MOTOROLA SEMICONDUCTOR **APPLICATION NOTE**

AN456

Using PCbug11 as a Diagnostic Aid for - W Droho Expanded Mode M68HC11 Systems

By Steven McAslan. Motorola I td. 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.

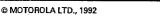
The command interpreter has four main functions and two simple communications handlers. The functions are:

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.

OTOROLA



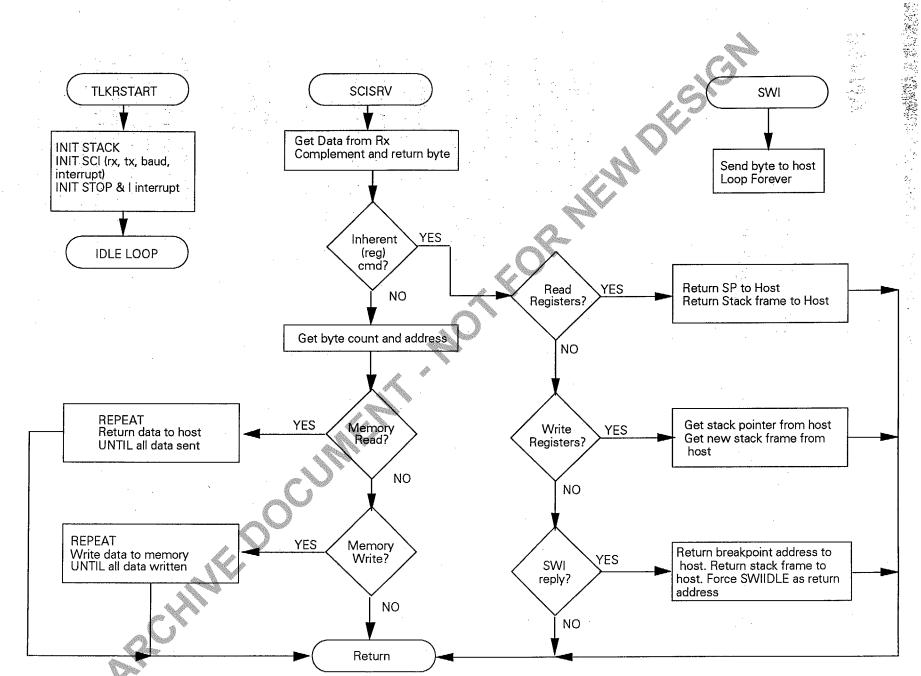


Figure 1. TALKE.ASC Flow Chart

MOTOROLA 2 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

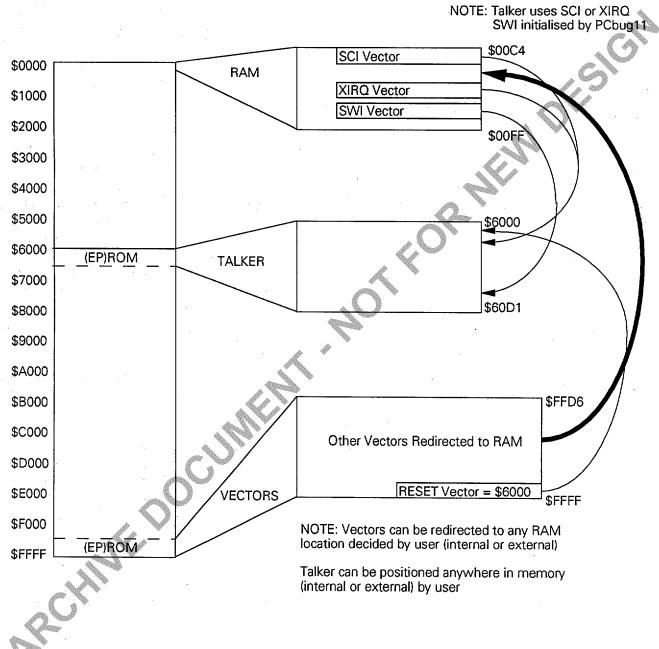
be altered according to the function in use. In bootstrap mode all of the interrupt vectors point to RAM. From RAM, an appropriate jump to an interrupt service routine can be carried out. This allows the interrupt vectors to be easily customised for the PCbug11 environment. In expanded mode the interrupt 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.)

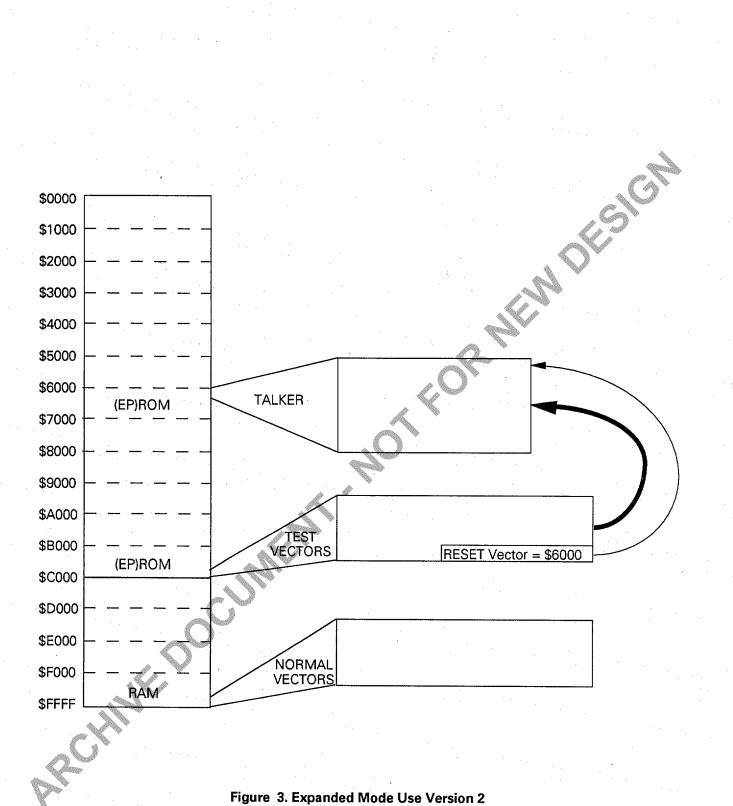
Another approach is to use the special test mode of the M68HC11 MCU. This mode is normally used for factory test purposes, as it allows access to normally protected features of the chip. However, it does have a notable additional feature, which is that the interrupt and reset vectors are transposed from 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.

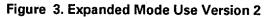
Special test mode could be accessed using a switch or key on the system. The talker interrupt vectors could be placed at the special mode interrupt locations or the interrupt locations could point into RAM; cf. figure 2. In either case, the talker could be placed in some spare area of memory (the talker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









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 ARCHINE DOCUMENT - MOTOR MENDERS include a debugging aid for any expanded mode M68HC11 system. If a single chip system is used,

REFERENCES

- [1] PCbug11 User Manual, Motorola M68PCBUG11/D1
- [2] M68HC11 Reference Manual, Motorola M68HC11RM/AD

LISTING 1 – TALKE.ASC ASSEMBLY LISTING

M68HC11 Absolute Assemb	oler	Vera	ion 2.70g:talke	ASC
1 A				ASC 6/3/90 ************************************
2 A			pht 1988,1990	
3 A				Communication routines for 68HC11
4 A 5 A	* montcor	• Provide	es low level mem	ory and stack read/write operations.
5 A -	* This ta	lkor DOR	S NOT use XIRQ	
7 A	*		MOI USE MINY	
8 A	*			
9 A	* Works w	ith Host	user interface	program PCBUG11.EXE.
10 A	* N.B. TA	LKE.ASC	is designed to b	e downloaded through standard type of
11 A	* bootloa	der, and	communicate wit	h host through SCI.
12 A		otloader	relies on 4 cha	r idle line on SCI to terminate.
13 A	*		*	
14 A	* CONSTAN			
15 A 0000	TALKBASE	equ	\$0000	
16 A 00C4	BOOTVECT	egu	\$00C4	Start of bootstrap vector jump table.
17 A 01FF 18 A 1000	STACK REGBASE	equ	\$01FF \$1000	
19 A	KEGBASE	egu	\$1000	
20 A 00C4	JSCI	equ	\$00C4	
21 A 00F1	JXIRQ	equ	\$00F1	
22 A 00F4	JSWI	equ	\$00F4	
23 A 00F7	JILLOP	equ	\$00F7	
24 A 00FA	JCOP	equ	\$00FA	
25 A 008	uS500	equ	5000/35	Delay with DEY/BNE loop
26 A 007E 27 A 004A	JMPEXT BRKCODE	egu	\$7E	Mnemonic for jump extended
28 A 004A	BRKACK	egu egu	\$4A \$4A	Break point signal code to host. Break point acknowledge code
29 A	*	cqu	ý tr	bieak point acknowledge code
30 A	* REGISTE	RS	٩	
31 A 002B	BAUD	egu	\$2B	
32 A 002C	SCCR1	equ	\$2C 🥢 🛷	
33 A 002D	SCCR2	equ	\$2D	
34 A 002E	SCSR	equ	\$2E	
35 A 002F	SCDR	equ	\$2F	
36 A	*	<i>M</i>	40.0	
37 A 0020 38 A 0080	RDRF TDRE	equ	\$20	
39 A	TDRE *	equ	\$80	
40 A	* PROGRAM	<i>₹ ¶ _</i> ∂		
41 A 0000	r noolii u	org	TALKBASE	
42 A 0000	TLKRSTART	EQU	*	
43 A 0000 8E01FF		LDS	#STACK	
44 A 0003 CE1000		LDX	#REGBASE	
45 A 0006 6F2C	S	CLR	SCCR1,X	
46 A 0008 CC302C	s ser i	LDD	#\$302C	
47 A 000B A72B 48 A 000D E72D	Car -	STAA	BAUD, X	Init SCI to 9600 baud, no parity
49 A 000F 8640	W.	STAB LDAA	SCCR2,X	and enable SCI tx & rx.
50 A 0011 06		TAP	#\$4 0	Enable STOP & I bit, disable XIRQ.
51 A		*		
52 A 0012 7E0012	IDLE	JMP	IDLE	Wait for SCI interrupt from host.
53 A				jump destination to start of user code.
54 A	*			
55 A 0015	SCISRV	EQU	*	On detecting interrupt,
56 A 0015 B6102E		LDAA	SCSR+REGBASE	assume receiver caused it.
57 A 0018 8420		ANDA	#RDRF	
58 A 001A 27F9 59 A	*	BEQ	SCISRV	otherwise program will hang up
60 A 001C	RXSRV	EQU	*	
61 A 001C B6102F	MANDIN V	LDAA	SCDR+REGBASE	Talker code processes data.
62 A 001F 43		COMA	SCON HEGENGE	Get command byte, & echo as ack Inverted
63 A 0020 8D46		BSR	OUTSCI	to host.
64 A 0022 2A51		BPL	INH1	If bit 7 set, process inh. command
65 A 0024 8D33		BSR	INSCI	else read byte count into B

66 A 0026 8F		XGDX		Save command and byte count.
67 A 0027 8D30		BSR	INSCI	Read high address byte
68 A 0029 17		TBA		into ACCA
69 A 002A 8D2D		BSR	INSCI	then low address byte into ACCB
70 A 002C 8F		XGDX		Cmd in A, count in B, addr in X
71 A 002D		CMPA	#\$FE	
72 A 002F 260D		BNE	RXSRV1	If command is memory read, then
73 A	*	22	10101112	11 Johnand 15 Monoly 1644, Chem
74 A 0031	TREADMEM	EQU	*	REPEAT
75 A 0031 A600	11121011211	LDAA	, X	read required address
76 A 0033 8D33		BSR	OUTSCI	send it to host
77 A 0035 17		TBA		save byte count)
78 A 0036 8D21	· · · ·	BSR	INSCI	and wait for acknowledge
79 A 0038 16		TAB		(restore byte count)
80 A 0039 08		INX		Increment address
81 A 003A 5A		DECB		Decrement byte count
82 A 003B 26F4		BNE	TREADMEM	UNTIL all done
83 A 003D 3B		RTI		return to idle loop or user code
84 A	*			
85 A 003E	RXSRV1	EQU	*	
86 A 003E 81BE	IGIDI(VI	CMPA	#\$BE	
87 A 0040 2616		BNE	RXSRVEX	If command is memory write then
88 A	*	DILL	NYON ATY	II command is memory write then
89 A 0042 17		TBA		move byte count to ACCA
90 A 0043			•	
91 A 0043 8D14	TWRITMEM	EQU	TNOOT	REPEAT
92 A 0045 E700		BSR	INSCI	Read next byte from host into B,
	· · ·		· , X	and store at required address.
93 A 0047 18CE0001 94 A 004B 1809		LDY	#\$0001	Set up wait loop
95 A 004B 1809	WAITPOLL	DEY		Y operand must be manually set
		BNE	WAITPOLL	
96 A 004F E600		LDAB	,X	Read stored byte and
97 A 0051 F7102F		STAB	SCDR+REGBASE	echo it back to host,
98 A 0054 08		INX		
99 A 0055 4A		DECA		Becrement byte count
100 A 0056 26EB	DVODUDV	BNE	TWRITMEM	UNTIL all done
101 A 0058	RXSRVEX	EQU	• ((ea)	and return
102 A 0058 3B	NULLSRV	RTI	· · · · · · · · · · · · · · · · · · ·	
103 A	INSCI	501	• • • · ·	
		EQU		
104 A 0059	10301			
105 A 0059 F6102E	INSCI	LDAB	SCSR+REGBASE	Wait for RDRF=1
105 A 0059 F6102E 106 A 005C C50A	TINGCT	BITB	#\$0A	If break detected
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0	INCL	BITB BNE	#\$0A TLKRSTART	
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420	114301	BITB BNE ANDB	#\$0A TLKRSTART #RDRF	If break detected
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5	INSCI	BITB BNE ANDB BEQ	#\$0A TLKRSTART #RDRF INSCI	If break detected then restart talker
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F	INSCI	BITB BNE ANDB BEQ LDAB	#\$0A TLKRSTART #RDRF	If break detected then restart talker then read data received from host
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39	INSCI	BITB BNE ANDB BEQ	#\$0A TLKRSTART #RDRF INSCI	If break detected then restart talker
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A		BITB BNE ANDB BEQ LDAB RTS	#\$0A TLKRSTART #RDRF INSCI	If break detected then restart talker then read data received from host and return with data in ACCB
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068	* outsci	BITB BNE ANDB BEQ LDAB RTS EQU	#\$0A TLKRSTART #RDRF INSCI	If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified.
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F	* OUTSCI	BITB BNE ANDB BEQ LDAB RTS EQU XGDY	#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE	If break detected then restart talker then read data received from host and return with data in ACCB
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E		BITB BNE ANDB BEQ LDAB RTS EQU XGDY LDAA	#\$0A TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE	If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA.
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0064 F6102F 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB	* OUTSCI	BITE BNE ANDE BEQ LDAB RTS EQU XGDY LDAA BPL	#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE	If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified.
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F	* OUTSCI	BITE BNE ANDE BEQ LDAB RTS EQU XGDY LDAA BPL XGDY	#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1	If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0071 B7102F	* OUTSCI	BITB BNE ANDB BEQ IDAB RTS EQU XGDY IDAA BPL XGDY STAA	#\$0A TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE	If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA.
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0071 B7102F 119 A 0074 39	* OUTSCI	BITE BNE ANDE BEQ LDAB RTS EQU XGDY LDAA BPL XGDY	#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1	If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 0068 B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0071 B7102F 119 A 0074 39 120 A	OUTSCI OUTSCI	BITB BNE ANDB BEQ LDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS	#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1	If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0071 B7102F 119 A 0074 39 120 A 121 A 0075	* OUTSCI	BITB BNE ANDB BEQ LDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU	#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1 SCDR+REGBASE *	If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag Important - Updates CCR!
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0075 1817E	OUTSCI OUTSCI	BITB BNE ANDB BEQ LDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU CMPA	<pre>#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1 SCDR+REGBASE * #\$7E</pre>	If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0071 B7102F 119 A 007T B7102F 119 A 0075 122 A 0075 817E 123 A 0077 260C	• OUTSCI OUTSCI INH1	BITB BNE ANDB BEQ IDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU CMPA BNE	#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1 SCDR+REGBASE *	If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag Important - Updates CCR!
 105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0071 B7102F 119 A 0074 39 120 A 121 A 0075 122 A 0075 817E 123 A 0077 260C 124 A 	• OUTSCI1 • INH1 *	BITB BNE ANDB BDQ LDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU CMPA BNE	<pre>#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1 SCDR+REGBASE * #\$7E</pre>	<pre>If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag Important - Updates CCR! If command is read registers then</pre>
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0071 B7102F 119 A 0074 39 120 A 121 A 0075 122 A 0075 817E 123 A 0077 260C 124 A 125 A 0079 30	• OUTSCI OUTSCI INH1	BITB BNE ANDB BDQ LDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU CMPA BNE TSX	<pre>#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1 SCDR+REGBASE * #\$7E</pre>	<pre>If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag Important - Updates CCR! If command is read registers then Move stack pointer to X</pre>
 105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 0067 188F 118 A 0071 B7102F 119 A 0074 39 120 A 121 A 0075 122 A 0075 817E 123 A 0079 30 126 A 007A 8F 	• OUTSCI1 • INH1 *	BITB BNE ANDB BEQ LDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU CMPA BNE TSX XGDX	<pre>#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1 SCDR+REGBASE * #\$7E INH2</pre>	<pre>If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag Important - Updates CCR! If command is read registers then Move stack pointer to X then to ACCD</pre>
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0064 F6102F 112 A 113 A 0068 114 A 0068 188F 115 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0071 B7102F 119 A 0074 39 120 A 121 A 0075 122 A 0075 817E 123 A 0077 260C 124 A 125 A 0079 30 126 A 007A 8F 127 A 007B 8DEB	• OUTSCI1 • INH1 *	BITB BNE ANDB BEQ LDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU CMPA BNE TSX XGDX BSR	<pre>#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1 SCDR+REGBASE * #\$7E</pre>	<pre>If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag Important - Updates CCR! If command is read registers then Move stack pointer to X</pre>
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0064 F6102F 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0071 B7102F 119 A 0074 39 120 A 121 A 0075 122 A 0075 817E 123 A 0077 260C 124 A 125 A 0079 30 126 A 007A 8F 127 A 007B 8DEB 128 A 007D 17	• OUTSCI1 • INH1 *	BITB BNE ANDB BEQ LDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU CMPA BNE TSX XGDX BSR TBA	<pre>#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1 SCDR+REGBASE * #\$7E INH2 OUTSCI</pre>	<pre>If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag Important - Updates CCR: If command is read registers then Move stack pointer to X then to ACCD send SP to host (high byte first)</pre>
 105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 007F 188F 118 A 0075 817E 122 A 0075 817E 123 A 0077 260C 124 A 125 A 0079 30 126 A 0079 30 126 A 0078 8F 127 A 0078 8DEB 128 A 0071 17 129 A 0078 8DE8 	• OUTSCI1 • INH1 *	BITB BNE ANDB BEQ LDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU CMPA BNE TSX XGDX BSR TBA BSR	<pre>#SOA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI1 SCDR+REGBASE * #\$7E INH2</pre>	<pre>If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag Important - Updates CCR! If command is read registers then Move stack pointer to X then to ACCD send SP to host (high byte first) then low byte</pre>
 105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0075 817E 122 A 0075 817E 123 A 0077 260C 124 A 125 A 0079 30 126 A 0079 30 126 A 0078 8F 127 A 0078 8DEB 128 A 0071 17 129 A 0078 8DE8 130 A 0080 30 	• OUTSCI1 • INH1 *	BITB BNE ANDE BEQ IDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU CMPA BNE TSX XGDX BSR TBA BSR TSX	<pre>#\$ OA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI SCDR+REGBASE * #\$7E INH2 OUTSCI OUTSCI</pre>	<pre>If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag Important - Updates CCR! If command is read registers then Move stack pointer to X then to ACCD send SP to host (high byte first) then low byte Restore X (=stack pointer)</pre>
105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 0068 188F 115 A 0068 B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 007D B7102F 119 A 0074 39 120 A 121 A 0075 122 A 0075 817E 123 A 0077 260C 124 A 125 A 0079 30 126 A 0078 8F 127 A 0078 8DEB 128 A 007D 17 129 A 007E 8DE8 130 A 0080 30 131 A 0081 C609	• OUTSCI1 • INH1 *	BITB BNE ANDE BDQ DDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU CMPA BNE TSX XGDX BSR TBA BSR TSX LDAB	<pre>#\$0A TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI SCDR+REGBASE * #\$7E INH2 OUTSCI OUTSCI GUTSCI #9</pre>	<pre>If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag Important - Updates CCR! If command is read registers then Move stack pointer to X then to ACCD send SP to host (high byte first) then low byte Restore X (=stack pointer) then return 9 bytes on stack</pre>
 105 A 0059 F6102E 106 A 005C C50A 107 A 005E 26A0 108 A 0060 C420 109 A 0062 27F5 110 A 0064 F6102F 111 A 0067 39 112 A 113 A 0068 114 A 0068 188F 115 A 006A B6102E 116 A 006D 2AFB 117 A 006F 188F 118 A 0075 817E 122 A 0075 817E 123 A 0077 260C 124 A 125 A 0079 30 126 A 0079 30 126 A 0078 8F 127 A 0078 8DEB 128 A 0071 17 129 A 0078 8DE8 130 A 0080 30 	• OUTSCI1 • INH1 *	BITB BNE ANDE BEQ IDAB RTS EQU XGDY LDAA BPL XGDY STAA RTS EQU CMPA BNE TSX XGDX BSR TBA BSR TSX	<pre>#\$ OA TLKRSTART #RDRF INSCI SCDR+REGBASE * SCSR+REGBASE OUTSCI SCDR+REGBASE * #\$7E INH2 OUTSCI OUTSCI</pre>	<pre>If break detected then restart talker then read data received from host and return with data in ACCB Only register Y modified. Enter with data to send in ACCA. MS bit is TDRE flag Important - Updates CCR! If command is read registers then Move stack pointer to X then to ACCD send SP to host (high byte first) then low byte Restore X (=stack pointer)</pre>

134 A	0085		INH2
135 A	0085	813E	
136 A	0087	2612	
137 A			*
138 A		8DCE	
139 A	008B	17	
140 A		8DCB	
141 A		8F	
142 A	008F		
	0090		
144 A 145 A	0092	20AF	*
145 A	0094		SWISRV
140 A 147 A	0094	864A	OWIDKY
147 A 148 A	0094	8DD0	
148 A 149 A	0098	0E	SWIIDLE
140 A	0099	20FD	DATIDBO
151 A	0055	2010	*
152 A	009B		SWISRV1
153 A		814A	511251112
154 A		26B9	
155 A		30	
156 A		C609	
157 A		3A	
158 A	00A3	35	
159 A	00A4	EC07	
160 A	00A6	8DC0	·
161 A	00A8	17	
162 A	00A9	8DBD	
163 A	00AB	CC009	98
164 A	00AE	ED07	
165 A	00B0	2007	
166 A			*
167 A	00C4		ORG
168 A			
169 A		0015	
170 A		7E	
171 A		0058	
172 A		7E	
173 A		0058	
174 A 175 A		7E 0058	
175 A		7E	
170 A		0058	
178 A		7E	
179 A		0058	
180 A		7E	
181 A		0058	and the second se
182 A		7E	
183 A			
184 A		7E	and all
185 A	00DD	0058	
186 A	OODF	7E	- × - V
187 A	00E0	0058	.C.A.
188 A	00E2	7E	
189 A	00E3	0058	
	00E5	- 1660 - 1660	
191 A	1922	0058	P
192 A	500 m		
193 A	00E9		
194 A	Children.	7E	
195 A		0058	
196 A		7E	
197 A		0058	
198 A		7E	
199 A			
200 A 201 A			
ZUI A	UUF5	0058	

EQU	*
CMPA	#\$3E
BNE	SWISRV1
BSR	INSCI
TBA	
BSR	INSCI
XGDX	
TXS	
LDAA	#9
BRA	TWRITMEM
POU	* .
EQU LDAA	#BRKCODE
BSR	OUTSCI
CLI	001301
BRA	SWIIDLE
EQU	*
CMPA	#BRKACK
BNE	RXSRVEX
TSX	
LDAB	#9
ABX	
TXS	
LDD	7,X
BSR	OUTSCI
TBA	
BSR	OUTSCI
LDD	#SWIIDLE
STD	7,X
BRA	INH1A
	m
BOOTVEC	
FCB FDB	JMPEXT SCISRV
FCB	JMPEXT
FDB	NULLSRV
FCB	JMPEXT
FDB	NULLSRV
FCB	JMPEXT
	NULLSRV
FCB 🦿	JMPEXT
FDB	NULLSRV
FCB	JMPEXT
FDB	NULLSRV
FCB	JMPEXT
FDB	NULLSRV
FCB	JMPEXT
FDB	NULLSRV
FCB	JMPEXT
FDB	NULLSRV
FCB	JMPEXT
FDB FCB	NULLSRV JMPEXT
FDB	NULLSRV
FCB	JMPEXT
FDB	NULLSRV
FCB	JMPEXT
FDB	NULLSRV
FCB	JMPEXT
FDB	NULLSRV
FCB	JMPEXT
FDB	
	NULLSRV
FCB	NULLSRV JMPEXT
FCB FDB	
	JMPEXT
FDB	JMPEXT NULLSRV

If command is write registers then get SP from host (High byte first) Move to X reg and copy to stack pointer Then put next 9 bytes on to stack -5101 Breakpoints generated by SWI Force host to process breakpoints by sending it the break signal then wait for response from host. If host has acknowledged BP then move SP to SWI stack frame and Send user code BP return address (high byte first) (low byte next) ٥, force idle loop on return from BP but first return all registers to host Jump table only during bootstrap SCI SPI (Unused vectors point to RTI) Pulse acc. Input Edge Pulse acc. Overflow Timer Overflow OC5 OC4 OC3 OC2 OC1 IC3 IC2 IC1 Real Time Intr IRQ XIRQ

SWI Changed by Break point

202 A 00F7 7E 203 A 00F8 0000 204 A 00FA 7E 205 A 00FB 0058 206 A 00FD 7E 207 A 00FE 0058 208 A 209 A SYMBOL TABLE:	국 국 국 국 *	CB JMPEXT DB TLKRSTART CB JMPEXT DB NULLSRV CCB JMPEXT DB NULLSRV SND S39	ILLOP COP Fail Clock Monitor	
BAUD BOOTVECT BRKACK BRKCODE IDLE INH1 INH1A INH2 INSCI JCOP JILLOP JMEXT JSCI JSWI JXIRQ NULLSRV OUTSCI OUTSCI RDRF REGBASE	00C4 R 004A R 004A S 0012 S 0075 S 0076 S 0077 S 0077 S 0077 S 0077 S 00764 T 0071 T 0058 T 0068 T 0068 T 006A W	XXSRV XXSRV1 XXSRVEX CCCR1 CCCR2 CCR2 CCR8 CISRV CCSR TACK WIIDLE WISRV WIIDLE WISRV1 TALKBASE TORE LKRSTAR READMEM WRITMEM WAITPOLL SS00	001C 003E 0058 002C 002D 002F 0015 002E 01FF 0098 0094 0098 0094 0098 0090 0000 0080 0000 0031 0043 004B 008E	AFFINI DESIG
Total errors: 0				
		MERI		
	2007			
ARCHINE				

LISTING 2 - TALKSCI.ASC ASSEMBLY LISTING

M68HC11 Absolute Assemble	r	Version	2.70g:talk	sci.ASC
1 A 2 A 3 A 4 A	* Motorola (* MCU reside	Copyrigh ≥nt, Int	t 1988,1991 errupt driv	LKSCI.ASC 14/8/91 ************************************
5 A 6 A 7 A	*		NOT use XIR	
8 A 9 A 10 A	* placed in	the MC	J memory map	l purpose talker. It is intended to be at \$6000 but this can be changed by
11 A 12 A 13 A 14 A	* and can be * initialise	e used i ed. It i	in any mode Is therefore	ress. The talker is for general debug as long as the vectors are correctly useful for normal modes. The SCI is e TALKACIA when an external ACIA is
14 A 15 A 16 A 17 A 18 A	* to be used * pointing (* IMPORTANT	3. TALKS to RAM i : If yo	SCI assumes in the same ou change th	that the interrupt vectors are way as the boostrap ROM. e running address of this program e TALKSCI.MAP file so that the two
19 A 20 A 21 A	* match. *			h option TALKSCI, a 10mS break is
22 A 23 A 24 A				prior to establishing communication.
25 A 6000 26 A 003F 27 A 00C4	TALKBASE STACK BOOTVECT	egu egu egu	\$6000 \$003F \$00C4	User may alter this parameter Start of bootstrap vector jump table.
28 A 1000 29 A 30 A 00C4 31 A 00F1	REGBASE * JSCI JXIRQ	equ	\$1000 \$00C4 \$00F1	Change if registers are moved
32 A 00F4 33 A 00F7 34 A 00FA	JSWI JILLOP JCOP	equ equ equ equ	\$00F4 \$00F7 \$00FA	
35 A 007E 36 A 004A 37 A 004A 38 A	JMPEXT BRKCODE BRKACK	equ equ equ	\$7E \$4A \$4A	Mnemonic for jump extended Break point signal code to host. Break point acknowledge code
39 A 40 A 002B 41 A 002C	* REGISTERS BAUD SCCR1	equ equ	\$2B \$2C	Change if required for MCU
42 A 002D 43 A 002E 44 A 002F	SCCR2 SCSR SCDR	equ equ equ	\$2D \$2E \$2F	
45 A 46 A 0020 47 A 0080 48 A 0008	RDRF TDRE OR	equ equ equ	\$20 \$80 \$08	SCI Masks, change if required
49 A 0002 50 A 51 A 52 A 6000	FE * * PROGRAM	equ	\$02	
52 A 6000 53 A 54 A 55 A	org * * Initialis *	TALKBAS	SE CI and inter	rupts
56 A 6000 57 A 6000 867E 58 A 6002 18CE6078	TLKRSTART LDY	EQU LDAA #NULLSH	#JMPEXT XV	Dynamically set up Boot jump table.
59 A 6006 CE00C4 60 A 6009 61 A 6009 A700	SETVECT	LDX EQU STAA	#B00TVECT * , X	
62 A 600B 08 63 A 600C 1AEF00 64 A 600F 08 65 A 6010 08		INX STY INX INX	,X	

66 A 6011 8C0100		CPX	#\$100	
67 A 6014 26F3		BNE	SETVECT	
68 A 6016 CE6035		LDX	#SCISRV	
69 A 6019 DFC5		STX	JSCI+1	
70 A 601B CE6000		LDX	#TLKRSTART	
71 A 601E DFF8		STX	JILLOP+1	
72 A	*			
73 A 6020	USERSTART E	าม	* .	
74 A 6020 8E003F	ODDIND IMAA Da	~		
		LDS	#STACK	
75 A 6023 CE1000		LDX	#REGBASE	
76 A 6026' 6F2C		CLR	SCCR1,X	
77 A 6028 CC302C		LDD	#\$302C	
78 A 602B A72B		STAA	BAUD, X	Initialise SCI to 9600, no parity
79 A 602D E72D		STAB	SCCR2,X	and enable SCI tx & rx.
80 A 602F 8640		LDAA	#\$40	Enable STOP, I interrupts, disable X 🥂 🔪
81 A 6031 06		TAP		
82 A	* 15 1			
83 A	* User mav	ລດີດີ ຈຳນຫ	n to his own code	here or may move the above
84 A				
	- Inicialis	sacron c	o the start of his	s own program.
85 A	* .			
86 A 6032 7E6032	IDLE	JMP -	IDLE	Now hang around for SCI interrupt
87 A	*			
88 A 6035	SCISRV	EQU	*	On detecting interrupt,
89 A 6035 B6102E	0010114		COOD DECENTO	
		LDAA	SCSR+REGBASE	assume receiver caused it.
90 A 6038 8420		ANDA	#RDRF	
91 A 603A 27F9		BEQ	SCISRV	otherwise program will hang up
92 A	*			
93 A 603C	RXSRV	FOU	+	m-1)
	KASKV	EQU	•	Talker code processes rec'd data.
94 A 603C B6102F		LDAA	SCDR+REGBASE	Read command byte, & echo as ack
95 A 603F 43		COMA		Inverted
96 A 6040 8D46		BSR	OUTSCI	to host
97 A 6042 2A51		BPL	INH1	If bit 7 set, then process inh cmd
98 A 6044 8D33		BSR	INSCI	else read byte count from host into B
99 A 6046 8F		XGDX		Save command and byte count.
100 A 6047 8D30		BSR	INSCI 🐁	Read high address byte
101 A 6049 17		TBA		into ACCA
102 A 604A 8D2D		BSR	INSCI	
			TUSCI	then low address byte into ACCB
103 A 604C 8F		XGDX	10 M	Cmd in A, count in B, addr in X
104 A 604D 81FE		CMPA	#\$FE	
105 A 604F 260D		BNE	RXSRV1	If command is memory read, then
106 A	*		A CARLER AND A CARLE	
107 A 6051	TREADMEM	EQU 🏑	*	
	INEADMEN			REPEAT
108 A 6051 A600		LDAA	, Х	read required address
109 A 6053 8D33 -			5. <i>389</i>	Tord Todattod dddTCDD
105 11 0055 0555		BSR	OUTSCI	send it to host
110 A 6055 17		BSR TBA	5. <i>389</i>	-
		TBA	OUTSCI	send it to host (save byte count)
110 A 6055 17 111 A 6056 8D21		TBA BSR	5. <i>389</i>	send it to host (save byte count) and wait for acknowledge
110 A 6055 17 111 A 6056 8D21 112 A 6058 16		TBA BSR TAB	OUTSCI	send it to host (save byte count) and wait for acknowledge (restore byte count)
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08	G	TBA BSR TAB INX	OUTSCI	send it to host (save byte count) and wait for acknowledge
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A	, cì	TBA BSR TAB	OUTSCI	send it to host (save byte count) and wait for acknowledge (restore byte count)
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08	SCI.	TBA BSR TAB INX	OUTSCI	send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4	Joch	TBA BSR TAB INX DECB BNE	OUTSCI INSCI	send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B	,0 ⁰¹	TBA BSR TAB INX DECB	OUTSCI INSCI	send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A	ooci	TBA BSR TAB INX DECB BNE RTI	OUTSCI INSCI	send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E	RXSRV1	TEA BSR TAB INX DECB BNE RTI EQU	OUTSCI INSCI TREADMEM	send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 6058 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 119 A 605E 81BE	RXSRV1	TBA BSR TAB INX DECB BNE RTI	OUTSCI INSCI	send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 6058 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 119 A 605E 81BE	RXSRV1	TEA BSR TAB INX DECB BNE RTI EQU CMPA	OUTSCI INSCI TREADMEM * #\$BE	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code.</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 119 A 605E 81BE 120 A 6060 2616	RXSRV1	TEA BSR TAB INX DECB BNE RTI EQU	OUTSCI INSCI TREADMEM	send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 119 A 605E 81BE 120 A 6060 2616 121 A	RXSRV1	TEA BSR TAB INX DECB BNE RTI EQU CMPA BNE	OUTSCI INSCI TREADMEM * #\$BE	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 119 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17	*	TBA BGR TAB INX DECB BNE RTI EQU CMPA BNE TBA	OUTSCI INSCI TREADMEM * #\$BE	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 119 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063	RXSRV1	TEA BSR TAB INX DECB BNE RTI EQU CMPA BNE	OUTSCI INSCI TREADMEM * #\$BE	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 119 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17	*	TBA BGR TAB INX DECB BNE RTI EQU CMPA BNE TBA	OUTSCI INSCI TREADMEM * #\$BE	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 119 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063	*	TEA BSR TAB INX DECB BNE RTI EQU CMPA BNE TEA EQU BSR	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB,</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 119 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063 124 A 6063 8D14 125 A 6065 E700	*	TEA BSR TAB INX DECB BNE RTI EQU CMPA BNE TEA EQU BSR STAB	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI ,X	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB, and store at required address.</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 6058 26F4 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063 124 A 6063 8D14 125 A 6065 E700 126 A 6067 18CE0001	* TWRITMEM	TBA BSR TAB INX DECB BNE RTI EQU CMPA BNE TBA EQU BSR STAB LDY	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB, and store at required address. Set up wait loop</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 6058 26F4 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063 8D14 125 A 6065 E700 126 A 6067 18CE0001 127 A 606B 1809	*	TBA BSR TAB INX DECB BNE RTI EQU CMPA BNE TBA EQU BSR STAB LDY DEY	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI ,X #\$0001	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB, and store at required address.</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063 124 A 6065 E700 126 A 6067 18CE0001 127 A 606B 1809 128 A 606D 26FC	* TWRITMEM	TBA BSR TAB INX DECB BNE RTI EQU CMPA BNE TBA EQU BSR STAB LDY	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI ,X	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB, and store at required address. Set up wait loop</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 6058 26F4 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063 8D14 125 A 6065 E700 126 A 6067 18CE0001 127 A 606B 1809	* TWRITMEM	TBA BSR TAB INX DECB BNE RTI EQU CMPA BNE TBA EQU BSR STAB LDY DEY	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI ,X #\$0001	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB, and store at required address. Set up wait loop Y may take on suitable value</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063 124 A 6065 E700 126 A 6067 18CE0001 127 A 606B 1809 128 A 606D 26FC 129 A 606F E600	* TWRITMEM	TBA BSR TAB INX DECB BNE RTI EQU CMPA BNE TBA EQU BSR STAB LDY DEY BNE LDAB	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI ,X #\$0001 WAITPOLL ,X	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB, and store at required address. Set up wait loop Y may take on suitable value Read stored byte and</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063 124 A 6063 8D14 125 A 6065 E700 126 A 6061 18CE0001 127 A 606B 1809 128 A 606D 26FC 129 A 606F E600 130 A 6071 F7102F	* TWRITMEM	TBA BSR TAB INX DECB BNE RTI EQU CMPA BNE TBA EQU BSR STAB LDY DEY BNE LDAB STAB	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI ,X #\$0001 WAITPOLL	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB, and store at required address. Set up wait loop Y may take on suitable value</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063 124 A 6063 8D14 125 A 6065 E700 126 A 6067 18CE0001 127 A 606B 1809 128 A 606D 26FC 129 A 606F E600 130 A 6071 F7102F 131 A 6074 08	* TWRITMEM	TEA BSR TAB INX DECB BNE RTI EQU CMPA BNE TBA EQU ESR STAB LDY DEY BNE LDAB STAB INX	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI ,X #\$0001 WAITPOLL ,X	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB, and store at required address. Set up wait loop Y may take on suitable value Read stored byte and echo it back to host,</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063 124 A 6063 8D14 125 A 6065 E700 126 A 6067 18CE0001 127 A 606B 1809 128 A 606D 26FC 129 A 606F E600 130 A 6071 F7102F 131 A 6074 08 132 A 6075 4A	* TWRITMEM	TBA BSR TAB INX DECB BNE RTI EQU CMPA BNE TBA EQU BSR STAB LDY DEY BNE LDAB STAB INX DECA	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI ,X #\$0001 WAITPOLL ,X SCDR+REGBASE	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB, and store at required address. Set up wait loop Y may take on suitable value Read stored byte and echo it back to host, Decrement byte count</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063 124 A 6063 8D14 125 A 6065 E700 126 A 6067 18CE0001 127 A 606B 1809 128 A 606D 26FC 129 A 606F E600 130 A 6071 F7102F 131 A 6074 08	* TWRITMEM	TEA BSR TAB INX DECB BNE RTI EQU CMPA BNE TBA EQU ESR STAB LDY DEY BNE LDAB STAB INX	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI ,X #\$0001 WAITPOLL ,X	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB, and store at required address. Set up wait loop Y may take on suitable value Read stored byte and echo it back to host,</pre>
110 A 6055 17 111 A 6056 8D21 112 A 6058 16 113 A 6059 08 114 A 605A 5A 115 A 605B 26F4 116 A 605D 3B 117 A 118 A 605E 81BE 120 A 6060 2616 121 A 122 A 6062 17 123 A 6063 124 A 6063 8D14 125 A 6065 E700 126 A 6067 18CE0001 127 A 606B 1809 128 A 606D 26FC 129 A 606F E600 130 A 6071 F7102F 131 A 6074 08 132 A 6075 4A	* TWRITMEM	TBA BSR TAB INX DECB BNE RTI EQU CMPA BNE TBA EQU BSR STAB LDY DEY BNE LDAB STAB INX DECA	OUTSCI INSCI TREADMEM * #\$BE RXSRVEX * INSCI ,X #\$0001 WAITPOLL ,X SCDR+REGBASE	<pre>send it to host (save byte count) and wait for acknowledge (restore byte count) Increment address Decrement byte count UNTIL all done and return to idle loop or user code. If command is memory write then move byte count to ACCA REPEAT Read next byte from host into ACCB, and store at required address. Set up wait loop Y may take on suitable value Read stored byte and echo it back to host, Decrement byte count</pre>

MOTOROLA 13

		•						
135	Α	6078	3B	NULL	SRV	RTI		
136				*				
137				* I	NSCI ge	ets the i	received byte form	the SCI
138		6079		INSC	т	EQU	*	
			F6102E	THOU	1	LDAB	SCSR+REGBASE	Wait for RDRF=1
		607C				BITB	# (FE+OR)	If break detected then
		607E				BNE	TLKRSTART	restart talker.
		6080				ANDB	#RDRF	
144	Α	6082	27F5			BEQ	INSCI	
145	Α	6084	F6102F			LDAB	SCDR+REGBASE	then read data received from host
146	A	6087	39			RTS		and return with data in ACCB
147	A			*				
148	Α			* C	UTSCI j	is the su	ubroutine which tra	nsmits a byte from the SCI
149				*			1 (C)	
		6088		OUTS	CI	EQU	*	Only register Y modified.
		6088		01100	~~ ~	XGDY		Enter with data to send in ACCA.
			B6102E	OUTS	C11	LDAA	SCSR+REGBASE	
		608D 608F				BPL XGDY	OUTSCI1	MS bit is TDRE flag
			B7102F			STAA	SCDR+REGBASE	Important - Updates CCR!
		6094				RTS	SCDRENBODADE	Important - opdates con
157				*				
158	Α			* N	low deci	ide which	h inherent command	was sent
159	A			* *				
160	Α	6095		INH1		EQU	* .	
161	A	6095	817E			CMPA	#\$7E	If command is read registers then
		6097	260C			BNE	INH2	
163				*				
164				* C	Command	was to :	read MCU registers	
165		c 0 0 0	20	*		mou		
		6099 609A		INH	IIA.	TSX		Move stack pointer to X
		609B				XGDX BSR	OUTSCI	then to ACCD
		609D				TBA	001301	send SP to host (high byte first)
		609E				BSR	OUTSCI	then low byte
		60A0				TSX		Restore X (=stack pointer)
172	Α	60A1	C609			LDAB	. #9	then return 9 bytes on stack
173	A	60A3	20AC			BRA	TREADMEM	i.e CCR, B, A, XH, XL, YH, YL, PCH, PCL
174	A			*				
175	Α			* C	Command	was to	write MCU registers	
176				*		đ		
		60A5		INH2		EQU	*	
		60A5				CMPA	#\$3E	If command is write registers then
180		60A7	2012	*		BNE	SWISRV1	
		60A9	8DCE		.	BSR	INSCI	get SP from bost (Nigh but o first)
		60AB			ALC IN	TBA	INSCI	get SP from host (High byte first)
		60AC			. 1	BSR	INSCI	get low byte next
184	A	60AE	8F	ALLES	all serves.	XGDX		Move to X reg
		60AF		e de la come	J	TXS		and copy to stack pointer
186	A	60B0	8609 《		annan.	LDAA	#9	Put next 9 bytes from host onto stack
187	A	60B2	20AF			BRA	TWRITMEM	
188				*				
189			. \\\	* A	n SWI :	interrup	t was generated	
190			A BANK	*				
		60B4	88. 998.	SWIS	SRV	EQU	*	Breakpoints generated by SWI
		60B4	· · · · · · · · · · · · · · · · · · ·			LDAA	*BRKCODE	Force host to process breakpoints
193	- 339	6086	סתניפ	*		BSR	OUTSCI	by sending it the break signal
194				* 0	WITDLE	is the	infinite loon which	allows the 'STOPPED' mode of PCbug11
195				*	, , , , , , , , , , , , , , , , , , ,	Ta cite.	initiate roop which	arrows the STORPED, Wode of ACDAGII
dan. White		60B8	0E		IDLE	CLI		
- MC+CT -		60B9				BRA	SWIIDLE	then wait for response
199				*				
200	Α	60BB		SWIS	RV1	EQU	*	
201	Α	60BB	814A			CMPA	#BRKACK	If host acknowledged then

	202 A 60BD	2689		BNE	RXSRVEX	
	203 A 60BF			TSX		
	204 A 60C0			LDAB	#9	
	205 A 60C2			ABX		
	206 A 60C3			TXS		
	200 A 60CJ			LDD	7,X	
	208 A 60C6			BSR	OUTSCI	
	209 A 60C81			TBA		
	210 A 60C9			BSR	OUTSCI	
	211 A 60CB	CC60B8		LDD	#SWIIDLE	
	212 A 60CE	ED07		STD	7,X	
	213 A 60D0	2007		BRA	INH1A	
	214 A		· *.			
	215 A				END	
	SYMBOL TABL	LE :	Total E	ntries=42		
	BAUD		002B	REGBAS	E	
	BOOTVECT		0004	RXSRV	-	
	BRKACK		004A	RXSRV1		
	BRKCODE	· · ·	004A	RXSRVE	X	
	FE		0002	SCCR1		
	IDLE		6032	SCCR2		
e.	INH1		6095	SCDR		
	INH1A		6099	SCISRV		
	INH2		60A5	SCSR		
	INSCI		6079	SETVEC	т	
	JCOP		00FA	STACK	1 - E - E - E E E E E E E	
	JILLOP		00F7	SWIIDL	E	
	JMPEXT		007E	SWISRV		
	JSCI		00C4	SWISRV		
	JSWI		00F4	TALKBA		
				TDRE	<u>.</u>	
	JXIRQ		00F1	TURE		
	NULLSRV		6078	TLKRST		
	NULLSRV OR		0008	TREADM	EM	
	NULLSRV				EM	4
	NULLSRV OR		0008	TREADM	EM EM	
	NULLSRV OR OUTSCI		0008 6088	TREADM TWRITM	EM EM AR	۹ ۵
	NULLSRV OR OUTSCI OUTSCI1		0008 6088 608A	TREADM TWRITM USERST	EM EM AR	۹ 1
	NULLSRV OR OUTSCI OUTSCI1	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	9 1
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	1
	NULLSRV OR OUTSCI OUTSCI1 RDRF	r5: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	1
· .	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	•
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: O	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	*
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	 . .
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: O	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	•••
· ·	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	4 1
· ·	NULLSRV OR OUTSCI OUTSCI1 RDRF	r5: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	*
•	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	(
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	•
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	*
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	*
· · · · · · · · · · · · · · · · · · ·	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	r5: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	• • • • • • • • • • • • • • • • • • •
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	• • • • • • • • • • • • • • • • • • •
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	r5: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: O	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	
	NULLSRV OR OUTSCI OUTSCI1 RDRF	rs: 0	0008 6088 608A	TREADM TWRITM USERST	EM EM AR	

move SP to SWI stack frame and

send user code BP return address (high byte first)

RAFFINITIES (low byte next) force idle loop on return from BP

but first return all MCU registers

1000 603C 605E 6078 002C 002D 002F 6035 002E 6009 003F 60B8 60B4 60BB 6000 0080 6000 6051 6063 6020 606B

LISTING 3 – TALKSCI.MAP

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

<pre>talker_start talker_idle user_start xirq_ujmp relocate_buf xirq_srv swi_srv swi_srv swi_idle null_srv xirq_jmp swi_jmp cme_jmp</pre>		\$6032 Talker \$6020 User's \$00F2 Address \$00A0 Address \$6035 Talker \$60B4 Talker \$6078 Talker \$00F2 XIRQ vs \$0055 SWI ve	vector.
н. н. ^н	· .		
	· .	· .	
	• • •		
	:		
		poc ^y .	
Motorala ree		make changes without furth	her notice to any products herein to improve reliability, function or design. Motorola does not assume

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola 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 Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola 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 Motorola was negligent regarding the design or manufacture of the part. Motorola and *C* are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

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.

M MOTOROLA