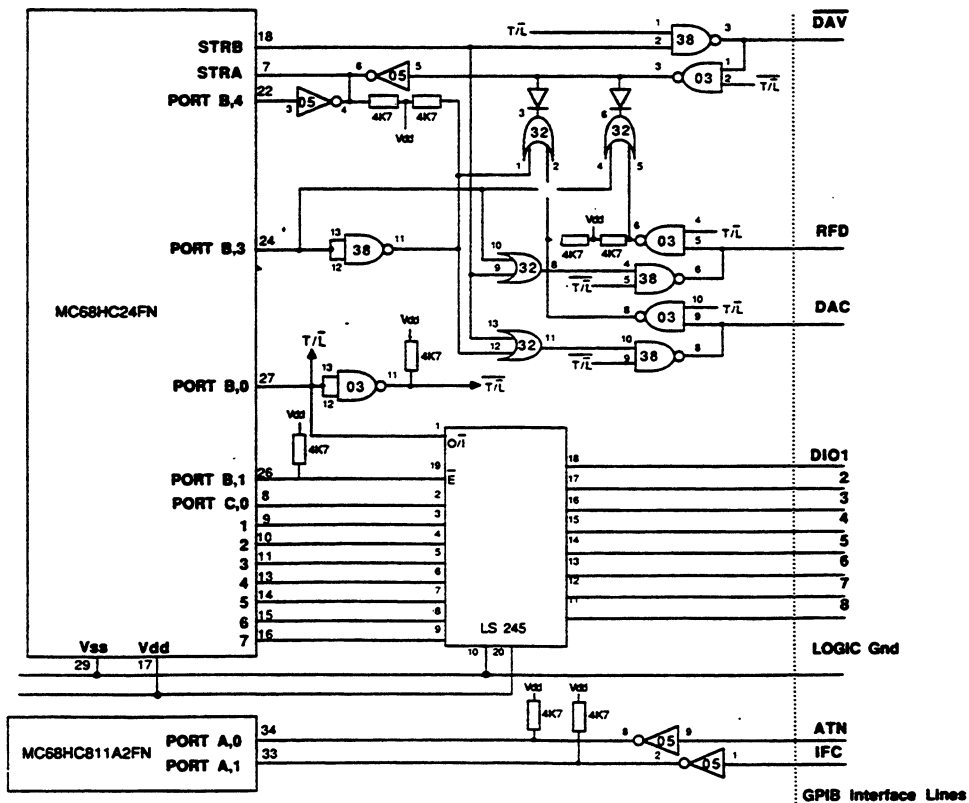


Fig. 3 GPIB Handshake Logic and I/O Buffer.

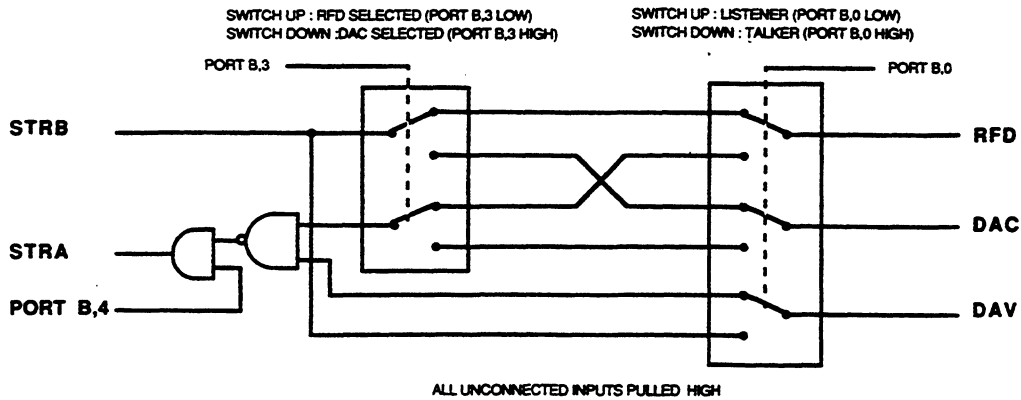


Fig 4 - Switch Representation

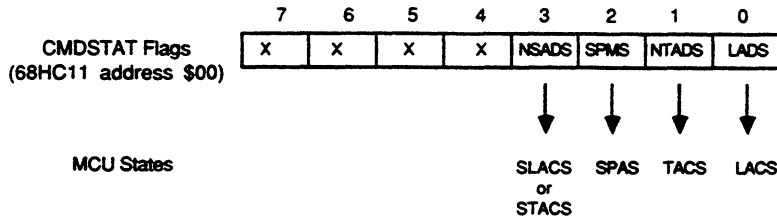


Fig 5 - CMDSTAT Flags and MCU States

**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.

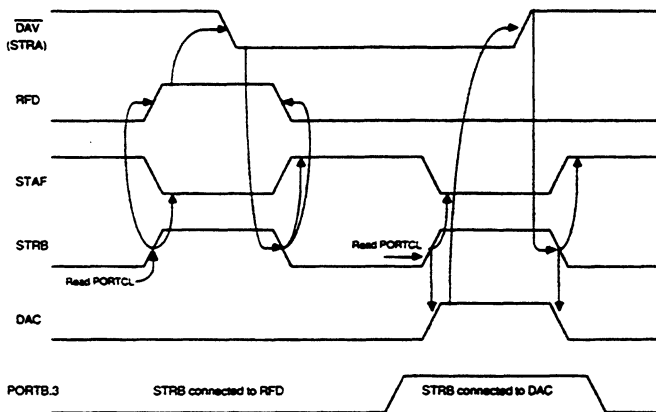


Fig 6 - MC68HC11 Acceptor Handshake Sequence

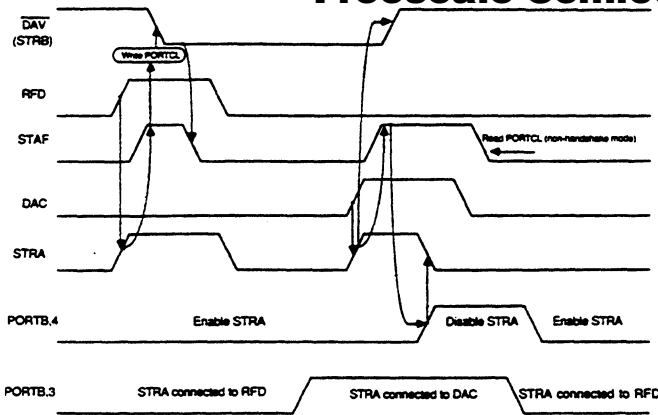


Fig 7 - MC68HC11 Source Handshake Sequence

ACDS	Acceptor Data State
LACS	Listener Active State
LADS	Listener Addressed State
LIDS	Listener Idle State
SIDS	Source Idle State
SPAS	Serial Poll Active State
SPIS	Serial Poll Idle State
SPMS	Serial Poll Mode State
TACS	Talker Active State
TADS	Talker Addressed State
TIDS	Talker Idle State

Table 1 - State Mnemonics used in Application Note

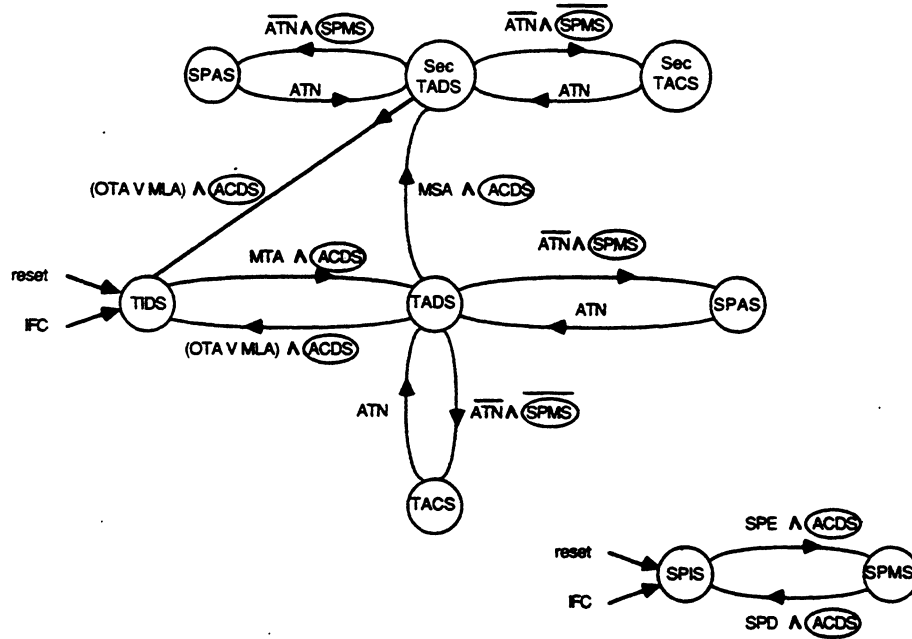


Fig 8 - Primary and Secondary Talker, Serial Poll State Diagram

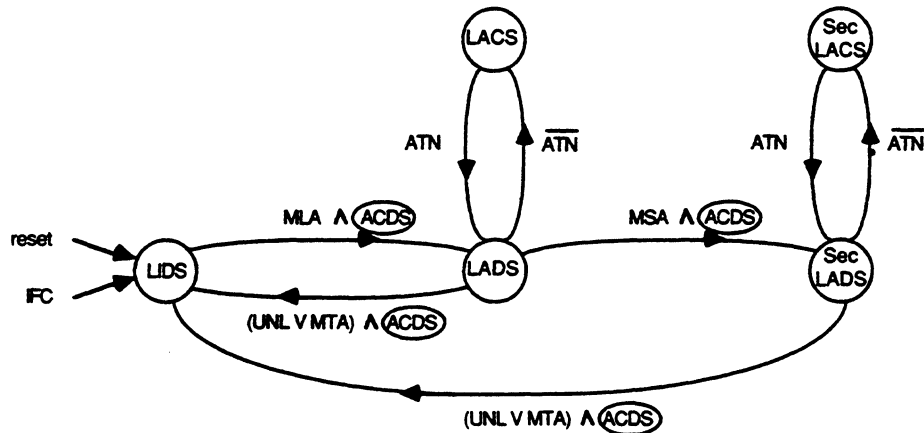


Fig 9 - Primary & Secondary Listener State Diagram

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 PORTCL 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 INITIP, 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 <READBYTE>, 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 cleanly 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 <READBYTE>.

### 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 the 74LS138 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
0 1 0 0 0 X X X	HEN0
0 1 0 0 1 X X X	HEN1
0 1 0 1 0 X X X	HEN2
0 1 0 1 1 X X X	HEN3
0 1 1 0 0 X X X	HEN0-3
Bit 7 6 5 4 3 2 1 0	

Bits 0-2 specify the address of the required DSP host register.

**Table 2 - DSP Enable Addresses**

HA2	HA1	HA0	Register Function	Mnemonic	Access
0	0	0	Interrupt Control Reg.	ICR	R/W
0	0	1	Command Vector Reg.	CVR	R/W
0	1	0	Interrupt Status Reg.	ISR	R only
0	1	1	Interrupt Vector Reg.	IVR	R/W
1	0	0	Not used		
1	0	1	Receive data byte Regs. (during Host reads) or Transmit data byte Regs. (during Host writes)	RXH/TXH	R/W
1	1	0		RXM/TXM	R/W
1	1	1		RXL/TXL	R/W

**Table 3 - Summary of DSP Host Registers**

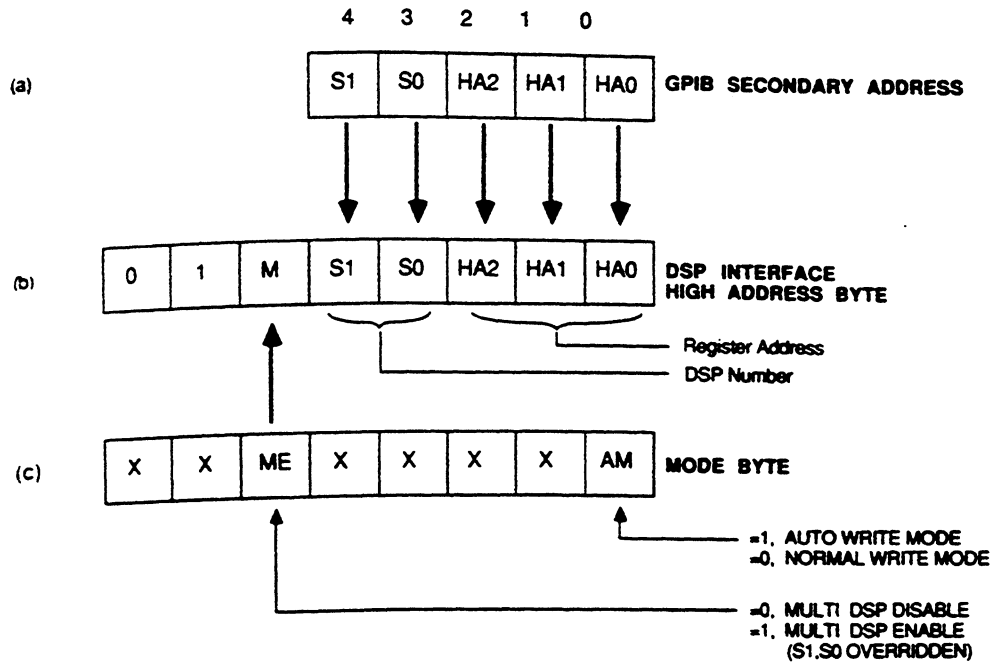


Fig 10

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 listener active state (SLACS), while data reads cause the secondary talker active state (STACS) to be entered.

When initiating 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(a) 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. (See figure 10(c))

The two write modes are: Normal - executed in program segment <SLNORM>

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 this way any number of registers can be accessed 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,CHRS(0);CHRS(1);CHRS(147);
```

The Auto Write 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, reg5 data, reg6 data, reg7 data, reg5 data, reg6 data, etc...

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

```
OUTPUT 70016,CHRS(1);CHRS(170);CHRS(85);CHRS(170);CHRS(85);
```

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

```
e.g. OUTPUT 70008 USING *B,C,CHRS(1);Program1$*
```

will download the entire contents of the data (e.g. user program) stored in string array <Program1\$>, to DSP number 1.

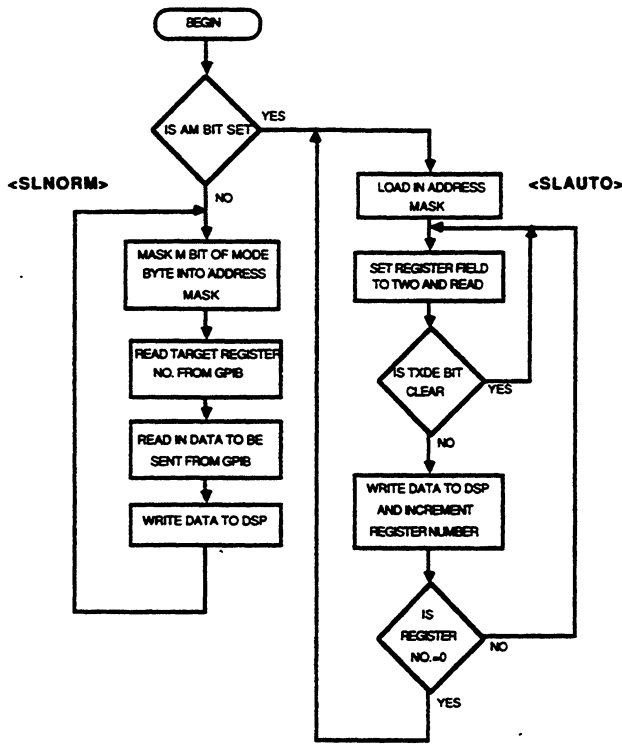


Fig 11 - <SLACS> Program Segment Flowchart

**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.

```
e.g DIM Program2$(30)[3] ! Reserve 30 x 3 byte words for data
    ENTER 70005 USING "#,3A";Program2$(*)
```

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.

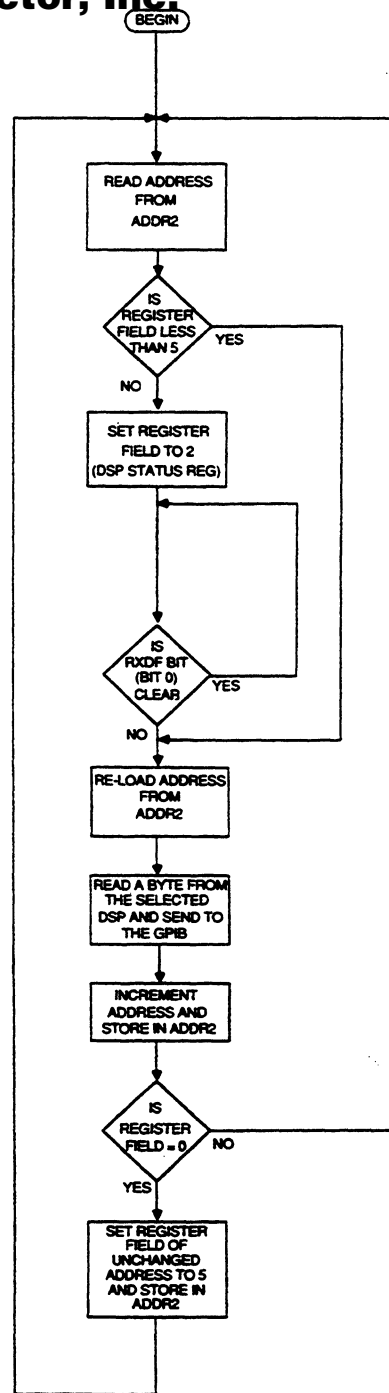


Fig 12 - <STACS> Program Segment Flowchart

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

Typical syntax for reading from the SCI is:

```
ENTER 700 USING "#,A";A$
```

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.





```

1 A          ***** GPIBDSP.ASC  28/10/87 *****
2 A          * Interrupt driven GPIB Talker/Listener function
3 A          *
4 A          *****
5 A          *
6 A          *          HARDWARE PROTOCOL : DSP ADDRESS DECODE HARDWARE
7 A          *
8 A          *          The DSP interface uses the high address byte to enable the required
9 A          *          DSP(s) and drive the DSP register select lines HA2,HA1 and HA0 as
10 A         *          shown below:
11 A         *
12 A         *          High address byte : 0,1,M,S1,S0,HA2,HA1,HA0
13 A         *
14 A         *          M bit - Master Select, enables all DSPs simultaneously
15 A         *          S1 + S0 - Select DSP to be enabled (overriden by M bit)
16 A         *          HA2 + HA1 + HA0 - Select DSP internal address register to be accessed
17 A         *
18 A         *          Note: The low address byte is not used by the interface and therefore
19 A         *          does not need to be setup.
20 A         *
21 A         *****
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         002B   BAUD    EQU    $2B
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         * Workspace
42 A         0000   ORG     $0
43 P 0000 0001   CMDSTAT  RMB    1
44 P 0001 0001   STATUS   RMB    1
45 P 0002 0001   ADDR2    RMB    1
46 P 0003 0001   CURRENT  RMB    1
47 A         *
48 A         * Constants
49 A         0002   TXDE    EQU    $02          DSP status bit mask (Transmit Data Empty)
50 A         0001   RXDF    EQU    $01          " " " " (Receive Data Full)
51 A         0018   SPE     EQU    $18
52 A         0019   SPD     EQU    $19
53 A         0020   LAG     EQU    $20
54 A         0040   TAG     EQU    $40
55 A         0060   SCG     EQU    $60
56 A         0020   MLA     EQU    LAG+0
57 A         0040   MTA     EQU    TAG+0
58 A         0060   MSAL    EQU    SCG+0

```

Freescale Semiconductor, Inc.

59 A	0067	MSAH	EQU	SCG+7	
60 A	003F	UNL	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	\$C4	
72 A	00C7	JTIC2	EQU	\$C7	
73 A	00FF	USTACK	EQU	\$FF	
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 8E00FF		LDS	#USTACK	Initialise stack
82 A	F803 BDF922		JSR	INIT	
83 A	F806 3E	IDLE	WAI		Wait for ATM or IFC.
84 A		*			
85 A	F807	CMDMODE	EQU	*	
86 A	F807 BDF967		JSR	READBYTE	Get command byte
87 A	F80A 847F		ANDA	#\$7F	Remove MS Bit
88 A	F80C 16		TAB		
89 A	F80D C460		ANDB	#\$60	
90 A	F80F C160		CMPB	#\$60	If SCG then
91 A	F811 2742		BEQ	SECCMD	process the address field
92 A	F813 C140		CMPB	#\$40	else
93 A	F815 2611		BNE	CMD1	! If talk address then
94 A	F817 8140		CMPA	#MTA	! If my talk address then
95 A	F819 2608		BNE	OTA	
96 A	F81B 150003		BCLR	CMDSTAT,#(LADS+NTADS)	! idle listener (LADS=0)
97 A	F81E 140008		BSET	CMDSTAT,#NSADS	! enable talker (NTADS=0)
98 A	F821 20E4		BRA	CMDMODE	! and clear sec. addr. mode
99 A	F823 14000A	OTA	BSET	CMDSTAT,#(NTADS+NSADS)	! else idle talker
100 A	F826 200F		BRA	CMDMODE	
101 A	F828 8120	CMD1	CMPA	#HLA	! else If my listen address then
102 A	F82A 2605		BNE	CMD2	! idle talker &
103 A	F82C 14000B		BSET	CMDSTAT,#(LADS+NTADS+NSADS)	
104 A	F82F 2006		BRA	CMDMODE	! enable listener.
105 A	F831 813F	CMD2	CMPA	#UNL	! else If unlisten then
106 A	F833 2605		BNE	CMD3	
107 A	F835 150001		BCLR	CMDSTAT,#LADS	! idle listener
108 A	F838 20CD		BRA	CMDMODE	
109 A	F83A 815F	CMD3	CMPA	#UNT	! else If Untalk then
110 A	F83C 2605		BNE	CMD4	! idle talker
111 A	F83E 14000A	CMD3A	BSET	CMDSTAT,#(NTADS+NSADS)	
112 A	F841 20C4		BRA	CMDMODE	
113 A	F843 8118	CMD4	CMPA	#SPE	
114 A	F845 2605		BNE	CMD5	
115 A	F847 140004		BSET	CMDSTAT,#SPMS	
116 A	F84A 20BB		BRA	CMDMODE	

117 A F84C 8119  
 118 A F84E 2603  
 119 A F850 150004  
 120 A F853 2082  
 121 A  
 122 A  
 123 A  
 124 A  
 125 A F855  
 126 A F855 150008  
 127 A F858 845F  
 128 A F85A 9702  
 129 A F85C 20A9  
 130 A  
 131 A F85E  
 132 A F85E BDF967  
 133 A F861 BDF953  
 134 A F864 20F8  
 135 A  
 136 A F866  
 137 A F866 BDF94A  
 138 A F869 BDF984  
 139 A F86C 20F  
 140 A  
 141 A  
 142 A  
 143 A  
 144 A  
 145 A  
 146 A  
 147 A  
 148 A F86E  
 149 A F86E 9602  
 150 A F870 16  
 151 A F871 C407  
 152 A F873 C105  
 153 A F875 2500  
 154 A F877 8458  
 155 A F879 8A02  
 156 A F87B 188F  
 157 A F87D 0E  
 158 A F87E 181F0001FB  
 159 A F883 0F  
 160 A F884 18DE02  
 161 A F887 18A600  
 162 A F88A BDF984  
 163 A F88D 9602  
 164 A F88F 16  
 165 A F890 4C  
 166 A F891 9702  
 167 A F893 8407  
 168 A F895 2607  
 169 A F897 C458  
 170 A F899 CA05  
 171 A F89B D702  
 172 A F89D 20CF  
 173 A  
 174 A

```

CMD5      CMPA      #SPD
          BNE       CMD6
          BCLR      CMDSTAT,#SPMS
CMD6      BRA       CMDMODE
*****
*         /SECCMD/ Prepares secondary command address to correct format
*         and stores it for STACS or SLACS
*****
SECCMD    EQU       *
          BCLR      CMDSTAT,#NSADS      Enter Sec. address state
          ANDA      #$5F                Mask off Master Enable bit
          STAA      ADDR2               Store
          BRA       CMDMODE
*
LACS      EQU       *
          JSR       READBYTE
          JSR       OUTPUT
          BRA       LACS
*
TACS      EQU       *
          JSR       INPUT
          JSR       SENDBYTE
          BRA       TACS
*****
*         /STACS/ Data read from DSP and sent to GPIB
*         On entry the secondary address mask is stored in ADDR2
*         DSP ready testing is performed only if the register to be
*         read is no. 5,6 or 7
*         Multiple bytes of data are read from successive registers until
*         register 7 has been read when the next register becomes 5.
*****
STACS     EQU       *
          LDAA      ADDR2               Load in address mask
          TAB
          ANDB      #$07                Get register field
          CMPB      #$05                Test B and branch if less than 5
          BLO       STMIS5              (i.e. 0,1,2,3 or 4)
          ANDA      #$58                Clear reg. address field
          ORAA      #$02                Set address to DSP status register
          XGDY
          STLOOP   CLI                  Allow IFC and ATN interrupts
          BRCLR     0,Y,#RXDF,*         Wait till DSP ready (+delay)
          SEI
          STMIS5  LDY       ADDR2        Re-load original address into Y
          LDAA      ,Y                  Read data from DSP
          JSR       SENDBYTE            Send to GPIB
          LDAA      ADDR2               Get current address mask
          TAB
          INCA
          STAA      ADDR2               + increment
          ANDA      #$07                + store
          BNE      STACS                Get register no.
          ANDB      #$58                Loop if not last register
          ORAB      #$05                Clear reg. field of address
          STAB      ADDR2               and set to 5
          BRA       STACS               Update ADDR2
          BRA       STACS               ..and loop
*****
*         /SLACS/ Data read from GPIB and sent to DSP
  
```

```

175 A
176 A
177 A
178 A
179 A
180 A
181 A
182 A
183 A
184 A
185 A
186 A
187 A      F89F
188 A F89F BDF967
189 A F8A2 16
190 A F8A3 C401
191 A F8A5 2731
192 A
193 A
194 A
195 A
196 A
197 A F8A7 D602
198 A F8A9 C458
199 A F8AB D702
200 A F8AD 8605
201 A F8AF 9A02
202 A F8B1 188F
203 A F8B3 183C
204 A F8B5 BDF967
205 A F8B8 36
206 A F8B9 8602
207 A F8BB 9A02
208 A F8BD 188F
209 A F8BF 0E
210 A F8C0 181F0002FB
211 A F8C5 0F
212 A F8C6 32
213 A F8C7 1838
214 A F8C9 18A700
215 A F8CC 188F
216 A F8CE 4C
217 A F8CF 16
218 A F8D0 C407
219 A F8D2 188F
220 A F8D4 26DD
221 A F8D6 20D5
222 A
223 A
224 A
225 A
226 A
227 A      F8D8
228 A F8D8 C660
229 A F8DA 8420
230 A F8DC 2604
231 A F8DE D602
232 A F8E0 C458

```

```

*      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,.....
*
*      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
SLAUTO  LDAA     #$05
        ORAA     ADDR2          Set first address to $05
        XGDY
SLLOOP  PSHY
        JSR      READBYTE      ...and store
        PSHA
        LDAA     #$02          Get a data byte from GPIB
        ORAA     ADDR2          ...and store
        XGDY          DSP 'ready' esting
        SLTL     CLI           Set address to DSP status register
        BRCLR    0,Y,#TXDE,*   Load address into Y
        SEI           Allow ATN and IFC ints.
        PULA          Wait till DSP ready (+delay)
        PULY          Retrieve data for DSP
        STAA     ,Y           Retrieve DSP write address
        XGDY          Write data to DSP
        INCA          Put address into D for processing
        TAB         Set to next register
        ANDB     #$07          Test for last register in cycle.
        XGDY          (mask off register field and set flags)
        BNE      SLLOOP       Address back in Y
        BRA      SLAUTO       Continue if not last reg.
        SLNORM  EQU      *           Repeat whole cycle
*****
*      /SLNORM/ - Normal data write mode
*      Multiple addressing used if indicated in mode byte
*      No DSP ready testing used
*****
SLNORM  EQU      *
        LDAB     #$60          Normal write routine
        ANDA     #$20          Prepare as if M bit is set
        BNE     SLNMISS       Test M bit of mode byte
        LDAB     ADDR2        Miss if M is set
        ANDB     #$58          Prepare for M bit clear
        SLNORM  EQU      *           (Clear Reg. field and M bit)

```

# Freescale Semiconductor, Inc.

233 A F8E2 D702	SLNMISS	STAB	ADDR2	Update ADDR2
234 A F8E4 BDF967	SLNLOOP	JSR	READBYTE	Get DSP reg. address field
235 A F8E7 8407		ANDA	#S07	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	SLNLOOP	Start again
241 A	*****			
242 A	*			
243 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 BDF984		JSR	SENBYTE	
247 A F8FA 20F9		BRA	SPAS	
248 A	*			
249 A F8FC	SENDSTRG	EQU	,*	
250 A F8FC 18A600	SSTRG1	LDAA	,Y	
251 A F8FF BDF984		JSR	SENBYTE	
252 A F902 1808		INY		
253 A F904 81F5		CMPA	#F5	(=0A):ACCA is inverted by SENDBYTE
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 F908 26FC		BNE	WAIT1	
260 A F90D 39		RTS		
261 A	*			
262 A F90E 18CE000A	ABORT	LDY	#10	
263 A F912 1D23FD		BCLR	TFLG1,X,#SFD	Clear I/C 2 flag only
264 A F915 8DF2		BSR	WAIT	Wait to see if IFC glitch only
265 A F917 1F000206		BRCLR	PORTA,X,#S2,ABORTEX	If IFC line still asserted then
266 A F91B 8E00FF		LDS	#USTACK	! reset stack &
267 A F91E 7EF800		JMP	MAIN	! re-start,
268 A F921 38	ABORTEX	RTI		else return.
269 A	*			
270 A F922	INIT	EQU	*	
271 A F922 0F		SEI		Disable interrupts.
272 A F923 CE1000		LDX	#S1000	
273 A F926 860A		LDAA	\$(NTADS+NSADS)	
274 A F928 9700		STAA	CHDSTAT	Set talker & listener to idle states.
275 A F92A BDF9A01		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	\$(PIFC+PATN)	enable interrupt on ATN assertion.
279 A F933 A721		STAA	TCTL2,X	
280 A F935 8603		LDAA	#3	
281 A F937 A722		STAA	TMSK1,X	Enable IC3,IC2 interrupt,
282 A F939 6F0C		CLR	OC1N,X	and disable all others.
283 A F93B 6F24		CLR	TMSK2,X	
284 A F93D 6F28		CLR	SPCR,X	
285 A F93F CC300C		LDD	#S300C	
286 A F942 6F2C		CLR	SCCR1,X	Enable SCI Tx + Rx
287 A F944 E72D		STAB	SCCR2,X	for 9600 baud
288 A F946 A72B		STAA	BAUD,X	
289 A F948 0E		CLI		
290 A F949 39		RTS		before returning.

```

_____ low ATN and IFC interrupts
_____ it for data

_____ ad in char

_____ able IFC and ATN interrupts

_____ en ready, send char and
_____ turn

_____ low ATN,IFC or T/O interruptions.

_____ ke sure RFD is asserted.
_____ it for DAV (when RFD de-asserted)
_____ itch STRB (low) to DAC line.
_____ t up detection of DAV de-assertion.
_____ hibit interruptions until data checked
_____ ad data and assertDAC,
_____ d invert data,
_____ d wait for DAV to go false.
_____ itch STRB (low) to RFD line
_____ set DAV neg detection,
_____ d return with data in ACCA.

_____ it for RFD assertion, then send
_____ ta to listener, with DAV asserted.
_____ itch DAC to STRA,
_____ d wait for DAC assertion.
_____ ll STRA low.
_____ lect non-handshake mode
_____ clear STAF without re-asserting DAV
_____ -select handshake mode.
_____ connect STRA to RFD,
_____ d release STRA before
_____ turning with STAF set if RFD asserted

_____ ATN asserted then
_____ make sure RFD is de-asserted
_____ until all registers are configured.
_____ Set up input handshake mode
    
```

# Freescale Semiconductor, Inc.

9 A F9AF C606		LDAB	\$(PIFC+MATN)	! enable interrupt on
350 A F9B1 E721		STAB	TCTL2,X	! deassertion of ATN,
351 A F9B3 18CEF807		LDY	\$(CMDMODE	! Get required return address
352 A F9B7 203E		BRA	SRVEXIT	! and go there.
353 A F9B9 C605	DATAMODE	LDAB	\$(PIFC+PATN)	else enable interrupt on
354 A F9BB E721		STAB	TCTL2,X	! assertion of ATN
355 A F9BD 13000110		BRCLR	CMDSTAT,#LADS,TALKER	! If listener then
356 A F9C1 13000806		BRCLR	CMDSTAT,#NSADS,LISTEN2!	! If not sec. listener then
357 A F9C5 18CEF85E		LDY	\$(LACS	! enter LACS state,
358 A F9C9 202C		BRA	SRVEXIT	
359 A F9CB 18CEF89F	LISTEN2	LDY	\$(SLACS	! else enter SLACS state.
360 A F9CF 2026		BRA	SRVEXIT	
361 A	*			
362 A F9D1 1200021C	TALKER	BRSET	CMDSTAT,#NTADS,NOADR	! else if talker then
363 A F9D5 8D4B		BSR	INITOP	! enable buffer for output
364 A F9D7 12000410		BRSET	CMDSTAT,#SPMS,SPOLL	! if not serial poll mode then
365 A F9DB 13000806		BRCLR	CMDSTAT,#NSADS,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	! else
369 A F9E9 200C		BRA	SRVEXIT	enter secondary TACS
370 A F9EB 18CEF8F5	SPOLL	LDY	\$(SPAS	! else activate serial poll
371 A F9EF 2006		BRA	SRVEXIT	
372 A F9F1 8D0E	NOADR	BSR	IDLEBUS	! else go to idle mode
373 A F9F3 18CEF806		LDY	\$(IDLE	
374 A F9F7 1D23FE	SRVEXIT	BCLR	TFLG1,X,\$(SFE	Clear interrupt flag generated by ATN
375 A F9FA 8E00FF		LDS	\$(JUSTACK	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	*	
381 A FA01 CC1211		LDD	\$(1211	Select handshake mode, to prevent STRB pulses
382 A FA04 E702		STAB	PIOC,X	when writing to PORTB. Select +ive STRB.
383 A FA06 A704		STAA	PORTB,X	Disable bus driver, and de-assert STRB.
384 A FA08 6C04		INC	PORTB,X	Now release RFD & DAC, by selecting talker mode
385 A FA0A 39		RTS		
386 A	*			
387 A FA0B	INITIP	EQU	*	
388 A FA0B 6F07		CLR	DDRC,X	Set PortC to input.
389 A FA0D 1D0407		BCLR	PORTB,X,\$(7	Connect DAC to STRB,select input buffer mode.
390 A FA10 C610		LDAB	\$(10	Select -ive STRA, -ive STRB,
391 A FA12 E702		STAB	PIOC,X	full i/p handshake mode.
392 A FA14 E602		LDAB	PIOC,X	
393 A FA16 1C0410		BSET	PORTB,X,\$(10	Make sure STRB is de-asserted
394 A FA19 1D0410		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 1D0408		BCLR	PORTB,X,\$(8	before connecting RFD to STRB (low).
397 A FA21 39		RTS		
398 A	*			
399 A FA22	INITOP	EQU	*	Entered with data & handshake lines in Listener mode
400 A FA22 1D040A		BCLR	PORTB,X,\$(0A	Route RFD to STRA.
401 A FA25 1C0410		BSET	PORTB,X,\$(10	Pull STRA low, to deassert STRB.
402 A FA28 C608		LDAB	\$(08	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 & PORTCL in turn,
405 A FA2E E605		LDAB	PORTCL,X	to clear STAF without asserting STRB.
406 A FA30 1C0210		BSET	PIOC,X,\$(10	Now select full output handshake mode.

407 A FA33 1C0405	BSET	PORTB,X,#5	Connect RFD to STRA,STRB to DAV,enable o/p buffer
408 A FA36 C6FF	LDAB	#\$FF	
409 A FA38 E707	STAB	DDRC,X	Then put PortC in output mode
410 A FA3A 100410	BCLR	PORTB,X,#\$10	& release STRA.
411 A FA3D 39	RTS		Exit with STAF set only if RFD asserted.
412 A	*		
413 A FA3E	NULLSRV	EQU	*
414 A FA3E 3B		RTI	This service routine should never be executed, as all unused interrupts are disabled.
415 A	*		
416 A FA3F	ILLOPSRV	EQU	*
417 A FA3F CF		STOP	Stop processor if there's a problem with code.
418 A FA40 7EFA3F		JMP	ILLOPSRV
419 A	*		Just in case.(Should never be executed).
420 A	*		
421 A	*		
422 A FFEA	ORG	\$FFEA	
423 A FFEA F9A5	FDB	ATNSRV	I/P capture 3
424 A FFEC F90E	FDB	ABORT	I/P capture 2
425 A FFEE 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	

Interrupt vector assignments





ART F90E  
 RTEX F921  
 ADDR2 0002  
 ATNSRV F9A5  
 BAUD 002B  
 CMD1 F828  
 CMD2 F831  
 CMD3 F83A  
 CMD3A F83E  
 CMD4 F843  
 CMD5 F84C  
 CMD6 F853  
 CMDMODE F807  
 CMDSTAT 0000  
 CURRENT 0003  
 DATAMODE F9B9  
 DAV F979  
 DDRC 0007  
 DUART A200  
 EOS 0000  
 IDLE F806  
 IDLEBUS FA01  
 ILLOPSRV FA3F  
 INIT F922  
 INITIP FA0B  
 INITOP FA22  
 INPUT F94A  
 JTIC2 00C7  
 JTIC3 00C4  
 LACS F85E  
 LADS 0001  
 LAG 0020  
 LISTEN2 F9CB  
 MAIN F800  
 MLA 0020  
 MSAH 0067  
 MSAL 0060  
 MTA 0040  
 NATN 0002  
 NDAC F98F  
 NDAV F968  
 NIFC 0008  
 NOADDRD F9F1  
 NRFD F986  
 NSADS 0008  
 NTADS 0002  
 NULLSRV FA3E  
 OC1M 000C  
 OTA F823  
 OUTPUT F953  
 OUTSTRG F95C  
 PATN 0001

# Freescale Semiconductor, Inc.

PIFS 0007  
 PIOC 0002  
 PORTA 0000  
 PORTB 0004  
 PORTC 0003  
 PORTCL 0005  
 RDRF 0020  
 READBYTE F967  
 RXDF 0001  
 SCCR1 002C  
 SCCR2 002D  
 SCDR 002F  
 SCG 0060  
 SCSR 002E  
 SECCMD F855  
 SENDBYTE F984  
 SENDSTRG F8FC  
 SLACS- F89F  
 SLAUTO F8AD  
 SLLOOP F883  
 SLNLOOP F8E4  
 SLNMISS F8E2  
 SLNORM F8D8  
 SLTL F88F  
 SPAS F8F5  
 SPCR 0028  
 SPD 0019  
 SPE 0018  
 SPMS 0004  
 SPOLL F9EB  
 SRVEXIT F9F7  
 SSTRG1 F8FC  
 STACS F86E  
 STATUS 0001  
 STLOOP F87D  
 STMISS F884  
 STRGEX F966  
 TACS F866  
 TAG 0040  
 TALKER F9D1  
 TALKER2 F9E5  
 TCTL2 0021  
 TDRE 0080  
 TFLG1 0023  
 TMSK1 0022  
 TMSK2 0024  
 TXDE 0002  
 UNL 003F  
 UNT 005F  
 USTACK 00FF  
 WAIT F909  
 WAIT1 F909

Freescale Semiconductor, Inc.



**Freescale Semiconductor, Inc.**

**Freescale Semiconductor, Inc.**

**For More Information On This Product,  
Go to: [www.freescale.com](http://www.freescale.com)**



**Freescale Semiconductor, Inc.**

**Freescale Semiconductor, Inc.**

**For More Information On This Product,  
Go to: [www.freescale.com](http://www.freescale.com)**

**How to Reach Us:****Home Page:**

www.freescale.com

**E-mail:**

support@freescale.com

**USA/Europe or Locations Not Listed:**

Freescale Semiconductor  
Technical Information Center, CH370  
1300 N. Alma School Road  
Chandler, Arizona 85224  
+1-800-521-6274 or +1-480-768-2130  
support@freescale.com

**Europe, Middle East, and Africa:**

Freescale Halbleiter Deutschland GmbH  
Technical Information Center  
Schatzbogen 7  
81829 Muenchen, Germany  
+44 1296 380 456 (English)  
+46 8 52200080 (English)  
+49 89 92103 559 (German)  
+33 1 69 35 48 48 (French)  
support@freescale.com

**Japan:**

Freescale Semiconductor Japan Ltd.  
Headquarters  
ARCO Tower 15F  
1-8-1, Shimo-Meguro, Meguro-ku,  
Tokyo 153-0064  
Japan  
0120 191014 or +81 3 5437 9125  
support.japan@freescale.com

**Asia/Pacific:**

Freescale Semiconductor Hong Kong Ltd.  
Technical Information Center  
2 Dai King Street  
Tai Po Industrial Estate  
Tai Po, N.T., Hong Kong  
+800 2666 8080  
support.asia@freescale.com

**For Literature Requests Only:**

Freescale Semiconductor Literature Distribution Center  
P.O. Box 5405  
Denver, Colorado 80217  
1-800-441-2447 or 303-675-2140  
Fax: 303-675-2150  
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.