Order this document
as AN456/D

MOTOROLA
SEMICONDUCTOR
APPLICATION NOTE

AN456

Using PCbug11 as a Diagnostic Aid for Expanded Mode M68HC11 Systems

By Steven McAslan,
Motorola Ltd.,
East Kilbride, Scotland

INTRODUCTION

This application note describes some advanced uses of the PCbug11 software package for the M68HC11. The techniques described here allow the user to optimise the debugging environment (perhaps for diagnostic purposes), by moving the communications program into external memory and making full use of the mode programming of the M68HC11. Firstly, the communications routine itself is explained, then the system architecture required is examined and finally the task of customising the talker for the application system is considered. The PCbug11 software is available from Motorola and provides a complex debugging environment for simple hardware platforms.

HOW TALKERS WORK

The PCbug11 environment consists of two pieces of software: the executable on the PC and the communications program which runs on the M68HC11. The communications program is called the *talker*. The talker is an interrupt-driven and very compact piece of code; either the SCI or XIRQ interrupt can be used.

The purpose of the talker is shown in the flow chart in figure 1. An example of the code used to implement the function is shown in listing 1. This is specifically for the MC68HC11E9. However, the code blocks and label names are normally common to all talkers.

There are three main sections to the code: initialisation; command interpreting; and breakpoint handling. The last two of these are driven by interrupts, while the initialisation is performed only once, whenever the talker is activated.

Briefly, initialisation sets up the internal SCI or external ACIA, enables the appropriate interrupt and ensures that the interrupt vector for this is pointing to the interrupt server, in this case the command interpreter.

The command interpreter has four main functions and two simple communications handlers. The functions are:

Read Memory (command: $01; label: TREADMEM)

Write Memory (command: $41; label: TWRITMEM)

Read Registers (command: $81; label: INH1)

Write Registers (command: $C1; label: INH2)

The register operations are specific examples of the memory reads and writes, as the register modifications only involve an alteration of the active stack frame in memory.

The functions are selected using the command received. The register commands involve a set number of bytes being transferred from the host to the M68HC11 or vice versa, therefore only a single command byte is required. The memory commands involve communication from the host to instruct the M68HC11 how much memory is to be read/written and the appropriate addresses. For full details, refer to the flow chart and listing software.

MOTOROLA

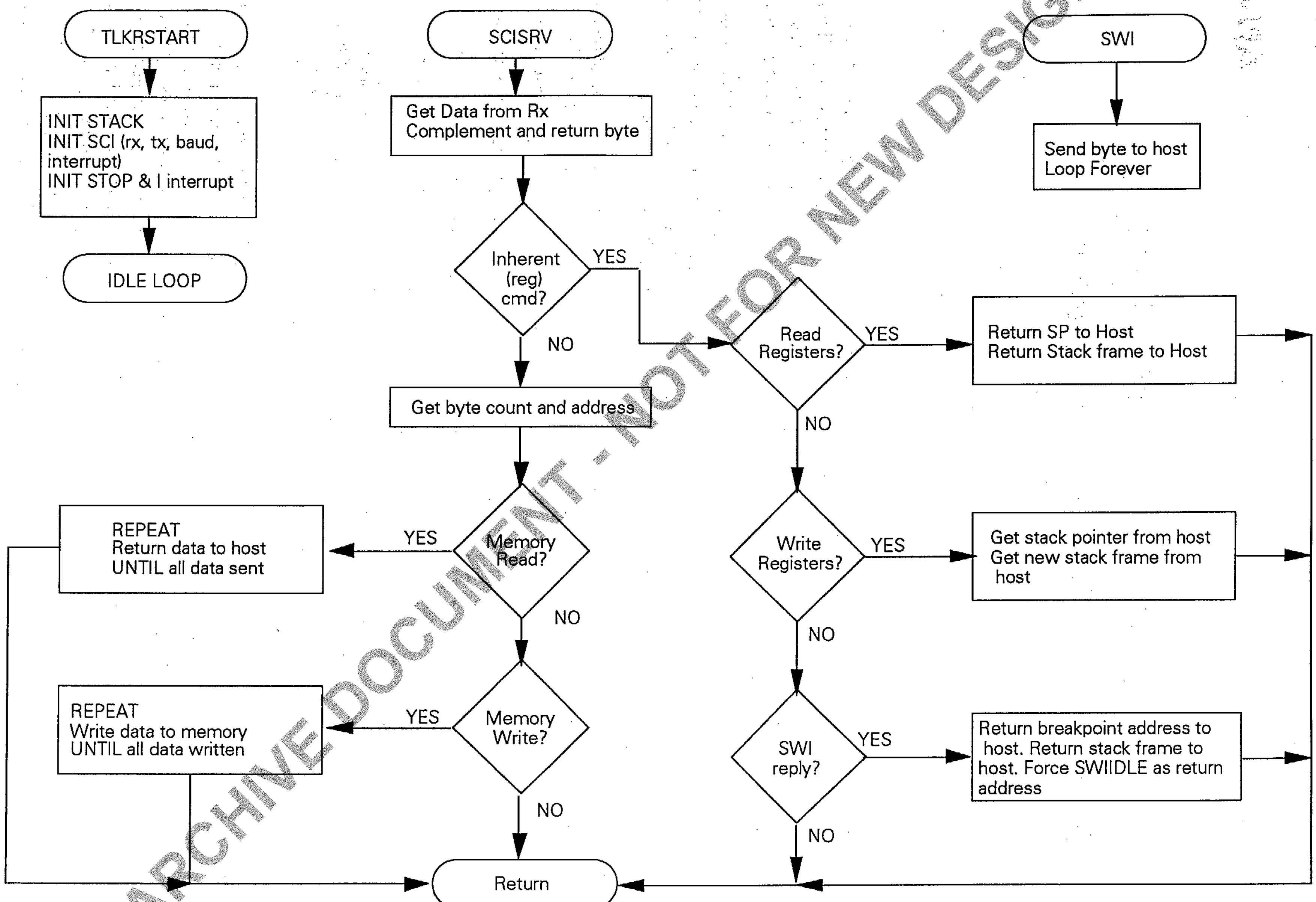

Figure 1. TALKE.ASC Flow Chart

Two communications routines are also used here. These perform reads and writes of the SCI/ACIA (INSCI, OUTSCI). Every command received by the talker is echoed back to the host *complemented* to confirm communications integrity.

In addition, there is breakpoint handling software. This is more complex, as it involves at least two interrupts to provide full functionality. Before the software can be run, the SWI interrupt vector must be initialised. This is done by the host computer before a go, call or trace command. (See [1], section 4.3.)

The first interrupt occurs when the M68HC11 executes a SWI opcode in its program. This causes a jump to the breakpoint handling software. The SWI interrupt handler then transmits a byte to the host to inform it that an SWI has been found. The M68HC11 enters an idle loop while the PC host determines whether the SWI found is a breakpoint, tracepoint or user SWI. Having decided on the nature of the SWI, the host sends a byte to the MCU to cause the second interrupt. If a user SWI is found, then the code at the user interrupt is simply executed. If a break or trace point is found, then the code suspends at the idle loop until the user decides either to trace again, continue or stop the code execution.

USING TALKERS IN EXPANDED MODE

Most users of PCbug11 communicate with a M68HC11 running in bootstrap mode. This involves downloading a talker each time communications begin or using internal EEPROM. However, in embedded systems using an expanded mode M68HC11 it would be more useful to place the talker in external memory with any self-test software. This approach also allows an alternative to the M68HC11 SCI system to be used; a feature which may be useful when the user requires to test software running on the SCI.

To use a talker in expanded memory, the basic blocks described in the preceding section must be implemented and the interrupt structure must be able to accommodate the requirements of the talker. The basic blocks are easily moved to an area of expanded memory. However, the interrupt structure does require to be examined quite closely.

The PCbug11/talker environment requires that certain vectors are pointing to certain pieces of code. For trace and breakpoint it is normal for the SWI vector to be altered according to the function in use. In bootstrap mode all of the interrupt vectors point to RAM. From RAM, an appropriate jump to an interrupt service routine can be carried out. This allows the interrupt vectors to be easily customised for the PCbug11 environment. In expanded mode the interrupt vectors point to the top of memory. From here, the user must either redirect them to an writable area of memory or have the block at the top of memory itself writable. Unfortunately, interrupt vectors in RAM would not normally be considered a sound system decision. Such techniques, however, are valuable when developing designs.

The BUFFALO monitor for the M68HC11 redirects the interrupt vectors to internal RAM. In PCbug11 systems, the RESET vector should point to something which will initialise the rest of the vectors in RAM and the talker code. After this the user may load application dependent addresses into the RAM and run his code. The disadvantages here are that there will be a slight processing overhead to reach the interrupt service routine (one extended JMP instruction = 4 cycles) and some 60 bytes of internal or external RAM will be lost to interrupt re-direction. (See [2,3] bootstrap ROM listings. See figure 2 for the memory map arrangement of this system.)

Another approach is to use the special test mode of the M68HC11 MCU. This mode is normally used for factory test purposes, as it allows access to normally protected features of the chip. However, it does have a notable additional feature, which is that the interrupt and reset vectors are transposed from their normal positions in memory at $FFXX to the special mode area $BFXX. Note that bootstrap mode also has the same effect. The key difference is that in special test mode the vectors are taken from external memory, rather than the internal bootstrap ROM.

Special test mode could be accessed using a switch or key on the system. The talker interrupt vectors could be placed at the special mode interrupt locations or the interrupt locations could point into RAM; cf. figure 2. In either case, the talker could be placed in some spare area of memory (the talker is normally less than 200 bytes) and only accessed in the special mode.

This approach allows the M68HC11 to be run in expanded mode while retaining the full features of PCbug11. An example of this approach is illustrated in figure 3.

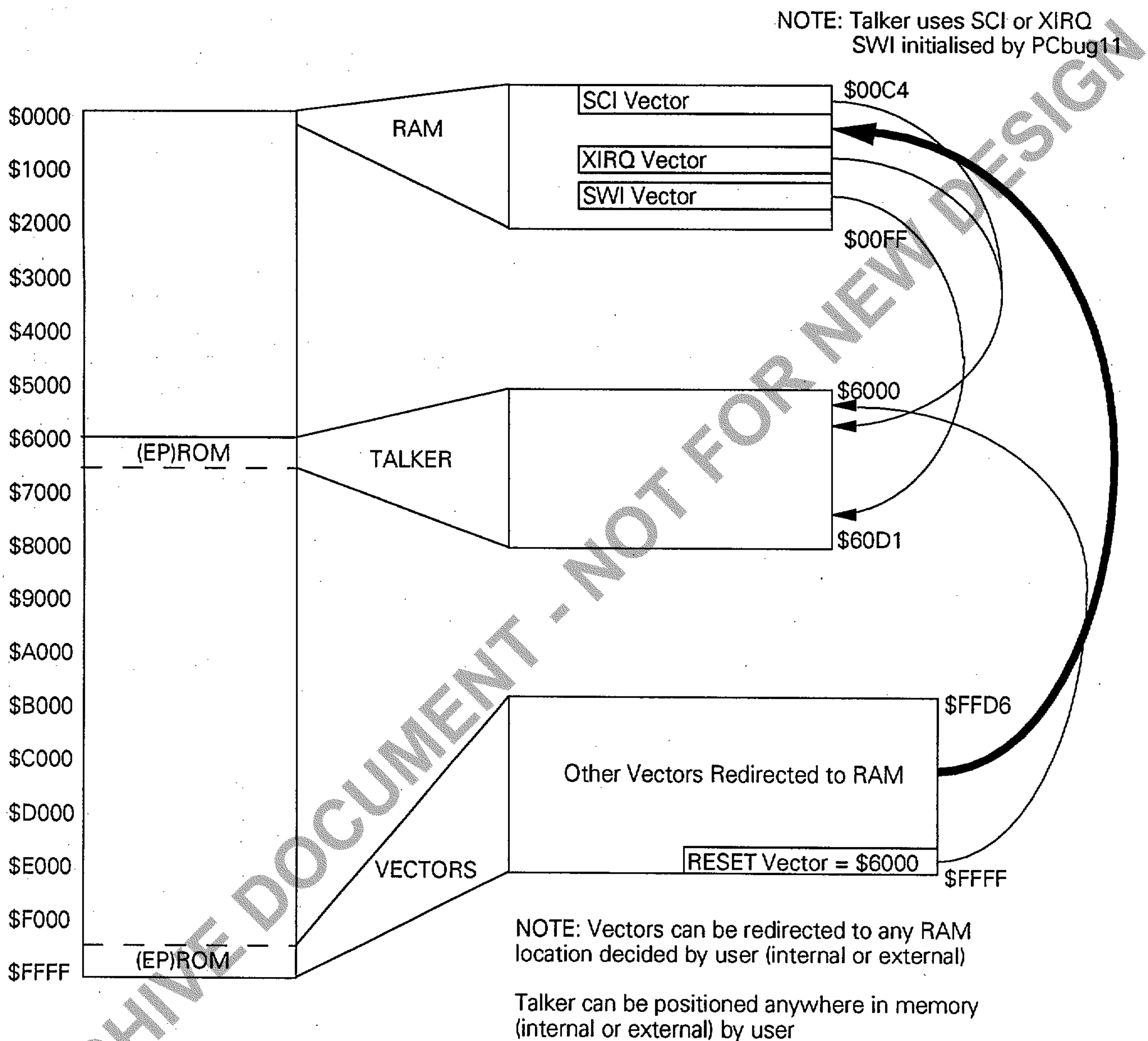

Figure 2. Expanded Mode Use Version 1

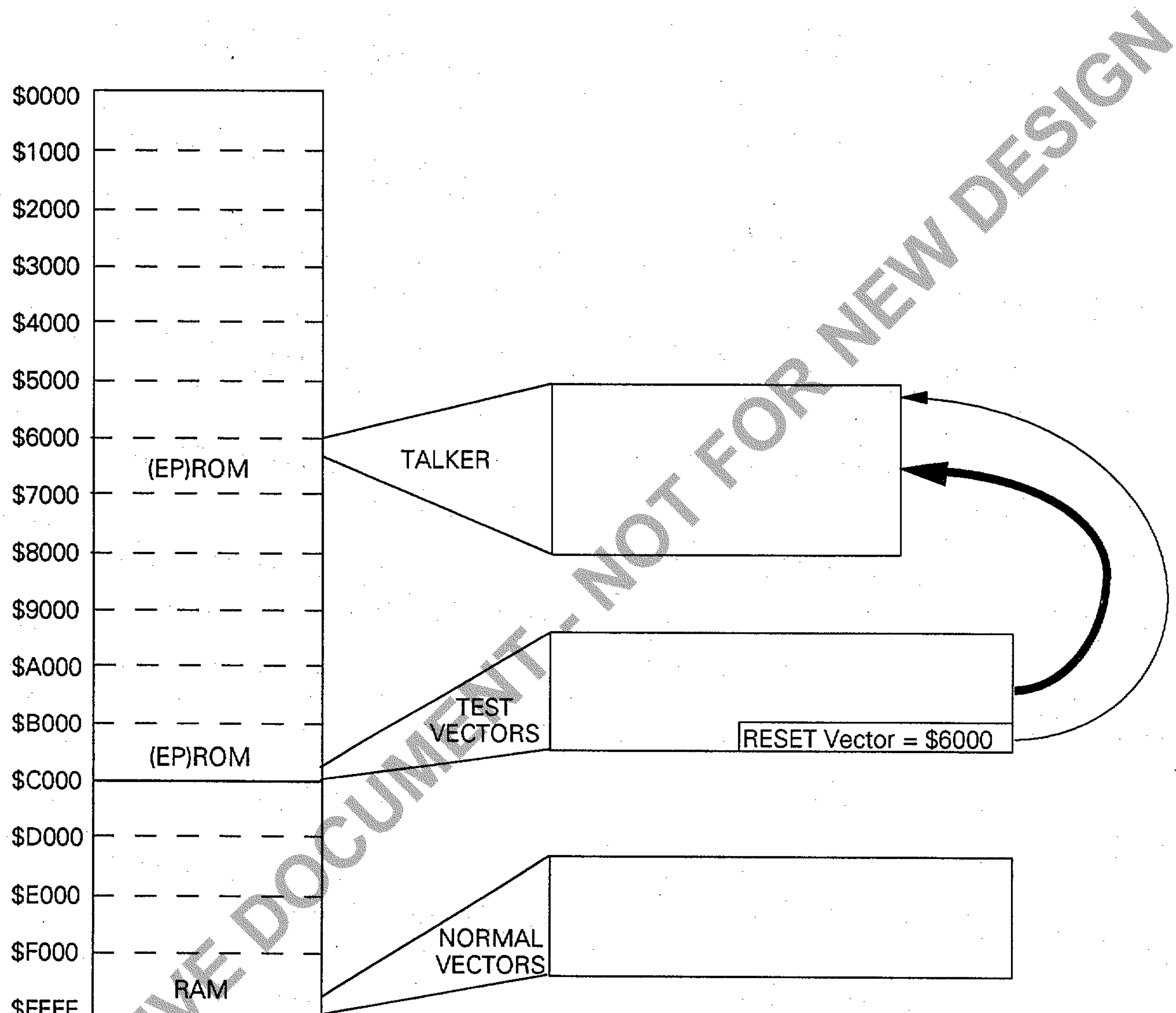

Figure 3. Expanded Mode Use Version 2

IMPLEMENTING THE EXPANDED MODE TALKER

The following discussion assumes that the user is going to modify an existing talker. If a new talker is to be written, care should be taken that the general principles described in the above sections are adhered to. A general purpose talker for the M68HC11 in expanded mode using the SCI is shown in listing 2.

The first decision to make when implementing a talker in an expanded mode is whether the internal SCI or an external ACIA device is to be used. If the SCI is used, then normally the SCI interrupt or the XIRQ interrupt would be used. It is also possible to use the IRQ interrupt or a timer input capture pin. However, these offer little advantage over the SCI interrupt itself. If an external communications device is used, then the choice is normally the XIRQ interrupt. Again, other interrupt sources can be used, but the XIRQ interrupt should ensure that the communications from the host are responded to.

The use of the XIRQ pin for the external communications device does not prevent the use of the XIRQ for other external resources. If another resource requires to use this pin, then internal arbitration could be used to select which source caused the interrupt. It is essential in this case that there is no possibility of the alternate source causing an endless loop from which the program could never recover.

Once the communications system is chosen and the interrupt to be used is selected, the initialisation section for the talker can be implemented. At this stage any baud, parity, number of bits and interrupt enable bits are set up. It is usually best to perform this function immediately after RESET but it could be performed later if required, for example, if an error is found.

The rest of the talker is not normally changed. However, take care that the M68HC11 registers are not moved using the INIT register and that the INSCI and OUTSCI routines are changed to handle an external device if required.

The last change required is to update the talker .MAP file.

UPDATING THE .MAP FILE

The .MAP file contains essential address information for PCbug11. In bootstrap mode the program knows where certain parameters are by default. However, in expanded mode the talker could be anywhere in memory and so the PCbug11 has to be told where to find it. It is important that the .MAP file corresponds correctly to the talker or malfunction of the software can occur.

Listing 3 shows the .MAP for the general purpose talker in listing 2. The requested addresses may be determined by assembling the talker and noting the location of each of the important labels.

Change the .MAP file using a text editor and place it in the current working directory. The address parameters must begin in the 15th column or higher.

USING THE TALKER AS A DIAGNOSTIC AID

The exact use of the talker in this situation will depend largely on the system which is being examined. However, with the talker installed the user can interactively examine the system. Self-test routines could be run, loaded into RAM from the user PC. EEPROM integrity and preset values could be checked and updated if necessary. If required, the MCU mode could be changed by writing into the HPRIO register. The upper nibble of this register is accessible only in special modes (see [2]).

If the MCU SCI port is available, the device could be placed in special bootstrap mode and PCbug11 run as normal. In this case, the data and address bus integrity of the system could be checked. Here, mode control of the M68HC11 is again the key feature. By changing the HPRIO register (MDA bit), the external data and address buses are turned on while the bootstrap ROM is still present and readable by the CPU. Now the user can perform reads and verifies on the external memory to see if any problem exists with either bus, while still having full control on the MCU via PCbug11.

CONCLUSION

By using the techniques described, the user can include a debugging aid for any expanded mode M68HC11 system. If a single chip system is used, then the additional overhead of PCbug11 RAM requirements is the only drawback.

REFERENCES

[1] PCbug11 User Manual, Motorola M68PCBUG11/D1

[2] M68HC11 Reference Manual, Motorola M68HC11RM/AD

[3] MC68HC11 Bootstrap Mode, Motorola Application Note AN1060/D

LISTING 1 – TALKE.ASC ASSEMBLY LISTING

```
M68HC11 Absolute Assembler                   Version 2.70g:talke.ASC

A                        *************************** TALKE.ASC 6/3/90 *****************************
A                        * Motorola Copyright 1988,1990
A                        * MCU resident, Interrupt driven Communication routines for 68HC11
A                        * monitor. Provides low level memory and stack read/write operations.
A                        *
A                        * This talker DOES NOT use XIRQ
A                        * ______________
A                        *
A                        * Works with Host user interface program PCBUG11.EXE.
A                        * N.B. TALKE.ASC is designed to be downloaded through standard type of
A                        * bootloader, and communicate with host through SCI.
A                        * This bootloader relies on 4 char idle line on SCI to terminate.
A                        *
A                        * CONSTANTS
A 0000                   TALKBASE     equu     $0000
A 00C4                   BOOTVECT     equ      $00C4            Start of bootstrap vector jump table.
A 01FF                   STACK        equ      $01FF
A 1000                   REGBASE      equ      $1000
A                        *
A 00C4                   JSCI         equ      $00C4
A 00F1                   JXIRQ        equ      $00F1
A 00F4                   JSWI         equ      $00F4
A 00F7                   JILLOP       equ      $00F7
A 00FA                   JCOP         equ      $00FA
A 008                    uS500        equ      5000/35          Delay with DEY/BNE loop
A 007E                   JMPEXT       equ      $7E              Mnemonic for jump extended
A 004A                   BRKCODE      equ      $4A              Break point signal code to host.
A 004A                   BRKACK       equ      $4A              Break point acknowledge code
A                        *
A                        * REGISTERS
A 002B                   BAUD         equ      $2B
A 002C                   SCCR1        equ      $2C
A 002D                   SCCR2        equ      $2D
A 002E                   SCSR         equ      $2E
A 002F                   SCDR         equ      $2F
A                        *
A 0020                   RDRF         equ      $20
A 0080                   TDRE         equ      $80
A                        *
A                        * PROGRAM
A 0000                                org      TALKBASE
A 0000                   TLKRSTART    EQU      *
A 0000 8E01FF                         LDS      #STACK
A 0003 CE1000                         LDX      #REGBASE
A 0006 6F2C                           CLR      SCCR1,X
A 0008 CC302C                         LDD      #$302C
A 000B A72B                           STAA     BAUD,X           Init SCI to 9600 baud, no parity
A 000D E72D                           STAB     SCCR2,X          and enable SCI tx & rx.
A 000F 8640                           LDAA     #$40             Enable STOP & I bit, disable XIRQ.
A 0011 06                             TAP
A                                     *
A 0012 7E0012            IDLE         JMP      IDLE             Wait for SCI interrupt from host.
A                        * A RESET from host changes above jump destination to start of user code.
A                        *
A 0015                   SCISRV       EQU      *                On detecting interrupt,
A 0015 B6102E                         LDAA     SCSR+REGBASE     assume receiver caused it.
A 0018 8420                           ANDA     #RDRF
A 001A 27F9                           BEQ      SCISRV           otherwise program will hang up
A                        *
A 001C                   RXSRV        EQU      *                Talker code processes data.
A 001C B6102F                         LDAA     SCDR+REGBASE     Get command byte, & echo as ack
A 001F 43                             COMA                      Inverted
A 0020 8D46                           BSR      OUTSCI           to host.
A 0022 2A51                           BPL      INH1             If bit 7 set, process inh. command
A 0024 8D33                           BSR      INSCI            else read byte count into B
```

```
A 0026 8F                        XGDX                   Save command and byte count.
A 0027 8D30                      BSR    INSCI           Read high address byte
A 0029 17                        TBA                    into ACCA
A 002A 8D2D                      BSR    INSCI           then low address byte into ACCB
A 002C 8F                        XGDX                   Cmd in A,count in B,addr in X
A 002D                           CMPA   #$FE
A 002F 260D                      BNE    RXSRV1          If command is memory read, then
A                     *
A 0031                TREADMEM   EQU    *               REPEAT
A 0031 A600                      LDAA   ,X              read required address
A 0033 8D33                      BSR    OUTSCI          send it to host
A 0035 17                        TBA                    save byte count)
A 0036 8D21                      BSR    INSCI           and wait for acknowledge
A 0038 16                        TAB                    (restore byte count)
A 0039 08                        INX                    Increment address
A 003A 5A                        DECB                   Decrement byte count
A 003B 26F4                      BNE    TREADMEM        UNTIL all done
A 003D 3B                        RTI                    return to idle loop or user code.
A                     *
A 003E                RXSRV1     EQU    *
A 003E 81BE                      CMPA   #$BE
A 0040 2616                      BNE    RXSRVEX         If command is memory write then
A                     *
A 0042 17                        TBA                    move byte count to ACCA
A 0043                TWRITMEM   EQU    *               REPEAT
A 0043 8D14                      BSR    INSCI           Read next byte from host into B,
A 0045 E700                      STAB   ,X              and store at required address.
A 0047 18CE0001                  LDY    #$0001          Set up wait loop
A 004B 1809           WAITPOLL   DEY                    Y operand must be manually set
A 004D 26FC                      BNE    WAITPOLL
A 004F E600                      LDAB   ,X              Read stored byte and
A 0051 F7102F                    STAB   SCDR+REGBASE    echo it back to host,
A 0054 08                        INX
A 0055 4A                        DECA                   Decrement byte count
A 0056 26EB                      BNE    TWRITMEM        UNTIL all done
A 0058                RXSRVEX    EQU    *               and return
A 0058 3B             NULLSRV    RTI
A                     *
A 0059                INSCI      EQU    *
A 0059 F6102E                    LDAB   SCSR+REGBASE    Wait for RDRF=1
A 005C C50A                      BITB   #$0A            If break detected
A 005E 26A0                      BNE    TLKRSTART       then restart talker
A 0060 C420                      ANDB   #RDRF
A 0062 27F5                      BEQ    INSCI
A 0064 F6102F                    LDAB   SCDR+REGBASE    then read data received from host
A 0067 39                        RTS                    and return with data in ACCB
A                     *
A 0068                OUTSCI     EQU    *               Only register Y modified.
A 0068 188F                      XGDY                   Enter with data to send in ACCA.
A 006A B6102E         OUTSCI1    LDAA   SCSR+REGBASE
A 006D 2AFB                      BPL    OUTSCI1         MS bit is TDRE flag
A 006F 188F                      XGDY
A 0071 B7102F                    STAA   SCDR+REGBASE    Important - Updates CCR!
A 0074 39                        RTS
A                     *
A 0075                INH1       EQU    *
A 0075 817E                      CMPA   #$7E            If command is read registers then
A 0077 260C                      BNE    INH2
A                     *
A 0079 30             INH1A      TSX                    Move stack pointer to X
A 007A 8F                        XGDX                   then to ACCD
A 007B 8DEB                      BSR    OUTSCI          send SP to host (high byte first)
A 007D 17                        TBA
A 007E 8DE8                      BSR    OUTSCI          then low byte
A 0080 30                        TSX                    Restore X (=stack pointer)
A 0081 C609                      LDAB   #9              then return 9 bytes on stack
A 0083 20AC                      BRA    TREADMEM        - CCR,B,A,XH,XL,YH,YL,PCH,PCL
A                     *
```

```
A 0085                INH2       EQU    *
A 0085 813E                      CMPA   #$3E         If command is write registers then
A 0087 2612                      BNE    SWISRV1
A                     *
A 0089 8DCE                      BSR    INSCI        get SP from host (High byte first)
A 008B 17                        TBA
A 008C 8DCB                      BSR    INSCI
A 008E 8F                        XGDX                Move to X reg
A 008F 35                        TXS                 and copy to stack pointer
A 0090 8609                      LDAA   #9           Then put next 9 bytes on to stack
A 0092 20AF                      BRA    TWRITMEM
A                     *
A 0094                SWISRV     EQU    *            Breakpoints generated by SWI
A 0094 864A                      LDAA   #BRKCODE     Force host to process breakpoints
A 0096 8DD0                      BSR    OUTSCI       by sending it the break signal
A 0098 0E             SWIIDLE    CLI
A 0099 20FD                      BRA    SWIIDLE      then wait for response from host.
A                     *
A 009B                SWISRV1    EQU    *
A 009B 814A                      CMPA   #BRKACK      If host has acknowledged BP then
A 009D 26B9                      BNE    RXSRVEX
A 009F 30                        TSX                 move SP to SWI stack frame and
A 00A0 C609                      LDAB   #9
A 00A2 3A                        ABX
A 00A3 35                        TXS
A 00A4 EC07                      LDD    7,X          Send user code BP return address
A 00A6 8DC0                      BSR    OUTSCI       (high byte first)
A 00A8 17                        TBA
A 00A9 8DBD                      BSR    OUTSCI       (low byte next)
A 00AB CC0098                    LDD    #SWIIDLE     force idle loop on return from BP
A 00AE ED07                      STD    7,X
A 00B0 20C7                      BRA    INH1A        but first return all registers to host
A                     *
A 00C4                ORG        BOOTVECT            Jump table only during bootstrap
A 00C4 7E                        FCB    JMPEXT       SCI
A 00C5 0015                      FDB    SCISRV
A 00C7 7E                        FCB    JMPEXT       SPI  (Unused vectors point to RTI)
A 00C8 0058                      FDB    NULLSRV
A 00CA 7E                        FCB    JMPEXT       Pulse acc. Input Edge
A 00CB 0058                      FDB    NULLSRV
A 00CD 7E                        FCB    JMPEXT       Pulse acc. Overflow
A 00CE 0058                      FDB    NULLSRV
A 00D0 7E                        FCB    JMPEXT       Timer Overflow
A 00D1 0058                      FDB    NULLSRV
A 00D3 7E                        FCB    JMPEXT       OC5
A 00D4 0058                      FDB    NULLSRV
A 00D6 7E                        FCB    JMPEXT       OC4
A 00D7 0058                      FDB    NULLSRV
A 00D9 7E                        FCB    JMPEXT       OC3
A 00DA 0058                      FDB    NULLSRV
A 00DC 7E                        FCB    JMPEXT       OC2
A 00DD 0058                      FDB    NULLSRV
A 00DF 7E                        FCB    JMPEXT       OC1
A 00E0 0058                      FDB    NULLSRV
A 00E2 7E                        FCB    JMPEXT       IC3
A 00E3 0058                      FDB    NULLSRV
A 00E5 7E                        FCB    JMPEXT       IC2
A 00E6 0058                      FDB    NULLSRV
A 00E8 7E                        FCB    JMPEXT       IC1
A 00E9 0058                      FDB    NULLSRV
A 00EB 7E                        FCB    JMPEXT       Real Time Intr
A 00EC 0058                      FDB    NULLSRV
A 00EE 7E                        FCB    JMPEXT       IRQ
A 00EF 0058                      FDB    NULLSRV
A 00F1 7E                        FCB    JMPEXT       XIRQ
A 00F2 0058                      FDB    NULLSRV
A 00F4 7E                        FCB    JMPEXT       SWI Changed by Break point
A 00F5 0058                      FDB    NULLSRV
```

```
202 A 00F7 7E                    FCB    JMPEXT          ILLOP
203 A 00F8 0000                  FDB    TLKRSTART
204 A 00FA 7E                    FCB    JMPEXT          COP Fail
205 A 00FB 0058                  FDB    NULLSRV
206 A 00FD 7E                    FCB    JMPEXT          Clock Monitor
207 A 00FE 0058                  FDB    NULLSRV
208 A                     *
209 A                            END

SYMBOL TABLE:             Total Entries=39

BAUD                      002B   RXSRV                  001C
BOOTVECT                  00C4   RXSRV1                 003E
BRKACK                    004A   RXSRVEX                0058
BRKCODE                   004A   SCCR1                  002C
IDLE                      0012   SCCR2                  002D
INH1                      0075   SCDR                   002F
INH1A                     0079   SCISRV                 0015
INH2                      0085   SCSR                   002E
INSCI                     0059   STACK                  01FF
JCOP                      00FA   SWIIDLE                0098
JILLOP                    00F7   SWISRV                 0094
JMPEXT                    007E   SWISRV1                009B
JSCI                      00C4   TALKBASE               0000
JSWI                      00F4   TDRE                   0080
JXIRQ                     00F1   TLKRSTAR               0000
NULLSRV                   0058   TREADMEM               0031
OUTSCI                    0068   TWRITMEM               0043
OUTSCI1                   006A   WAITPOLL               004B
RDRF                      0020   uS500                  008E
REGBASE                   1000

Total errors: 0
```

LISTING 2 – TALKSCI.ASC ASSEMBLY LISTING

```
M68HC11 Absolute Assembler          Version 2.70g:talksci.ASC

A                 ************************** TALKSCI.ASC 14/8/91 *************************
A                 * Motorola Copyright 1988,1991
A                 * MCU resident, Interrupt driven Communication routines for 68HC11
A                 * monitor. Provides low level memory and stack read/write operations.
A                 *
A                 * This talker DOES NOT use XIRQ
A                 * ----------------------------
A                 *
A                 * N.B. TALKSCI.ASC is a general purpose talker. It is intended to be
A                 * placed in the MCU memory map at $6000 but this can be changed by
A                 * the user to any suitable address. The talker is for general debug
A                 * and can be used in any mode as long as the vectors are correctly
A                 * initialised. It is therefore useful for normal modes. The SCI is
A                 * used for communications - use TALKACIA when an external ACIA is
A                 * to be used. TALKSCI assumes that the interrupt vectors are
A                 * pointing to RAM in the same way as the boostrap ROM.
A                 * IMPORTANT : If you change the running address of this program
A                 * then you MUST also change the TALKSCI.MAP file so that the two
A                 * match.
A                 *
A                 * When PCBUG11 is executed with option TALKSCI, a 10mS break is
A                 * output to the 68HC11's SCI, prior to establishing communication.
A                 *
A                 * CONSTANTS
A 6000            TALKBASE    equ    $6000
A 003F            STACK       equ    $003F        User may alter this parameter
A 00C4            BOOTVECT    equ    $00C4        Start of bootstrap vector jump table.
A 1000            REGBASE     equ    $1000        Change if registers are moved
A                 *
A 00C4            JSCI        equ    $00C4
A 00F1            JXIRQ       equ    $00F1
A 00F4            JSWI        equ    $00F4
A 00F7            JILLOP      equ    $00F7
A 00FA            JCOP        equ    $00FA
A 007E            JMPEXT      equ    $7E          Mnemonic for jump extended
A 004A            BRKCODE     equ    $4A          Break point signal code to host.
A 004A            BRKACK      equ    $4A          Break point acknowledge code
A                 *
A                 * REGISTERS                     Change if required for MCU
A 002B            BAUD        equ    $2B
A 002C            SCCR1       equ    $2C
A 002D            SCCR2       equ    $2D
A 002E            SCSR        equ    $2E
A 002F            SCDR        equ    $2F
A                 *
A 0020            RDRF        equ    $20          SCI Masks, change if required
A 0080            TDRE        equ    $80
A 0008            OR          equ    $08
A 0002            FE          equ    $02
A                 *
A                 * PROGRAM
A 6000            org         TALKBASE
A                 *
A                 * Initialise the SCI and interrupts
A                 *
A 6000            TLKRSTART   EQU                 Dynamically set up Boot jump table.
A 6000 867E                   LDAA   #JMPEXT
A 6002 18CE6078   LDY         #NULLSRV
A 6006 CE00C4                 LDX    #BOOTVECT
A 6009            SETVECT     EQU    *
A 6009 A700                   STAA   ,X
A 600B 08                     INX
A 600C 1AEF00                 STY    ,X
A 600F 08                     INX
A 6010 08                     INX
```

```
A 6011 8C0100                 CPX     #$100
A 6014 26F3                   BNE     SETVECT
A 6016 CE6035                 LDX     #SCISRV
A 6019 DFC5                   STX     JSCI+1
A 601B CE6000                 LDX     #TLKRSTART
A 601E DFF8                   STX     JILLOP+1
A                   *
A 6020              USERSTART EQU     *
A 6020 8E003F                 LDS     #STACK
A 6023 CE1000                 LDX     #REGBASE
A 6026 6F2C                   CLR     SCCR1,X
A 6028 CC302C                 LDD     #$302C
A 602B A72B                   STAA    BAUD,X          Initialise SCI to 9600, no parity
A 602D E72D                   STAB    SCCR2,X         and enable SCI tx & rx.
A 602F 8640                   LDAA    #$40            Enable STOP, I interrupts, disable X
A 6031 06                     TAP
A                   *
A                   * User may add jump to his own code here or may move the above
A                   * initialisation to the start of his own program.
A                   *
A 6032 7E6032       IDLE      JMP     IDLE            Now hang around for SCI interrupt
A                   *
A 6035              SCISRV    EQU     *               On detecting interrupt,
A 6035 B6102E                 LDAA    SCSR+REGBASE    assume receiver caused it.
A 6038 8420                   ANDA    #RDRF
A 603A 27F9                   BEQ     SCISRV          otherwise program will hang up
A                   *
A 603C              RXSRV     EQU     *               Talker code processes rec'd data.
A 603C B6102F                 LDAA    SCDR+REGBASE    Read command byte, & echo as ack
A 603F 43                     COMA                    Inverted
A 6040 8D46                   BSR     OUTSCI          to host.
A 6042 2A51                   BPL     INH1            If bit 7 set, then process inh cmd
A 6044 8D33                   BSR     INSCI           else read byte count from host into B
A 6046 8F                     XGDX                    Save command and byte count.
A 6047 8D30                   BSR     INSCI           Read high address byte
A 6049 17                     TBA                     into ACCA
A 604A 8D2D                   BSR     INSCI           then low address byte into ACCB
A 604C 8F                     XGDX                    Cmd in A,count in B,addr in X
A 604D 81FE                   CMPA    #$FE
A 604F 260D                   BNE     RXSRV1          If command is memory read, then
A                   *
A 6051              TREADMEM  EQU     *               REPEAT
A 6051 A600                   LDAA    ,X              read required address
A 6053 8D33                   BSR     OUTSCI          send it to host
A 6055 17                     TBA                     (save byte count)
A 6056 8D21                   BSR     INSCI           and wait for acknowledge
A 6058 16                     TAB                     (restore byte count)
A 6059 08                     INX                     Increment address
A 605A 5A                     DECB                    Decrement byte count
A 605B 26F4                   BNE     TREADMEM        UNTIL all done
A 605D 3B                     RTI                     and return to idle loop or user code.
A                   *
A 605E              RXSRV1    EQU     *
A 605E 81BE                   CMPA    #$BE
A 6060 2616                   BNE     RXSRVEX         If command is memory write then
A                   *
A 6062 17                     TBA                     move byte count to ACCA
A 6063              TWRITMEM  EQU     *               REPEAT
A 6063 8D14                   BSR     INSCI           Read next byte from host into ACCB,
A 6065 E700                   STAB    ,X              and store at required address.
A 6067 18CE0001               LDY     #$0001          Set up wait loop
A 606B 1809         WAITPOLL  DEY                     Y may take on suitable value
A 606D 26FC                   BNE     WAITPOLL
A 606F E600                   LDAB    ,X              Read stored byte and
A 6071 F7102F                 STAB    SCDR+REGBASE    echo it back to host,
A 6074 08                     INX
A 6075 4A                     DECA                    Decrement byte count
A 6076 26EB                   BNE     TWRITMEM        UNTIL all done
A 6078              RXSRVEX   EQU     *               and return
```

```
A 6078 3B         NULLSRV   RTI
A                           *
A                           * INSCI gets the received byte form the SCI
A                           *
A 6079            INSCI     EQU    *
A 6079 F6102E               LDAB   SCSR+REGBASE      Wait for RDRF=1
A 607C C50A                 BITB   #(FE+OR)          If break detected then
A 607E 2680                 BNE    TLKRSTART         restart talker.
A 6080 C420                 ANDB   #RDRF
A 6082 27F5                 BEQ    INSCI
A 6084 F6102F               LDAB   SCDR+REGBASE      then read data received from host
A 6087 39                   RTS                      and return with data in ACCB
A                           *
A                           * OUTSCI is the subroutine which transmits a byte from the SCI
A                           *
A 6088            OUTSCI    EQU    *                 Only register Y modified.
A 6088 188F                 XGDY                     Enter with data to send in ACCA.
A 608A B6102E     OUTSCI1   LDAA   SCSR+REGBASE
A 608D 2AFB                 BPL    OUTSCI1           MS bit is TDRE flag
A 608F 188F                 XGDY
A 6091 B7102F               STAA   SCDR+REGBASE      Important - Updates CCR!
A 6094 39                   RTS
A                           *
A                           * Now decide which inherent command was sent
A                           *
A 6095            INH1      EQU    *
A 6095 817E                 CMPA   #$7E              If command is read registers then
A 6097 260C                 BNE    INH2
A                           *
A                           * Command was to read MCU registers
A                           *
A 6099 30          INH1A    TSX                      Move stack pointer to X
A 609A 8F                   XGDX                     then to ACCD
A 609B 8DEB                 BSR    OUTSCI            send SP to host (high byte first)
A 609D 17                   TBA
A 609E 8DE8                 BSR    OUTSCI            then low byte
A 60A0 30                   TSX                      Restore X (=stack pointer)
A 60A1 C609                 LDAB   #9                then return 9 bytes on stack
A 60A3 20AC                 BRA    TREADMEM          i.e CCR,B,A,XH,XL,YH,YL,PCH,PCL
A                           *
A                           * Command was to write MCU registers
A                           *
A 60A5            INH2      EQU    *
A 60A5 813E                 CMPA   #$3E              If command is write registers then
A 60A7 2612                 BNE    SWISRV1
A                           *
A 60A9 8DCE                 BSR    INSCI             get SP from host (High byte first)
A 60AB 17                   TBA
A 60AC 8DCB                 BSR    INSCI             get low byte next
A 60AE 8F                   XGDX                     Move to X reg
A 60AF 35                   TXS                      and copy to stack pointer
A 60B0 8609                 LDAA   #9                Put next 9 bytes from host onto stack
A 60B2 20AF                 BRA    TWRITMEM
A                           *
A                           * An SWI interrupt was generated
A                           *
A 60B4            SWISRV    EQU    *                 Breakpoints generated by SWI
A 60B4 864A                 LDAA   #BRKCODE          Force host to process breakpoints
A 60B6 8DD0                 BSR    OUTSCI            by sending it the break signal
A                           *
A                           * SWIIDLE is the infinite loop which allows the 'STOPPED' mode of PCbug11
A                           *
A 60B8 0E          SWIIDLE  CLI
A 60B9 20FD                 BRA    SWIIDLE           then wait for response
A                           *
A 60BB            SWISRV1   EQU    *
A 60BB 814A                 CMPA   #BRKACK           If host acknowledged then
```

```
A 60BD 26B9        BNE   RXSRVEX      
A 60BF 30          TSX                move SP to SWI stack frame and
A 60C0 C609        LDAB  #9
A 60C2 3A          ABX
A 60C3 35          TXS
A 60C4 EC07        LDD   7,X          send user code BP return address
A 60C6 8DC0        BSR   OUTSCI       (high byte first)
A 60C817           TBA
A 60C9 8DBD        BSR   OUTSCI       (low byte next)
A 60CB CC60B8      LDD   #SWIIDLE     force idle loop on return from BP
A 60CE ED07        STD   7,X
A 60D0 20C7        BRA   INH1A        but first return all MCU registers
A           *
A                        END
```

SYMBOL TABLE: Total Entries=42

BAUD	002B	REGBASE	1000
BOOTVECT	00C4	RXSRV	603C
BRKACK	004A	RXSRV1	605E
BRKCODE	004A	RXSRVEX	6078
FE	0002	SCCR1	002C
IDLE	6032	SCCR2	002D
INH1	6095	SCDR	002F
INH1A	6099	SCISRV	6035
INH2	60A5	SCSR	002E
INSCI	6079	SETVECT	6009
JCOP	00FA	STACK	003F
JILLOP	00F7	SWIIDLE	60B8
JMPEXT	007E	SWISRV	60B4
JSCI	00C4	SWISRV1	60BB
JSWI	00F4	TALKBASE	6000
JXIRQ	00F1	TDRE	0080
NULLSRV	6078	TLKRSTAR	6000
OR	0008	TREADMEM	6051
OUTSCI	6088	TWRITMEM	6063
OUTSCI1	608A	USERSTAR	6020
RDRF	0020	WAITPOLL	606B

Total errors: 0

LISTING 3 – TALKSCI.MAP

```
Name of constant must not exceed 14 characters.
Value of constant must start in column 15 or higher.

talker_start          $6000     Talker code start address.
talker_idle           $6032     Talker code idle loop address.
user_start            $6020     User's reset entry into talker code.
xirq_ujmp             $00F2     Address in talker code of user's XIRQ server address.
relocate_buf          $00A0     Address to where user's code is relocated on break point.
xirq_srv              $6035     Talker's XIRQ service address.
swi_srv               $60B4     Talker's SWI service address for break points.
swi_idle              $60B8     Talker's SWI idle loop.
null_srv              $6078     Talker RTI.
xirq_jmp              $00F2     XIRQ vector.
swi_jmp               $00F5     SWI vector.
cme_jmp               $00FE     COP clock monitor vector.
```


Literature Distribution Centres:
EUROPE: Motorola Ltd., European Literature Centre, 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
ASIA PACIFIC: Motorola Semiconductors (H.K.) Ltd., Silicon Harbour Center, No. 2, Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.
JAPAN: Nippon Motorola Ltd., 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan.
USA: Motorola Literature Distribution, P.O. Box 20912, Phoenix, Arizona 85036.