MicroNator

USER

UNIVERSAL
DEVELOPMENT BOARD
Version 4.04

User's Manual
Version 4.04a

RF-232

http://www.micronator.com

ISBN 2-9803460-1-2

Dépôt Légal - Bibliothèque Nationale du Québec, avril 1995.

PRINTED IN CANADA

MicroNator

UNIVERSAL
DEVELOPMENT BOARD
Version 4.04

User's Manual
Version 4.004a

RF-232

http://www.micronator.com

MicroNator

MicroNator UNIVERSAL DEVELOPMENT BOARD USER MANUAL

RF-232
1404 rue Galt
Montréal, Qc H4E 1H9
CANADA
Tél: (514) 761-4201

RF-232
21 rue André Gide
59123 ZUYDCOOTE
FRANCE
Tél: 03 28 58 28 39

micronator@micronator.com

ISBN 2-9803460-1-2

Dépôt Légal - Bibliothèque Nationale du Québec, avril 1995.

MicroNator

UNIVERSAL
DEVELOPMENT BOARD

User's Manual

Information contained in this manual applies to
Version (4.04) MicroNator UNIVERSAL DEVELOPMENT BOARD
serial numbers 4000 through 9999

PREFACE

Unless otherwise specified, all address references are in hexadecimal throughout this document.

An asterisk (*) following the signal denotes that the signal is asserted, valid, or true when the signal is low.

MicroNator, CPU-11/64e2, and UCT-11/64e2 System refer to the same system.

All Vcc & Gnd are shown on schematics.

Contents at a Glance

CHAPTER 1

GENERAL INFORMATION

1.1 INTRODUCTION

This manual provides general information, hardware preparation, installation, operating instructions, functional description, and support information for the Universal Development Board, hereafter referred to as MicroNator System or MicroNator "Central Processing Unit 68HC11 EEPROM").

Downloading S-record information are contained in appendix A.

1.2 SPECIFICATIONS

MCU	***TOSHIBA***	TMP68HC11**A0**T, or TMP68HC11**A1**T, or TMP68HC11**A8**T, or TMP68HC11**E0**T, or TMP68HC11**E1**T. (52 pins PLCC with socket)
	MOTOROLA	MC68HC11**A0**FN, or MC68HC11**A1**FN or MC68HC-11**A8**FN or MC68HC11**E0**FN, or MC68HC11**E1**FN. (52 pins PLCC with socket)
Clock:		4.9152 MHz, crystal controlled, giving 1.2288 MHz bus operation
Monitor Size:		8.1 KBytes: all replaceable by user code
Memory Size:		32 KBytes EEPROM & 32 KBytes of RAM
Real Time Clock		MC68HC68T1P @ ***32.768 KHz*** (default). [1.048576 MHz, 2.097152 MHz, or 4.194304 MHz].
MCU Extension I/O Ports		HCMOS compatible
Terminal I/O Port		RS-232C compatible: XON/XOFF protocol
Temperature		
Operating		-40 to +70°C
Storage		-55 to +125°C
Relative Humidity		0 to 90%: non-condensing
Power Requirements:		5 VDC @ 80 mA (MAX)

Communication Cable	AT-Cross-over: DB25F->DB9F
Dimensions:	
Width	4.75 in. (12.06 cm)
Length	5.725 in (14.54 cm)
Wire-wrap Area:	
Area	Approx. 4.3 in. square (10.92 cm^2)
Holes	437

OPTIONS:

- Casing — 6.06" x 6.25" x 2.5"
6.06" x 6.25" x 3.25"
- Expansion Board — To be inserted on top or on the bottom of CPU-11/64 system
- Extra Connector — BUS-Connector for additional wire wrap expansion board
- USER BOARD — LCD: (2 x 16) or (4 x 20)
Keyboard: (4 x 4) hex
- U I/O Board — 8 Opto-Input & 8 Relay-Output

1.3 FEATURES

- An economical means of debugging user assembled code and evaluating **68HC11A0/A1/A8/E0/E1** micro controller.
- Monitor/debugger firmware.
- One-line assembler/disassembler.
- PC downloading capability.
- 68HC11 based debugging/evaluating circuitry.
- RS-232C compatible using XON/XOFF communication.
- 20 I/O pins, IRQ*, XIRQ*, +5Vdc, and GND on the DB25F I/O

connector.

- Real Time Clock **INCLUDED**
- Communication Cable DB25F-DB9F **INCLUDED**
- Single (+5 Vdc) input power source **INCLUDED**
- 32 KBytes of EEPROM and 32 KBytes of RAM **INCLUDED**
- Motorola (AS11.EXE) Cross-Assembler **INCLUDED**
- Complete monitor program “MONITEUR” **INCLUDED**
- "C" language from DDSystem **OPTION**
- Casing: 6.06" x 6.25" x 2.5" or 6.06" x 6.25" x 3.25" **OPTION**

No jumper, no PCB TRACE to cut,
no EPROM eraser nor EPROM programmer required.

1.4 GENERAL DESCRIPTION

The MicroNator System provides a tool for designing, debugging, and evaluating 68HC11A1 Micro controller Unit **“MCU”** based system equipment. By providing all of the essential MCU timing and I/O circuitry, the MicroNator System simplifies user evaluation of his prototype hardware/software design. The MicroNator System requires a PC compatible computer with one COM port.

Entering data, program debugging, and programming external EEPROM is accomplished by the monitor EEPROM firmware via a IBM-PC, or compatible, connected to the MicroNator System SCI port connector.

Downloading programs directly from a IBM-PC, or compatible, to the MicroNator System is accomplished via the SCI port of the MCU.

The MicroNator System standard mode of operation is the expanded-multiplexed mode. It is also possible to put the MCU in single-chip mode by applying the proper signal on the CPU-BUS pins labelled MODA BUS pin # 61 and MODB BUS pin # 58.

1.4.1 Program Development

MCU code may be generated using the resident one-line assembler/disassembler, or may be downloaded from S-Record files to the user EEPROM through the PC serial port. User code may then be executed using various debugging commands in the monitor. User code may also be started using the

reset switch.

Independent 150 to 19.2K baud rate selection capabilities are provided for the SCI port connector. All baud rates are software selectable.

“MONITEUR” enables the user to, check, erase, program, verify, and copy the entire external EEPROM ***32 KBytes*** without the use of EPROM eraser or programmer.

1.5 EXTERNAL EQUIPMENT REQUIRED

• Host computer (PC compatible).

CHAPTER 2

HARDWARE PREPARATION

2.1 INTRODUCTION

This chapter provides unpacking instructions, hardware preparation, and installation instructions for the MicroNator System. For detailed hardware discussions refer to CHAPTER 6 .

2.2 UNPACKING INSTRUCTIONS

NOTE

If shipping carton is damaged upon receipt, request carrier's agent be present during unpacking and inspection of the MicroNator System package

Unpack MicroNator System from shipping box. Refer to packing list and verify that all items are present. Save packing material for storing or reshipping.

2.2.1 Registration

Fill the included registration card and mail it as soon as possible so as to be kept informed of the latest developments and to receive upgrades of "MONITEUR" free of charge for a period of one year.

2.3 HARDWARE PREPARATION

This portion of text describes the inspection/preparation of MicroNator System components prior to installation. This description will ensure the user that the MicroNator System components are properly connected together for system operation. The MicroNator System has been factory-tested before shipment.

MicroNator System should be inspected prior to installation. Figure: 1 illustrates the MicroNator System's connectors and switches:.

- ON/OFF (Switch SW2) is used for turning on/off the MicroNator System.

- Power LED (D3) Power-on LED.

- Reset (Switch SW1) is used for resetting the MicroNator System.

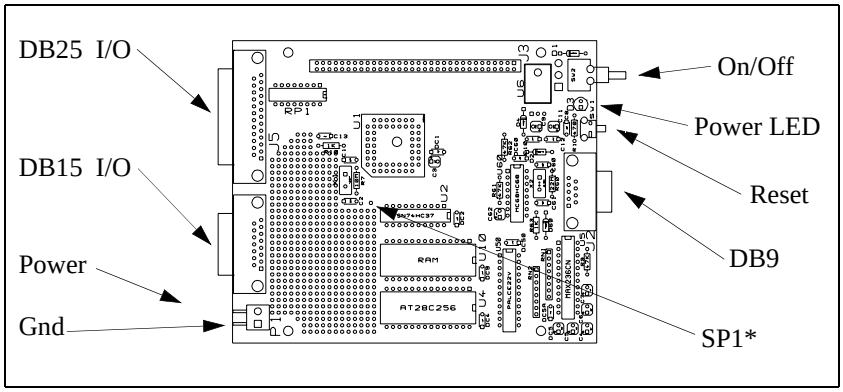


Figure: 1 MicroNator System

- DB9 (Connector J2) connects the MicroNator System to the PC.

- SP1 (SP1) spare chip-select for user WW area. Active LOW from $0240-$027F.

- Gnd (Connector P1-1) DC ground, square pad on PCB.

- POWER (Connector P1) wall-mounted power supply. Gnd is pin #1 (square).

- DB15 I/O Is used for interconnection of external I/O signals to the user WW area.

- DB25 I/O (Connector J5) is used for interconnection of external I/O signals to the MicroNator System.

2.3.1 Wall Mounted DC Power Supply input

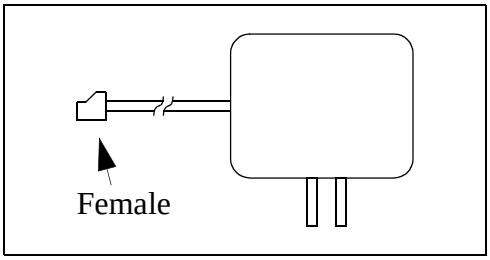


Figure: 2 Wall-mounted power supply

Figure: 2 shows the wall-mounted power supply. The unit plug into an AC wall socket and the small female connector plug into the MicroNator System. The user should take care to respect the polarity of the plug or he might damage the units.

2.3.2 Serial Cable

Figure: 3 shows the standard cable used for the communication between the PC and the system. The connector to be inserted into the serial port of the PC is a DB25 female socket and the one to be inserted into the MicroNator System is a DB9 female socket. The plug on the MicroNator System is a DB9 male connector.

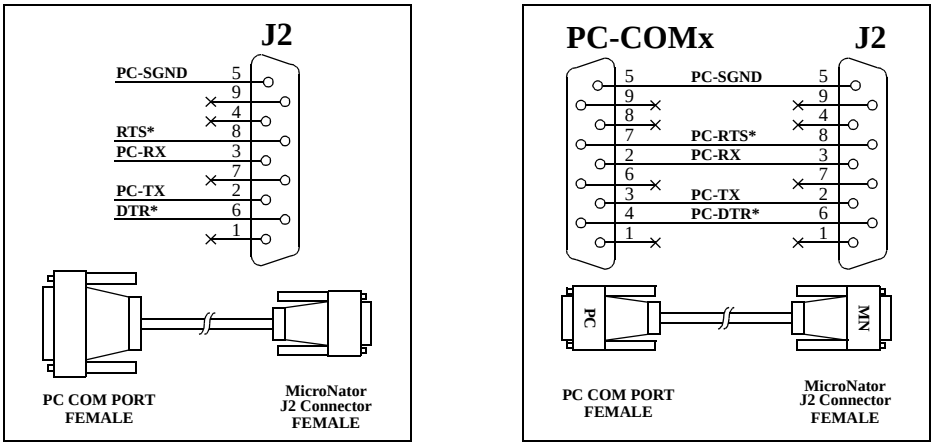


Figure: 3 Serial Connector & Signals

2.3.3 Software Installation

In order to install the software all the user has to do is to put the supplied disk into the computer and to run INSTALLA or INSTALLB. The INSTALL program will create a directory in drive C:\ and "xcopy/s" disk A: to it.

INSTALL ***does not update the PATH***.

2.3.4 Final Installation

• Place the MicroNator System near the PC.

• Installation of the communication cable:

- Plug the DB25 female connector into the serial port of the PC.
- Plug the DB9 female socket into the DB9 male plug of MicroNator System.

• Installation of the power supply:

- Connect the female connector of the wall-mounted power supply into the MicroNator System male input power connector.

- Plug the wall-mounted power supply into a wall AC socket.
- Turn on the MicroNator System power-on switch.

• Installation of the software:

- Place the diskette into the drive A.
- Type A: to go to disk A.
- Type INSTALLA.

• Go to the UCT11 directory (on drive C:)created by the INSTALL program:

- Execute TALK.EXE.

- (Refer to section “Communication with MicroNator System” in section 3.4.2 , for TALK switches).

- The prompt from the MicroNator System will appear.

Now all the Monitor commands are available.

This is all what is required to do.

No jumpers installation, no PCB trace to cut nor anything else to do.

...Just enjoy...

CHAPTER 3

OPERATING INSTRUCTIONS

3.1 INTRODUCTION

This chapter provides the user with the necessary information to initialize and operate the MicroNator System. This section consists of the standard (defaults) communication settings , command line format, monitor command, and operating procedures. The operating procedures consist of assembly/disassembly and downloading descriptions and examples.

3.2 STANDARD (DEFAULTS) COMMUNICATION SETTINGS

The MicroNator System MCU SCI has been set for 19200 baud using a 4.9152 MHz crystal. This baud rate can be changed with software by reprogramming the BAUD register in the MCU. The BAUD register can be changed by instructions in the user program or by the memory change (MM) command.

Another way to change the baud rate is to change the content of memory location @ $FF66: $02 will give 19 200, $03 9600, $04 4800, $05 2400, and $06 will give 1200 BAUD. “MONITEUR” take the value stored at this address (default is $02) to initialize the SCI baud rate upon reset.

The MicroNator System can transfer data faster than some terminal devices can receive them, which at certain times, can cause missing characters on the terminal display screen. Memory display (MD), trace (*TRACE*), and help (*HELP*) commands may be affected by this problem. The user can either ignore the problem, switch to a slower baud rate, or use a different communications program. When using the MD or TRACE commands, the missing character problem can be resolved by displaying fewer address locations or tracing fewer instructions at a time, respectively. This problem will sometime show on a PC, even on a 486/33.

NOTE:

> At 19 200, with some PC, there might be some characters mis-aligned on the display when dumping (MD) a long array of memory but it does not affect the content of the memory nor the downloading of programs as there is communication error recovery in the downloading procedure.

The monitor program uses part of the MCU external RAM located at $7F00-$7FFF. That way, ***all the internal memory is available to the user program***. The control registers are located at $1000-$103F. The monitor program reserves the software INTERRUPT of the Output Compare #5 (OC5) for the TRACE instruction, therefore TRACE cannot be used in user routine which uses OC5. Since PROCEED and STOPAT commands indirectly use the TRACE function these commands also rely on the

OC5. Please remember that **IRQ have to be enable** to use those commands. TRACE can be used to verify an IRQ routine as it is not the IRQ routine itself that disable the interrupts but the response of the CPU to an interrupt. If you want the CPU to service an interrupt, they have to be enable, and if they are enabled then TRACE can go through it.

3.2.1 Special (Reserved) Settings

Addresses $FFD0 to $FFD5 need special attention in order for the RESET vector to work properly when using downloading procedures with "MONITEUR".

Addresses $FFD0 and $FFD1 contain the version number of "MONITEUR".

Addresses $FFD2 and $FFD3 contain a duplication of the RESET vector. When using "ALT" "B" or "ALT" "L", while in "MONITEUR" for downloading S-Record, TALK download a small special program ("BS_EEPRM.BIN" ***which has to be in the same directory as TALK***) into RAM for the MCU to be able to program the EEPROM. When this small program is executed it writes the HPRIO register to go to special bootstrap mode of operation and doing so the RESET vector at address $FFFE get corrupted. That is why, when this small special program runs, it re-initialize the RESET vector by reading the duplicated RESET vector at $FFD2-$FFD3 and copy it to $FFFE-$FFFF.

If ***only*** the user wants ***to integrate "MONITEUR***" in his own program ***and uses downloading procedure*** he has to make sure that addresses $FFD0 to $FFD5 contain the following:

```
 *** Vectors ***

ffd0 04 00 FDB $0400
ffd2 xx xx FDB MONITEUR
ffd4 00 00 FDB 0
```

($xxxx is the RESET vector, stored at $FFFE-$FFFF, i.e starting address of "MONITEUR" or user program)

Most of the time the user will not have to modify those addresses. If the user doesn't want to include "MONITEUR" in his program, he doesn't have to bother with the above.

3.3 MONITOR MEMORY

The MicroNator System allows the operator to use all the features of "MONITEUR" to evaluate his software, however it should be noted (when designing) that "MONITEUR" uses some of the MCU external RAM locations $7F00-$7FFF ***so leaving all of the internal 256 bytes to the user*** (i.e. $0000 to $00FF).

The user must be aware of the "MONITEUR" address location restrictions. TABLE: 1 lists the monitor memory map.

TABLE: 1 "Moniteur" Memory Map

ADDRESS	DESCRIPTION
$0000-$00FF	Available to the user
$0100-$01FF	Other HC11 internal RAM, i.e. HC11E0, HC11E1, HC11E8...
$0200-$020F	Reserved
$0210-$021F	LCD & KBY expansion board
$0220-$022F	UIO (Relays & Opto couplers) expansion board
$0230-$023F	GAL Programmer expansion board
$0240-$027F	SPARE chip select for WW
$0280-$02BF	*** If Read, enables (HIGH) the **RTC** chip select for SPI *** If Written, disables (LOW) the **RTC** chip select for SPI
$02C0-$0FFF	Reserved, by 16 bytes increment, for future expansion and I/O
$1000-$103F	MCU Registers
$1040-$7EFF	User RAM
	"MONITEUR"
$7F00-$7F40	"MONITEUR" uses those addresses for the user stack
$7F41-$7F53	Reserved
$7F54	B7..B4, 10 of seconds // B3..B0, unit of seconds
$7F55	B7..B4, 10 of minutes // B3..B0, unit of minutes
$7F56	B7..B4, 10 of hours // B3..B0, unit of hours *** To set the RTC time, the order is SSMMHH for $7F54 to $7F56 and HHMMSS to read the RTC time ***($7F54-$7F56 will be modified in near future)***
$7F57	Seconds for RTC
$7F58	Minutes for RTC
$7F59	Hour for RTC
$7F5A	DOW, Day Of the Week for RTC
$7F5B	DOM, Day Of the Month for RTC
$7F5C	Month for RTC
$7F5D	Year for RTC
$7F5E-$7F95	"MONITEUR" Stack
$7F95-$7FF0	"MONITEUR" Storage RAM
$7FF1	JMP SCI, see TABLE: 4 on page 53
$7FF4	JMP SPI
$7FF7	JMP TOC5
$7FFA	JMP XIRW
$7FFD	JMP SWI
$8000-$DEFF	**User program in EEPROM**
$DF00-$E857	In-line Assembler ***(Can be remove if user need more room for his program)***
$E858-$EB11	Disassembler ***(Can be remove if user need more room for his program)***
$EB12-$FF64	"MONITEUR" (Look for the reset vector @ $FFFE-$FFFF in newer version of "MONITEUR")

$FF65-$FFCE	Indirect JUMP, see TABLE: 3 on page 47 All JMPs are Buffalo 3.XX Compatible except for VECTINIT
$FFD0-$FFD5	Special Reserved, see 3.2.1 on page 16
$FFD6-$FFFF	CPU Vector Table, see TABLE: 5 on page 55

3.4 OPERATING PROCEDURES

The MicroNator System is a simplified debugging/evaluating tool designed for debugging user programs and evaluation of the 68HC11 family devices.

3.4.1 Generating S-Record

After the user has written his assembly program he pass it through the included Motorola Cross-Assembler until it is error-free. The Cross-Assembler then generates a S-Record file with the extension .S19. The generated S-Record has to be in the same directory than TALK.EXE, if it is not, the user copy the S-RECORD into the same directory as TALK.EXE.

3.4.2 Communication with MicroNator System

To communicate with MicroNator System the user executes TALK.EXE with the following parameters:

TALK [-f] [-p1 | -p2] [-r-]

< >	No parameter (default).
< -f>	To have all texts displayed in French.
< -p1>	Same as no parameter (COM1).
< -p2>	Communication port will be COM2.
< -r- >	When TALK start communication, it will not reset CPU-11/64-e2 System.

TALK will initialize the specified communication port of the PC, generates a RESET of the MicroNator System if the switch [-r-] was not specified, and will communicate with MicroNator System.

3.4.3 Downloading S-Record

The generated S-Record has to be in the same directory as TALK.EXE, if it is not the user copy the S-RECORD into the same directory as TALK.EXE then he executes TALK.EXE with the proper switches.

When the MicroNator System screen appears press “ALT” and “L” together. “Enter name of file to download: ” will appear and the user type the name of the S-Record. The .S19 extension is

not necessary. Press RETURN and TALK will download the file.

After downloading “Load complete” will appear. Press any key to re-establish communication with MicroNator System. “MONITEUR” MICRO CONTROLLER 68HC11 UCT11 V4.0 will be displayed and all the “MONITEUR” resources will be available to debug your program.

3.4.4 Reset

To RESET MicroNator System the user:

- Press the RESET SWITCH, on the front side of the enclosure, or...
- Turn the POWER SWITCH off then on again, or...
- While in communication with the MicroNator System, press “ALT” and “R” together, or...
- When executing TALK, do not specify the -r- switch.

3.4.5 Help File

While the user is in communication with the MicroNator System, he can press the F1 key and a small help file will be displayed. Please note that not all of the commands in the HELP file are available with version 4.04 (those related to BASIC).

3.4.6 Exit

To exit TALK and return to DOS the user types “ALT - X”.

3.4.7 No Communication with MicroNator System

After making sure that MicroNator System is connected to the right communication port and TALK was used with the proper COMx port switch (-p1) or (-p2) and if there is still no communication with MicroNator System it might mean that there is no “MONITEUR” in the EEPROM or that the user program has change some of the codes of “MONITEUR”. In such a case, the user has to re-initialize MicroNator System to the original factory settings:

To initialize MicroNator System to the original factory settings:

- Make sure the file UCT2.S19 is in the same directory as TALK.EXE.
- Execute TALK specifying the COMx port.

- When the screen appears press “ALT” and “B” together and TALK will download UCT2.S19 (“MONITEUR”) to the EEPROM.

- After downloading “BOOTSTRAP OK” will appear.

- Press any key to re-establish the communication with MicroNator System.

- “MONITEUR” MICRO CONTROLLER 68HC11 UCT11 v4.0 will appear and all the “MONITEUR” resources will be available.

3.4.8 Monitor Program

“MONITEUR” program is included in the distribution diskette and is called UCT2.S19. BS_EEPRM.BIN is used by TALK for the downloading of programs from the PC into the system memory. This special program **has to be in the same directory** as TALK.EXE. The same applies to UCT2.S19 if you want to download “MONITEUR”, and to CONFIG.S19 if you want to change the CONFIG register of the MCU.

3.5 COMMAND LINE FORMAT

The command line format is as follows:

> < command > [< parameters >] (RETURN)

>	MicroNator System monitor prompt.
< command >	Command mnemonic (single letter for most commands).
< parameters >	Expression or address.
(RETURN)	RETURN keyboard key - depressed to enter command.

NOTES

(1) The command line format is defined using special characters which have the following syntactical meanings:

< >	Enclose syntactical variable
[ ]	Enclose optional fields
[ ]...	Enclose optional fields repeated

These characters are not entered by the user, however they are for definition purposes only.

(2) Fields are separated by any number of spaces, commas, or tab characters.

(3) All input numbers are interpreted as hexadecimal.

(4) All input commands can be entered in either upper or lower case lettering. All input commands are converted automatically to upper case lettering.

(5) A maximum of 35 characters may be entered on a command line. After the 36th character is entered, the monitor automatically terminates the command entry and the terminal CRT display the message “Too long”.

(6) Command line errors may be corrected by backspacing, or by aborting “CTRL-X”.

(7) Pressing “RETURN” will repeat the most recent command.

3.6 MONITOR COMMANDS

“MONITEUR” program commands are listed alphabetically by mnemonic in TABLE: 2 . Each of the commands are described in detail following the tabular command listing. In most cases the initial single letter of the command mnemonic or specific symbol can be used. A minimum number of characters must be entered to at least guarantee uniqueness from other commands (i.e., MO=MOVE, ME=MEMORY). If the letter M is entered, “MONITEUR” uses the first command in table which starts with the letter M.

Additional terminal keyboard functions are as follows:

(CTRL) A	Exit assembler
(CTRL) H	Backspace
(CTRL) J	Line feed (LF)
(CTRL) W	Wait/freeze screen. Execution is restarted by any terminal keyboard key.
(CTRL) X	Abort/cancel command
(RETURN)	Enter command/repeat last command

NOTES:

- When using the control key for a specialized command such as “CTRL A”, the “CTRL” key is depressed and held, then the key “A” is depressed. Both keys are then released.

- Command line input examples in this chapter are amplified with the following:

  • Command line input is entered when (RETURN) key is depressed.

  • Typical example of this explanation is as follows:

  >MD F000 F100

TABLE: 2 Monitor Program Commands

COMMAND	DESCRIPTION
ASM [<address>]	Assembler/disassembler
ASSEM	(same as ASM)
BF <addr1> <addr2> <data>	Block fill memory with data
BR [-] [<address>]...	Breakpoint set
BREAK	(same as BR)
CALL [<address>]	Execute subroutine
COPY	(same as MOVE)
DUMP	(same as MD)
FILL	(same as BF)
G [<address>]	Execute program
GO	(same as G)
HELP	Display monitor commands
MEMORY	(same as MM)
MD [<addr1> [<addr2>]]	Dump memory to terminal
MM [<address>]	Memory modify
MOVE <addr1> <addr2> [<dest>]	Move memory to new location
P	Proceed/continue from breakpoint
PROCEED	(same as P)
RD	(same as RM)
READ	(same as MOVE)
REGISTER	(same as RM)
RM [p,x,y,a,b,c,s]	Register modify/display user registers
STOPAT <address>	Stop at address
T [<n>]	Trace $1-$FF instructions
TRACE	(same as T)
?	(same as HELP)
[<address>]/	(same as MM [<address>])

3.6.1 Assembler/disassembler

ASM [<address>]

< address > is the starting address for the assembler operation.

Assembler operation defaults to internal RAM if no address is given.

The assembler/disassembler is an interactive assembler/editor. Each source line is converted into the proper machine language code and is stored in memory overwriting previous data on a line-by-line basis at the time of the entry. In order to display an instruction, the machine code is disassembled and the instruction mnemonic and operands are displayed. All valid opcodes are converted to assembly language mnemonics. All invalid opcodes are displayed on the PC screen as “ILLOP”.

The syntax rules for the assembler are as follows:

a. All numerical values are assumed to be hexadecimal. Therefore no base designators (e.g., $ = hex, % = binary, etc.) are allowed.

b. Operands must be separated by one or more space or tab characters.

c. Any characters after a valid mnemonic and associated operands are assumed to be comments and are ignored.

Addressing mode are designated as follows:

a. Immediate addressing is designated by preceding the address with a # sign.

b. Indexed addressing is designated by a comma. The comma must be preceded by a one byte relative offset (even if the offset is 00), and the comma must be followed by an X or Y designating which index register should be use (e.g., LDAA 0,X)

c. Direct and extended addressing is specified by the length of the address operand (1 or 2 digits specifies direct, 3 or 4 digits specified extended). Extended addressing can be forced by padding the address operand with leading zeros

d. Relative offsets for branch instructions are computed by the assembler. Therefore the valid operand for any branch instructions is the branch-if-true address, not the relative offset.

When a new source line is assembled, the assembler overwrites what was previously in memory. If no source line is submitted, or if there is an error in the source line, then the contents of memory remain unchanged. Four instruction pairs have the same opcode, so disassembly will display the following mnemonics:

- Arithmetic Shift Left (ASL) / Logical Shift Left (LSL) displays as ASL
- Arithmetic Shift Left Double (ASLD) / Logical Shift Double (LSDL) displays as LSLD

- Branch if Carry Clear (BCC) / Branch if Higher or Same (BHS) displays as BCC
- Branch if Carry Set (BCS) / Branch if Lower (BLO) displays as BCS

<u>It is possible</u> for the assembler to assemble in the complete memory map <u>even if it is in the external EEPROM</u> because it uses *WRITE* subroutine to put the assembled code in memory.

Assembler/disassembler subcommands are as follows. If the assembler detects an error in the new source line, the assembler will output an error message and then re-open the same address location.

/ , = Assemble the current line and then disassemble the same address location.

^ , - Assemble the current line and then disassemble the previous sequential address location.

(CTRL) J , + Assemble the current line. If there isn't a new line to assemble, then disassemble the next sequential address location. Otherwise, disassemble the next opcode address.

RETURN Disassemble next opcode.

(CTRL) A , . Exit the assembler mode of operation.

EXAMPLES **DESCRIPTION**

```
>ASM FB00

FB00 STX $FFFF >LDAA #55      Immediate mode addressing, require
86 55                         # before operand.
FB02 STX $FFFF >STAA C0       Direct mode addressing.
97 C0
FB04 STX $FFFF >LDS 0,X       Index mode, if offset = 0 (,X) will
AE 00                         not be accepted.
FB06 STX $FFFF >BRA C500      Branch out of range message.

Branch out of range
FB06 STX $FFFF >BRA FB50      Branch offsets calculated automatic-
20 48                         ally, address required as branch opr
FB08 RTS >(CTRL)A             Assembler operation terminated.
>
```

NOTE:

In the above example memory locations $FB00-$FB08 previously contain $FF data which disassembles to STX $FFFF.

3.6.2 Block Fill

BF <address 1> <address 2> <data>

<address 1> Lower limit for fill operation.

<address 2> Upper limit for fill operation.

<data> Fill pattern hexadecimal value.

The BF command allows the user to repeat a specific pattern throughout a determined user memory range in RAM or EEPROM.

EXAMPLES	**DESCRIPTION**
`>BF FB00 FBFF FF`	`Fill each byte of memory from $FB00 to $FBFF with data pattern $FF.`
`>BF FC00 FC00 0`	`Set location $FC00 to 0.`

3.6.3 Breakpoint Set

BR [-] [<address>]

[-] by itself	Removes (clears) all breakpoints.
[-] preceding [<address>]	Removes individual or multiple addresses from breakpoint table

The BR command sets the address into the breakpoint address table. During program execution, a halt occurs to the program execution immediately preceding the execution of any instruction address in the breakpoint table. A maximum of 4 breakpoints may be set. After setting the breakpoint, the current breakpoint addresses, if any, are displayed. Whenever the G, CALL, or P commands are invoked, the monitor program inserts breakpoints into the user code at the address specified in the breakpoint table.

Breakpoints are accomplished by the placement of a Software Interrupt (SWI) at each address specified in the breakpoint table. The SWI service routine saves and displays the internal machine state, then restores the original opcodes at the breakpoint locations before returning control back to the monitor program.

Normally SWI opcodes cannot be executed or breakpointed in user code because the monitor program uses the SWI vector.

However the user can put his own vector at the SWI vector address ($FFF6-$FFF7) using the WRITE command as described in section 5.2.5 on page 49 or use the indirect RAM SWI vector address ($7FFD-$7FFF). But remember to put the original vectors back because “MONITEUR” will not work properly if the proper indirect vectors are not properly initialized.

One way to avoid confusion after the user has manipulated the SWI vectors is to re-download “MONITEUR”.

SWI opcodes can be put anywhere in the memory map (RAM or external EEPROM).

COMMAND	**DESCRIPTION**
BR	Display all current breakpoints.
BR <address>	Set breakpoint.
BR <addr1> <addr2> ...	Set several breakpoints.
BR -	Remove all breakpoints.

BR - <addr1> <addr2> ... Remove <addr1> and add <addr2>.

BR <addr1> - <addr2> ... Add <addr1>, clear all entries, then add <addr2>.

BR <addr1> - <addr2> ... Add <addr1>, then remove <addr2>.

```
>BR FB03                      Set breakpoint at address location
                              $FB03.
FB03 0000 0000 0000
>
>BR FB03 FB05 FB07 FB09       Sets four breakpoints. Breakpoints
                              at same address will result in only
FB03 FB05 FB07 FB09           one breakpoint being set.

>BR                           Display all current breakpoints.

FB03 FB05 FB07 FB09

>BR -FB09                     Remove breakpoint at address $FB09.

FB03 FB05 FB07 0000

>BR - FB09                    Clear breakpoint table & add $FB09.

FB09 0000 0000 0000

>BR -                         Remove all breakpoints.

0000 0000 0000 0000

>BR E000 E003 E005 E007 E009  Maximum of 4 breakpoints can be set.

Full                          Buffer full message.
E000 E003 E005 E007
>
```

3.6.4 CALL

CALL [<address>]

[<address>] Address is the starting address where user subroutine begins.

The CALL command allows the user to execute a user subroutine program. Execution starts at the current program counter (PC) address location, unless starting address is specified. Two extra bytes are placed onto the stack before "MONITEUR" calls the subroutine so that the first unmatched return from subroutine (RTS) encountered will return control back to the monitor program. Thus any user subroutine can be called and executed via the monitor program. Program execution continues until an unmatched RTS is encountered, a breakpoint is encountered, or the MicroNator System reset switch is activated (pressed).

EXAMPLE PROGRAM FOR: CALL, G, P, and STOPAT

```
>ASM FB00

FB00 STX $FFFF >LDAA #44
86 44
FB02 STX $FFFF >STAA 0040
B7 00 40
FB05 STX $FFFF >NOP
01
FB06 STX $FFFF >NOP
01
FB07 STX $FFFF >NOP
01
FB08 STX $FFFF >RTS
39
FB09 STX $FFFF >(CTRL)A
>
```

EXAMPLE	**DESCRIPTION**
`>CALL FB00`	`Execute program subroutine.`
`P-FB00 Y-FB18 X-0818 A-44 B-BE C-D0 S-005E` `>`	`Displays register status at time RTS is encountered (except P register contains original call address or a breakpoint address if encountered).`

G **Go** **G**

3.6.5 GO

GO [<address>]

[<address>] Address is the starting address where program execution is to begin.

The G command allows the user to initiate user program execution (free run in real time). The user may optionally specify a starting address where execution is to begin. Execution starts at the current program counter (PC) address location, unless a starting address is specified. Program execution continues until a breakpoint is encountered, or the MicroNator System reset switch is activated (pressed).

NOTE:

Refer to example program shown in section 3.6.4 and insert breakpoints at locations $FB05 and $FB07 for the following G command example.

```
>BR FB05 FB07                  Insert breakpoints.

FB05 FB07 0000 0000            Display of breakpoint table.


>G FB00                        Begin program execution at PC
                               address location $FB00.

P-FB05 Y-FB18 X-0818 A-44 B-BE C-D0 S-005E
>
                               Breakpoint encountered at $FB05.
```

3.6.6 HELP

The HELP command enables the user available MicroNator System command information to be displayed on the terminal CRT for quick reference purposes.

At 19 200, with some PC, there might be some characters mis-aligned on the display when dumping (MD) a long array of memory but it does not affect the content of the memory nor the down-loading of programs as there is communication error recovery in the downloading procedure.

EXAMPLE

```
>HELP
```

FIRST SCREEN

```
ASM [<addr>] => Line assembler/disassembler.
/ , =      :  Do same address.
^ , -      :  Do previous address.
CTRL-J , + :  Do next address.
RETURN     :  Do next opcode.
CTRL-A , . : Quit.

MM [<addr>]        : Memory modify.
/ , =              : Open same address.
CTRL-H , ^ , -     : Open previous address.
CTRL-J , + , space : Open next address.
<addr>O            : Compute offset to <addr>.
RETURN             : Quit.

************************************************
******   SPACE for more OTHER KEY to exit  ******
************************************************
```

SECOND SCREEN

```
BF <addr1> <addr2> [<data>] =>Block fill.
BR [-][<addr>] => Set up breakpoint table.
CALL [<addr>] => Call user subroutine.
G [<addr>] => Execute user code.
MD [<addr1> [<addr2>]] => Memory dump.
MOVE <s1> <s2> [<d>] => Block move.
P => Proceed/continue execution.
RM [P, Y, X, A, B, C, or S] => Register modify.
T [<n>] => Trace n instructions.
STOPAT <addr> => Trace instructions until <addr>.

CTRL-H => Backspace.

CTRL-W => Wait for any key.

CTRL-X => Abort/cancel command.

RETURN => Repeat last command.
```

```
>
```

3.6.7 MEMORY DISPLAY

MD [<address 1> [<address 2>]]

<address 1> Memory starting address (optional).

[<address 2> Memory ending address (optional).

The MD command allows the user to display a block of user memory beginning at address 1 and continuing to address 2. If address 2 is not entered, 9 lines of 16 bytes are displayed beginning at address 1. If address 1 is greater than address 2, the display will default to the first address. If no addresses are specified, 9 lines of 16 bytes are displayed near the last memory location accessed.

Each memory display line consists of a four digits hexadecimal address (applicable to the memory location displayed), followed by 16 two digits hexadecimal values (contents of the sixteen memory locations), followed by the ASCII equivalents (if applicable) of the 16 memory locations. Since not all 8-bit values correspond to a displayable ASCII character, some of the character positions at the end of a line may be blank.

At 19 200, with some PC, there might be some characters mis-aligned on the display when dumping (MD) a long array of memory but it does not affect the content of the memory nor the down-loading of programs as there is communication error recovery in the downloading procedure.

EXAMPLES

```
>MD E7D0

E7D0 20 8A 03 97 20 86 08 97 0B 39 CE E7 E1 BD E3 02 9
E7E0 39 0D 41 53 4D 20 5B 3C 61 64 64 72 3E 5D 20 20 9 ASM [<addr>]
E7F0 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 3D =
E800 3E 20 4C 69 6E 65 20 61 73 73 65 6D 62 6C 65 72 > Line assembler
E810 2F 64 69 73 61 73 73 65 6D 62 6C 65 72 2E 0D 20 /disassembler.
E820 20 20 20 2F 20 20 20 20 20 20 20 20 20 3D 3E 20 / =>
E830 44 6F 20 73 61 6D 65 20 61 64 64 72 65 73 73 2E Do same address.
E840 20 20 20 20 20 5E 20 20 20 20 20 20 20 20 20 20 ^
E850 20 20 20 3D 3E 20 44 6F 20 70 72 65 76 69 6F 75 => Do previou
>MD FB30 FB20

FB30 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
>MD FB00 FB20

FB00 86 44 B7 00 40 01 01 01 39 FF FF FF FF FF FF FF D @ 9
FB10 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FB20 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

3.6.8 MEMORY MODIFY

MM [<address>]

<address> Is the memory location at which to start display / modify.

The MM command allows the user to examine/modify contents in user memory at specified locations in a interactive manner. The MM command will also erase any external EEPROM or internal RAM location, and will reprogram the location with the corresponding value (external EEPROM locations treated as if standard RAM).

(CTRL)J or (SPACE BAR) or + Examine next location

(CTRL)H or ^ or - Examine/modify previous location.

/ or = Re-examine/modify same location.

(RETURN) Terminate MM operation.

<ADR> O Compute branch instruction relative offset.

If an attempt is made to change an invalid address, the invalid address message “WRITING ERROR” is displayed on the terminal CRT. An invalid address is any memory location which cannot be read back immediately after a change in order to verify that change. A good example is a location in memory which have no memory installed. CONFIG is a special case refer to B.1 on page 71.

EXAMPLES	**DESCRIPTION**
`>MM FB80`	`Display memory location $FB80`
`FB80 FF 66/`	`Change data at $FB80 and re-examine location.`
`FB80 66 55^`	`Change data at $FB80 and backup one location.`
`FB7F AA AA`	`Change data at $FB7F and terminate MM operation.`
`>MM FB80`	`Display memory location.`
`FB3C FF FB8EO 51` `FB3C FF`	`Compute offset, result = $51.`
`>MM 0040`	`Examine $0040 (internal RAM).`
`0040 44 04 25 97 3F D7 (RETURN)`	

```
                                Examine next location(s) using
                                (SPACE BAR) & ending with (RETURN).
>

>MM 1000                        Examine $1000 (no memory installed).

1000 00 FF                      Trying to write $FF at $1000.

WRITING ERRO                    Error message since after writing,
>                               the monitor reads back and if data
                                read is different than data written
                                then the monitor display “WRITING
                                ERROR
```

3.6.9 MOVE

MOVE < source start-address > < source end-address > [< dest-start >]

< source start-address > Memory source starting address.

< source end-address > Memory source ending address.

[< dest-start >] Destination starting address (optional).

The MOVE command allows the user to copy/move memory to new memory locations. If the destination is not specified, the block of data residing from address 1 to address 2 will be moved up one byte.

No message will be displayed on the CRT upon completion of the copy / move operation, only the prompt is displayed.

Take note that it takes about 10 msec to program one byte of EEPROM.

EXAMPLES	**DESCRIPTION**

`>MOVE E000 E0FF F000`
 	
`>`
	`Move data from locations $E000-$E0FF to locations $F000-$F0FF.`

3.6.10 PROCEED / CONTINUE

P

This command is used to proceed or continue program execution without having to remove assigned breakpoints. This command is used to bypass assigned breakpoints in a program executed by the G command.

NOTE:

Refer to example program shown in section 3.6.4 on page 30 for the following P command. Breakpoint have been inserted at location $FB05 and $FB07 (refer to example in section 3.6.3).

EXAMPLE	DESCRIPTION
>BR	To see the breakpoint table.
FB05 FB07 0000 0000	Breakpoints table.
>G FB00	Start execution at $FB00
P-FB05 Y-FB18 X-0818 A-44 B-BE C-D0 S-005E	Breakpoint encountered at $FB05.
>P	Continue execution.
P-FB07 Y-FB18 X-0818 A-44 B-BE C-90 S-005E	
>	Breakpoint encountered at $FB07.

3.6.11 REGISTER MODIFY

RM [p, y, x, a, b, c, s]

The RM command is used to modify the MCU program counter (P), Y index (Y), X index (X), A accumulator (A), B accumulator (B), condition code register (C), and stack pointer (S) register contents.

EXAMPLE	DESCRIPTION
>RM	Display P register contents.
P-FB07 Y-FB18 X-0818 A-44 B-BE C-90 S-005E	
P-FB07 FB00	Modify P register contents.
>RM X	Display X register contents.
P-FB00 Y-FB18 X-0818 A-44 B-BE C-90 S-005E	
X-0818 00E0	Modify X register contents.
>RM	Display P register contents.
P-FB00 Y-FB18 X-00E0 A-44 B-BE C-90 S-005E	
P-FB00 (SPACE BAR) Y-FB18 (SPACE BAR) X-00E0 (SPACE BAR) A-44 (SPACE BAR) B-BE (SPACE BAR) C-90 (SPACE BAR) S-005E (SPACE BAR) >	Display remaining registers. (SPACE BAR)entered following stack pointer display will terminate RM command.

3.6.12 STOP AT

STOPAT <address>

<address> Is the specified user program counter (PC) stop address.

The STOPAT command causes a user program to be executed one instruction at a time until the specified address is encountered. Execution begins with the current PC address and stop just before execution of the instruction at the specified stop address. The STOPAT command should only be used when the current value of the user PC register is known, (e.g., after a breakpoint is reached or after an RD command is used to set the user PC).

Since the STOPAT command traces one instruction at a time with a hidden return to the monitor after each user instruction, some user programs will appear to execute slowly.

The stop address specified in the STOPAT command must be the address of an opcode just as breakpoints can only be set at opcode addresses.

NOTE:

Refer to example program shown in section 3.6.4 on page 30 for the following STOPAT command example. The RD command was used prior to this example to set the user PC register to $FB00.

EXAMPLE	DESCRIPTION
`>STOPAT FB08`	`Execute example program until $FB08 is reached.`

```
P-FB08 Y-FB18 X-0818 A-44 B-BE C-90 S-005E
>
```

3.6.13 TRACE

Trace [<n>]

<n> N is the number (in hexadecimal, $1-$FF) of instructions to execute. A default value of 1 is used if <n> is not specified.

The T command allows the user to monitor program execution on an instruction-by-instruction basis. The user may optionally execute several instructions at a time by entering count value (up to $FF). Execution starts at the current program counter (PC). Each event message line includes a disassembly of the instruction that was traced and a register display showing the CPU state after the execution of the traced instruction. The trace command operates by setting the OC5 interrupt to time out after the first cycle of the first user opcode fetched.

At 19 200, with some PC, there might be some characters mis-aligned on the display when dumping (MD) a long array of memory but it does not affect the content of the memory nor the downloading of programs as there is communication error recovery in the downloading procedure.

NOTE:

The RM command was used to set the user PC register to $FB00 prior to starting the following trace examples.

```
 EXAMPLE                          DESCRIPTION

>RM
P-FFFF Y-FFFF X-FFFF A-FF B-FF C-94 S-004E

P-FFFF FB00(SPACE)                Init user PC register.
Y-FFFF (SPACE)
X-FFFF (SPACE)
A-FF 61 (RETURN)                  Init accumulator A.

>T 1
JMP $FB10 P-FB10 Y-FFFF X-FFFF A-61 B-FF C-94 S-004E
>T 2
PSHA P-FB11 Y-FFFF X-FFFF A-61 B-FF C-94 S-004D
PSHB P-FB12 Y-FFFF X-FFFF A-61 B-FF C-94 S-004C
>T 3
PSHX P-FB13 Y-FFFF X-FFFF A-61 B-FF C-94 S-004A
JSR $FB20 P-FB20 Y-FFFF X-FFFF A-61 B-FF C-94 S-0048
CMPA #$61 P-FB22 Y-FFFF X-FFFF A-61 B-FF C-94 S-0048
>T 4
BEQ $FB40 P-FB40 Y-FFFF X-FFFF A-61 B-FF C-94 S-0048
RTS P-FB16 Y-FFFF X-FFFF A-61 B-FF C-94 S-004A
LDX #$EFAE P-FB19 Y-FFFF X-EFAE A-61 B-FF C-98 S-004A
JSR $E500 P-E500 Y-FFFF X-EFAE A-61 B-FF C-98 S-0048
>
```

CHAPTER 4

HARDWARE DESCRIPTION

4.1 INTRODUCTION

This chapter provides an overall description of the MicroNator System hardware. The MicroNator System schematic diagrams can also be referred to for the following descriptions.

4.2 GENERAL DESCRIPTION

Overall evaluation/debugging control of the MicroNator System is provided by the "MONITEUR" program residing in the standard memory map in the EEPROM outside the CPU. The RS-232C terminal I/O port interface circuitry provides communication and data transfer operations between the MicroNator System and external terminal/host computer devices while the I/O port DB25 is used for the external I/O connections.

4.3 POWER

The power supply is a wall-mounted one. The module plugs into the wall AC socket and the female connector plugs into the connector of the MicroNator System. Refer to Figure: 2 on page 12.

The output of the wall-mounted power supply is about 7.5 Vdc, unregulated. Inside the MicroNator the input power is protected with diode D1 in case the power is plugged the wrong way. It is filtered with caps C9 and C10 then it is regulated to 5 Vdc with regulator U6 and filtered again with caps C11 and C12. The diode D2 is used in the case the power comes from the bus and is more than 5 Vdc. Refer to Fig: 5 on page 57.

4.4 MICRO CONTROLLER

4.4.1 Software

The MCU configuration register "CONFIG $103F" is programmed such that the NOCOP bit is set (COP system is disabled) and the EEON (Enable on-chip EEPROM) bit is reset (internal 512 bytes of EEPROM is disabled and takes no space in the memory map). When the EEON bit is reset, MCU internal ROM is disabled, and that memory space becomes internally unaccessible. This allowed the external EEPROM memory to contain "MONITEUR", the monitor program.

The control registers are located at $1000-$103F after RESET.

The MicroNator System allows the user to use all the features of “MONITEUR” program, however it should be noted that the monitor program uses part of the MCU external RAM locations $7F00-$7FFF leaving the entire 256 bytes of internal RAM to the user (i.e., $0000-$00FF). When you are using “MONITEUR” to debug your program, about 23.9 Kbytes of EEPROM is available. After your program is debugged you can remove “MONITEUR” and you will have the complete (32 Kbytes) of EEPROM available.

4.4.2 Mode / RS-232 Communication

The MicroNator System resident 68HC11A1 MCU device (U1) is factory configured for normal expanded multiplexed mode of operation.

The normal expanded multiplexed mode is accomplished by applying +5 Vdc to the MCU MODA and MODB pins during reset. Refer to Figure: 4 below.

Inputs		Mode description
MODB	MODA	
1	0	Single Chip
1	1	Expanded
0	0	Special Bootstrap
0	1	Special Test

Figure: 4 Mode Selection

The MicroNator System can be re configured for the special-bootstrap mode of operation without additional circuitry. This special-bootstrap mode of operation is activated remotely through the RS-232C communication channel. Refer to Figure: 3 on page 13 for the cable connection.

Lines from the PC and internal circuitry of U50 (PALCE22V10) are used to change and latch the mode of the MicroNator System from expanded multiplexed standard mode of operation to special bootstrap mode of operation and vice versa. The special bootstrap mode is one of the way used to download program from the PC to the MicroNator System board.

If nothing is plugged in the DB9 of the MicroNator System the MCU will be in standard expanded multiplexed mode of operation

4.4.3 Remote Reset through the RS-232 Communication from the PC

When the RESET* pin is LOW the MCU stop running just like any other MCU under reset. Lines from the PC and internal circuitry of the PALCE22V10 are used to reset the MicroNator System.

If nothing is plugged in the DB9 of the MicroNator System the MCU will be in standard expanded multiplexed mode of operation when its comes out of reset.

4.4.4 Trace

The TRACE command uses the INTERRUPT generated by the OC5 on pin PA3. This is the reason why the PA3 signal **should be used with care** on the I/O connector. The user should pay attention if he wants to use the OC5 or PA3 and still be able to TRACE his program so not to interfere with the TRACE command.

4.4.5 MEMORY

The MicroNator System map is a single map design reflecting the permanently resident 68HC11A1 device. The MicroNator System is configured for expanded-multiplexed mode of operation, but can be configured by the PC for special boot-strap mode of operation. Refer to the M68HC11 Reference Manual (M68HC11RM/AD REV2) for the specific memory map information on the modes of operation.

4.4.6 RAM

The amount of internal RAM available is 256 bytes and is situated in the lower part of the memory map inside the CPU. ***The monitor uses none of it*** for its own variables. When the user is developing his program and wants to use the monitor, he can use the entire internal 256 bytes and the entire external RAM except $7F00 to $7FFF. After the program is developed, the user can remove the monitor from the memory so he will be free to use all of the RAM available (internal & external).

4.4.7 EEPROM

The amount of EEPROM available is 32Kbytes and is situated in the higher part of the memory map outside the CPU. The 512 bytes of EEPROM inside the CPU is still there and is disabled because of memory conflict with the external EEPROM. The EEON bit (bit 0) of the CONFIG register is reset so as to disable the 512 bytes of internal EEPROM.

If the user wants to write a byte in the external EEPROM, he just has to put the byte in the accumulator A and the address to write to in the accumulator X, then call the WRITE subroutine. The address of this subroutine is founded in TABLE: 3 on page 47.

For more information, please refer to the paragraph 5.2.5 on page 49 “Special EEPROM Writing Routine”.

4.5 UCT BUS

Please refer to “UCT BUS CONNECTOR” on page 59 for a complete description of signals and pinout. All of the signals necessary to implement a complete micro controller system are included on this bus. If the user wants more expansion he can purchase an optional Wire Wrap board that plugs

directly into the CPU-BUS connector of the MCU. The board comes with all the necessary hardware to secure the expansion board to the CPU board. The user in not restricted to only one expansion board, he can adds as many expansion boards as he wants as long as he respect the loading of the pins. Please refer to "EXPANSION BOARD" on page 65 for more details on expansion boards.

4.6 Serial I/O Communication

The MicroNator System uses a +5 volt RS-232C driver/receiver device (U5) to communicate to a PC via the MicroNator System DB9 I/O port. The terminal I/O port baud rate defaults to 19,200 baud. The baud rate can be changed by software by programming the MCU BAUD register ($102B) or changing the memory location $FF66 as explained in section 3.2 on page 15.

The terminal I/O port is also used as a host computer communication port for downloading Motorola S-records via the special-bootstrap mode of operation, with the help of U50 (PALCE) and a special communication program residing in the host computer (BS_EEPRM.BIN).

Figure: 3 on page 13 show the standard cable used to communicate between the PC and MicroNator System. The DB25 to be inserted into the communication port of the PC is a female connector and the one to be inserted into the MicroNator System is also a DB9 female. The DB9 on the MicroNator System is a male connector.

4.7 I/O Port Interface

Section 6.6 describes the MCU I/O port connector. The user is able to connect I/O wires to this connector. All of the analog pins have a 1K ohms resistor in serie to limit the current flowing into the MCU analog pin. For more explanations about those serie resistors the user is referred to section 12.3 A/D PIN CONNECTION CONSIDERATIONS on page 12-16 in Motorola HC11 reference manual (# M68HC11M/AD REV 2) included with the MicroNator System.

CHAPTER 5

MONITOR PROGRAM

5.1 INTRODUCTION

This chapter provides the overall description of the monitor program. This description will enable the user to understand the basic organization of the program.

5.2 PROGRAM DESCRIPTION

The monitor program supplied with the MicroNator System is called “MONITEUR”. This monitor is the standard Motorola BUFFALO monitor modified by RF-232. The program communicates via the MCU serial communication interface (SCI). “MONITEUR” is contained in the distribution diskette and it is also in the external EEPROM of the MicroNator System.

“MONITEUR” consist of six parts as follows:

a. Initialization
b. Command interpreter
c. I/O routines
d. Utility subroutines
e. Command table
f. Special EEPROM writing routines

5.2.1 Initialization

This part of “MONITEUR” contains all of the reset initialization code. In this section, external RAM locations and the I/O channel for the communication with the PC are set up. The baud rate is initialize to 19200 bps, 8 bits, 1 stop, no parity.

5.2.2 Command interpreter

The next section of “MONITEUR” is the command interpreter. American Standard Code for Information Interchange (ASCII) characters are read from the PC into the input buffer until a carriage return or a slash (/) is received.

The command field is then parsed out of the input buffer and placed into the command buffer. A table of commands is then searched and if a match is found, the corresponding command module is called as a subroutine.

All commands return control back to the command interpreter upon completion of the operation.

5.2.3 I/O Routines

The I/O section of "MONITEUR" consists of a set of three supervisor routines. The supervisor routines consists of an initialization routine INIT, an input routine INPUT, and an output routine OUTPUT.

All I/O routines use the SCI for communication with the PC. ***There are no*** IODEV, EXTDEV nor HOSTDEV.

The INIT routine sets up a serial transmission format of 8 data bits, one stop bit, and no parity. The SCI has a baud rate of 19200 bps for a 4.9152 MHz crystal. A different baud rate can be achieved by modifying the BAUD register at address location $102B (refer to MCU data sheet, SCI baud rate selection). The baud rate can also be modify by changing the content of memory location $FF66 as explained in section 3.2 on page 15.

The INPUT routine reads from the SCI. If a character is received, the character is returned to accumulator A. If no character is received, a logic zero (0) is returned to accumulator A. This routine does not wait for a character to be received before returning (that function is performed by INCHAR utility subroutine).

The output routine takes the ASCII character in accumulator A and writes the character to the SCI. This routine waits until the character begins transmitting before returning.

5.2.4 Utility Subroutines

Several subroutines exist that are available for performing I/O tasks. Those subroutines are in a jump table set up in EEPROM directly before the interrupt vectors. To use these subroutines, execute a jump to subroutine (JSR) command to the appropriate entry in the jump table. By default, all I/O performed with these routines are sent to the SCI. The utility subroutines available to the user are listed in TABLE: 3 below. Those routine do the same thing and are at the same place as those from the BUFFALO monitor. So they are exactly compatible with the programs developed with the EVB of Motorola. ***The only exception is VECTINIT*** which is at $FF79 instead of $FFD0 and "MONITEUR" has only 5 pseudo-vectors in RAM. Please refer to TABLE: 4 for those pseudo-vectors.

TABLE: 3 Utility Parameters and Subroutines Jump Table

SPECIAL PARAMETERS & SUBROUTINES of "MONITEUR"

ADR	ROUTINE	DESCRIPTION
$FF65	RTCXTAL	Contains the variable to initialize the Real Time Clock $85 => 4.194304 MHz $95 => 2.097152 MHz $A5 => 1.048576 MHz $B5 => 32.768 KHz (default)
$FF66	BBAUD	Contains the variable to initialize the SCI $02=19,200 $03=9600 $04=4800 $05=2400 Default is $02.
$FF67	WRCH	Same as OUTA, endless loop until SCI TX ready.
$FF6A	JMP RTCINIT	Initialize the PORTD and the SPI system for communication with the RTC.
$FF6D	JMP SETDFLT	Subroutine to set up default time of: 12:00:00 Mon 01/01/1995 and alarm set to same time but turned off. RTC turned on.
$FF70	JMP SETTIME	Set time sec/min/hr as pointed to by Y-register.
$FF73	JMP DISPTIM	Display current time to the PC (alternate entry to do a CR, LF first)
$FF76	JMP WRITE	Write the content of accumulator A to the address in X. (Operation applicable to all locations in internal and external RAM or external EEPROM ***except for CONFIG $103F***).
$FF79	JMP VECINIT	Used during initialization to pre-set 5 indirect interrupt vectors in RAM (SCI, SPI, TOC5, XIRQ and SWI) located @ $7FF1 to $7FFF. This routine or a similar routine should be included in a user program which is invoked by the jump to START feature of "MONITEUR".***This is the only routine that is not compatible with BUFFALO as MicroNator System uses only 5 pseudo-vectors.***

ROUTINES of "MONITEUR" COMPATIBLE with BUFFALO 3.xx

$FF7C	JMP WARMST	Go to the ">" prompt point (skip "MONITEUR" message)
$FF7F	JMP BPCLR	Clear breakpoint table.
$FF82	JMP RPRINT	Display user's registers.

$FF85	JMP HEXBIN	Convert ASCII character in A register to 4-bit binary number. Shift binary number into SHIFTREG from the right. SHIFTREG is a 2-byte (4 hexadecimal digits) buffer. If A register is not hexadecimal, location TMP1 is incremented and SHIFTREG is unchanged.
$FF88	JMP BUFFAR	Read 4-digit hexadecimal argument from input buffer to SHIFTREG.
$FF8B	JMP TERMAR	Read 4-digit hexadecimal argument from SCI device to SHIFTREG.
$FF8E	JMP CHGBYT	Write value (if any) from SHIFTREG + 1 to memory location pointed to by X. (Operation applicable to all locations (except for CONFIG at $003F) in internal RAM or external EEPROM).
$FF91	JMP READBU	Read next character from INBUFF.
$FF94	JMP INCBUF	Increment pointer into input buffer.
$FF97	JMP DECBUF	Decrement pointer into input buffer.
$FF9A	JMP WSKIP	Read input buffer until non-white space character found.
$FF9D	JMP CHKABR	Monitor input for (CTRL)X or (CTRL)W requests.
$FFA0	JMP UPCASE	If character in accumulator A is lower case alpha, convert to upper case.
$FFA3	JMP WCHEK	Test character in accumulator A and return with Z bit set if character is white space (space, comma, tab).
$FFA6	JMP DCHEK	Test character in accumulator A and return with Z bit set if character is delimiter (carriage return or white space).
$FFA9	JMP INIT	Initialize I/O device (SCI).
$FFAC	JMP INPUT	Read I/O device (SCI).
$FFAF	JMP OUTPUT	Write I/O device (SCI).
$FFB2	JMP OUTLHL	Convert left nibble of accumulator A contents to ASCII and output to terminal port (SCI).
$FFB5	JMP OUTRHL	Convert right nibble of accumulator A contents to ASCII and output to terminal port (SCI).
$FFB8	JMP OUTA	Output accumulator A ASCII character.

$FFBB	JMP OUT1BY	Convert binary byte at address in index register X to two ASCII characters and output. Returns address in index register X pointing to next byte.
$FFBE	JMP OUT1BS	Convert binary byte at address in index register X to two ASCII characters and output followed by a space. Returns address in index register X pointing to next byte.
$FFC1	JMP OUT2BS	Convert two consecutive binary bytes starting at address in index register X to four ASCII characters and output followed by a space. Returns address in index register X pointing to next byte.
$FFC4	JMP OUTCRL	Output ASCII carriage return followed by a line feed.
$FFC7	JMP OUTSTR	Output string of ASCII bytes pointed to by address in index register X until character is end of transmission ($04).
$FFCA	JMP OUTST0	Same as OUTSTR except leading carriage return and line feed is skipped.
$FFCD	JMP INCHAR	Input ASCII character to accumulator A and echo back. This routine loops until character is actually received.

When addressing “MONITEUR” utility routines, always reference the routines by the applicable address ($FF67 through $FFCD) in the JMP table rather then the actual address in the “MONITEUR” program. Jump table addresses remain the same when a new version of “MONITEUR” is developed even though the actual addresses of the routine may change. Programs that reference routines by the jump table addresses are not required to be changed to operate on revised versions of the “MONITEUR” program.

5.2.5 Special EEPROM Writing Routine

The only disadvantage of EEPROM is that their I/O pins turn into high impedance state after the CPU has finished writing data into them.

If the CPU tries to read back the data right away after writing, it will read the complement of the last byte written into the EEPROM. The state of high impedance lasts from 5 to 10 msec then the EEPROM returns to its normal operating state. The CPU can read it as a normal memory device only after such a delay

If someone wants to write into an EEPROM, he has to find out one of the main parameter of the device which is the page size. The page size is the number of bytes you can write into the EEPROM

at any one time. It can be anything from 8 to 64 bytes to the entire EEPROM.

The next parameter to find out is how much time, in micro-seconds after the WRITE pin of the EEPROM goes inactive, the CPU has to write another byte before the EEPROM goes into high impedance. This amount of time ranges from 10 usec to 100 usec.

Also if the CPU writes more than one byte, the user must make sure he does not cross the page boundary while writing the string.

Normally a programmer writes the string into the EEPROM taking care of all the requirements of the device. Then the CPU jumps to a time-delay subroutine stored into another memory device. After looping through the delay he returns to the calling routine and he checks the bytes written. If everything is verified satisfactory, he continues his program.

Another way for the verification is to write one byte at a time to the EEPROM. After writing, you continuously read back the device bit7 (EEPROM busy flag). This read cycle can be at any address of the EEPROM just written into. If the bit7 is HIGH (1) the device is still busy writing the byte. If the bit7 is LOW (0) the device has finished writing and the user can check back the written byte. This sequence has the advantage of taking less time because the EEPROM becomes available as soon as it detects the written bytes is actually well programmed.

This verify-flag-after-writing program takes more bytes to code than just a write-then-delay program.

All the 32Kbytes EEPROM are guaranteed for 100,000 writes but they will go far beyond that limit (300,000 to 400,00 times).

The MicroNator System wants to be one taking the least amount of silicon so as to keep the price of the system as low as possible. Since there is only one EEPROM in the system the monitor has to take another way to write into it because it can not loop through the delay in the EEPROM for the memory device is in high impedance state.

The only other kind of memory available to the monitor is the internal RAM. The external one ***might not be present in the user design***. “MONITEUR” takes the delay subroutine stored in itself (refer to 5.2.5.1 below) and uses the available RAM to store it into the STACK; jumps to the routine in the STACK; writes the byte into the EEPROM; loops through the delay; returns to the WRITE routine in “MONITEUR”; re-ajusts the STACK; cheks if the byte was written correctly, display an error message if it was not; and finally return from subroutine. If there are many bytes to be written the program loops until end of string.

Downloading a program is done differently through the bootstrap mode of operation however it uses almost the same idea.

If the user has to write into the external EEPROM, it is only necessary to incorporate the WRITE subroutine into it. Make sure the STACK has enough space to contain the delay subroutine and stacking

information which, in all, take 19 bytes of the STACK.

Note

Please remember that those routines are copyrighted and that you need a licence if you want to incorporate them into you program.

5.2.5.1 Part of Write.asm *(Copyright 1990 by Michel-André Robillard of T.Sc.A.)*

Before executing those routines, the byte to be written is stored into "A" and the address into "X".

Those routines take 19 bytes of stack.

The part that verifies if the address is "CONFIG" and the one that displays error messages are not included in this assembler program.

NOTE

The addresses below may change with different version of "MONITEUR".

(COPYRIGHT 1990 by Michel-André Robillard of RF-232)

```
4478                        ********************
4428                        *
4429                        ***************
4430                        *   -   WRITE() This routine is used to write the content
4431                        *           of A to the address in X.
4432                        *   -   If the user try to write to CONFIG
4433                        *           an error message is displayed.
4434                        *   -   All registers are saved.
4435                        ***************
4436                        *
...
4503 fd89 18 3c             WRITE_OK    PSHY
4504 fd8b 37                           PSHB
4505 fd8c 3c                           PSHX
4506 fd8d 36                           PSHA
4507
4508 fd8e c6 39                        LDAB    #$39
4509 fd90 37                           PSHB
4510 fd91 c6 fd                        LDAB    #$FD
4511 fd93 37                           PSHB
4512 fd94 c6 26                        LDAB    #$26
4513 fd96 37                           PSHB
4514 fd97 c6 09                        LDAB    #$09
4515 fd99 37                           PSHB
4516 fd9a c6 00                        LDAB    #$00
4517 fd9c 37                           PSHB
4518 fd9d c6 08                        LDAB    #$08
```

```
4519 fd9f 37                            PSHB
4520 fda0 c6 ce                         LDAB    #$CE
4521 fda2 37                            PSHB
4522 fda3 c6 00                         LDAB    #$00
4523 fda5 37                            PSHB
4524 fda6 c6 a7                         LDAB    #$A7
4525 fda8 37                            PSHB
4526
4527 fda9 18 ce fd b8                   LDY     #WR_RTS
4528 fdad 18 3c                         PSHY
4529 fdaf 18 30                         TSY
4530 fdb1 18 08                         INY
4531 fdb3 18 08                         INY
4532 fdb5 18 3c                         PSHY
4533 fdb7 39                            RTS
4534
4535 fdb8 38                WR_RTS      PULX
4536 fdb9 38                            PULX
4537 fdba 38                            PULX
4538 fdbb 38                            PULX
4539 fdbc 32                            PULA
4540 fdbd 32                            PULA
4541 fdbe 38                            PULX
4542 fdbf 33                            PULB
4543 fdc0 18 38                         PULY
4544 fdc2 a1 00                         CMPA    0,X
4545 fdc4 27 0a                         BEQ     END_END
4546 fdc6 3c                            PSHX
4547 fdc7 36                            PSHA
4548 fdc8 ce fd d1                      LDX     #MSG_ERR
4549 fdcb bd ee 2d                      JSR     OUTSTRG
4550 fdce 32                            PULA
4551 fdcf 38                            PULX
4552 fdd0 39                END_END     RTS
```

5.3 WRITING INTERRUPT VECTORS

The user looks upon the interrupt vectors residing in EEPROM as ordinary bytes which are accessible as follows:

Writing the high byte of the vector subroutine address:

• Load the high byte of the subroutine's address into the accumulator A.
• Load the vector's high byte address in the index register X.
• Call the WRITE subroutine.

Writing the low byte of the vector subroutine address:

• Load the low byte of the subroutine's address into the accumulator A.
• Load the vector's low byte address in the index register X.
• Call the WRITE subroutine.

...That's all there is to it...

Since EEPROM are guaranteed to be written only 100 000 times and subroutines like TRACE, STOPAT, ASM etc. modify often the addresses of the vectors then it is better to put the often modified vectors into RAM instead of EEPROM.

It is for this reason that there are 5 indirect interrupt vectors which are assigned a three bytes field residing in the upper MicroNator ***System external RAM*** locations near $7FFF. Each real interrupt vector in EEPROM points to one of those three bytes field which is used as a jump table to the actual interrupt service subroutine. TABLE: 4 below lists those indirect interrupt vectors and associated three byte fields.

TABLE: 4 Interrupt Pseudo-Vector Jump Table

INTERRUPT VECTOR	FIELD
Serial Communication Interface (SCI)	$7FF1-$7FF3
Serial Peripheral Interface (SPI)	$7FF4-$7FF6
Timer Output Compare 5 (TOC5)	$7FF7-$7FF9
XIRQ	$7FFA-$7FFC
Software Interrupt (SWI)	$7FFD-$7FFF

To use vectors specified in TABLE: 4 above , the user inserts a jump extended opcode ($7E) and an address in the three bytes field of the required indirect interrupt vector.

EXAMPLE:

For the SCI vector, the following is performed:

1) Place *$7E* (extended JMP) at location $7FF1.

2) Place SCI interrupt subroutine address at location $7FF2 and $7FF3

In this example the SCI interrupt subroutine starts at **$FC00**:

```
$ADDR   OC OP OP   label   opcod   oper     *comment

$FB00   86 7E      S_INIT  LDAA    #$7E     *JMP instruction
$FB02   B7 7F F1           STAA    $7FF1    *Store it in $7FF1
$FB05   CE FC 00           LDX     #SCI_AD  *Get SCI Routine address
$FB08   FF 7F F2           STX     $7FF2    *Store address @ $7FF2/3
...
...
$FC00   BD FF AC   SCI_AD  JSR     INPUT    *Start of SCI Interrupt
...                                         *located in vector table
```

Of course the user may write directly to the real vector address with the WRITE subroutine as

explained above in paragraph 5.2.5 on page 49 and in paragraph 5.3 on page 52.

During initialization “MONITEUR” checks the first byte of each of the five locations. If a jump opcode ($7E) is not found, “MONITEUR” will install a jump to a routine called STOPIT. This assures there will be no uninitialized interrupt vectors which would cause undesirable operation during power up and power down. If an interrupt is accidentally encountered, the STOPIT routine will force a STOP instruction sequence to be executed.

NOTE

The STOPIT routine is an endless loop. ***Only a reset can take the CPU out from that routine*** as it is in STOP mode and the XIRQ is also pointing to STOPIT.

A user may replace any of the JMP STOPIT instructions with a JMP to a user written interrupt service routine. If a hardware reset is issued via the RESET switch (SW1), “MONITEUR” will not overwrite these user jump instructions so they need not be re-initialized after every reset.

TABLE: 5 Vector Table

```
ffd0                                ORG     $FFD0

                          *** Vectors ***

*Do not change MONITEUR (used when downloading to adjust RESET vector) for more
*information see section 3.2.1 on page 16
*
ffd0                      VERSION   EQU     *
ffd0 04 00                          FDB     $0400    *Version number of MONITEUR
ffd2 eb 12                          FDB     MONITEUR *Used for downloading
ffd4 00 00                          FDB     0
ffd6 7f f1                VSCI      FDB     JSCI
ffd8 7f f4                VSPI      FDB     JSPI
ffda ed 98                VPAIE     FDB     STOPIT
ffdc ed 98                VPAO      FDB     STOPIT
ffde ed 98                VTOF      FDB     STOPIT
ffe0 7f f7                VTOC5     FDB     JTOC5
ffe2 ed 98                VTOC4     FDB     STOPIT
ffe4 ed 98                VTOC3     FDB     STOPIT
ffe6 ed 98                VTOC2     FDB     STOPIT
ffe8 ed 98                VTOC1     FDB     STOPIT
ffea ed 98                VTIC3     FDB     STOPIT
ffec ed 98                VTIC2     FDB     STOPIT
ffee ed 98                VTIC1     FDB     STOPIT
fff0 ed 98                VRTI      FDB     STOPIT
fff2 ed 98                VIRQ      FDB     STOPIT
fff4 7f fa                VXIRQ     FDB     JXIRQ
fff6 7f fd                VSWI      FDB     JSWI
fff8 ed 98                VILLOP    FDB     STOPIT
fffa ed 98                VCOP      FDB     STOPIT
fffc ed 98                VCLM      FDB     STOPIT
fffe eb 12                VRST      FDB     MONITEUR
```

CHAPTER 6

SUPPORT INFORMATION

6.1 INTRODUCTION

This chapter provides the connector signal descriptions for the MicroNator System.

6.2 CONNECTOR SIGNAL DESCRIPTIONS

Connector P1 (VIN) interconnects an external power supply to the MicroNator System. Connector J2 (Serial Connector) is provided to facilitate interconnection to a personnel computer (PC). Connector J3 (BUS CONNECTOR) is the standard bus "proprietary of RF-232" for expansion I/O board. J5 is the MCU I/O interface connector used to connect all MCU I/O pins to the outside world.

Pins assignments for the above connectors are identified in following tables. Connector signals are identified by pin number, signal mnemonic, I/O type, signal name and a brief description.

6.3 POWER INPUT CONNECTOR

TABLE: 6 Input Power Connector (P1) Pin Assignments

PIN NUMBER	SIGNAL MNEMONIC	I/O	SIGNAL NAME AND DESCRIPTION
1	GND	-	MicroNator System Ground.
2	VIN	I	+VDC input to the 5 volts on-board regulator used by the MicroNator System logic circuits.

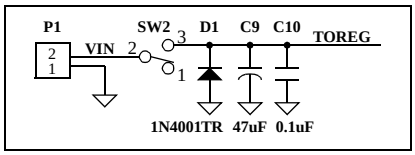


Fig: 5 Input Power "VIN"

NOTE:

VIN is also on bus connector J3 pin 63.

6.4 RS-232C CONNECTOR

TABLE: 7 Serial I/O communication Port (J2)

PIN NUMBER	SIGNAL MNEMONIC	I/O	SIGNAL NAME AND DESCRIPTION
1		-	Not connected.
2	PC-TX	I	PC TRANSMIT-DATA - Serial data input line to MicroNator.
3	PC-RX	O	PC RECEIVE-DATA - Serial data output line from MicroNator.
4		-	Not connected
5	PC-SGND	-	Signal Ground.
6	DTR*	I	Standard RS-232C Data Terminal Ready.
7		-	Not connected.
8	RTS*	I	Standard RS-232C Request To Send.
9		-	Not connected.

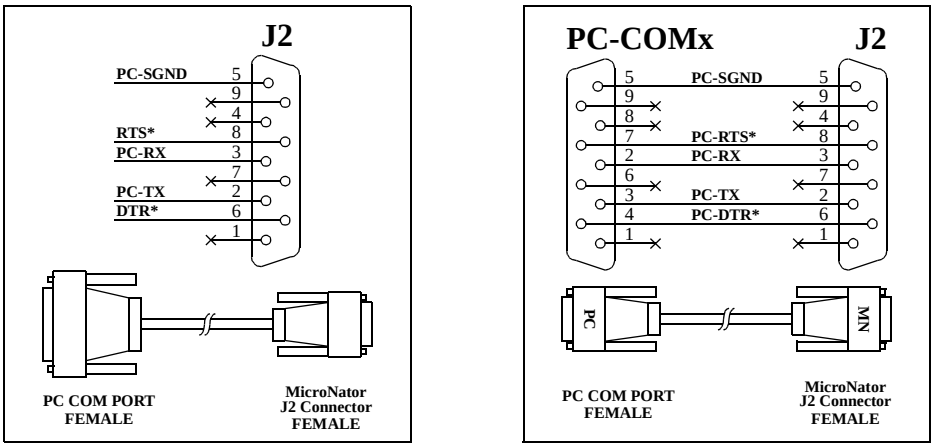


Fig: 6 Serial Connector & Signals

Fig: 6 shows the schematic and the standard cable used for the communication between the PC and the MicroNator System. The DB25 female plugs into the COM port of the PC and the DB9 male into J2 of the MicroNator System.

6.5 UCT BUS CONNECTOR

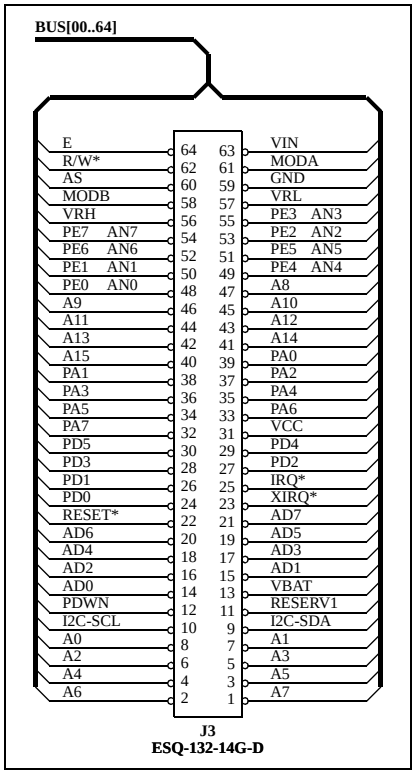


Fig: 7 Bus Description

PIN #	SIGNAL	I/O	SIGNAL NAME AND DESCRIPTION
29	GND	-	GND.
58	MODB*	I	MODE B - An input control line used in conjunction with the MODA pin to select the MCU mode of operation.
	VSTBY	I	STANDBY VOLTAGE - An input MCU RAM standby power line.
61	MODA*	I	MODE A - An input line used in conjunction with the MODB pin to select the MCU mode of operation
	LIR*	O	LOAD INSTRUCTION REGISTER - An open-drain output signal used to indicate an instruction is starting.
60	AS	O	ADDRESS STROBE - An output control line used to de-multiplexed port C address and data signals in the expanded multiplexed mode of operation.
	STRA*	BI	STROBE A - An input edge detecting signal for parallel I/O device

			handshaking in the single-chip mode of operation.
64	E	O	ENABLE CLOCK - An output control line used for timing reference. E clock frequency is one fourth the frequency of the XTAL and EXTAL pins.
62	R/W*	O	READ/WRITE - An output control line used to control the direction of transfers on the MCU external data bus in the expanded multiplexed mode of operation.
	STRB*	O	STROBE B - An output strobe signal for parallel I/O device handshaking in the single-chip mode of operation.
11	RESERV1	-	Not connected, reserved for future expansion.
10	I2C-SCL	-	Not connected, reserved for I2C communication in future expansion.
09	I2C-SDA	-	Not connected, reserved for I2C communication in future expansion.
13	VBAT	I/O	Battery back-up voltage (one diode drop below VCC).
15	PC0/AD0	I/O	PORT C (bits 0-7) - General purpose I/O lines.
15	PC1/AD1		
16	PC2/AD2		
17	PC3/AD3		
18	PC4/AD4		
19	PC5/AD5		
20	PC6/AD6		
21	PC7/AD7		
22	RESET*	BI	RESET - An active low bi-directional control line used to initialized the MCU.
23	XIRQ*	I	NON-MASKABLE INTERRUPT - An active low input line used to request asynchronous non-maskable interrupts to the MCU.
25	IRQ*	I	INTERRUPT REQUEST - An active low input line used to request asynchronous interrupts to the MCU.
24	PD0/RXD	I/O	PORT D (bits 0-5) - General purpose I/O lines. These lines can be used with the MCU Serial Communications Interface (SCI) and Serial Peripheral Interface (SPI).
26	PD1/TXD		
27	PD2/MISO		
28	PD3/MOSI		
29	PD4/SCK		
30	PD5/SS*		

12	PDWN	BI	POWER-DOWN - Active HIGH signal to put the system in low power mode.
32	PA7/OC1	I/O	PORT A (bits 7-0) - General purpose I/O lines and/or timer signals.
33	PA6/OC2		
34	PA5/OC3		
35	PA4/OC4		
36	PA3/OC5		
37	PA2/IC1		
38	PA1/IC2		
39	PA0/IC3		
40	PB7/A15	O	PORT B (bits 7-0) - General purpose output lines.
41	PB6/A14		
42	PB5/A13		
43	PB4/A12		
44	PB3/A11		
45	PB2/A10		
46	PB1/A9		
47	PB0/A8		
48	PE0/AN0	I/O	PORT E (bits 0-7) - General purpose input or A/D channel input lines.
50	PE1/AN1		
53	PE2/AN2		
55	PE3/AN3		
49	PE4/AN4		
51	PE5/AN5		
52	PE6/AN6		
54	PE7/AN7		
08	A0	O	De-multiplexed output A0-A7 addresses.
07	A1		
06	A2		
05	A3		
04	A4		
03	A5		
02	A6		
01	A7		
31	VCC	-	+5 Vdc outputted directly from the regulator.
63	VIN	-	Unregulated input power voltage. In case the expansion requires more than 1 Amp of input power it will be possible to implement a more powerful regulator to add instead of the one on the CPU board. The power pin from the standard power connector (P2) is also connected to this pin.

6.6 I/O PORT CONNECTOR

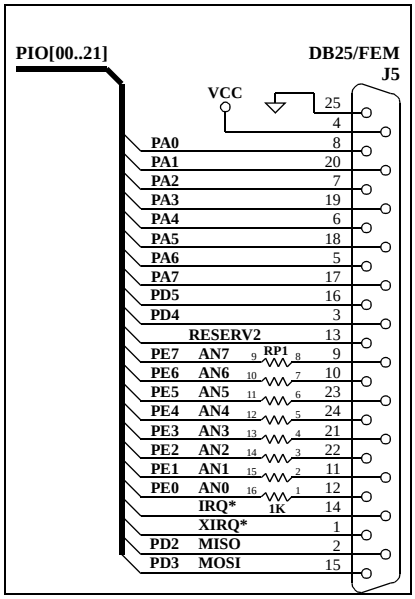


Fig: 8 MCU I/O Connector

PIN #	SIGNAL NAME	I/O	SIGNAL NAME AND DESCRIPTION
25	GND	-	GND.
4	VCC		+5Vdc - System main voltage supply.
8	PA0/IC3	I/O	PORT A (bits 7-0) - General purpose I/O lines and/or
20	PA1/IC2		timer signals.
7	PA2/IC1		
19	PA3/OC5		*** Special for TRACE.
6	PA4/OC4		
18	PA5/OC3		
5	PA6/OC2		
17	PA7/OC1		
16	PD5/SS*	I/O	PORT D (bits 5-2) - General purpose I/O lines./ SPI.
3	PD4/SCK	I/O	PORT D (bits 5-2) - General purpose I/O lines./ SPI.
13	RESERV2	-	Not connected, reserved for future expansion. Most probably E.
9	PE7/AN7	IO	PORT E (bits 0-7) - General purpose input or A/D channel input lines.
10	PE6/AN6		
23	PE5/AN5		
24	PE4/AN4		
21	PE3/AN3		
22	PE2/AN2		

11	PE1/AN1		
12	PE0/AN0		
14	IRQ*	I	INTERRUPT REQUEST - An active LOW input line used to request asynchronous interrupts to the MCU.
1	XIRQ*	I	NON-MASKABLE INTERRUPT - An active LOW input line used to request asynchronous non-maskable interrupts to the MCU.
2	PD2/MISO	IO	PORTD-2 / Master In Slave OUT
15	PD3/MOSI	IO	PORTD-3 / Master Out Slave In.

CHAPTER 7

EXPANSION OPTIONS

7.1 INTRODUCTION

This chapter provides information on the expansion options.

7.2 EXPANSION BOARD

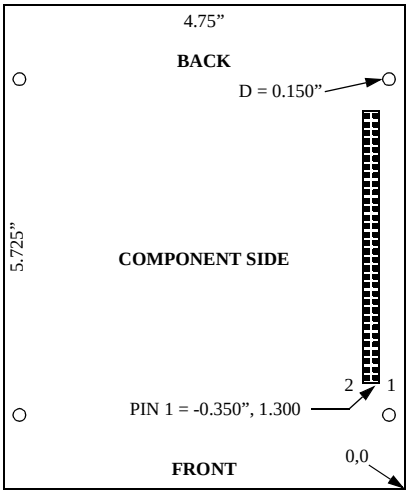


Fig: 9 Standard CPU-1/64e2

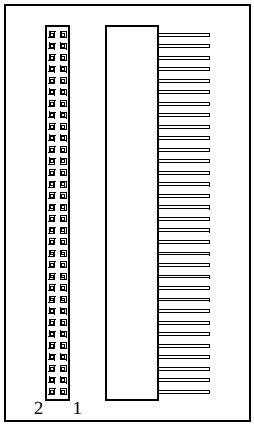


Fig: 9 BUS Connector

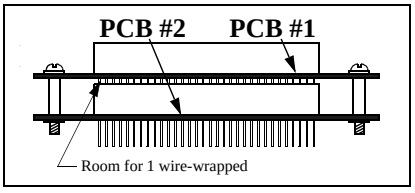


Fig: 10 System with two boards

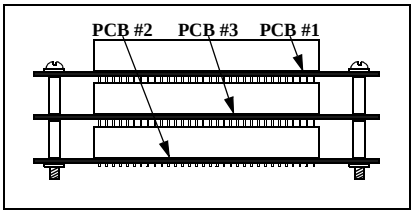


Fig: 11 System with three boards

Fig: 9, Fig: 9, Fig: 10, and Fig: 11 show the optional expansion Wire Wrap boards and their expansion mechanism. The boards go on top or on the bottom of the MicroNator System board. **No need for a mother board or cable to connect them together**. ***The BUS connectors mate with each other*** and form a sandwich with a “thickness”, center to center, of 11/16 inch. (0.625” + 0.0625” = 0.6875”).

If the user wants to add expansion board, using ***wire-wrapped*** sockets, on top of the MCU board, he should order an extra expansion connector to put between the two boards so as to extend the distance between them. That way ***the extra length of the wire wrap pins*** will fit between the two boards and they will not touch any component on the CPU board.

Usually the user is able to debug his developed board from the top of it. After the board is fully debugged the user can plug it underneath the MCU board.

APPENDIX A

S-RECORD INFORMATION

A.1 INTRODUCTION

The S-record format for input modules was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. The transportation process can thus be visually monitored and the S-records can be more easily edited.

A.2 S-RECORD CONTENT

When viewed by the user, S-records are essentially character made of several fields which identify the record type, record length, memory address, code/data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first character representing the high-order 4 bits, and the second the low-order 4 bits of the byte.

The 5 fields which comprise an S-record are shown below:

TYPE
RECORD LENGTH
ADDRESS
CODE/DATA
CHECKSUM

Where the fields are composed as follows:

TABLE: 1 S-RECORD Format

TYPE	PRINTABLE CHARACTER	CONTENTS
Type	2	S-record type - S0, S1, etc.
Record length	2	The count of the character pairs in the record, excluding the type and record length
Address	4,6, or 8	The 2-, 3-, or 4-byte address at which the data field is to be loaded into memory.
Code/data	0-2n	From 0 to n bytes of executable code, memory loadable

		data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S-record).
Checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

Each record may be terminated with a CR/LF/NULL. Additionally, an S-record may have an initial field to accommodate other data such as line numbers generated by some time-sharing systems.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

A.3 S-RECORD TYPES

Eight types of S-records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. The various Motorola upload, download, and other record transportation control programs, as well as cross-assemblers, linkers, and other file creating or debugging programs, utilize only those S-record which serve the purpose of the program. For specific information on which S-records are supported by a particular program, the user manual for that program must be consulted.

NOTE

The MicroNator System monitor supports only the S1 and S9 records. All data before the first S1 record is ignored. Thereafter, all records must be S1 type until the S9 record terminates data transfer.

An S-record format module may contain S-records of the following type:

S0 The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. The address field is normally zeroes.

S1 A record containing code/data and the 2-byte address at which the code/data is to reside.

S2-S8 Not applicable to MicroNator System.

S9 A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed. If not

specified, the first entry point specification encountered in the object module input will be used. There is no code/data field.

Only one termination record is used for each block of S-records. Normally, only one header record is used, although it is possible for multiple header records to occur.

S-record format programs may be produced by several dump utilities, debuggers, or several cross assemblers or cross linkers. Several programs are available for downloading a file in S-record format from a host system to an 8-bit or 16-bit microprocessor-based system.

A.4 S-RECORD EXAMPLE

Shown below is a typical S-record format module, as printed or displayed:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The above value module consists of an S0 header record, four S1 code/data records, and an S9 termination record.

The S0 header record contains the following character pairs:

S0 S-record type S0, indicating a header record.

06 Hexadecimal 06 (decimal 6), indicating six character pairs (or ASCII bytes) follow.

00 00 Four-character 2-byte address field, zeroes.

44 48 52 ASCII “H”, “D”, and “R”- (HDR).

1B Checksum of S0 record.

The first S1 code/data record is explained as follows:

S1 S-record type S1, indicating a code/data record to be loaded/verified at a 2 byte address.

13 Hexadecimal 13 (decimal 19), indicating 19 character pairs, representing 19 bytes of binary data, follow.

00 Four-character 2-byte address field; hexadecimal address 0000, in-
00 dicates location where the following data is to be loaded.

The next 16 character pairs are the ASCII bytes of the actual program code/data. In this assembly language example, the hexadecimal opcodes of the program are written in sequence in the code/data fields of the S1 records:

28	5F		BHCC	$0161
24	5F		BCC	$0163
22	12		BHI	$0118
22	6A		BHI	$0172
00	04	24	BRSET	0,$04,$012F
29	00		BHCS	$010D
08	23	7C	BRSET	4,$23,$018C

(Description of this code/data fields of the remaining S1 records, and stored in memory location 0010, etc...)

2A Checksum of the first S1 record.

The second and third code/data records each also contain $13 (19) character pairs and are ended with checksums 13 and 52, respectively. The fourth S1 code/data record contains 07 character pairs and has a checksum of 92.

The S9 termination record is explained as follows:

S9 S-record type S9, indicating a termination record.

03 Hexadecimal 03, indicating three character pairs (3 bytes) follow.

00 Four-character 2-byte address field, zeroes.
00

FC Checksum of S9 record.

APPENDIX B

B.1 CONFIG REGISTER

```
* To change the config REGISTER:
*
* 1) Edit “CONFIG.ASM”. Go to the first line of the EQUATES and
*    replace $0C with the new HEX value you want in the CONFIG register.
*
*  ***************
*  * EQUATES *
*  ***************
*
*    CF_VALUE EQU $0C value of CONFIG register
*
*
* 2) Assemble CONFIG.ASM with AS11.EXE:
*
*    - ex: AS11 CONFIG.ASM -L > CONFIG.LST
*
* 3) Check CONFIG.LST for errors.
*
* 4) Convert CONFIG.S19 to CONFIG.BIN with MOTO2BIN.EXE
*
*    - Execute MOTO2BIN.EXE
*
*    - Choose: 1. Convertir S19 en BIN (use arrow to move around)
*    - Answer: with ENTER (CARRIER RETURN)
*
*    - Then: Nom du fichier:
*    - Answer: CONFIG
*
*    - Then: Déplacement: 00000
*    - Answer: with ENTER (CARRIER RETURN)
*
*    - Then: 0. FIN (you can move around with the arrow keys)
*    - Answer: with ENTER (CARRIER RETURN) to exit
*
*    - Now MOTO2BIN.EXE will generate a CONFIG.BIN
* 5) You have to rename CONFIG.BIN to BS_EEPRM.BIN
*
```

Save the original BS_EEPRM.BIN so you will be able to find it later. (Very important because if you don’t have the original BS_EEPRM.BIN you will never be able to download regular programs to MicroNator System).

```
*
*    - Rename CONFIG.BIN to BS_EEPRM.BIN
*
* 6) Execute TALK.EXE
```

```
*
* 7) Now press ALT B just like you would download ordinary boot.
*
*    - The program generate an ERROR6 Boostrap error:-4 after a few seconds
*      This is normal.
*
*    - Press any key to get out of bootstrap.
*
*    - The CONFIG register is now programmed to your new value.
*
* 8) Exit TALK by pressing ALT X.
*
* 9) Get back your original BS_EEPRM.BIN
*
* 10)Verify your new CONFIG register.
*
*    - Execute TALK. (If the prompt does not appear It may be necessary
*      to redownload “MONITEUR” with ALT B)
*
*    - Examine memory:
*      MM 103f
*
*    - Your new value for CONFIG register will be displayed.
*
* ------------------------------------------------------------------------
```

APPENDIX C

REAL TIME CLOCK ROUTINES

C.1 REAL TIME CLOCK DECODING

From $0280 to $03BF the decoder will select the Real Time Clock. The chip select of the RTC is its SS pin # 7. It is active HIGH and it has to stay HIGH for the entire communication period when using the SPI protocol. When not communicating it has to be LOW. So the decoder is made to latch HIGH or LOW by the PALCE22V10.

If you ***read*** address $0280, the RTC chips delect will latch to a ***HIGH*** level and stay that way.

If you ***write*** to address $0280, the RTC chips delect will latch to a ***LOW*** level and stay that way.

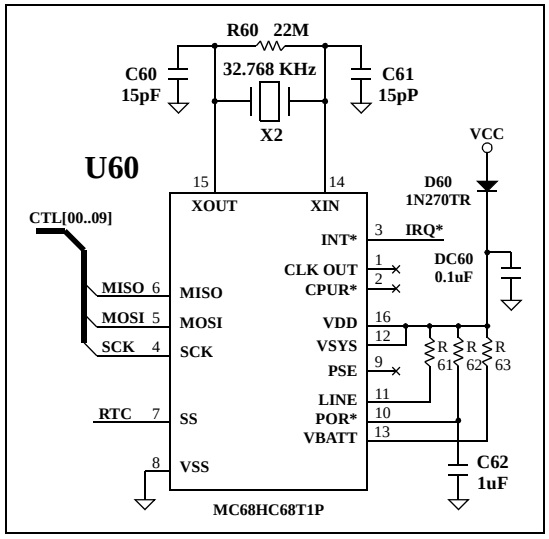


Fig: 12 Real Time Clock

So before using the SPI to communicate with the Real Time Clock, you can do a:

LDAA $0280 .....*PALCE22V10 latches SS pin #7 ACTIVE*

and the SS pin of the RTC will become active i.e. HIGH and stay that way because you are reading address $0280.

You exchange information with the RTC using the SPI protocol.

After the communication is finished, you do a:

STAA $0280.....*PALCE22V10 latches SS pin #7 NOT ACTIVE

and the SS pin of the RTC will become inactive i.e. LOW and stay that way because you wrote at the address $0280

This way of implementing the circuit is to free the SS* pin of the CPU in case the user wants to add another slave or even another master on the SPI bus.

C.2 REAL TIME CLOCK ROUTINES

NOTE

The addresses of those routines may vary with version number.

```
                        ************************************************************************
                        * Software support routines for the MC68HC68T1 Real-Time-Clock/RAM       *
                        * Routines include...                                                   *
                        * RTCINIT - Subroutine to initialize Port D and SPI system.             *
                        * SETDFLT - Subroutine to set up a default time 12:00:00 Mon 1/1/1995   *
                        *           and alarm set to same time but turned off.  RTC turned on.  *
                        * SETTIME - Subroutine to set time sec/min/hr as pointed to by Y-reg    *
                        * SETDATE - Subroutine to set date dow/dom/month/yr as pointed to by Y  *
                        * SETALRM - Subroutine to set alarm time sec/min/hr as pointed to by Y  *
                        * W1RTC   - Subroutine to write A data to addr B in RTC                 *
                        * R1RTC   - Subroutine to read A data from addr B in RTC                *
                        * WBURST  - Burst write B bytes starting at addr A in RTC, uses Y index *
                        * RBURST  - Burst raed B bytes starting at addr A in RTC, uses Y index  *
                        * DISPTIM - Display current time (alternate entry to do CR,LF first)    *
                        ************************************************************************
                        *
                        * Send one char to the terminal
                        *
fddf f6 10 2e           WRCH        LDAB    SCSR            Wait for the port to be empty
fde2 c5 80                          BITB    #$80
fde4 27 f9                          BEQ     WRCH
fde6 84 7f                          ANDA    #$7F            Take out parity
fde8 b7 10 2f                       STA     SCDAT
fdeb 39                             RTS
                        *
                        ************************************************************************
                        * RTCINIT - Subroutine to initialize Port D and SPI system.             *
                        ************************************************************************
fdec ce 10 00           RTCINIT     LDX     #REGBS          Point at start of register block
fdef 86 00                          LDAA    #$00
fdf1 a7 08                          STAA    PORTD,X         Init port D data
fdf3 86 18                          LDAA    #$18            SS=in,SCK,and MOSI=OUTS, others=INS...
                        *                                   ... to enable other master on SPI
fdf5 a7 09                          STAA    DDRD,X          Set directions for port D pins
fdf7 86 54                          LDAA    #$54            SPIE,SPE,DWOM,MSTR;CPOL,CPHA,SPR1,SPR0
fdf9 a7 28                          STAA    SPCR,X          SPI On, Master, CPOL:CPHA=0:1, 1MHz
fdfb 39                             RTS                     ** Return from RTCINIT **
```

```
                        ************************************************************************
                        * SETDFLT - Subroutine to set up a default time 12:00:00 Mon 1/1/1995    *
                        *       and alarm set to same time but turned off.  RTC turned on.       *
                        * SETOTHR - This is alt entry point to set time to that pointed-to by Y  *
                        *       Y must point to ordered set of 11 bytes (see DFLTSEC as Ex)      *
                        ************************************************************************
fdfc 18 ce fe 27        SETDFLT     LDY     #DFLTSEC        Point at data to set RTC for defaults
fe00 86 20              SETOTHR     LDAA    #RTCSEC         Addr of first clock data loc in RTC
fe02 c6 0b                          LDAB    #11             Number of bytes to burst transfer to RTC
fe04 8d 4e                          BSR     WBURST          Do burst transfer of default settings
fe06 b6 ff B5                       LDAA    RTCXTAL         RTC Clk ON, 32 KHz crystal, 60Hz,
                        *                                   CLK OUT = 1 HZ.
                        *                                   $85 => 4.194304 MHz   $95 => 2.097152 MHz
                        *                                   $A5 => 1.048576 MHz   $B5 => 32.768 KHz
                        *
fe09 c6 31                          LDAB    #RTCCTRL        Address for RTC control reg
fe0b 8d 25                          BSR     W1RTC           Transfer init control word to start RTC
fe0d 86 00                          LDAA    #$00            RTC Clk ON, 32KHz crystal, 60Hz,
                        *                                   CLK OUT = Xtal
fe0f c6 32                          LDAB    #RTCINT         Address for RTC interrupt control reg
fe11 8d 1f                          BSR     W1RTC           Turn off RTC interrupts
fe13 39                             RTS                     ** Return from SETDFLT or SETOTHR **
                        *
                        ************************************************************************
                        * SETTIME - Subroutine to set time sec/min/hr as pointed to by Y-reg     *
                        ************************************************************************
fe14 86 20              SETTIME     LDAA    #RTCSEC         Address of first byte of burst Sec/Min/Hr
fe16 c6 03                          LDAB    #3              Number of bytes to burst transfer
fe18 20 0a                          BRA     BRSTOUT         Go to common exit point
                        *
                        ************************************************************************
                        * SETDATE - Subroutine to set date dow/dom/month/yr as pointed to by Y   *
                        ************************************************************************
fe1a 86 23              SETDATE     LDAA    #WEEKDAY        Address of 1st byte of burst
                        *                                   WkDay/MoDat/Mo/Yr
fe1c c6 04                          LDAB    #4              Number of bytes to burst transfer
fe1e 20 04                          BRA     BRSTOUT         Go to common exit point
                        *
                        ************************************************************************
                        * SETALRM - Subroutine to set alarm time sec/min/hr as pointed to by Y  *
                        ************************************************************************
fe20 86 28              SETALRM     LDAA    #ALRMSEC        Address of first byte of burst Sec/Min/Hr
fe22 c6 03                          LDAB    #3              Number of bytes to burst transfer
fe24 8d 2e              BRSTOUT     BSR     WBURST          Burst transfer to RTC
fe26 39                             RTS                     *Return from SETALRM, SETDATE or SETTIME
fe27 00 00 92           DFLTSEC     FCB     $00,$00,$12+AM  12:00:00 AM
fe2a 01 01 01 95        DFLTDAT     FCB     $01,$01,$01,$95 Sun, 01/01/95
fe2e 00                             FCB     $00             Place holder for unused loc $27 in RTC
fe2f 00 00 92           DFLALRM     FCB     $00,$00,$12+AM  12:00:00 AM ($12+AM=$92)
                        *
                        ************************************************************************
                        * W1RTC   - Subroutine to write A data to addr B in RTC                  *
                        ************************************************************************
                        * R1RTC   - Subroutine to read A data from addr B in RTC                 *
                        ************************************************************************
fe32 ca 80              W1RTC       ORAB    #$80            Set MSB of Addr byte to indicate write
fe34 3c                 R1RTC       PSHX                    Save X for now
fe35 ce 10 00                       LDX     #REGBS          Point to start of register block
fe38 36                             PSHA
fe39 b6 02 80                       LDAA    $0280           READ $0280 => CPU's SS=HIGH
fe3c 32                             PULA
fe3d e7 2a                          STAB    SPDR,X          Write Addr byte to RTC
fe3f 1f 29 80 fc                    BRCLR   SPSR,X SPIF     * Wait for SPIF
```

```
fe43 a7 2a                      STAA    SPDR,X          Initiate xfer of A to RTC
fe45 1f 29 80 fc                BRCLR   SPSR,X SPIF     * Wait for SPIF


fe49 a6 2a                      LDAA    SPDR,X          Get data from RTC into A
fe4b 36                         PSHA
fe4c b7 02 80                   STAA    $0280           WRITE $0280 => CPU's SS=LOW
fe4f 32                         PULA
fe50 c4 7f                      ANDB    #$7F            Restore original value to B
fe52 38                         PULX                    Restore X
fe53 39                         RTS                     ** Return from W1RTC or R1RTC **
                    *
                    ********************************************************************************
                    * WBURST  - Burst write B bytes starting at addr A in RTC, uses Y index  *
                    ********************************************************************************
fe54 8a 80          WBURST      ORAA    #$80            Set MSB of Addr byte to indicate write
fe56 3c                         PSHX                    Save X for now
fe57 ce 10 00                   LDX     #REGBS          Point to start of register block
fe5a 36                         PSHA
fe5b b6 02 80                   LDAA    $0280           READ $0280 => CPU's SS=HIGH
fe5e 32                         PULA
fe5f a7 2a                      STAA    SPDR,X          Write Addr byte to RTC
fe61 1f 29 80 fc                BRCLR   SPSR,X SPIF     *  Wait for SPIF
fe65 18 a6 00       MOREW       LDAA    0,Y             Get next data byte to transfer
fe68 a7 2a                      STAA    SPDR,X          Initiate xfer of A to RTC
fe6a 1f 29 80 fc                BRCLR   SPSR,X SPIF     *  Wait for SPIF
fe6e a6 2a                      LDAA    SPDR,X          Get data from RTC into A (clears SPIF)
fe70 18 08                      INY                     Advance data pointer
fe72 5a                         DECB                    Continue for B bytes
fe73 26 f0                      BNE     MOREW           Loop till B gets to zero
fe75 36                         PSHA
fe76 b7 02 80                   STAA    $0280           WRITE $0280 => CPU's SS=LOW
fe79 32                         PULA
fe7a 38                         PULX                    Restore X
fe7b 39                         RTS                     ** Return from WBURST **
                    *
                    ********************************************************************************
                    * RBURST  - Burst raed B bytes starting at addr A in RTC, uses Y index   *
                    ********************************************************************************
fe7c 3c             RBURST      PSHX                    Save X for now
fe7d ce 10 00                   LDX     #REGBS          Point to start of register block
fe80 36                         PSHA
fe81 b6 02 80                   LDAA    $0280           READ $0280 => CPU's SS=HIGH
fe84 32                         PULA
fe85 a7 2a                      STAA    SPDR,X          Write Addr byte to RTC
fe87 1f 29 80 fc                BRCLR   SPSR,X SPIF     *   Wait for SPIF
fe8b a7 2a          MORER       STAA    SPDR,X          Initiate xfer to RTC (any data)
fe8d 1f 29 80 fc                BRCLR   SPSR,X SPIF     *   Wait for SPIF
fe91 a6 2a                      LDAA    SPDR,X          Get data from RTC into A (clears SPIF)
fe93 18 a7 00                   STAA    0,Y             Store new data byte
fe96 18 08                      INY                     Advance data pointer
fe98 5a                         DECB                    Continue for B bytes
fe99 26 f0                      BNE     MORER           Loop till B gets to zero
fe9b 36                         PSHA
fe9c b7 02 80                   STAA    $0280           WRITE $0280 => CPU's SS=LOW
fe9f 32                         PULA
fea0 38                         PULX                    Restore X
fea1 39                         RTS                     ** Return from RBURST **
                    *
                    ********************************************************************************
                    * DISPTIM - Display current time in the form HH:MM SS                    *
                    * CRLFTIM - alternate entry point to do CR,LF first                      *
                    ********************************************************************************
fea2 bd ee 22       CRLFTIM     JSR     OUTCRLF         Send leading CR,LF to display
7f54                TIMTMP      EQU     TEMP_RTC
```

```
fea5 18 ce 7f 54        DISPTIM     LDY     #TIMTMP         Point at 3 byte RAM holding area
fea9 c6 03                          LDAB    #3              Request read of 3 bytes...
feab 86 20                          LDAA    #RTCSEC         Starting with seconds address
fead 18 3c                          PSHY                    Will need it again
feaf 8d cb                          BSR     RBURST          Read in current time from MC68HC68T1
feb1 38                             PULX                    Original value of Y now in X TIMTMP
                        * I don't really like this sequence but I need to accomodate the calling
                        * requirements of the Buffalo 3.2 routine "OUT1BYT".  Bytes are converted
                        * to two ASCII hex characters and displayed.  Data needs to be in the order
                        * it will be displayed and pointed-to by X.
feb2 a6 02                          LDAA    2,X             Data in wrong order, get Hours
feb4 84 1f                          ANDA    #$1F            Strip off 12 Hr coding bits
                        * This routine doesn't handle 24 Hr mode time (you would change $1F to $7F)
feb6 e6 00                          LDAB    0,X             Get Seconds
feb8 e7 02                          STAB    2,X
feba a7 00                          STAA    0,X             Data now in correct order
febc bd ee 0a                       JSR     OUT1BYT         Display hours (X moves to Minutes)
febf 86 3a                          LDAA    #':'            An ASCII colon
fec1 bd ed ab                       JSR     OUTPUT          Display colon between Hr and Min
fec4 bd ee 19                       JSR     OUT1BSP         Display Minutes with trailing space
fec7 bd ee 0a                       JSR     OUT1BYT         Display Seconds
feca 39                             RTS
```

NOTE

Those RTC routines were taken from the monitor of the evaluation board SBC68HC11 from:

CEGEP André Laurendeau, Lasalle, Qc.

and modified by:

Michel-André Robillard T.Sc.A. for RF-232

C.3 REAL TIME CLOCK CRYSTAL FREQUENCY

If the user decides to use another crystal for the RTC, default is 32.768 KHz, he just has to change the byte at $FF65 as shown below. Room is provided on the PCB to accomodate any of those crystals.

```
ff65 b5                 .RTCXTAL    FCB $B5                 $85 => 4.194304 MHz $95 => 2.097152 MHz
*                                                           $A5 => 1.048576 MHz $B5 => 32.768 KHz
```

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

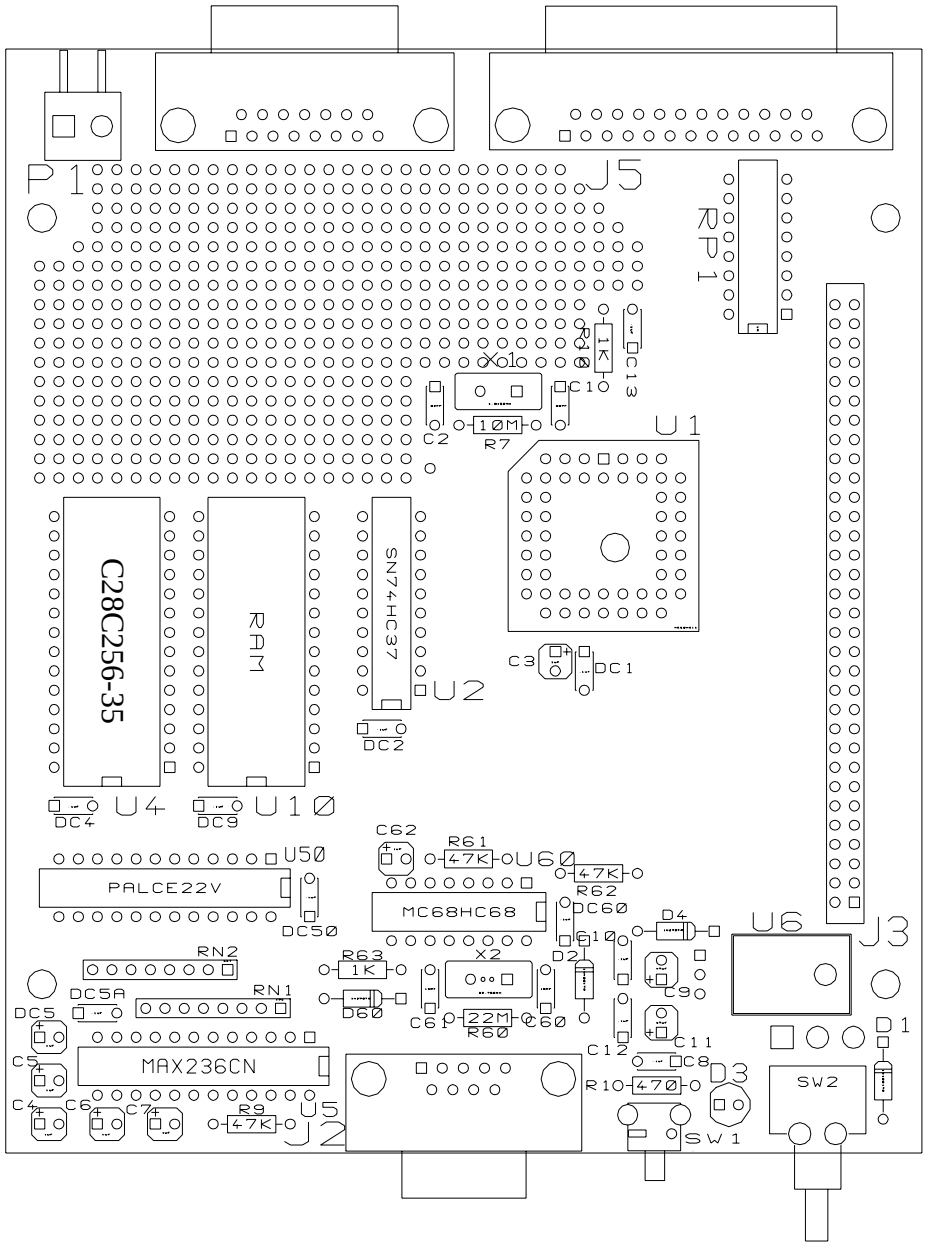
U

V

W

X

Y



RF-232

1404, rue Galt
Montréal Qc H4E 1H9
(514) 761-4201