

**M68HC11EVB2/AD1**

November 1991

**M68HC11EVB2**  
**EVALUATION BOARD**  
**USER'S MANUAL**

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

**Information contained in this document applies to  
REVision (A) M68HC11EVB2 Evaluation Boards**

IBM-PC is a registered trademark of International Business Machines Corporation.

Apple and MacTerminal are trademarks of Apple Computer, Inc.

Macintosh is a trademark licensed to Apple Computer, Inc.

Macintosh is a trademark of McIntosh Laboratory, Inc.

Red Ryder is a trademark of Freesoft Company

ProComm is a trademark of Datastorm Technologies, Inc.

The computer program stored in the Read Only Memory of the device contains material copyrighted by Motorola Inc., first published 1985, and may be used only under a license such as the License For Computer Programs (Article 14) contained in Motorola's Terms and Conditions of Sale, Rev. 1/79.

## ACKNOWLEDGEMENTS

The M68HC11EVB2 Evaluation Board project was organized at Georgia Tech by Dr. John Peatman and funded through the Motorola MCU Applications and University Support Program. Dr. Peatman organized a group of undergraduate and graduate students to design and build the EVB2 and logic analyzer. Upon acceptance, these devices were included in the Motorola MCU Support product line.

The success of the EVB2 project is due to the quality of the Georgia Tech students involved, Georgia Tech administrative support, and the Motorola University Support Program. The following contributors deserve specific credit:

Joe Bazzell	EVB2LA setup & return
Sean Doyle	EVB2LA self-test & drivers
Jan Glott	Software integration
Kennon Guglielmo	RUN command
Jeff Jenkins	Software integration
Clinton Knight	Real-time trace
Joe Loftus	Single-step trace
Steve Lowe	ONLY V specification
Larry Madar	EVB2LA data manipulation
John Peatman	Circuit design
Dan Willey	User program checksum

Each of the students involved in the development of this project, received a production set of M68HC11EVB2CPU and M68HC11EVB2LA boards in return for their development work.

## **PREFACE**

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

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

---

---

## CONTENTS

### CHAPTER 1 GENERAL INFORMATION

1.1	INTRODUCTION .....	1-1
1.2	FEATURES .....	1-1
1.2.1	EVB2CPU Features .....	1-1
1.2.2	EVB2LA Logic Analyzer Features.....	1-2
1.3	EVB2CPU DESCRIPTION.....	1-2
1.4	EVB2LA DESCRIPTION.....	1-3
1.5	SPECIFICATIONS.....	1-5
1.6	EQUIPMENT REQUIRED.....	1-7

### CHAPTER 2 HARDWARE PREPARATION AND INSTALLATION

2.1	INTRODUCTION .....	2-1
2.2	UNPACKING INSTRUCTIONS.....	2-1
2.3	EVB2CPU AND EVB2LA HARDWARE PREPARATION.....	2-1
2.3.1	EVB2CPU – EVB2LA Interconnection .....	2-3
2.3.2	EVB2CPU Jumper Headers.....	2-5
2.3.2.1	Jumper Header Types .....	2-5
2.3.2.2	Jumper Header Descriptions.....	2-7
2.3.3	Terminal – EVB2 Connection .....	2-11
2.3.4	Host Device – EVB2 Connection.....	2-15
2.3.5	Power Supply – EVB2 Connection .....	2-16
2.3.6	Target System – EVB2 Connection.....	2-17
2.4	EVB2 CHECKOUT PROCEDURE.....	2-19

## CHAPTER 3 FUNCTIONAL DESCRIPTION

3.1	INTRODUCTION .....	3-1
3.2	EVB2CPU HARDWARE DESCRIPTION.....	3-1
3.2.1	Microcontroller (MCU) .....	3-1
3.2.2	Port Replacement Unit (PRU) .....	3-2
3.2.3	EPROM Memory .....	3-3
3.2.4	RAM Memory.....	3-3
3.2.5	Clock Circuitry .....	3-3
3.2.6	Power Supply .....	3-4
3.2.7	Reset Circuitry .....	3-4
3.2.8	Decode Circuitry.....	3-4
3.2.9	External Communications .....	3-5
3.3	EVB2LA HARDWARE DESCRIPTION .....	3-5
3.3.1	EVB2LA RAM.....	3-6
3.3.2	EVB2LA Counters.....	3-6
3.3.3	Three-State Buffers.....	3-7
3.3.4	Trigger RAM .....	3-7
3.3.5	State Counter .....	3-8
3.3.6	Programmable Logic Array .....	3-8
3.3.7	Post-Trigger Counter .....	3-10

## CHAPTER 4 OPERATION

4.1	INTRODUCTION .....	4-1
4.2	OVERVIEW .....	4-1
4.3	EVB2 LIMITATIONS.....	4-1
4.4	EVB2 OPERATING PROCEDURE.....	4-3
4.4.1	User Program Evaluation Procedure.....	4-4
4.4.1.1	User Registers .....	4-6
4.4.1.2	Breakpoints .....	4-7
4.4.1.3	Single-Step Trace.....	4-7
4.4.1.4	Real-Time Trace .....	4-7
4.4.2	Downloading Procedure .....	4-8
4.4.2.1	Personal Computer – EVB2 Downloading.....	4-8
4.4.2.2	HOST – EVB2 Downloading .....	4-10
4.4.2.3	Downloading to EEPROM .....	4-11
4.4.3	User-Reset Procedure .....	4-11

---

---

## CHAPTER 4 OPERATION (continued)

4.4.4	CONFIG Register Programming Procedure.....	4-12
4.4.5	EEPROM Programming Procedure.....	4-13
4.4.6	ROM Debugging Procedure.....	4-13
4.5	EVB2LA OPERATING PROCEDURE.....	4-14
4.5.1	Definitions.....	4-14
4.5.2	Overview.....	4-15
4.5.3	Trace Set-Up.....	4-16
4.5.4	Trace Displays.....	4-17
4.5.5	Moving Through the Capture Buffer.....	4-18
4.5.6	Timing Captured Events.....	4-19
4.5.7	User Program Error Triggering.....	4-19
4.5.8	User Test Points.....	4-20
4.5.9	Logic Analyzer Debugging Examples.....	4-20
4.5.9.1	Trace After Reset.....	4-20
4.5.9.2	Trace About an Interrupt.....	4-21
4.5.9.3	Watch Interrupt Sequencing.....	4-23
4.5.9.4	Time Between Interrupts.....	4-24
4.5.9.5	Trace Writes to Variables.....	4-25
4.5.9.6	Trace Faults.....	4-25
4.5.10	Logic Analyzer Self-Test Error Messages.....	4-27
4.5.11	Internal Read Visibility.....	4-27
4.6	SOFTWARE PREPARATION.....	4-28
4.6.1	RAM/Control Register Mapping.....	4-28
4.6.2	Stack.....	4-28
4.6.3	Emulation ROM.....	4-28
4.6.4	Interrupt Vectors.....	4-29
4.6.5	User Program Preparation.....	4-29

## CHAPTER 5 MONITOR COMMANDS

5.1	INTRODUCTION.....	5-1
5.2	MONITOR COMMANDS.....	5-5
5.2.1	Assembler/Disassembler.....	5-6
5.2.2	Block Fill.....	5-9
5.2.3	Breakpoint Set.....	5-10
5.2.4	Bulk.....	5-12

---

---

## CHAPTER 5 MONITOR COMMANDS (continued)

5.2.5	Bulkall.....	5-13
5.2.6	Execute User Subroutine .....	5-14
5.2.7	Execute User Program .....	5-15
5.2.8	Primary Command Menu.....	5-16
5.2.9	Load S-Records into Memory.....	5-17
5.2.10	Memory Display .....	5-19
5.2.11	Memory Command Menu.....	5-20
5.2.12	Memory Modify.....	5-21
5.2.13	Move Memory .....	5-23
5.2.14	Remove Breakpoints.....	5-24
5.2.15	Offset Value.....	5-25
5.2.16	Continue Past Breakpoint .....	5-26
5.2.17	Register Display.....	5-28
5.2.18	MCU Hardware Reset.....	5-29
5.2.19	Register Modify .....	5-31
5.2.20	Execute User Program from Reset Vector.....	5-33
5.2.21	Single-Byte/Double-Byte Real-Time Trace.....	5-34
5.2.22	Stack Pointer/Program Counter Real-Time Trace.....	5-36
5.2.23	Set Host Port Baud Rate .....	5-37
5.2.24	Single-Step Trace.....	5-38
5.2.25	Execute External Command .....	5-40
5.2.26	Transparent Mode.....	5-41
5.2.27	Trace .....	5-42
5.2.28	User-Reset Preparation .....	5-44
5.2.29	Verify S-Records .....	5-46
5.2.30	Send Data to Another MCU .....	5-48
5.3	EVB2LA COMMANDS.....	5-50
5.3.1	Move Capture Buffer Trigger Offset.....	5-51
5.3.2	Set-Up Trace About Trigger.....	5-52
5.3.3	Set-up Trace After Trigger.....	5-54
5.3.4	Set-up Trace Before <n>th Execution of Trigger.....	5-56
5.3.5	Disassemble Instructions in Buffer.....	5-58
5.3.6	Fault .....	5-61
5.3.7	Find Data in Capture Buffer .....	5-62
5.3.8	Display Logic Analyzer Commands Help Menu.....	5-65
5.3.9	Set-up Trace of Only Trigger Addresses .....	5-66



---

---

## CHAPTER 5 MONITOR COMMANDS (continued)

5.3.10 Display Bus-Cycle Data from Buffer.....	5-69
5.3.11 Move the Trigger Offset in Buffer.....	5-71
5.3.12 Display Cycle Count Between Events in Buffer.....	5-72

## CHAPTER 6 SUPPORT INFORMATION

6.1 INTRODUCTION .....	6-1
6.2 I/O CONNECTOR SIGNAL DESCRIPTIONS.....	6-1
6.3 PARTS LISTS .....	6-6
6.4 SCHEMATIC DIAGRAMS.....	6-10

## APPENDIX A S-RECORD INFORMATION

A.1 INTRODUCTION .....	A-1
A.2 S-RECORD CONTENT .....	A-1
A.3 S-RECORD TYPES .....	A-2
A.4 S-RECORD CREATION .....	A-2
A.5 S-RECORD EXAMPLE.....	A-3

## APPENDIX B PROGRAMMABLE ARRAY LOGIC FLOW DIAGRAMS

B.1 TRACE ONLY INTERRUPT VECTORS.....	B-1
B.2 TRACE ONLY READS OF RAM OR REGISTERS.....	B-2
B.3 TRACE ONLY WRITES TO RAM OR REGISTERS.....	B-3
B.4 TRACE ONLY ACCESSES OF RAM OR REGISTERS .....	B-4
B.5 TRACE ONLY WRITES TO EEPROM.....	B-5
B.6 INITIALIZE LOGIC ANALYZER STATE.....	B-6
B.7 TRACE ABOUT EXECUTION OF SELECTED USER INSTRUCTION.....	B-7
B.8 TRACE BEFORE "INSTRUCTION FETCH OUT-OF-RANGE" FAULT OR "WRITE TO USER PROGRAM ADDRESS" FAULT.....	B-7
B.9 TRACE AFTER EXECUTION OF SELECTED USER INSTRUCTION.....	B-9
B.10 TRACE BEFORE <n>TH EXECUTION OF SELECTED USER INSTRUCTION....	B-10
B.11 DUMP CAPTURED DATA.....	B-11
B.12 TRACE BEFORE <n>TH READ OF SELECTED RAM OR REGISTER(S).....	B-12
B.13 TRACE BEFORE <n>TH WRITE OF SELECTED RAM OR REGISTER(S).....	B-13
B.14 TRACE BEFORE <n>TH ACCESS OF SELECTED RAM OR REGISTER(S).....	B-14

## APPENDIX C USING AN UNREGULATED POWER SUPPLY

C.1 INTRODUCTION .....	C-1
C.2 VOLTAGE REGULATOR INSTALLATION PROCEDURE.....	C-1

### FIGURES

1-1. EVB2 Memory Map.....	1-4
2-1. EVB2CPU Jumper Header, Switch, and Connector Layout.....	2-2
2-2. 50-Pin Ribbon Cable Interconnection .....	2-3
2-3. EVB2CPU/EVB2LA Factory Configuration .....	2-4
2-4. EVB2CPU Connector P6 .....	2-12
2-5. DB-25 to DB-25 Cable Assembly Diagram.....	2-12
2-6. DB-25 to DB-9 Cable Assembly Diagram for PC .....	2-13
2-7. DB-25 to DB-9 Cable Assembly Diagram for Macintosh.....	2-13
2-8. DB-25 to 8-pin DIN Cable Assembly Diagram .....	2-14
2-9. DB-9 to DB-25 Cable Assembly Diagram for Host Device.....	2-15
2-10. EVB2CPU Connector P1 .....	2-17
2-11. EVB2 Power-On Reset Screen.....	2-20
2-12. EVB2 Power-up Screen.....	2-20
2-13. EVB2LA Error Code Interpretation .....	2-21
4-1. EVB2 Power-On Reset Screen.....	4-3
4-2. EVB2 Command Example .....	4-4
4-3. User Program Downloading with KERMIT.....	4-9
4-4. HOST Device Downloading.....	4-10
4-5. RAW Command Display.....	4-17
4-6. DIS Command Display.....	4-18
4-7. Trace After Reset.....	4-21
4-8. Trace About an Interrupt .....	4-22
4-9. Watch Interrupt Sequencing.....	4-23
4-10. Time Between Interrupts .....	4-24
4-11. Trace Writes to Variables.....	4-25
4-12. Trace Faults .....	4-26
6-1. M68HC11EVB2CPU Schematic Diagrams (1 of 6).....	6-11
6-1. M68HC11EVB2CPU Schematic Diagrams (2 of 6).....	6-13
6-1. M68HC11EVB2CPU Schematic Diagrams (3 of 6).....	6-15
6-1. M68HC11EVB2CPU Schematic Diagrams (4 of 6).....	6-17

---

---

## FIGURES (continued)

6-1. M68HC11EVB2CPU Schematic Diagrams (5 of 6).....	6-19
6-1. M68HC11EVB2CPU Schematic Diagrams (6 of 6).....	6-21
6-2. M68HC11EVB2LA Schematic Diagrams (1 of 6).....	6-23
6-2. M68HC11EVB2LA Schematic Diagrams (2 of 6).....	6-25
6-2. M68HC11EVB2LA Schematic Diagrams (3 of 6).....	6-27
6-2. M68HC11EVB2LA Schematic Diagrams (4 of 6).....	6-29
6-2. M68HC11EVB2LA Schematic Diagrams (5 of 6).....	6-31
6-2. M68HC11EVB2LA Schematic Diagrams (6 of 6).....	6-33
C-1. Voltage Regulator and Phone Jack Installation.....	C-2
C-2. Heat Sink Installation .....	C-2

## TABLES

1-1. M68HC11EVB2CPU Specifications.....	1-5
1-2. M68HC11EVB2LA Specifications .....	1-6
1-3. External Equipment Requirements .....	1-7
2-1. Jumper Header Types .....	2-6
2-2. EVB2CPU Jumper Header Descriptions .....	2-7
2-3. Power Supply Requirements.....	2-16
2-4. Terminal Device Serial Communications Parameters .....	2-19
3-1. Logic Analyzer RAM Configuration.....	3-6
3-2. Three-State Buffer Configuration.....	3-7
3-3. FPLA Input .....	3-9
3-4. FPLA Output .....	3-9
4-1. User Register Definitions .....	4-6
4-2. Logic Analyzer Command Organization.....	4-15
4-3. Trace Types.....	4-16
5-1. Standard BUFFALO Command Set.....	5-2
5-2. M68HC11EVB2LA Logic Analyzer Command Set .....	5-4
5-3. Extended Disassembly Mnemonic Codes.....	5-59
6-1. MCU I/O Port Connector (EVB2CPU P1) Pin Assignments.....	6-1
6-2. EVB2CPU/EVB2LA Connector P2 Pin Assignments .....	6-3
6-3. RS-232 HOST Port EVB2CPU P5 Pin Assignments .....	6-4
6-4. RS-232 TERMINAL Port EVB2CPU P6 Pin Assignments.....	6-5

**TABLES (continued)**

6-5. EVB2LA Connector P1 Test Point Pin Assignments..... 6-5  
6-6. EVB2CPU Parts List ..... 6-6  
6-7. EVB2LA Parts List..... 6-9  
C-1. Unregulated Power Supply Specifications.....C-1

## **CHAPTER 1**

### **GENERAL INFORMATION**

#### **1.1 INTRODUCTION**

This manual contains user information on the M68HC11EVB2 Evaluation Board (EVB2). The EVB2 includes two boards and a debug monitor program:

- M68HC11EVB2CPU Central Processor Unit Board (EVB2CPU)
- M68HC11EVB2LA Logic Analyzer (EVB2LA).
- Bit User's Fast Friendly Aid to Logical Operation (BUFFALO) monitor program

This information has been organized into general information, hardware preparation and installation, functional description, operation, monitor commands, and support information. Several appendices are also included. This chapter lists the features and general description of the EVB2 and lists additional user-supplied hardware requirements.

#### **1.2 FEATURES**

The BUFFALO monitor program has been modified to include logic analyzer commands. This modified version of the BUFFALO monitor program is BUFFALO/GATECH but for brevity is called BUFFALO in the text references.

##### **1.2.1 EVB2CPU Features**

- Compatible with 52-pin PLCC package M68HC11 (A- and E-family members)
- Emulates M68HC11 single-chip operating mode
- Single-chip mode memory map emulation
- Download user code from a host computer
- Single-line assembler/disassembler resident in monitor program
- Includes monitor program single-step tracing and breakpoint debugging tools
- Master reset into monitor program, or user reset into user code
- Includes M68HC11EVB2LA Logic Analyzer connector
- Capable of programming CONFIG control register and internal EEPROM

- Mapping of MCU internal RAM and control registers to addresses \$00xx or \$10xx
- Includes a MC34064 under-voltage sensing circuit for the reset line
- Includes two options for connecting a power supply

### 1.2.2 EVB2LA Logic Analyzer Features

- Real time debugging with interrupts
- Captures address bus, data bus,  $\overline{R/W}$ ,  $\overline{LIR}$  and six user test points
- 8192 cycle capture buffer for bus cycle data
- Adds fourteen commands to the standard BUFFALO command set
- Trace instruction execution mode:
  - Permits triggering about, after, or before execution of an instruction
  - Displays captured data as disassembled instructions or in bus-cycle form
  - Calculates time between any two events captured
- Trace only mode:
  - Permits capturing of reads, writes, or accesses of any combination of RAM or control register addresses, writes to internal EEPROM, or fetches of interrupt vectors
  - Displays captured data in raw bus cycle form
- Capture buffer search command for finding the presence of any condition
- Uses M68HC11E-series *internal read visibility* to capture reads of internal resources
- Extensive power-on performance verification

## 1.3 EVB2CPU DESCRIPTION

The M68HC11EVB2CPU is an emulator/debugger for the A- and E-series of the M68HC11 Microcontroller (MCU) Family. The M68HC11 device includes a central processing unit, on-chip memory, and peripheral functions. Refer to the *MC68HC11 MCU Reference Manual*, M68HC11RM/AD and the specific technical summary or data book for additional device information. The EVB2CPU is a cost-effective board for demonstrating M68HC11 features and a powerful tool for emulating and debugging MCU-based target system software and hardware.

Although the on-board MC68HC11E1 has four modes of operation, only the single-chip mode of operation may be emulated with this product. Other more powerful emulators such as the Motorola M68HC11EVM Evaluation Module are available for more complete M68HC11 MCU emulation.

The EVB2CPU contains an M68HC11 MCU for executing the BUFFALO monitor program and user developed programs. You control EVB2CPU functionality, using the BUFFALO commands. BUFFALO is contained in an EPROM on the EVB2CPU board. This EPROM is mapped to the \$8000 – \$BFFF range of MCU addresses. The monitor program gains control of the EVB2CPU at power-up and reset by using the M68HC11 special test mode of operation. In special test mode, the MCU fetches its reset vector from \$BFFE – \$BFFF, which is within the EPROM. Immediately after reset, the monitor program switches from the special test mode to expanded multiplexed mode.

The expanded mode is used because its memory map is a superset of the single-chip mode memory map. The MC68HC24 Port Replacement Unit (PRU) is used on the EVB2CPU to rebuild the parallel I/O subsystem lost when in expanded mode. Use of the PRU makes single-chip mode transparent to a software programmer. All internal MCU features are addressable: the standard control registers, the 512 bytes of internal RAM, and the internal EEPROM. Additionally, 16K bytes of external static RAM are addressable which is mapped to the MCU \$C000 – \$FFFF address range. The upper 12K bytes of this range is the **user-map**. User programs and interrupt vectors are loaded into the **user-map** for emulation and debugging. The MCU internal ROM is disabled via the ROMON bit of the CONFIG register. Figure 1-1 shows the EVB2 internal memory map.

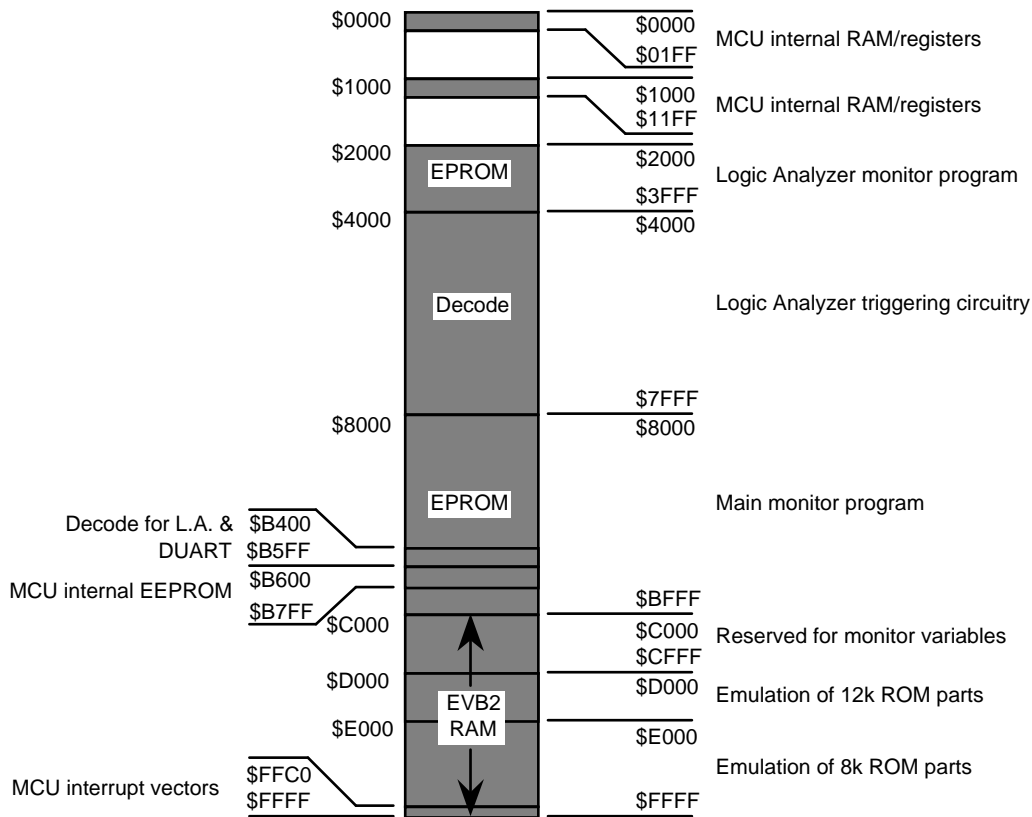
Communication with the monitor program is through an RS-232 compatible interface via a user-supplied terminal or personal computer (with terminal emulation program). The user issues commands and downloads user programs through this interface. Additionally, a second RS-232 compatible interface is present for communicating with a host computer which may be used to cross-assemble and download programs to the EVB2CPU.

## 1.4 EVB2LA DESCRIPTION

The EVB2LA is used with the EVB2CPU. The EVB2LA connects to the EVB2CPU through a 50-pin ribbon cable. The software required to run the logic analyzer is contained in an EPROM that is installed on the EVB2CPU. This EPROM adds commands to the BUFFALO command set when the EVB2LA is present. These commands:

- Set-up the EVB2LA to capture selected cycle-by-cycle data when a user program is executed
- Execute user code
- Display the data from the logic analyzer's 8192 cycle capture buffer

The EVB2LA capture buffer contains the MCU address bus, data bus,  $\overline{R/W}$  signal,  $\overline{LIR}$  signal, and six user test points for each cycle of captured data. This includes read cycles from internal resources such as the internal RAM which are usually not visible externally from the MCU.



**Figure 1-1. EVB2 Memory Map**

The EVB2LA has two data capture modes: trace instruction execution mode and trace only mode. Four functions comprise trace instruction execution mode: trace about, trace after, trace before, and trace fault. The set-up commands let you specify the trace function and trigger instruction addresses. The trace only mode lets you capture either reads from or writes to specific internal RAM or control register addresses, both of the above, writes to internal EEPROM at \$B600 – \$B7FF, or fetches of interrupt vectors.

When data capture is complete, a non-maskable interrupt is generated which returns control to the monitor program. The data capture terminates when the capture buffer fills with data or you press a key on the display terminal. The data in the capture buffer is then available for display. Data captured from trace instruction execution mode may be displayed as disassembled instructions or in raw bus-cycle form. Additionally, the time in bus-cycles may be calculated between the occurrence of any two events captured in the buffer. Data captured in trace only mode displays the individual bus-cycles captured.



## 1.5 SPECIFICATIONS

Tables 1-1 and 1-2 list the EVB2CPU and EVB2LA Specifications.

**Table 1-1. M68HC11EVB2CPU Specifications**

<b>Characteristic</b>	<b>Specification</b>
MCU	MC68HC11E1FN
PRU	MC68HC24FN
DUART	MC2681 or SCC2692AC1N40
I/O ports: Terminal (P6) Host computer (P5) Target System (P1) Logic Analyzer (P2)	RS-232 compatible DB-25 (female) RS-232 compatible DB-9 (female) HCMOS-TTL compatible, 60-pin header 50-pin header (logic analyzer only)
Temperature: Operating Storage	0 to +50° C -40 to +85° C
Relative Humidity	0 to 90% (non-condensing)
Power Requirements	+5 Vdc @ 500 mA (max)
Dimensions	7.00 x 5.75 in. (17.78 x 14.61 cm)

**Table 1-2. M68HC11EVB2LA Specifications**

Characteristic	Specification
Triggering Capability: Opcode fetch from any address Read cycle from any address Write cycle to any address Write cycle to EEPROM address Interrupt vector fetch from address	\$D000 – \$FFFF \$0000 – \$1FFF \$0000 – \$1FFF \$B600 – \$B7FF \$FFC0 – \$FFFE (even addresses)
Capture Buffer Size	8192 cycles x 32 bits
I/O ports: Test Point connector P1 EVB2 connector P2	TTL compatible, 7-pin header with 10k½ pull-up resistors to +5V. Dedicated to EVB2, 50-pin header
Temperature: Operating Storage	0 to +50° C -40 to +85° C
Relative Humidity	0 to 90% (non-condensing)
Power Requirements	+5 Vdc at 500 mA (max), supplied by EVB2
Dimensions	7.00 x 5.75 in. (17.78 x 14.61 cm)

## 1.6 EQUIPMENT REQUIRED

Table 1-3 lists the external equipment requirements for EVB2LA operation.

**Table 1-3. External Equipment Requirements**

An RS-232 compatible terminal or host computer (with a terminal emulation package—PCKERMIT, PROCOMM, MacTerminal, Red Ryder, etc.) <sup>(1)</sup>
Host computer (RS-232 compatible)
Serial communication cable for the terminal or host computer <sup>(2)</sup> .
+5 Vdc @ 500 mA power supply <sup>(2)</sup>  Additionally, an unregulated +9 Vdc power supply may be used with the EVB2. To use an unregulated power supply, a MC7805CT (rated 5V @ 1A) voltage regulator must be installed at location VR1. A heat sink must be used with the voltage regulator. A user-supplied and -installed 3.5 mm phone jack at P3 is also required when using an unregulated power supply <sup>(3)</sup> .
1. Refer to Chapter 4 for an example of downloading with PCKERMIT. 2. Refer to Chapter 2. 3. Refer to Appendix C.



## **CHAPTER 2**

### **HARDWARE PREPARATION AND INSTALLATION**

#### **2.1 INTRODUCTION**

This chapter provides unpacking instructions, hardware preparation, and installation instructions for the EVB2CPU and EVB2LA.

#### **2.2 UNPACKING INSTRUCTIONS**

Unpack the EVB2 from the shipping carton. Refer to the packing list and verify that all items are present. Save the packing material for storing or shipping the equipment.

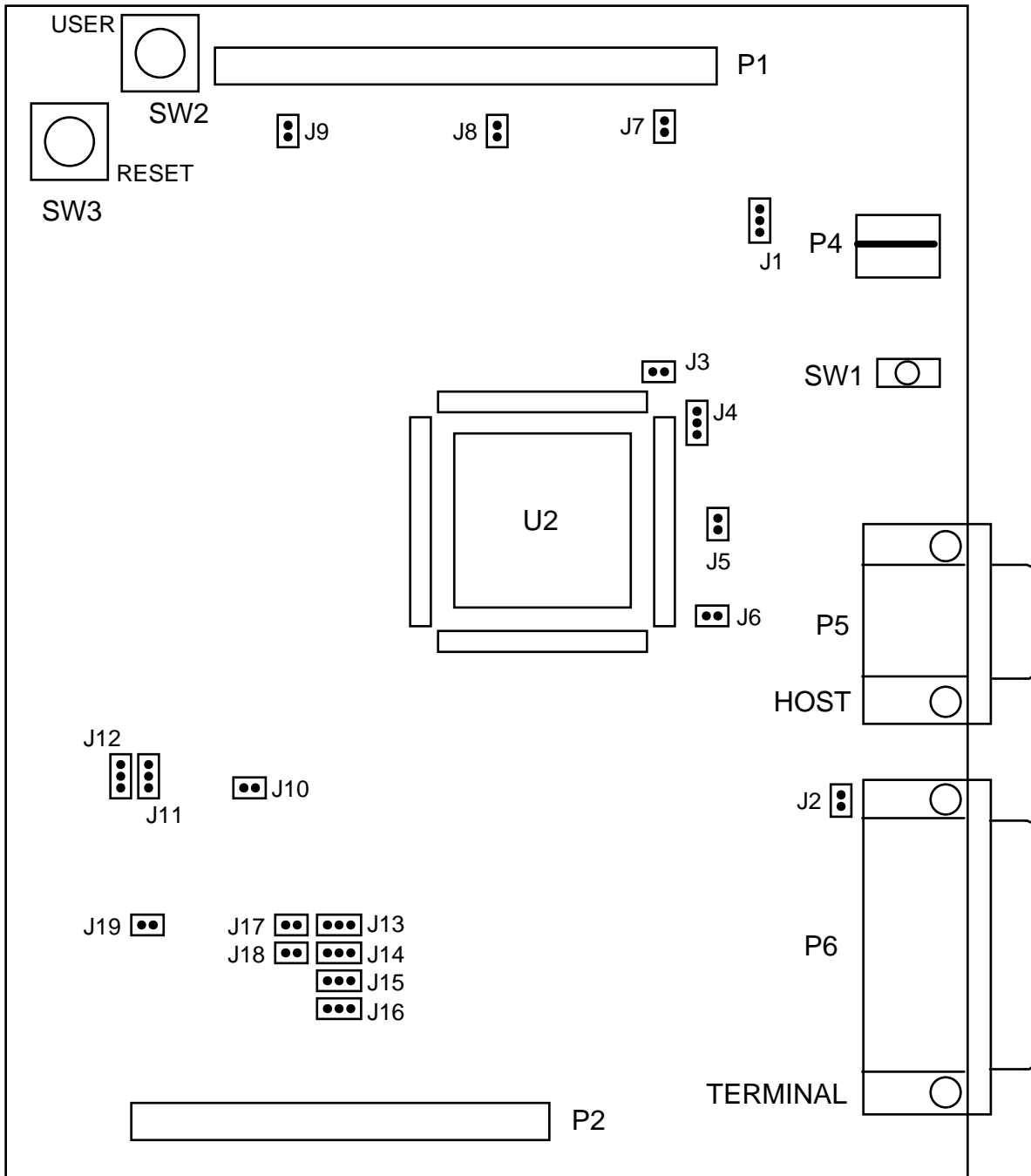
#### **NOTE**

Should the product arrive damaged, save all packing material and contact the carrier's agent.

#### **2.3 EVB2CPU AND EVB2LA HARDWARE PREPARATION**

The EVB2CPU is shipped from the factory with the EVB2LA connected and jumpers installed on the jumper headers. You must connect a user-supplied: power supply, communications cables, and target system cable to the EVB2CPU. Figure 2-1 shows the location of the EVB2CPU jumper headers, I/O connectors, reset switches, and power connector.

The EVB2LA contains no hardware options and, if connected to the EVB2CPU, requires no hardware preparation prior to use.



**Figure 2-1. EVB2CPU Jumper Header, Switch, and Connector Layout**

### 2.3.1 EVB2CPU – EVB2LA Interconnection

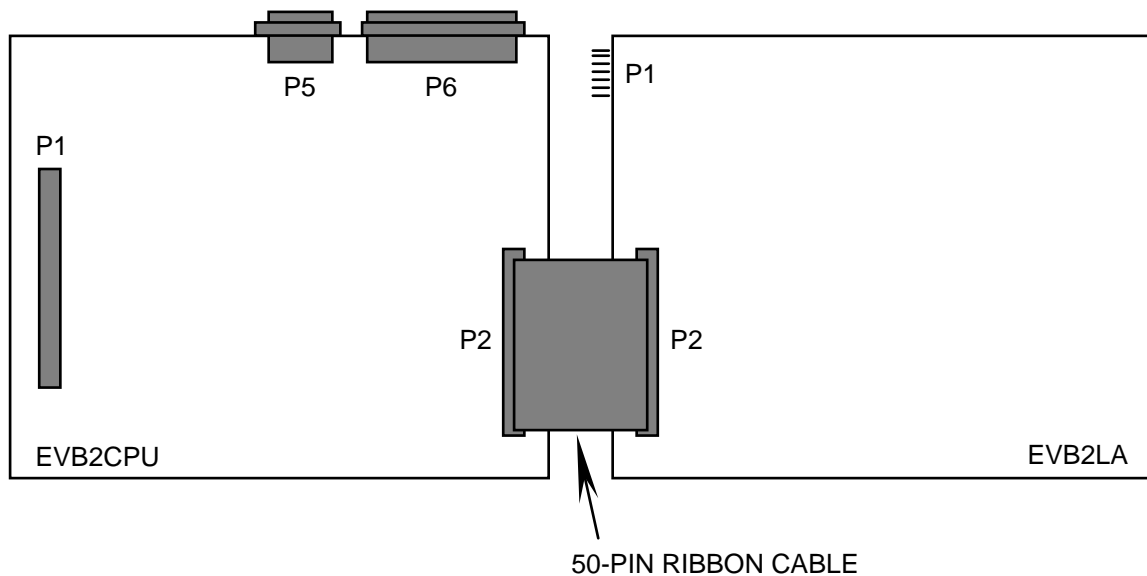
To use the EVB2LA requires connection to the EVB2CPU via a 50-pin ribbon cable (see the factory configuration in Figures 2-2 and 2-3). The EVB2CPU and EVB2LA may be mounted back-to-back or laid flat on a table top, side-by-side. The 50-pin ribbon cable is shipped with the EVB2CPU and EVB2LA.

Follow these steps to connect the EVB2LA to the EVB2CPU:

#### CAUTION

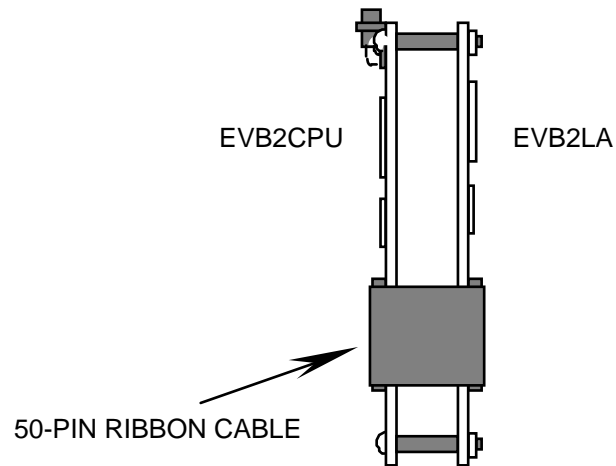
Sudden power surges could damage EVB2CPU, EVB2LA, and target system integrated circuits. Turn off EVB2 and target system power when installing or removing the EVB2LA or the target system from the EVB2CPU.

1. Mate the 50-pin ribbon cable to connector P2 on the EVB2CPU (Figure 2-2). Press the cable connector down firmly while supporting the EVB2CPU from beneath.
2. Mate the 50-pin ribbon cable to connector P2 on the EVB2LA. Again press the cable connector down firmly while supporting the board from beneath.



**Figure 2-2. 50-Pin Ribbon Cable Interconnection**

3. Optionally, the EVB2CPU and EVB2LA may be stacked back-to-back in a sandwich arrangement using four threaded spacers between the boards (Figure 2-3). The EVB2LA faces downward using the original EVB2CPU feet to protect the components.



**Figure 2-3. EVB2CPU/EVB2LA Factory Configuration**



## 2.3.2 EVB2CPU Jumper Headers

The EVB2CPU has been factory tested and is shipped with installed jumpers. These jumper may be moved to customize the EVB2CPU functionality. These paragraphs are a detailed description of each jumper header function. Refer to the schematic diagrams in Chapter 6 for additional details concerning hardware jumper headers.

### 2.3.2.1 Jumper Header Types

A jumper installed on a jumper header provides a connection between two points in the EVB2CPU circuit. There are four types of jumper headers on the EVB2CPU: two-post, and two-post with a cut-trace short, three-post, and three-post with a cut-trace short. Each jumper header type consists of feed-thru holes on the EVB2CPU printed circuit board (PCB). The jumper header types that are cut-trace shorts have a copper trace between the feed-thru holes (bottom or solder side of the EVB2CPU). Table 2-1 describes each type of jumper header.





#### CAUTION

Depending on the application, it may be necessary to cut wiring trace shorts (cut-trace shorts) on the PCB. Be careful not to cut adjacent PCB wiring traces.

#### NOTE

If the cut-trace short on a jumper header is cut, a user-supplied fabricated jumper must be installed on the jumper header to return the EVB2CPU to its default setting.






**Table 2-1. Jumper Header Types**

<b>Jumper Header Type</b>	<b>Symbol</b>	<b>Description</b>
two-post		Two-post jumper header and designated as JX (X = the jumper header number). Use a fabricated jumper to create a short between the two posts of the jumper header.
two-post with cut-trace short		Two-post jumper header with cut-trace short, designated as JX (X = the jumper header number). After removing the cut-trace short, use a fabricated jumper to return the jumper header to its factory default state.
three-post		Three-post jumper header, designated as JX (X = the jumper header number). Use a fabricated jumper to create a short between two of the three posts of the jumper header.
three-post with cut-trace short		Three-post jumper header with cut-trace short and designated as JX (X = the jumper header number). To change the factory jumper header configuration, cut the trace on the bottom of the board and install a jumper on the two desired posts.







### 2.3.2.2 Jumper Header Descriptions

Table 2-2 lists the jumper headers on the EVB2CPU and a description of its function in each position.







**Table 2-2. EVB2CPU Jumper Header Descriptions**

Jumper Header	Type	Description
J1		<p>Jumper between pins 1 and 2 (factory default); a regulated power supply supplies +5 Vdc to the EVB2CPU via P4.</p> <p>Jumper between pins 2 and 3; an unregulated power supply supplies +5 Vdc to the EVB2CPU via P3. A voltage regulator is required at location VR1. VR1 and P3 are user supplied and installed.</p>
J2		<p>Jumper installed or cut-trace short intact (factory default); TERMINAL port (P6) pin 1 connected to EVB2CPU ground (GND).</p> <p>No jumper or cut-trace short; P6 pin 1 disconnected from EVB2CPU GND.</p>
J3		<p>Jumper installed or cut-trace short intact (factory default); On-board oscillator connected to MCU XTAL line.</p> <p>No jumper or cut-trace short; Disconnects the MCU XTAL line from the on-board 8 MHz oscillator clock. J3 must be open if the target system supplies the clock signal (refer to J4).</p>
J4		<p>Jumper or cut-trace short between pins 1 and 2 (factory default); MCU EXTAL line connected to on-board oscillator.</p> <p>Cut-trace short removed and a jumper between pins 2 and 3; MCU EXTAL disconnected from the on-board 8 MHz oscillator clock and connected to target system connector P1 pin 7. The cut-trace short must be removed and a jumper installed on pins 2 and 3 if the target system supplies the clock signal (refer to J3).</p> <p style="text-align: center;"><b>NOTE</b></p> <p style="text-align: center;">If the trace is cut and no jumper is installed between J4 pins 2 and 3, the MCU will not operate.</p>
J5		<p>Jumper installed or cut-trace short intact (factory default); MCU V<sub>RL</sub> line connected to ground.</p> <p>No jumper or cut-trace short; Disconnects the MCU line V<sub>RL</sub> from the EVB2CPU ground bus. Useful when an external A-D converter low reference voltage is used.</p>



**Table 2-2. EVB2CPU Jumper Header Descriptions (continued)**

Jumper Header	Type	Description
J6		<p>Jumper installed or cut-trace short intact (factory default); MCU <math>V_{RH}</math> line connected to +5 Vdc bus through <math>1K\frac{1}{2}</math> R6.</p> <p>No jumper or cut-trace short; Disconnects the MCU line <math>V_{RH}</math> from the EVB2CPU +5 Vdc bus. Useful when an external A-D converter high reference voltage is used.</p>
J7		<p>No jumper installed (factory default); Isolates the target system XIRQ (connector P1 pin 18) from the on-board MCU XIRQ line.</p> <p>Jumper installed; Connects the target system <math>\overline{XIRQ}</math> signal to MCU XIRQ (U2 pin 18) .</p>
J8		<p>No jumper installed (factory default); Target system <math>V_{CC}</math> line isolated from EVB2CPU +5 Vdc bus.</p> <p>Jumper installed; EVB2CPU +5 Vdc connected to target system via P1 pin 26. Use jumper header J8 when the target system and EVB2CPU are powered by a common power supply.</p>
J9		<p>No jumper installed (factory default); Normal MCU operation.</p> <p>Jumper installed; MODA signal connected to ground. Reset or power-up causes MCU to enter special bootstrap mode as long as the USER RESET switch (SW2) is not pressed. May be used to download a program to the MCU RAM through the MCU SCI (refer to J11 and J12).</p>
J10		<p>Jumper installed (factory default); EVB2LA generated <math>\overline{XIRQ}</math> signal connected to MCU.</p> <p>No jumper installed; Isolate the P2 EVB2LA <math>\overline{XIRQ}</math> signal from the MCU <math>\overline{XIRQ}</math> pin. Useful when the EVB2LA generated <math>\overline{XIRQ}</math> signal interferes with target system operation.</p> <p style="text-align: center;"><b>NOTE</b></p> <p style="text-align: center;">Removing the factory installed jumper disables EVB2LA capture functions.</p>
J11		<p>Jumper or cut-trace short between pins 1 and 2 (factory default); DUART RXDB input connected to P5 pin 2.</p> <p>Cut-trace short removed and a jumper between pins 2 and 3; MCU RXD input connected to P5 pin 2. Isolate the P5 HOST port DB-9 pin 2 from the DUART RXDB input pin 10 and connect the MCU RXD input pin 20 to P5 pin 2. May be used to connect the MCU SCI to the P5 connector (refer to J12).</p> <p>Cut-trace removed and no jumper; P5 pin 2 not connected.</p>

**Table 2-2. EVB2CPU Jumper Header Descriptions (continued)**

Jumper Header	Type	Description
J12		<p>Jumper or cut-trace short between pins 1 and 2 (factory default); DUART TXDB output connected to P5 pin 3.</p> <p>Cut-trace short removed and a jumper between pins 2 and 3; MCU TXD output connected to P5 pin 3. Isolate the P5 HOST port DB-9 pin 3 from the DUART TXDB input pin 11 and connect the MCU TXD output pin 21 to P5 pin 3. May be used to connect the MCU SCI to the P5 connector (refer to J11).</p> <p>Trace cut and no jumper: P5 pin 3 not connected.</p>
J13		<p>J13 provides input to the DUART (U12 pin 39).</p> <p>No jumper installed (factory default); ties IP1 to VCC.</p> <p>Jumper installed; ties IP1 to GND.</p>
J14		<p>J14 provides input to the DUART (U12 pin 38).</p> <p>No jumper installed (factory default); ties IP1 to VCC.</p> <p>Jumper installed; ties IP1 to GND.</p>
J15		<p>J15 tells the monitor program to locate the MCU control registers at \$0000 or \$1000 after reset.</p> <p>Jumper installed on pins 1 and 2 (factory default); MCU control registers mapped to \$1000.</p> <p>Jumper installed on pins 2 and 3 or no jumper installed; MCU control registers mapped to \$0000.</p>
J16		<p>J16 tells the monitor program to locate the MCU internal RAM at \$0000 or \$1000 after reset.</p> <p>Jumper installed on pins 2 and 3 (factory default); MCU internal RAM mapped to \$0000.</p> <p>Jumper installed on pins 1 and 2; MCU internal RAM mapped to \$1000.</p>
J17		<p>No jumper installed (factory default); EVB2CPU operates normally.</p> <p>Jumper installed; special test mode entered after reset. Provides input to the monitor program to keep the MCU in special test mode after reset. May be used to access MCU special test mode features through the monitor.</p>

**Table 2-2. EVB2CPU Jumper Header Descriptions (continued)**

Jumper Header	Type	Description
J18		<p>J18 provides input to the DUART (U12 pin 4).</p> <p>No jumper installed (factory default); ties IP1 to V<sub>CC</sub>.</p> <p>Jumper installed; ties IP1 to GND.</p>
J19		<p>Jumper installed (factory default); DUART generated <math>\overline{XIRQ}</math> signal connected to MCU.</p> <p>No jumper installed; <u>DUART</u> generated <math>\overline{XIRQ}</math> signal disconnected from MCU. Isolates the <u>DUART</u> INTR signal from the MCU <math>\overline{XIRQ}</math> pin. Use when the DUART generated <math>\overline{XIRQ}</math> signal interferes with target system operation.</p> <p style="text-align: center;"><b>NOTE</b></p> <p style="text-align: center;">Removing the factory installed jumper on this jumper header disables the TRACE, PROCEED, and STOPAT commands, as well as breakpoint operation.</p>

### 2.3.3 Terminal – EVB2 Connection

There are two ways to communication with the EVB2:

1. An RS-232 compatible personal computer (PC) with a terminal emulation package may be connected to EVB2CPU TERMINAL port (P6). In this configuration, the PC can cross-assemble M68HC11 programs and download S-Record files to the EVB2. The TERMINAL port baud rate is 9600 baud.
2. An RS-232 compatible personal computer (PC) with a terminal emulation package or a terminal connected to EVB2CPU TERMINAL port (P6) and a host computer connected to EVB2CPU HOST port (P5). Refer to paragraph 2.3.4 for a description of this configuration.

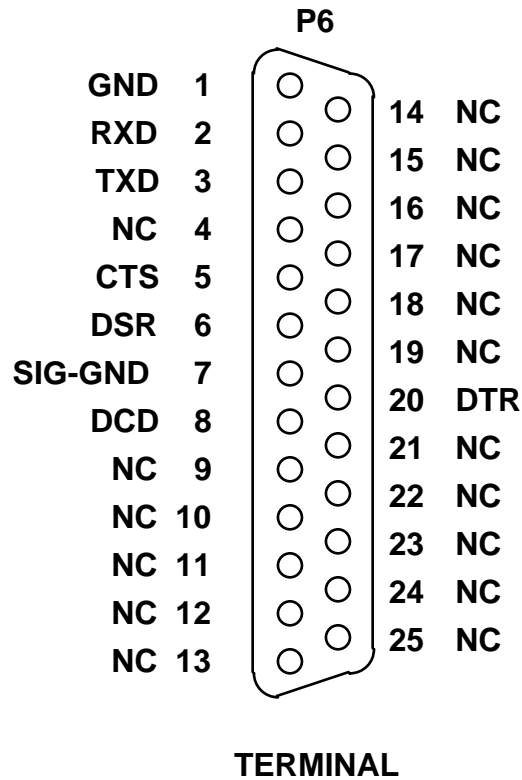
The RS-232 cable, as a minimum, must contain three wires: the *signal-ground* wire (GND), the *transmit-data* wire (TXD), and the *receive-data* wire (RXD). The connector (P6) on the EVB2CPU is a DB-25 female (Figure 2-4).

For hardware handshaking two additional wires are required: the request-to-send wire (RTS) and clear-to-send wire (CTS). The RTS signal lets you suspend EVB2 data transmissions using the display terminal device while internal processing is taking place. The CTS line lets the EVB2 interrupt terminal data transmissions. Basic EVB2 operation does not require the RTS or CTS signals, but they may be useful for downloading S-record files directly into internal EEPROM.

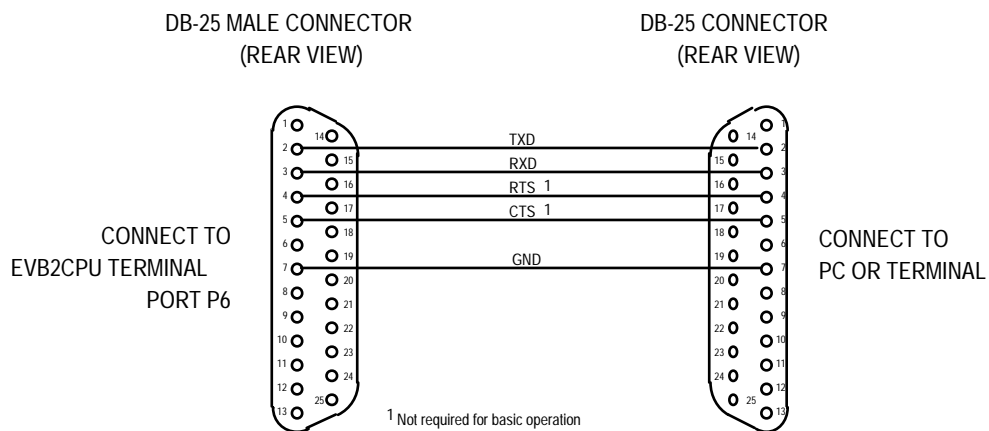
Figures 2-5 through 2-8 show several variations of the display terminal cable for different devices. The RS-232 connector P6 (*TERMINAL* port) is a data carrier equipment (DCE) port.

#### NOTE

The cable required to connect your personal computer to the EVB2 is a standard Hayes modem cable. This cable is available in most computer stores.

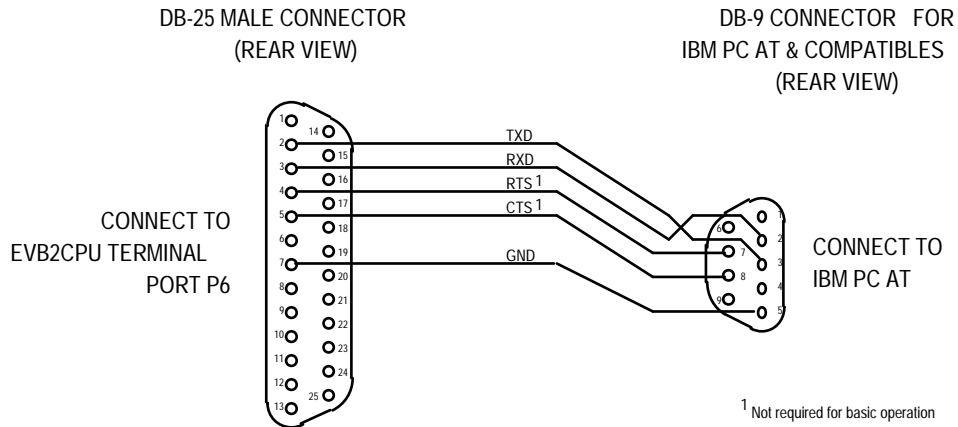


**Figure 2-4. EVB2CPU Connector P6**

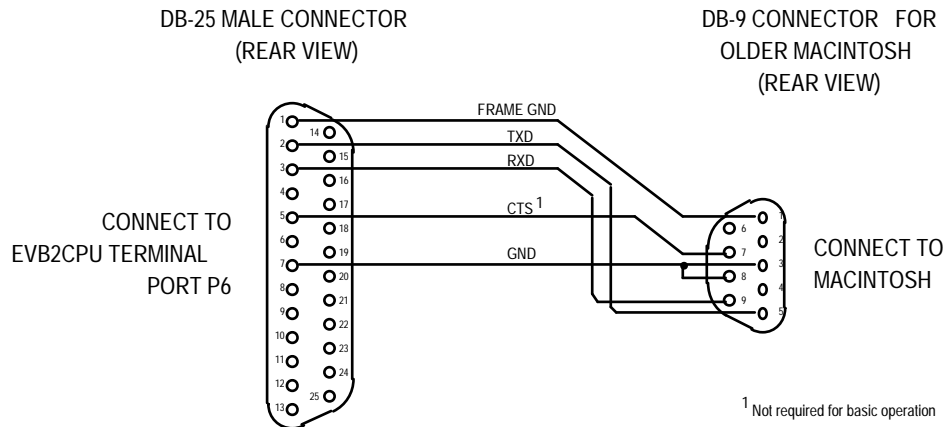


**Figure 2-5. DB-25 to DB-25 Cable Assembly Diagram**

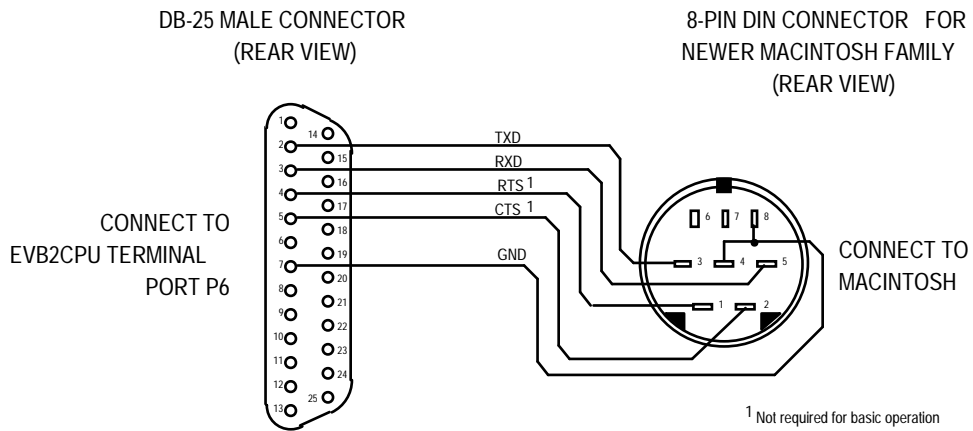




**Figure 2-6. DB-25 to DB-9 Cable Assembly Diagram for PC**



**Figure 2-7. DB-25 to DB-9 Cable Assembly Diagram for Macintosh**



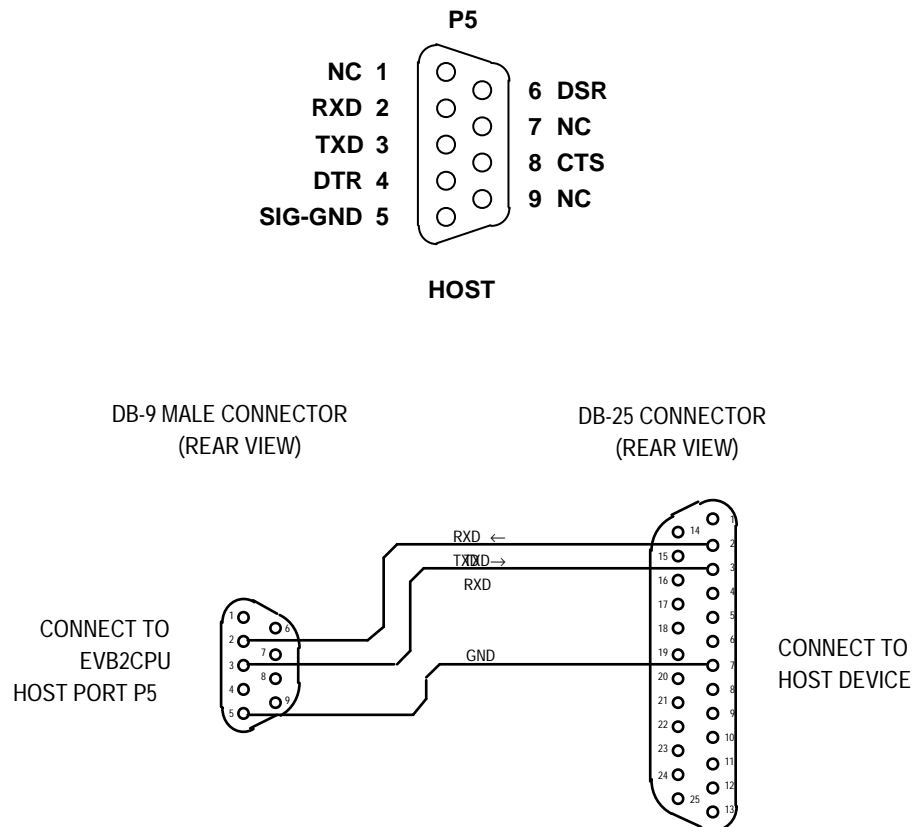
**Figure 2-8. DB-25 to 8-pin DIN Cable Assembly Diagram**

### 2.3.4 Host Device – EVB2 Connection

A host computer connected to EVB2CPU P5 (HOST port) may be used to download S-record files to the EVB2. The host device must be a computer with an RS-232 compatible serial connection. When a host device is connected to EVB2CPU P5 (*HOST* port) and a terminal is connected to EVB2CPU P6 (*TERMINAL* port), the transparent mode (**TM**) command may be used. In the transparent mode all data sent from either the *TERMINAL* port or *HOST* port is passed through the EVB2 to the other device.

The *HOST* port connector (EVB2CPU P6) is a DB-9 female (shown below). No hardware handshaking capabilities are provided through this connector. Figure 2-9 shows one typical cable for connecting the EVB2 to a host device with a DB-25 style connector. The RS-232 connector P5 (*HOST* port) RS-232 is a Data Terminal Equipment (DTE) port.

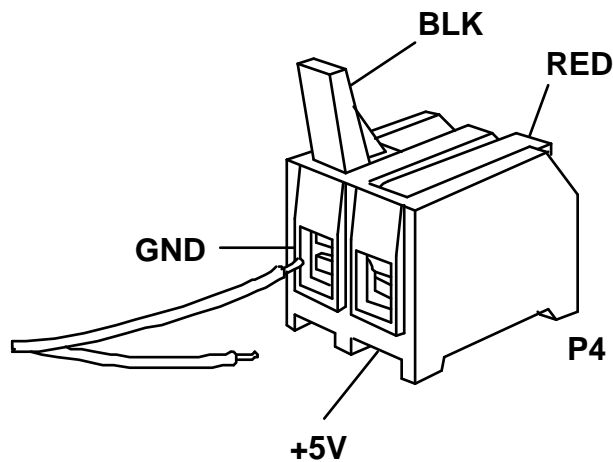
In this configuration, the host computer would be used to cross-assemble M68HC11 programs and download S-Record files to the EVB2 through the *HOST* port. The *TERMINAL* port baud rate is fixed at 9600 baud, while the *HOST* port may be set for 300, 1200, 2400, 4800, 9600, or 38400 baud.



**Figure 2-9. DB-9 to DB-25 Cable Assembly Diagram for Host Device**

### 2.3.5 Power Supply – EVB2 Connection

The EVB2 requires a +5 Vdc @ 500 mA power supply for operation (refer to Table 2-3). Use EVB2CPU connector P4 (shown below) to connect power to the EVB2CPU. The black lever is ground (GND). The red lever is +5 Vdc (V<sub>CC</sub>). Use 20 or 22 AWG wire for power connections. For each wire, trim back the insulation 1/4 in. (.635 cm), lift the appropriate lever of EVB2CPU P4 to release tension on the contacts, then insert the bare wire into EVB2CPU P4 and close the lever. Refer to Appendix C for specifications for using an unregulated power supply.



#### CAUTION

Do not use wire larger than 20 AWG in connector EVB2CPU P4. Such wire could damage the connector.

**Table 2-3. Power Supply Requirements**

Equipment	Supplier	Connector	Voltage	Current
Regulated supply	Lambda	P4	+5 Vdc	500 mA

### 2.3.6 Target System – EVB2 Connection

The target system is user-developed M68HC11 MCU-based hardware. The EVB2 is capable of emulating the MCU operating in single-chip mode via a 60-pin ribbon cable connected between the EVB2CPU P1 and the target system. The 60-pin ribbon cable is user-supplied. Use a standard 60-pin ribbon cable with one-to-one connectors at both ends. Construct the target system with a 60-pin connector that matches the EVB2CPU pinouts. Figure 2-10 shows the EVB2CPU P1 60-pin connector pinouts.

		P1			
GND	1	2	NC		
NC	3	4	STRA		
E	5	6	STRB		
NC or EXTAL	7	8	NC		
PC0	9	10	PC1		
PC2	11	12	PC3		
PC4	13	14	PC5		
PC6	15	16	PC7		
RESET*	17	18	XIRQ*		
IRQ*	19	20	PD0		
PD1	21	22	PD2		
PD3	23	24	PD4		
PD5	25	26	NC or VDD		
PA7	27	28	PA6		
PA5	29	30	PA4		
PA3	31	32	PA2		
PA1	33	34	PA0		
PB7	35	36	PB6		
PB5	37	38	PB4		
PB3	39	40	PB2		
PB1	41	42	PB0		
PE0	43	44	PE4		
PE1	45	46	PE5		
PE2	47	48	PE6		
PE3	49	50	PE5		
VRL	51	52	PE6		
NC	53	54	NC		
NC	55	56	NC		
NC	57	58	NC		
NC	59	60	MODB		

**Figure 2-10. EVB2CPU Connector P1**

---

Generally the EVB2CPU P1 connector is a one-for-one pinout of the on-board MCU. In the factory configuration five signals are not available on P1: XTAL, EXTAL, V<sub>CC</sub>, MODA, and MODB. Three of these signals (XTAL, EXTAL, and V<sub>CC</sub>) can be connected to P1 if you use the jumper header options (refer to paragraph 2.3.2). Two signals are not wired to the EVB2CPU P1 connector: MODA and MODB.

If the target system clock is a crystal it is not possible to drive the EVB2 clock signals XTAL and EXTAL. This is due to the high capacitance introduced by the cable. If the target system clock is a buffered oscillator you can overcome this restriction. When using a buffered oscillator to drive the EVB2 MCU it is necessary to modify the default jumper header settings on J3 and J4.

V<sub>CC</sub> (+5 Vdc) in the factory default is not connected to EVB2CPU P1 because there's a likelihood of overloading the EVB2 power supply. The target system may be powered by the EVB2 power supply, but the power supply connected to EVB2CPU P4 must have the available current to support the target system. When using the EVB2 power supply a jumper must be installed at J8 to connect the V<sub>CC</sub> to the target system via P1 pin 26.

MODA and MODB signals define the MCU operating mode at power-up or RESET. These signals are wired so the EVB2 powers up in single-chip mode and may not be manipulated.

These steps outline the target system installation procedure:

#### CAUTION

Failure to disconnect all cabling between the EVB2 or target system and other energized equipment during installation may damage the EVB2 or target system or both.

1. Switch the EVB2CPU power switch SW1 OFF.
2. Disconnect any energized circuits from the target system.
3. Mate the 60-pin ribbon cable from the target system to EVB2CPU connector P1. Press the cable connector down firmly while supporting the EVB2CPU from beneath.

## 2.4 EVB2 CHECKOUT PROCEDURE

With the power supply and terminal device installed, follow these steps to verify that the EVB2 is functioning.

1. Prepare the display terminal device (PC communications program or terminal) for the serial communication parameters shown in Table 2-4.
2. With the EVB2CPU power switch SW1 **OFF**, turn-on or plug-in the external power supply.
3. Switch EVB2CPU SW1 **ON**. Verify that EVB2CPU LED L1 is ON.
4. Watch for the power-up reset screen shown in Figure 2-11. If this screen does not appear or part of the data is garbled or missing, check that the serial communications parameters are properly set as shown in Table 2-4.
5. Press the terminal's carriage return to display the EVB2 primary help screen.

The preceding steps are sufficient to verify that the EVB2 is configured properly and communicating with the terminal device. The monitor program contains a power-up RAM test that performs a non-destructive checkerboard test of the external static RAM on the EVB2CPU. The results of this test (pass or fail) are displayed at power-up only and indicate the operability of the MCM60256 static RAM at U11.

**Table 2-4. Terminal Device Serial Communications Parameters**

<b>Parameter</b>	<b>Set Value</b>
data rate	9600 baud
data word length	8 data bits
stop bits	1 stop bit
parity	none
duplex (local echo)	full duplex (no local echo)
flow control	hardware (CTS/RTS)

```
BUFFALO/GATECH 1.0 - Bit User's Fast Friendly Aid to Logical Operation

Register Start (J15): 1000
MCU RAM Range (J16): 0000 01FF
EEPROM Range:      B600 B7FF
User Program Range: D000 FFFF
Internal ROM Disabled
COP System Disabled
EVB2 RAM test passed
EVB2LA Logic Analyzer Functioning           Says "Not Present" if there is no EVB2LA.
>
```

**Figure 2-11. EVB2 Power-On Reset Screen**

At power-up the BUFFALO monitor program identifies the existence of the EVB2LA and executes the self-test program. Test results are displayed on the screen which appears after power-up occurs.

With the EVB2LA, power supply, and data terminal device connected to the EVB2, follow this step-by-step checkout procedure:

1. Turn the EVB2CPU SW1 switch on.
2. Watch for the power-up reset screen shown in Figure 2-12. The following paragraphs explain the required action if this screen is not received correctly.

```
BUFFALO/GATECH 1.0 - Bit User's Fast Friendly Aid to Logical Operation

Register Start (J15): 1000
MCU RAM Range (J16): 0000 01FF
EEPROM Range:      B600 B7FF
User Program Range: D000 FFFF
Internal ROM Disabled
COP System Disabled
EVB2 RAM test passed
EVB2LA Logic Analyzer Functioning
>
```

**Figure 2-12. EVB2 Power-up Screen**



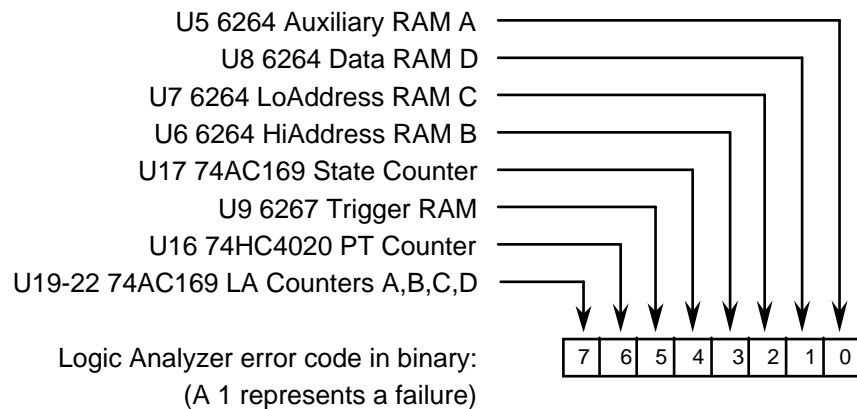
The following messages may be generated by the EVB2LA self-test program:

- EVB2LA Logic Analyzer Functioning
- EVB2LA Logic Analyzer Not Functioning XXXXXXXX
- EVB2LA Logic Analyzer Not Present

The error message *EVB2LA Logic Analyzer Not Functioning XXXXXXXX* means that the EVB2LA did not pass at least one of its power-up self-test functions. All EVB2LA commands will be disallowed by the monitor program until the EVB2LA successfully passes the self-test. The 8-bit binary value at the end of the error message is an error code giving helpful information about the potentially failing hardware. To interpret this code, consult Figure 2-13.

The error message *EVB2LA Logic Analyzer Not Present* means that the EVB2LA failed all power-up self-test functions. Check that the EVB2LA is properly connected to the EVB2CPU.

Note that the EVB2 only runs the EVB2LA self-test after a power-up **reset** (POR) or after a **RESET T** command. The EVB2LA status line does not appear when a RESET occurs. In some cases, switching the power switch SW1 off in itself will not produce a power-up reset either. The low-power CMOS chips and capacitive effects on the boards may require that SW1 be turned off for up to five seconds. Additionally, any energized circuits on the target system must be de-energized, and the RS-232 *TERMINAL* port and *HOST* port cables at P5 and P6 may need to be disconnected.



**Figure 2-13. EVB2LA Error Code Interpretation**



## **CHAPTER 3**

### **FUNCTIONAL DESCRIPTION**

#### **3.1 INTRODUCTION**

This chapter is a functional description of the M68HC11EVB2 Evaluation Board and its subsystems the EVB2CPU and EVB2LA.

The EVB2 is a hardware/software emulator and evaluation tool for the M68HC11 MCU in single-chip mode. EVB2 control is provided by an on-board MC68HC11E1 MCU executing the BUFFALO/GATECH monitor program. The BUFFALO monitor program is located in EVB2 EPROM. User programs may be downloaded into EVB2 RAM and executed by the on-board MCU.

The minimal interaction between the monitor program and the MCU resources lets the EVB2 closely emulate the M68HC11 and makes the EVB2 as general as possible. EVB2 control and communications use external components which are addressed within the undefined areas of the normal single-chip mode memory map.

#### **3.2 EVB2CPU HARDWARE DESCRIPTION**

The EVB2CPU is designed around an M68HC11 MCU with supporting systems such as power supply, clock, and reset circuitry, an MC68HC24 Port Replacement Unit (PRU) for single-chip mode emulation, and decoding circuitry for addressing external components. These external components include EPROM memory, static RAM memory, a DUART for communicating with a terminal or host computer, and the M68HC11EVB2LA Logic Analyzer (EVB2LA).

##### **3.2.1 Microcontroller (MCU)**

The MC68HC11E1 MCU is supplied with the EVB2CPU. This MCU contains 512 bytes of internal RAM, 512 bytes of internal EEPROM, and is shipped with the internal ROM disabled via the configuration (CONFIG) register ROMON bit. After power-on or a reset occurs, the EVB2 configures the MCU for special test operating mode (MODA = V<sub>CC</sub>, MODB = V<sub>SS</sub>). In this configuration the MCU fetches the reset vector from addresses \$BFFE – \$BFFF. The EVB2CPU EPROM (at location U9) is mapped to the address space from \$8000 – \$BFFF. This EPROM contains the main monitor program and special test mode interrupt vector table, thus allowing a reset to start the monitor program. Once started, the monitor program re-configures the MCU operating mode to expanded multiplexed mode by writing to the HPRI0 register SMOD bit.

---

The external static RAM (at location U11) is mapped to the MCU addresses \$C000 – \$FFFF. The lower 4K bytes of this RAM, from \$C000 – \$CFFF, are reserved and used by the monitor program. The upper 12K byte range is designated as the **user-map** and is intended to emulate the internal ROM for an MCU application program.

Simultaneously pressing the USER RESET (SW2) and RESET (SW3) switches on the EVB2CPU configures the MCU for expanded multiplexed mode (MODA = MODB = VCC). This allows a user program to be executed without monitor program intervention provided that the normal mode reset vector at \$FFFE – \$FFFF is properly initialized.

All 52-pin PLCC members of the M68HC11 family are capable of operating in the EVB2. Thus a user-supplied MCU may be swapped with the factory-supplied MCU. Special care should be exercised, however, when using the EVB2LA while operating the EVB2CPU with a replacement MCU. The EVB2LA requires that the internal read visibility (IRV) feature of the MCU be ON in order that read cycles from the MCU internal modules are captured in the EVB2LA. Only MC68HC11 E-Series parts enable the IRV feature in expanded multiplexed operating mode.

To change the contents of the CONFIG register the MCU must be in the special test mode, while normal MCU operation is expanded multiplexed mode. So a specific sequence must be followed by the monitor program to program the CONFIG register. While the MCU is in special test mode the EVB2 identifies that a CONFIG register change request exists following a reset into the monitor program. The CONFIG register programming operation is performed and the MCU is immediately forced into reset again to implement the CONFIG register change.

Although special care should be taken when programming the MCU CONFIG register due to the critical MCU features that it controls (internal ROM on/off, internal EEPROM on/off, computer operating properly (COP) on/off, security mode on/off, and EEPROM location for MCUs with 2K bytes of EEPROM), the EVB2 cannot be adversely affected by an improper setting in the CONFIG register. The monitor program operates regardless of the internal EEPROM and ROM status, and identifies when the COP system is enabled and resets the COP timer automatically. If an MCU with a re-locatable block of EEPROM is controlling the EVB2, the monitor program verifies that the EEPROM is not mapped over the external monitor EPROM U9 (\$8000 – \$BFFF) and disables the EEPROM if so.

### 3.2.2 Port Replacement Unit (PRU)

The MC68HC24 PRU operates in conjunction with the MCU in the expanded multiplexed and special test operating modes to replace the port B, port C, STRA, and STRB signals lost when the external address and data bus signals are brought out of the MCU. The PRU operates invisibly to a user program controlling the MCU and provides full single-chip mode functionality. Thus the MCU appears to the user program to be operating in single-chip mode, although the SMOD and MDA bits of the HPRI0 register reflect the true expanded multiplexed operating mode of the MCU.

### 3.2.3 EPROM Memory

Programs in the MCU internal ROM may be evaluated by first copying the program into the EVB2CPU RAM and then disabling the internal ROM.

The BUFFALO monitor program is addressed in the \$8000 – \$BFFF range. The EVB2CPU EPROM (at location U9) is enabled by a memory-decoder over the ranges \$8000 – \$B3FF and \$B800 – \$BFFF. This non-contiguous mapping allows for further decoding of the \$B400 – \$B5FF range to control other EVB2 features, and prevents external data bus contention when the MCU internal EEPROM is addressed at \$B600 – \$B7FF. This contention arises because the MCU internal read visibility feature (IRV) is enabled by the monitor program to facilitate capturing of MCU internal resource read cycles by the EVB2LA.

The EVB2CPU EPROM (at location U10) contains the portion of the monitor program required to drive the EVB2LA. U10 is enabled over the address range \$2000 – \$3FFF by a memory-decoder in conjunction with address line A13 tied to the active-high chip enable  $\overline{\text{CE/PGM}}$  (U10 pin 27). Special care must be taken when selecting a replacement part for U10 to insure that U9 PGM (pin 27) also acts as an active-high chip enable. The presence of U10 is identified by the monitor program code in U9 prior to attempting a jump into U10 code.

The high address lines of both U9 and U10 are independently tied to the DUART (at location U12) output port to allow for monitor controlled bank-switching. Bank-switching effectively doubles the addressable size of each EPROM (U9 and U10). This bank-switching is provided as a future expandability option and is currently not implemented in the monitor program.

### 3.2.4 RAM Memory

The EVB2CPU static RAM memory (at location U11) is decoded into the address range \$C000 – \$FFFF by a memory-decoder. This RAM is used to store monitor program variables and for emulation of MCU ROM. Like the EPROMs, the high address line of the RAM may be bank-switched under monitor control by the EVB2CPU DUART output port (at location U12). Again this feature is provided for future use and is not implemented in the monitor program.

### 3.2.5 Clock Circuitry

An on-board clock circuit produces a 2 MHz E-clock frequency unless a target system external clock driver is used via jumper headers J3 and J4. The on-board clock is a parallel resonant crystal oscillator consisting of an 8 MHz ceramic resonator with internal capacitors and a 10 M $\frac{1}{2}$  resistor.

When using a target system external clock signal to drive the EVB2 MCU, do not use a simple crystal oscillator. Only a CMOS compatible buffered oscillator clock source should be used as the target system external clock driver.

### 3.2.6 Power Supply

The EVB2 requires a user-supplied +5 Vdc regulated power supply. Connect this supply directly to EVB2CPU connector P4. Alternatively, an external +9 Vdc unregulated power supply may be connected to connector P3. If an unregulated power supply is to be used, you must mount an MC7805CT voltage regulator on the EVB2 board (refer to Appendix C).

The EVB2CPU SW1 power switch connects the external power supply (from either source) to the EVB2 +5 Vdc bus which supplies the EVB2CPU on-board components. The EVB2CPU red LED L1 lights when +5 Vdc is present on the bus.

### 3.2.7 Reset Circuitry

The EVB2CPU contains an under-voltage sensing circuit (at location Q1) on the EVB2CPU +5 Vdc bus. This under-voltage circuit drives the MCU  $\overline{\text{RESET}}$  line low when an under-voltage condition exists on the bus. The Q1 has dual thresholds which prevent noise on the power supply from false starting the MCU as the power supply voltage ramps up, while also forcing the MCU into reset and preventing runaway when the power supply is turned off.

The MCU  $\overline{\text{RESET}}$  line is also controlled by the EVB2CPU RESET switch (SW3). SW3 lets you drive the  $\overline{\text{RESET}}$  line low and restart the monitor program.

### 3.2.8 Decode Circuitry

Address decoding circuitry is provided on the EVB2CPU to allow the on-board components to be controlled by the MCU by addressing them in various areas of the MCU memory map. The decoding circuitry consists of two memory-decoders (at locations U3 and U4), a hex inverter (at location U7), and an 8-input NAND gate (at location U6).

The memory-decoder (U3) divides the MCU address range into 16K byte blocks, with separate outputs for read and write cycles. All decoder outputs are disabled throughout the address range \$B400 – \$B7FF. This prevents contention on the data bus when the MCU internal read visibility is enabled and internal EEPROM is addressed in \$B600 – \$B7FF. U3 is used to enable the EVB2CPU EPROM (at location U9) during reads of the address range \$8000 – \$B3FF and \$B800 – \$BFFF, while the EVB2CPU EPROM (at location U10) is enabled when reading the address range \$2000 – \$3FFF. The EVB2CPU RAM (U11) is enabled for reading and writing within the address range \$C000 – \$FFFF. Special  $\overline{\text{EVB2LA}}$  decoding is also performed over the range \$4000 – \$7FFF to generate the  $\overline{\text{WTRIG}}$  and  $\overline{\text{RTEST}}$  control signals.

The memory-decoder (U4) decodes the address range \$B400 – \$B5FF for communications and EVB2LA control. The EVB2CPU DUART (at location U12) is enabled for reading and writing over the address range \$B400 – \$B40F. The pulse generated by reading address \$B500 resets and initializes the DUART. The  $\overline{RLA}$ ,  $\overline{WNIB}$ , and PTRESET EVB2LA control signals are generated by reading addresses \$B480 – \$B4B0, writing to \$B490, and writing to \$B500 respectively.

### 3.2.9 External Communications

Communications with the terminal device (EVB2CPU connector P6) and the host device (EVB2CPU connector P5) are through the DUART (at location 12). The DUART contains dual asynchronous receiver/transmitters which operate independently. The DUART also performs several input/output functions to control the EVB2. The RS-232 driver (at location U5) uses the +5 Vdc power supply of the EVB2 to internally generate RS-232 voltage levels ( $\pm 10$  Vdc) for communications with RS-232 compatible equipment.

The MCU controls the DUART by reading and writing within the address range \$B400 – \$B40F. A hardware reset of the DUART is generated by reading address \$B500. The DUART is capable of generating a non-maskable XIRQ interrupt to the MCU after the proper initialization occurs. This feature passes CPU control from a user program back to the monitor program when a key is pressed on the user's terminal.

The DUART contains an input port, with six parallel binary inputs, that can be interrogated by the MCU. These inputs allow the user to control the EVB2 functions such as default RAM and control register mapping by repositioning the jumpers on EVB2CPU jumper headers J15 and J16. The DUART also contains an eight-bit parallel output port used by the MCU to control the LAMODE and S16 signals (EVB2LA) as well as providing bank-switching controls to EVB2CPU EPROM at locations U9 and U10 and RAM at location U11. These bank-switching controls are not currently utilized by the EVB2.

## 3.3 EVB2LA HARDWARE DESCRIPTION

The EVB2LA consists primarily of:

- A static RAM buffer (32 bits x 8K)
- A counter to address the RAM
- Three-state buffers to control data flow into and out of the RAM
- A trigger RAM for memory of trigger addresses
- A state counter
- Post-trigger counter
- Field programmable logic array (FPLA) to provide state machine control

### 3.3.1 EVB2LA RAM

The EVB2LA capture buffer consists of four 8K x 8 bit static RAM chips (at locations U5, U6, U7, and U8,) to provide a 32 bit wide data word. All RAM is addressed in parallel by the 13 low order outputs of the EVB2LA counters. These counters increment and decrement the current RAM address. Each RAM has access to a different input source as shown in Table 3-1.

**Table 3-1. Logic Analyzer RAM Configuration**

ID	Input	Output	Output Enable	Write Enable
U5	Auxiliary byte or test byte	Data bus	$\overline{\text{LAMODE}}$ , $\overline{\text{READ AUX}}$	$\overline{\text{WRITE}}$
U6	High byte of address bus	Data bus	$\overline{\text{LAMODE}}$ , $\overline{\text{READ HIADD}}$	$\overline{\text{WRITE}}$
U7	Low byte of address bus	Data bus	$\overline{\text{LAMODE}}$ , $\overline{\text{READ LOADD}}$	$\overline{\text{WRITE}}$
U8	Data bus	Data bus	$\overline{\text{LAMODE}}$ , $\overline{\text{READ DATA}}$	$\overline{\text{WRITE}}$

The auxiliary byte stored in the RAM at location U5 consists of six user test points from connector P1, the MCU R/W line, and the MARK bit. The three-state buffers may alternately be configured so that the input to RAM U5 is a test byte consisting of the state counter outputs S1-S8, the PTCLOCK, PT4096, PT8192, and TRIGGER control signals for performance testing.

All RAM chips are enabled in parallel by the  $\overline{\text{LAMODE}}$  signal from the EVB2CPU. Output from each RAM is also enabled by an independent signal from the memory-decoder at location U13. All of the RAMs are write enabled in parallel from the  $\overline{\text{WRITE}}$  output of the field programmable logic array (FPLA) at location U18.

The EVB2CPU can retrieve data from the capture buffer by reading one of four decoded addresses. Each address enables one of the four RAMs and enables the respective three-state buffer to place the output of the RAM onto the data bus. The data is retrieved from the address in the RAM which is determined by the EVB2LA counter value.

### 3.3.2 EVB2LA Counters

The EVB2LA counters are four synchronous 4-bit counters configured so that they effectively act as a single 16-bit counter. The lower 13 bits address the EVB2LA RAM, while the upper 2 bits are the control signals  $\text{EN}_A$  and  $\text{EN}_T$  that enable the three-state buffers (U4 and U10). These buffers are provided for performance testing of the EVB2LA.



The four EVB2LA counters (at locations U19, U20, U21, and U22) are high-speed, CMOS, up/down, synchronously pre-loadable counters. The up/down feature is controlled by the LADOWN signal output of the memory-decoder at location U13. While normally configured to count up while capturing data, the counters may count down to facilitate data retrieval by the EVB2CPU. The synchronous pre-load feature is implemented by connecting each of the four EVB2LA counters and the state counter in series. Connecting the EVB2LA counters and state counter in series lets you shift one nibble (4-bits) from the data bus into the high order EVB2LA counter, while each counter in the series is loaded with the previous counter's value. Thus after five nibble-shift operations, all of the EVB2LA counters and the state counter are pre-loaded with specific EVB2CPU values. By pre-loading the EVB2LA counters and the state counter with specific EVB2CPU values the EN<sub>A</sub> and EN<sub>T</sub> outputs are determined.

The LACOUNT signal from the FPLA (U18) causes a single increment/ decrement of the 14-bit counter chain. LACOUNT is generated automatically by the FPLA state machine while capturing data. While retrieving data, the EVB2CPU can increment the EVB2LA counter chain by reading address \$B4B0, and decrement it by reading address \$B4A0. The counters are disabled from counting by the EVB2CPU LAMODE signal being active.

### 3.3.3 Three-State Buffers

Seven three-state buffers are required to control data flow to and from the EVB2LA RAM as shown in Table 3-2.

**Table 3-2. Three-State Buffer Configuration**

ID	Input	Output
U4	Auxiliary byte	RAM (U5)
U10	Test byte	RAM (U5)
U1	RAM (U5)	data bus
U11	A8 – A15	RAM (U6)
U2	RAM (U6)	data bus
U12	A0 – A7	RAM (U7)
U3	RAM (U7)	data bus

Three-state buffer at location U10 may also be used to place the test byte directly onto the data bus using the RTEST signal output of the EVB2CPU. Therefore the test byte may be determined by reading anywhere in the address range \$4000 – \$7FFF.

### 3.3.4 Trigger RAM

A 16K x 1 bit static RAM (at location U9) is the memory for trigger addresses. The trigger RAM is write enabled by the EVB2CPU  $\overline{WTRIG}$  output, which is asserted by writing to the address range \$4000 – \$7FFF. The trigger RAM is addressed by the address bus lines A0 – A13 and takes its input from the data bus line D7. Therefore, when the EVB2CPU writes to the address space \$4000 – \$7FFF, D7 is stored in the corresponding address of the trigger RAM.

The output of the trigger RAM is fed into the FPLA as the TRIGGER signal. TRIGGER is asserted when addresses appear on the address bus that have been set in the Trigger RAM. When the EVB2LA state machine is setup for data capture, the state machine responds to the TRIGGER input depending on the particular EVB2LA state.

### 3.3.5 State Counter

A 4-bit binary counter (EVB2LA location U17) determines the current state for the EVB2LA state machine. The output of the state counter is fed into the FPLA. The FPLA changes states by enabling the state counter to count via the  $\overline{STCOUNT}$  signal. The state counter is clocked from the inverted MCU E-clock output. The up/down feature of the binary counter is not utilized, so the state counter always counts up. The state counter is disabled by the EVB2CPU  $\overline{LAMODE}$  signal.

The state counter is the fifth counter in the chain consisting of the EVB2LA counters and the state counter. The state counter is pre-loaded with the first nibble shifted into the counter chain after five nibbles have been loaded. The output of the state counter is also the lower four bits of the test byte which may be interrogated by the EVB2CPU using the  $\overline{RTEST}$  signal. By reading the test byte as each nibble is shifted into the counter chain, the value of the state counter as well as the EVB2LA counters can be determined by the EVB2CPU.

### 3.3.6 Programmable Logic Array

The FPLA (at location U18) is the heart of the EVB2LA state machine. The 24-pin FPLA is used on the EVB2LA to save board space and lower the number of discrete components required to generate the EVB2LA control signals. The FPLA uses 15 input signals to produce six output signals (shown in Tables 3-3 and 3-4) that drive the EVB2LA. For more information on the FPLA, refer to the FPLA flow diagrams in Appendix B.

**Table 3-3. FPLA Input**

Input Signal	Source
A15	MCU
A14	MCU
R/ $\overline{W}$	MCU
$\overline{E}$	MCU/74HC00
$\overline{LIR}$	MCU
S1 – S8	State Counter
S16	EVB2CPU
$\overline{COUNT}$	EVB2/74HC09
$\overline{LAMODE}$	EVB2CPU
PT8192	PT Counter
PT4096	PT Counter
TRIGGER	Trigger RAM

**Table 3-4. FPLA Output**

Output Signal	Purpose
$\overline{XIRQ}$	Non-maskable interrupt to MCU
$\overline{WRITE}$	Enable three-state buffers and RAMs for capture of one bus state
MARK	Auxiliary byte, inverted $\overline{LIR}$
PTCLOCK	Clock PT Counter
$\overline{LACOUNT}$	Clock EVB2LA Counters
$\overline{STCOUNT}$	State Counter increment

---

### 3.3.7 Post-Trigger Counter

The post-trigger (PT) counter (at location U16) is a 14-bit counter controlled by the FPLA state machine and is used as a memory device for the number of cycles elapsed after the trigger condition. Only two outputs are used from the PT counter: PT4096, asserted after 4096 counts and PT8192, normally asserted after 8192 counts.

State machine usage of the PT counter output depends on the EVB2LA function in progress. For example, the PT counter is clocked every cycle when a trace-after function is triggered. Therefore PT8192 signals that the capture buffer is full and to stop the trace. The PT counter is pre-loaded with the value 8192-N by the EVB2CPU for a trace before <n>th execution trace and the PT counter is clocked by each trigger. The PT8192 signal is again used to end the trace.

The PT counter is reset by the PTRESET signal from the EVB2CPU which is generated by writing to address \$B500. The EVB2CPU may clock up the PT counter by reading address \$B4B0 while in state 00000.

## CHAPTER 4

### OPERATION

#### 4.1 INTRODUCTION

This chapter contains operating procedures and limitations for the EVB2. Additionally this chapter addresses the advanced features available with the board and special instructions for performing some tasks. Finally, how to use the M68HC11EVB2LA Logic Analyzer is covered. This section is complemented by Chapter 5 Monitor Commands which is a detailed description of each BUFFALO (Bit User's Fast Friendly Aid to Logical Operation) monitor command.

Before attempting to use the EVB2 or logic analyzer for the first time, follow the steps outlined in Chapter 2.

#### 4.2 OVERVIEW

The EVB2 EPROM contains a monitor program called BUFFALO. This program controls the EVB2 and allows you to direct its functions. The user accomplishes this by entering commands on the terminal device.

#### 4.3 EVB2 LIMITATIONS

The EVB2 is a single-chip mode emulator/evaluation tool for the M68HC11 family. There are several differences between EVB2 and standard MCU operation. The following paragraphs list these EVB2 limitation.

- These memory locations in the MCU memory map are used by the EVB2, and so, are unavailable for user programs:
  - \$2000 – \$3FFF EVB2LA logic analyzer control program
  - \$4000 – \$7FFF EVB2LA triggering circuitry
  - \$8000 – \$B3FF Monitor program
  - \$B400 – \$B5FF Decoded control circuitry
  - \$B800 – \$BFFF Monitor program
  - \$C000 – \$CFFF Monitor program usage of EVB2 RAM

The EVB2 disables the monitor program EPROM over the \$B400 – \$B7FF address range to prevent conflicts with internal EEPROM at \$B600 – \$B7FF. Spurious read or write operations within the \$B400 – \$B5FF range can interfere with EVB2 operation, and may require a hardware reset to recover.

- The MCU non-maskable interrupt resource  $\overline{XIRQ}$  is used by the EVB2 to return control to the monitor program after execution of a user program. It is recommended that your target system ignore  $\overline{XIRQ}$ . If the target system must control  $\overline{XIRQ}$ , refer to paragraph 2.3.2 for methods of disengaging EVB2 control circuitry from  $\overline{XIRQ}$ . Regardless of the hardware settings, the monitor program always intervenes in  $\overline{XIRQ}$  interrupt sequencing before passing control to the user program.
- The MCU software interrupt resource SWI is used by the monitor program to implement breakpoints. A software interrupt instruction cannot be used in user programs.
- A RESET into the monitor program causes three things to occur:
  1. Starts a 64 cycle time-out sequence. Completion of the time-out sequence prevents you from changing some MCU control register bits, such as the INIT register. Also, user program instructions that modify these bits have no effect after the time-out sequence. The user-reset feature allows evaluation of user program initialization code (refer to paragraph 4.4.3).
  2. The state of jumper headers J15 and J16 are checked to determine RAM and control register mapping. This permits mapping of RAM and/or control registers starting at addresses \$0000 or \$1000 (refer to paragraph 2.3.2.2 for J15 and J16 position definitions).
  3. These MCU control registers are automatically modified from their default post-reset conditions:
 

BPROT	register	All bits are cleared (applies to MC68HC11E9 only)
HPRIO	register	IRV bit is set
INIT	register	Value determined by jumpers J15 and J16
OPTION	register	IRQE, CR1, and CR0 bits are set
TEST1	register	DISR bit is cleared
- The EVB2 is fully compatible with M68HC11 family members with 2K bytes of internal EEPROM. However, if the EEPROM is mapped over the monitor-program address space via the CONFIG register, the EEPROM is disabled.
- The EVB2 contains an MC34064 under-voltage sensing circuit that pulls the  $\overline{RESET}$  line low during low power supply voltage conditions.

- Generally the EVB2 P1 connector is a one-for-one pin-out of the on-board MCU. In the factory configuration five signals are not available on P1: XTAL, EXTAL, VDD, MODA, and MODB,. Three of these signals (XTAL, EXTAL, and VDD) can be connected to P1 if you use the jumper header options (refer to paragraph 2.3.2.2). While two signals are not wired to the EVB2 P1 connector: MODA and MODB.
- The EVB2 contains an MC68HC24 port replacement unit to rebuild the port B and C I/O lines lost in expanded multiplexed mode. Minor differences exist between the MC68HC11 and MC68HC24 timing implementation for the STRA and STRB signals. Refer to the MC68HC24 Advanced Information Data Sheet for more information.

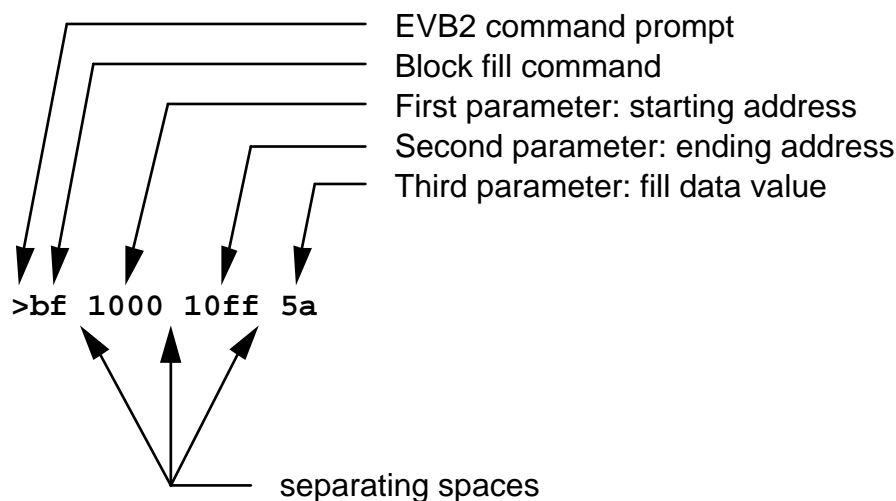
#### 4.4 EVB2 OPERATING PROCEDURE

When the EVB2 is turned on the on-board MCU comes out of reset in special test mode (so that the reset vector is fetched from EPROM at \$BFFE – \$BFFF). After the power-up sequence the BUFFALO monitor gains control of the MCU. The monitor program executes several self-tests, and then switches to expanded multiplexed operating mode. The board status is displayed for the user on the EVB2 power-on screen (Figure 4-1).

```
BUFFALO/GATECH 1.0 - Bit User's Fast Friendly Aid to Logical Operation
Register Start (J15): 1000
MCU RAM Range (J16): 0000 01FF
EEPROM Range:      B600 B7FF
User Program Range: D000 FFFF
Internal ROM Disabled
COP System Disabled
EVB2 RAM test passed
EVB2LA Logic Analyzer Functioning
>
```

**Figure 4-1. EVB2 Power-On Reset Screen**

The greater than character (>) in Figure 4-1 is the EVB2 command prompt. When > is displayed, the EVB2 is waiting for input from the user. The user controls the EVB2 by entering a valid command and a carriage return. One or more tab or space characters must separate the EVB2 command and each of its parameters. An example of an EVB2 command is shown in Figure 4-2. When the EVB2 completes the execution of a command, the command prompt is displayed.



**Figure 4-2. EVB2 Command Example**

These rules apply to command line entries:

- Pressing a carriage return with no preceding characters repeats the last command entered.
- Command line entries may be either upper or lower case. The EVB2 converts all commands to uppercase before parsing the command line and interpreting the entry.
- A maximum of 80 characters may be entered on the command line.
- Command line errors may be corrected by entering a backspace, (**CNTL-H**).
- Command line entries may be aborted by entering (**CNTL-X**) or (**DELETE**).

Valid EVB2 commands are listed in Chapter 5. All commands may be abbreviated by entering the minimum number of characters required to uniquely identify the command. If an abbreviated command entry matches multiple commands in the monitor's command table, the first match encountered in the table is executed.

The command line error messages generated by the EVB2 and error message definitions are:

What?	Invalid EVB2 command
Too Long	Command line entries must be less than 80 characters



---

#### 4.4.1 User Program Evaluation Procedure

After a reset into the monitor program, the monitor program maintains control of the MCU until instructed to execute a user program. A user program may either be entered line-by-line via the monitor program assembler/disassembler (refer to the **ASM** command), or may be downloaded into the EVB2 memory (refer to paragraph 4.4.2). There are several monitor commands that execute user programs:

<b>CALL</b>	Execute user subroutine in real time
<b>GO &amp; RUN</b>	Execute a user program in real time
<b>PROCEED</b>	Execute a user program in real time past a breakpoint
<b>TRACE &amp; STOPAT</b>	Single-step trace one instruction at a time
<b>SB, DB, SP, &amp; PC</b>	Execute user program with real-time tracing

The user program executes until control of the MCU returns to the monitor program through one of these:

- A key is pressed on the display terminal
- A breakpoint is encountered
- An RTS instruction returns to the monitor program from the **CALL** command
- The M68HC11EVB2LA logic analyzer completes a trace

To generate a hardware reset into the monitor program, press the RESET switch (SW3). When this switch is pressed, the EVB2 MCU comes out of reset in special test mode and follows an initialization procedure similar to power-on reset. EVB2 self-tests are not executed when SW3 is pressed.

Immediately prior to passing MCU control to the user program, a checksum value is calculated by the monitor program. The checksum value is over the user program range \$D000 – \$FFFF. When the user program completes execution, control of the EVB2 returns to the monitor program. The monitor program then calculates a new checksum value and compares it to the previous value. If the values are not identical, the following message is displayed on the user's terminal:

**\*\*WARNING\*\***: User code space modified

If this warning message appears, check the user program via the **ASM** command or the **VERIFY** command. Verify the integrity of the user program and ensure the user program is not writing to the memory range \$D000 – \$FFFF before again attempting execution.

The monitor program includes three principal debugging tools: breakpoints, single-stepping, and real-time tracing. These tools should not be confused with the tracing capabilities introduced by the EVB2LA logic analyzer. The logic analyzer board adds extensive bus state logic analysis features. Refer to paragraph 4.5 for logic analyzer operation.

A breakpoint may be set at an address in a user program to stop user program execution. When a breakpoint is encountered control returns to the monitor program. The user program stops before the instruction at the breakpoint address is executed. In this way the state of the CPU registers, MCU control registers, user program variables and target system hardware may be checked.

Single-stepping executes one user program instruction at a time. After execution of each instruction control returns to the monitor program and the state of the CPU registers is displayed. Single-stepping allows the user to watch execution of each program instruction. When single-stepping, execution is not in real time.

The real-time tracing feature displays a requested value on the user's terminal periodically during user program execution by briefly interrupting the user program to assist in program debugging.

#### 4.4.1.1 User Registers

The monitor program contains a set of user registers that emulate the CPU registers. The user registers values are loaded into the CPU registers immediately before the user program is executed by one of these commands: **CALL, GO & RUN, PROCEED, TRACE & STOPAT, SB, DB, SP, & PC**. When control returns to the monitor program, the value of each CPU register is saved in its corresponding user register. Table 4-1 shows the names of each user register and the CPU register correspondence. The user registers may be displayed with the RD command and modified with the RM command.

**Table 4-1. User Register Definitions**

User Register	CPU Register
P	Program counter
Y	Index register Y
X	Index register X
A	Accumulator A
B	Accumulator B
C	Condition code register
S	Stack pointer

After reset, the values in the user registers are undefined except for the C register. Like the CPU condition code register, the S, X, and I bits in the C register are always set after reset. However any time a user program is executed, the X-bit in the C register is cleared to allow the monitor program to regain control of the MCU.

After a user program has executed, the value of the I bit interrupt mask is always saved in the C register as execution returns to the monitor program. However interrupts enabled by the user program will not be serviced while the monitor program has control of the MCU. Instead, these interrupts will remain pending until reset or until the user program is executed again. At that time, if the I bit is clear then all pending interrupts will be serviced in the order of the established interrupt priority.

#### 4.4.1.2 Breakpoints

Breakpoints are set and removed using the **BR** command. Breakpoints appear in the breakpoint table. The breakpoints in the breakpoint table are installed in the user program when the **GO**, **CALL**, **RUN**, and **USER** commands are executed. When the monitor program regains control it removes the breakpoints from the user program.

Breakpoints are not installed when a single-step trace is executed, or when the logic analyzer is present and set-up to capture a trace.

#### 4.4.1.3 Single-Step Trace

A single-step trace occurs when either a **TRACE** or **STOPAT** command is executed. All user programs must reside in RAM or internal EEPROM to use the **PROCEED**, **STOPAT**, and **TRACE** commands. The **PROCEED** command is implemented in the monitor by single-stepping one user program instruction (past a breakpoint address), subsequently installing breakpoints, and then executing a **GO** command. The instruction to be executed by single-step trace is determined by the user P-register (program counter).

A single-step trace places a software interrupt (SWI) instruction after the instruction to be executed. When the SWI instruction is encountered control returns to the monitor program. The primary advantage of this single-stepping method is that the software interrupt has the lowest interrupt priority. This method allows all other pending interrupts to be serviced, followed by the execution of the traced instruction. Single-step trace execution is not in real time and may affect the operation of some user programs. Because the monitor uses the SWI opcode, the user cannot use the SWI instruction in user programs.

#### 4.4.1.4 Real-Time Trace

Real-time tracing periodically displays a value on the display terminal during user program execution. Four commands start real-time tracing: **SB**, **DB**, **SP**, and **PC**. The **SB** command displays a single-byte (8-bit) value from memory. The **DB** command displays a double-byte (16-bit) value from memory. The **SP** command displays the user stack-pointer value. The **PC** command displays the user program counter value. Each command executes the user program at the address contained in the user program's reset vector (\$FFFE – \$FFFF).

Real-time tracing interrupts the user program with a non-maskable interrupt (XIRQ) once every 0.25 seconds and displays the requested data on the terminal device. The length of the user program interruption needed to display the data value has been minimized, but this process in itself may affect execution of the user program. While using single-byte or double-byte tracing, reading one of the MCU control registers PIOC, SPSR, or SPCR will affect execution of the user program.

#### 4.4.2 Downloading Procedure

The monitor program includes two commands, **LOAD** and **VERIFY**, to download programs and data to the EVB2. All programs and data must be in Motorola S-record format as described in Appendix A. The S-record data may be downloaded from a personal computer through the **TERMINAL** port or from a **HOST** device through the **HOST** port.

The **LOAD** command is used to download S-record files into EVB2 memory, while the **VERIFY** command is used to check an S-record file against EVB2 memory. Both commands have the same syntax rules:

1. If the only parameter following the command is the letter t, the EVB2 waits for the S-record file to be sent from the **TERMINAL** port (P6).
2. If any other character sequence follows the command, that character sequence is transmitted to the **HOST** device. The EVB2 then waits for the S-record file to be sent from the **HOST** port (P5).

While the **LOAD** command may be used to download programs and data into EVB2 RAM, MCU RAM, and even MCU EEPROM (with restrictions). There are restrictions when downloading programs and data into MCU EEPROM (refer to paragraph 4.4.2.3).

Remember, data stored in RAM is erased when the EVB2 power is removed. Pay special attention when loading data into MCU RAM, since this data is cleared at power-down. Normally any data in the MCU RAM used by the user program should be stored in the MCU RAM by the user program during initialization.

#### 4.4.2.1 Personal Computer – EVB2 Downloading

Any personal computer running a terminal emulation program can download programs and data to the EVB2. Refer to paragraph 2.3.3 for the proper serial communication settings to communicate with the EVB2. The terminal emulation program should transmit ASCII text files in order to send S-records.

Figure 4-3 demonstrates downloading an S-record file using an IBM-PC or compatible computer and the terminal emulation program KERMIT. Connect the EVB2 to the computer COM1 serial port and start the KERMIT terminal emulation program.

C>KERMIT<CR>	Enter KERMIT program.
IBM-PC Kermit-MS VX.XX Type ? for help	
Kermit-MS>SET BAUD 9600<CR>	Set PC serial port baud rate begin terminal emulation.
Kermit-MS>CONNECT<CR>	
[Connecting to host, type Control-] C to return to PC]	
	Switch on EVB2 or press RESET.
BUFFALO/GATECH 1.0 - Bit User's Fast Friendly Aid to Logical Operation	
Register Start (J15): 1000 MCU RAM Range (J16): 0000 01FF EEPROM Range: B600 B7FF User Program Range: D000 FFFF Internal ROM Disabled COP System Disabled EVB2 RAM test passed EVB2LA Logic Analyzer Functioning	
>load t<CR>	Enter download command exit terminal emulation.
(CNTL)C	
Kermit-MS>PUSH<CR>	Shell to DOS.
The IBM Personal Computer DOS Version X.XX (C)Copyright IBM Corp 1981, 1982, 1983	
C>TYPE (File Name) > COM1<CR>	Copy S-record file to serial port.
C>EXIT<CR>	Return to KERMIT.
Kermit-MS>CONNECT<CR>	Return to terminal emulation.
<CR>	
>	Enter monitor commands.

**Figure 4-3. User Program Downloading with KERMIT**

#### 4.4.2.2 HOST – EVB2 Downloading

The EVB2 supports use of a host computer via the HOST port (P5). Two EVB2 commands may be used with a host computer: transparent mode (**TM**) and **LOAD**. Use the **TM** command to transmit data to and from the host computer and the terminal. In transparent mode the EVB2 effectively becomes transparent to the HOST computer. This allows editing, cross-assembling, and linking of 68HC11 programs on the HOST computer, as well as creation of S-record files. Pressing **(CNTL)A** terminates the transparent mode. The **LOAD** command may then be used to transmit a one-line command to the HOST computer. The one-line command instructs the host computer to transmit the ASCII S-record file. The host computer waits for the S-record file to be transmitted. Figure 4-4 presents a sample session in which the EVB2 is connected to a UNIX host computer system.

<code>&gt;tm&lt;CR&gt;</code>	Enter transparent mode.
<code>\$ xasm test.asm -l &gt; test.lst</code>	Cross-assemble program.
	•
	•
	•
<code>\$ (CNTL)A</code>	Exit transparent mode.
<code>&gt;load cat text.s19&lt;CR&gt;</code>	Load S-record file from HOST.
<code>cat text.s19</code>	EVB2 echoes command to HOST and terminal.
<code>done</code>	S-record file successfully received.
<code>&gt;</code>	Enter next command.

**Figure 4-4. HOST Device Downloading**

---

### 4.4.2.3 Downloading to EEPROM

When developing S-record files you define the S-record memory address, the place in memory where the S-record data is stored. The memory address of the S-record file is the destination address. When loading an S-record using the **LOAD** command, if the destination address is within the EEPROM address range, the destination cells are automatically byte-erased (as required).

There is a timing problem when downloading to EEPROM. Downloading takes approximately 20 ms per byte to erase and program, while incoming data bytes arrive approximately every 1 ms at 9600 baud. There are three possible solutions to downloading S-record data into EEPROM memory:

1. An offset value can be added (**OFFSET** command) to all S-record addresses so that data destined for EEPROM memory is actually loaded into EVB2 RAM. Once loaded, use the **MOVE** command to copy the data into the EEPROM memory. For example, to download S-record data to EEPROM memory at \$B600 – \$B7FF through the TERMINAL port:
  - a. Specify <offset> into EVB2 RAM                      **OFFSET 4000**
  - b. Issue load from terminal port command              **LOAD T**
  - c. Send S-record text file                                      (send S-record file)
  - d. Copy data into EEPROM memory                      **MOVE F600 F7FF B600**
2. Use hardware handshaking with the terminal device to prevent the terminal device from sending the S-record data too fast. Refer to paragraph 2.3.3 for hardware-handshaking set-up requirements.
3. Use the **SPEED** command to set the HOST port baud rate to 300 baud and use the **LOAD** command to download the S-record file through the HOST port.

### 4.4.3 User-Reset Procedure

The user-reset function bypasses the monitor program, resets the MCU into expanded multiplexed operating mode, and fetches the reset vector from \$FFFE – \$FFFF. Operation of the EVB2 MCU in expanded multiplexed mode is virtually identical to single-chip mode operation due to the presence of the MC68HC24 PRU. When a user-reset is performed, a user program must be present and the reset vector at \$FFFE – \$FFFF must contain the user program starting address.

To force a user-reset, press and hold the user switch (SW2) and then press and release reset switch (SW3). This switch combination resets the MCU, causing it to come out of reset in expanded multiplexed operating mode. Since the monitor program is bypassed, the reset switch must be pressed to return to the monitor program. If you issue the **USER** command prior to the user-reset, pressing any key on the terminal device returns control to the monitor program.

---

A user-reset lets a user program control the MCU after reset and permits the user to evaluate the MCU initialization and operation. The user-reset also bypasses the limitations detailed in paragraph 4.3. These limitations are imposed after a normal reset into the monitor program.

The **USER** command prepares the EVB2 for a user-reset and lets control return to the monitor program. Issuing the **USER** command and initiating a user-reset returns control to the monitor program when:

- Any key is pressed on the user's terminal
- A user program breakpoint occurs
- The EVB2LA captures a trace

The logic analyzer can capture a trace following a user-reset. This trace capture occurs when one of the trace capture commands and either a **RUN**, **GO**, or **CALL** command is executed (refer to paragraph 4.5). It must be noted, however, that MCU data bus read cycles from the MCU internal RAM, EEPROM, and control registers captured in the logic analyzer will contain irrelevant data unless the internal read visibility (IRV) feature is activated during a user-reset. Refer to paragraph 4.5.11 for instructions to turn IRV on within the user program.

#### 4.4.4 CONFIG Register Programming Procedure

The MCU CONFIG register is implemented as an EEPROM cell to retain its value when MCU power is removed. The CONFIG register controls some basic features of the MCU. The CONFIG register can normally only be changed by performing a bulk erase operation or a single-byte programming operation on the CONFIG register address. To change the CONFIG register, the MCU must be in special test mode.

Any attempt to alter the CONFIG register contents causes the EVB2 to store the new value and display the message, CONFIG programmed, reset to implement. The actual programming sequence will not occur until the next monitor reset sequence is performed. Initiate the reset sequence by either using the **RESET** command or pressing the reset switch (SW3). The programming sequence is aborted if the **RESET T** command is used or a user program is subsequently executed. Refer to the appropriate technical summary for the CONFIG register bit description.



#### 4.4.5 EEPROM Programming Procedure

The EVB2 MCU is the MC68HC11E1FN. The HC11E1 has 512 bytes of EEPROM memory in address range \$B600 – \$B7FF. You can replace the on-board MCU with any M68HC11 family member and the EVB2 will still operate. This is especially useful for programming the larger and more versatile EEPROM memories of some family members.

The internal EEPROM address range is automatically determined and displayed after reset into the monitor program. Additionally, EEPROM cells are programmed when the EVB2 programming command sequence is executed (refer to paragraph 4.4.2.3 for EEPROM programming limitations). Two other commands, **BULK** and **BULKALL** may be used to modify the MCU EEPROM. Each of these commands erase all EEPROM cells to the value \$FF. **BULKALL** additionally prepares the EVB2 to erase the CONFIG register following a subsequent reset.

#### 4.4.6 ROM Debugging Procedure

Programs contained in ROM, such as the internal MCU ROM, cannot be directly debugged with the EVB2 because the monitor program must be able to set breakpoints and change interrupt vectors in the user program. This limitation may be overcome by loading the ROM program into EVB2 RAM. This may be accomplished in two ways:

- S-record format user programs may be downloaded into EVB2 RAM using the LOAD command.
- If the user program is stored in EVB2 MCU internal ROM, it can be copied into the EVB2 RAM using these steps:
  1. Set the ROMON bit in CONFIG to 1                                  MM 103F → 0F
  2. Reset the MCU to implement the CONFIG change                  RESET
  3. Copy the ROM program to RAM    MOVE E000 FFFF E000
  4. Set the ROMON bit in CONFIG to 0                                  MM 103F → 0D
  5. Reset the MCU to implement the CONFIG change                  RESET

After the user program is copied into the EVB2 RAM, it may be evaluated using the BUFFALO monitor debug features.

---

## 4.5 EVB2LA OPERATING PROCEDURE

The M68HC11EVB2LA Logic Analyzer connects to the M68HC11EVB2 through a 50-pin ribbon cable. The logic analyzer monitors the MCU multiplexed address/data bus as well as several other signals in order to capture selected cycles of user program real-time execution. The captured data may be viewed after the monitor program regains control of the MCU.

### 4.5.1 Definitions

The following definitions are used throughout this manual and in the BUFFALO monitor program to define the basic operating characteristics of the logic analyzer :

- Trace instruction execution mode — captures every cycle of user program execution.
- Trigger — in trace instruction execution mode, an address in the user program that causes the logic analyzer to start or stop capturing user-program-execution cycles. In trace only mode, an address at which a read or write cycle is captured in the capture buffer.
- Trace only mode — captures only selected cycles during user program execution.
- Capture buffer — 32-bit wide by 8K RAM memory. Each 32-bit row of the buffer holds data concerning one cycle of user program execution. This data includes the 16-bit address bus, 8-bit data bus, and an 8-bit auxiliary byte.
- Auxiliary byte — 8 bits in the capture buffer. The byte is comprised of six user-defined test points, the MARK bit, and the MCU R/W signal.
- Test points — six user-defined TTL compatible inputs on the logic analyzer board that are captured in the capture buffer.
- MARK bit — control signal generated by the logic analyzer control circuitry and stored in the capture buffer. In trace instruction execution mode, MARK is asserted during the first cycle of each instruction executed by the MCU. In trace only mode, MARK is asserted for all cycles captured in the capture buffer.
- Trigger offset — pointer used to indicate the current position in the capture buffer from which data is displayed. The trigger offset value is relative to the trigger point and is in units of instructions for data captured from trace instruction execution mode, and units of cycles for trace only mode.
- Trace set-up command — BUFFALO command used to specify trigger addresses and prepare the logic analyzer to capture a particular type of trace the next time the user program is executed.
- Trace display command — BUFFALO command used to display data in the capture buffer from a previous user program trace.

### 4.5.2 Overview

The two principal logic analyzer hardware elements are: the capture buffer and the control circuitry. The control circuitry allows the EVB2 monitor program to control the logic analyzer. Monitor program control includes setting up the various traces and displaying data in the capture buffer.

The logic analyzer contains two modes for data capture: trace instruction execution mode and trace only mode. Trace instruction execution mode captures every cycle of user program execution, while trace only mode captures only selected cycles. The type of trace to be performed is determined by the command used to set-up the trace.

When the logic analyzer is installed, fourteen commands are added to the standard BUFFALO command set. There are five logic analyzer commands which are used to set-up the logic analyzer to capture data into the capture buffer. After a user program has been executed and data captured in the capture buffer, two commands are used to display the capture buffer data, and six commands are used to move through and analyze the capture buffer data. Table 4-2 shows the logic analyzer command names and organization.

**Table 4-2. Logic Analyzer Command Organization**

Function	Trace Commands	
	Trace Instruction Execution Mode	Trace Only Mode
Setup	ABOUT AFTER BEFORE FAULT	ONLY
Display	DIS RAW	RAW
Positioning	FIND SKIP TIME < ^ >	FIND SKIP < ^ >

Refer to paragraph 5.3 for detailed descriptions of each logic analyzer command.

### 4.5.3 Trace Set-Up

Issue a trace set-up command to capture a user-program-execution trace. The type of trace to be performed by the logic analyzer is determined by the command used to set-up the trace. Five types of traces can be performed as described in Table 4-3. When a trace set-up command is entered, the monitor program prepares the logic analyzer for the trace and then displays the command prompt. The user must issue one of the user program execution commands (**GO**, **CALL**, **RUN**, or **USER**) to start user program execution. Reset the EVB2 to cancel a trace set-up command, and issue another trace set-up command to cancel a previous set-up.

**Table 4-3. Trace Types**

Command	Mode	Description
AFTER	Instruction Execution	Capture 8K cycles of execution data following the trigger
BEFORE	Instruction Execution	Capture up to 8K cycles of execution data before the trigger
ABOUT	Instruction Execution	Capture up to 4K cycles of execution data before the trigger and 4K cycles after the trigger
FAULT	Instruction Execution	Capture up to 8K cycles of execution data before a fault condition, defined as an instruction fetched out of the user program area or a write to the user program area
ONLY	Only	Capture only read or write cycles to the trigger

The **ABOUT**, **AFTER**, and **BEFORE** commands require that at least one address within the user program area (\$D000 – \$FFFF) be supplied as a trigger. This address must be the start of an instruction in the user program. If multiple trigger addresses are given, executing any one satisfies the trigger condition.

The **ONLY** command requires a parameter to indicate which mode is desired: R – read cycles of RAM/registers, W – write cycles to RAM/registers, A – accesses (both read and write cycles to RAM/registers), E – write cycles to the internal EEPROM, and V – interrupt vector fetches. At least one address, or range of addresses, within the appropriate range of memory, must follow the R, W, A, and E parameters.

Some limitations are imposed once a trace set-up command is issued. Single-step tracing is not allowed. When a subsequent user program execution command (**GO**, **CALL**, **RUN**, or **USER**) is issued, breakpoints in the breakpoint table are not set in the user program. If desired, the logic analyzer trace may be aborted by resetting the EVB2. Also note that at no time will data capture begin until the user program begins executing in the upper 16K bytes of the MCU address space.

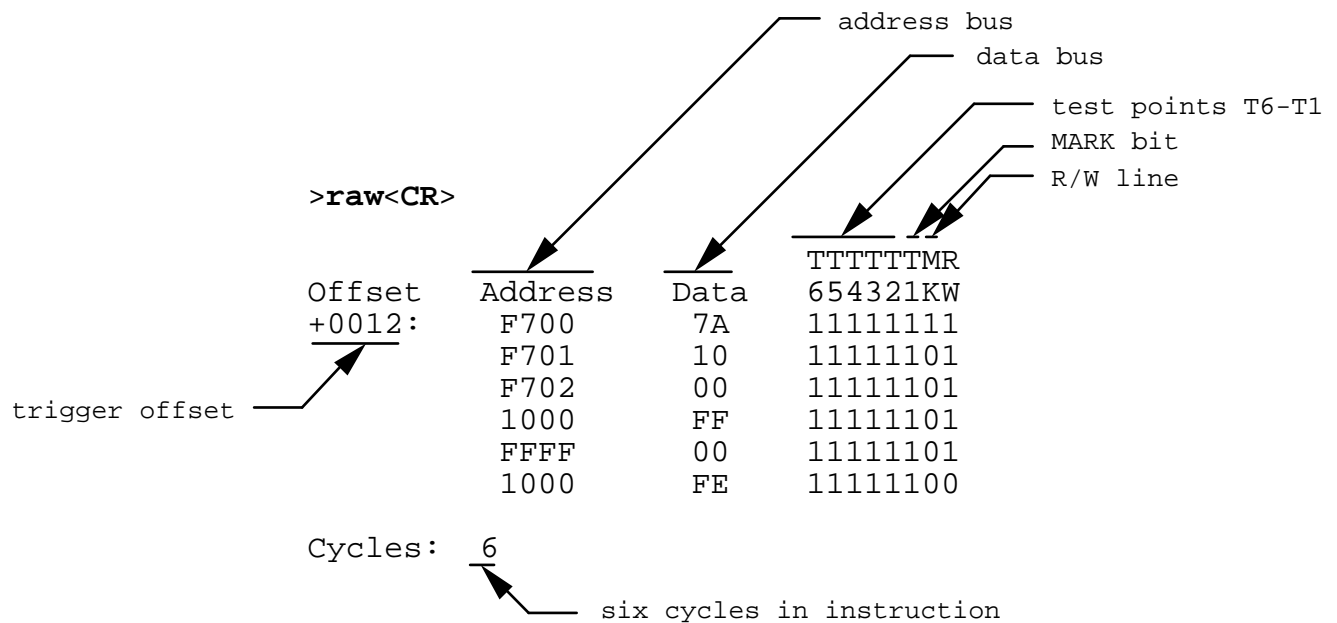
#### 4.5.4 Trace Displays

The capture buffer contents may be viewed once a trace has completed and the monitor program regains control of the CPU. This data remains in the capture buffer until one of the following events occurs:

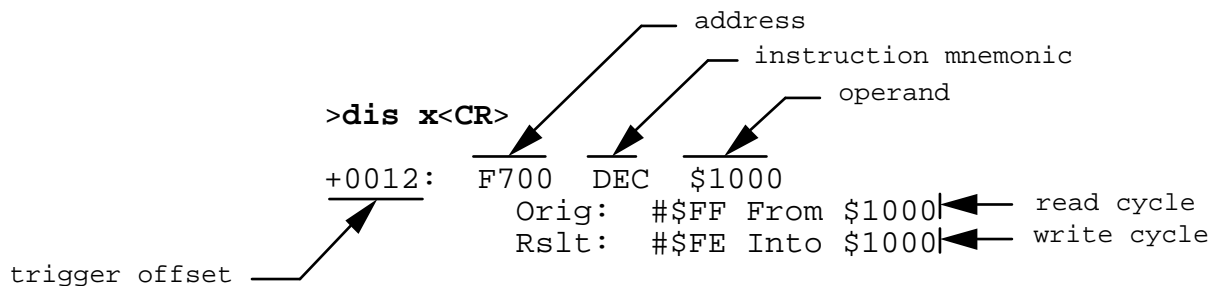
1. An MCU reset occurs
2. A trace set-up command is entered
3. The user program is executed again

Use the **DIS** and **RAW** commands to display the capture buffer contents. **DIS** and **RAW** display the same capture buffer data, but the data is displayed in different formats. The **DIS** command is only allowed with data captured from trace instruction execution mode. **DIS** shows the address and M68HC11 mnemonic that was executed, as well as any requested bus transaction data. The **RAW** command displays every cycle of data, with the address bus, data bus, and auxiliary byte values for each cycle.

Figures 4-5 and 4-6 show the equivalent instruction displayed from the logic analyzer by the **DIS** and **RAW** commands.



**Figure 4-5. RAW Command Display**



**Figure 4-6. DIS Command Display**

#### 4.5.5 Moving Through the Capture Buffer

Both the **DIS** and **RAW** commands display data at the trigger offset. After each instruction is displayed, the trigger offset is incremented so it points to the next instruction in the capture buffer.

The **SKIP** command moves the trigger offset forward or backward in the capture buffer from the current position. **SKIP** accepts a decimal number of instructions to skip, between -8192 and 8192, to move the trigger offset.

An alternative to the **SKIP** command is the **FIND O** command which moves the trigger offset to an absolute offset. **FIND O** accepts the desired trigger offset value as a decimal number between -8192 and 8192.

The **FIND** command has an alternate way of locating events in the capture buffer. An event is defined as the occurrence of a certain address bus, data bus, and auxiliary byte pattern during an MCU cycle. For instance, the first cycle of the instruction shown in Figures 4-5 and 4-6 is an event in which the address bus holds the value `$F700`, the data bus holds `$7A`, and the auxiliary byte is `$FF`. Use the **FIND** command and these parameters to locate the event:

```
FIND A F700 D 7A X FF
```

The **FIND** command begins searching from the current trigger offset forward in the capture buffer. **FIND** with a minus sign (-) parameter indicates a search backwards from the trigger offset. If a decimal number (<n>th) is appended to the **FIND** command line, the <n>th occurrence of an event may be located. Finally, any of the search parameters may be specified in a binary format if preceded by a percent (%) sign. A bit may be specified as “don’t care” by entering it as X. For instance, the following command locates the third occurrence of an event, looking backward in the capture buffer from the current trigger offset:

```
FIND A F700 D 7A X %XXXXXX11 -3
```

---

In the example, the auxiliary byte is specified in binary format, with the user test point bits T6-T1 specified as don't care. This prevents activity on on the test point input pins from affecting the results of the search. Only one address bus, data bus, and auxiliary byte value may be given to define an event, but all three of these parameters are not required.

#### 4.5.6 Timing Captured Events

The **TIME** command is incorporated in the monitor program to calculate the number of MCU cycles between two events captured in the capture buffer. The **TIME** command uses the same syntax rules as the **FIND** command, except that two events may be specified (separated with a colon). For instance, suppose the event shown in Figures 4-5 and 4-6 is captured more than once in the capture buffer because it is performed in a loop or within an interrupt service routine that is executed twice. To locate the first and second occurrence of the event shown in Figures 4-5 and 4-6, use the following **TIME** command:

```
TIME I F700 : I F700 2
```

The example demonstrates the use of the "I" instruction address parameter that defines the event of an instruction starting at the address specified. The calculated result of the **TIME** command is displayed for the user in MCU cycles and the trigger offset is moved to the instruction which contains the second event. The two events, the search direction, and <n>th occurrence may be different depending on the trigger offset position when the **TIME** command is issued. For example, if the trigger offset is positioned in between two events, use this command:

```
TIME I F700 -1 : I F700 1
```

This command searches backward in the capture buffer for the first event and positions the trigger offset there before searching forward for the second event.

#### 4.5.7 User Program Error Triggering

The logic analyzer detects an error during user program execution if the user program writes data into the upper 16K byte range of memory. Also errors are detected when an instruction is executed from outside of the upper 16K byte range of memory. The first of these events can occur due to a coding error in the user program, while the second may occur if the user program runs away. User program run away is often due to an improper stack operation in which an incorrect return address is loaded into the program counter during an RTS or RTI instruction.

The **FAULT** command is used to set-up a trace that triggers on one of the above error conditions. If the trigger condition occurs, the user program is interrupted and control is returned to the monitor program. The capture buffer then contains the user-program execution data leading up the the error, and the trigger offset is positioned at the user program instruction causing the error. An error message is displayed that describes the type of error.

---

#### 4.5.8 User Test Points

Six user-defined inputs (T1 – T6) are located on the logic analyzer connector P1. These are TTL compatible (0-5V) inputs that are pulled-up to 5V by 10K½ resistors on the logic analyzer. These inputs are captured in the logic analyzer auxiliary byte during user program traces. The state of these test points is stored and may be viewed using the **RAW** command.

The state of these inputs is captured at the same time that the data bus value is captured, requiring the test signal be valid and stable at the same time as the MCU data bus value. This prevents irrelevant test point data from being captured. Refer to the MC68HC11 MCU Advanced Information Data Sheet for external MCU address/data bus timing information.

#### 4.5.9 Logic Analyzer Debugging Examples

Debugging with the logic analyzer allows the user to view program execution. Alternately, when breakpoint debugging, register and memory contents must be pieced together from previous instruction execution. The logic analyzer is non-intrusive and so does not require monitor program intervention in the user program. While the single-step trace also allows viewing of program execution, single-step tracing requires monitor program intervention in the user program. The logic analyzer is especially useful for viewing the operation of interrupt intensive programs.

An example program is used throughout this section which is loaded into EVB2 RAM and starts at address \$F800. This program starts from reset, drives a target system while responding to keyboard input, and generates random pulses to drive LEDs, a stepper motor, and a liquid crystal display. The example program uses interrupts to perform these functions. Each section that follows demonstrates one of the debugging capabilities of the logic analyzer.

##### 4.5.9.1 Trace After Reset

The example in Figure 4-7 tests the vital steps in user program initialization. There are four principal steps:

1. Reset the EVB2 and check the MCU control register values (should contain the post-reset values)
2. Set up to trace after the user program's starting instruction
3. Run the user program
4. Disassemble the program from the top of the capture buffer



```

>reset<CR>                                Reset the EVB2

BUFFALO/GATECH 1.0 - Bit User's Fast Friendly Aid to Logical Operation

Register Start (J15): 0000
MCU RAM Range (J16): 1000 10FF
EEPROM Range:      0800 0FFF
User Program Range: D000 FFFF
Internal ROM Disabled
COP System Disabled
>after f800<CR>                            Setup to trace after starting instruction.
LA Set-up

>run<CR>                                    Execute user program from the reset vector address.

0 Pre-trigger Instructions
1887 Post-trigger Instructions
P-F905 Y-FD92 X-F929 A-04 B-00 C-80 S..... S-10FD
>dis x 10<CR>                               Extended disassemble the first ten instructions.
0000: F800 LDS  #$10FF
+0001: F803 LDAA #$10
+0002: F805 STAA $103D
      Data:  #$10 Into $103D
+0003: F808 BSR  $F819
      Low :  #$0A Into $10FF
      High:  #$F8 Into $10FE
+0004: F819 LDAA #$20
+0005: F81B STAA $02
      Data:  #$20 Into $0002
+0006: F81D LDAA $89
      Data:  #$FF From $0089
+0007: F81F STAA $26
      Data:  #$FF Into $0026
+0008: F821 LDAA #$C0
+0009: F823 STAA $24
      Data:  #$C0 Into $0024
>
  
```

**Figure 4-7. Trace After Reset**

#### 4.5.9.2 Trace About an Interrupt

If the user program fails in response to a particular interrupt, it is useful to capture the user program execution as the interrupt service routine is executed. The trace **ABOUT** function is effective because it captures data before and after the trigger. If the interrupt source can be wired to one of the six user test points, the interrupt latency (time delay before the MCU services the interrupt) can be measured. If the trace never triggers, this test determines if the interrupt is improperly set-up or disabled.

In Figure 4-8, a trace is set-up about the beginning of an interrupt service routine that services pulses generated by a rotary pulse generator. The active-low pulse is wired to test point T1.

<pre>&gt;about fa48&lt;CR&gt; LA Set-up</pre>	<p>Setup trace about interrupt service routine that starts at address \$FA48.</p>
<pre>&gt;run&lt;CR&gt;</pre>	<p>Execute user program from the reset vector address.</p>
<pre>919 Pre-trigger Instructions 936 Post-trigger Instructions P-F91D Y-FD92 X-F92B A-FE B-00 C-94 S..I.Z.. S-10FD</pre>	
<pre>&gt;skip -2&lt;CR&gt;</pre>	<p>Move trigger offset back two instructions.</p>
<pre>Trigger Offset Moved From : 0000 To : -0002</pre>	
<pre>&gt;dis x 3&lt;CR&gt;</pre>	<p>Extended disassemble three instructions.</p>
<pre>-0002: F817 BRA \$F80B -0001: F80B JSR \$F8F4         Low : #\$0E Into \$10FF         High: #\$F8 Into \$10FE         INTERRUPT</pre>	<p>Indicates beginning of interrupt stacking of CPU registers.</p>
<pre>registers.         PCL : #\$F4 Into \$10FD         PCH : #\$F8 Into \$10FC         IYL : #\$92 Into \$10FB         IYH : #\$FD Into \$10FA         IXL : #\$29 Into \$10F9         IXH : #\$F9 Into \$10F8         ACCA: #\$FC Into \$10F7         ACCB: #\$00 Into \$10F6         CCR : #\$80 Into \$10F5         V_HI: #\$FA From \$FFEC         V_LO: #\$48 From \$FFED</pre>	
<pre>0000: FA48 LDAA #\$02</pre>	<p>First instruction of interrupt service routine.</p>
<pre>&gt;time x%xxxxxx0xx - : i fa48&lt;CR&gt; interrupt</pre>	<p>Calculate time between T1=low and start of service routine.</p>
<pre>Trigger Offset Moved From : 0001 To : -0001 To : 0000</pre>	
<pre>Cycles Between Events: 19</pre>	
<pre>&gt;</pre>	

**Figure 4-8. Trace About an Interrupt**

### 4.5.9.3 Watch Interrupt Sequencing

The **ONLY** command V option captures the even interrupt vector addresses as each interrupt is fetched during user program execution. This feature does not provide timing information, but the sequencing of interrupts may be monitored. This command should be used if a specific interrupt is expected but is not serviced (the interrupt is improperly set-up or not enabled), or if the program does not execute properly after the first interrupt (an interrupt flag is improperly cleared after the interrupt is serviced). Using the **ONLY V** command is shown in Figure 4-9.

```

>only v<CR>                               Setup to capture fetches of even interrupt vectors.
LA Set-up

>run<CR>                                   Execute user program from the reset vector
address.

0 Pre-trigger Cycles
8191 Post-trigger Cycles
P-FB36 Y-FD92 X-F929 A-00 B-00 C-94 S..I.Z.. S-10F6
>raw 5<CR>                                 View first five cycles in raw bus cycle format.

Offset   Address   Data   TTTTTTMR
0000:    FFF0     FA    11111111
+0001:    FFE6     FC    11111111
+0002:    FFE0     FB    11111111
+0003:    FFDE     FB    11111111
+0004:    FFE6     FC    11111111

>find i ffec<CR>                           Find timer input capture 2 interrupt in capture
buffer.
Trigger Offset Moved From : +0005
To End Of Capture Buffer : +8191             Interrupt vector was not fetched (i.e. TOC2 interrupt
                                             is improperly set-up).
Trigger offset restored to previous position
>

```

**Figure 4-9. Watch Interrupt Sequencing**

#### 4.5.9.4 Time Between Interrupts

If an interrupt source produces interrupts faster than every 4.096 ms, then the 8K cycle capture buffer can hold execution data (including two or more occurrences of the interrupt service routine). The **TIME** command may then be used to determine the number of cycles, and thus the approximate interval between the actual interrupts (shown in Figure 4-10).

<code>&gt;before fcb7 n 100&lt;CR&gt;</code>	Trace before 100th execution of interrupt service routine at \$FCB7 (this allows the interrupts to stabilize after initialization).
LA Set-up	
<code>&gt;run&lt;CR&gt;</code>	
1841 Pre-trigger Instructions	
0 Post-trigger Instructions	
P-FCB9 Y-FD92 X-F929 A-40 B-00 C-90 S..I.... S-10F4	
<code>&gt;time i fcb7 -2&lt;CR&gt;</code>	Calculate time from trigger offset to previous start of
of	interrupt service routine.
Trigger Offset Moved From : 0000	
To : -0904	
Cycles Between Events: 4024	4024 cycles = 2.012 ms.
<code>&gt;&lt;CR&gt;</code>	Repeat previous command.
Trigger Offset Moved From : -0904	
To : -1815	
Cycles Between Events: 4024	Same as previous result, 2.012 ms.
<code>&gt;</code>	

**Figure 4-10. Time Between Interrupts**

#### 4.5.9.5 Trace Writes to Variables

Use the **ONLY** command with the W (write) option to verify that the user program is correctly writing data to the MCU control registers or the user program variables in the MCU RAM. Individual RAM or control register addresses may be specified with the command or the entire range of addresses may be traced as shown in Figure 4-11.

```

>only w 1000-10ff<CR>           Trace write cycles to address $1000 – $10FF.
LA Set-up

>run<CR>                         Execute user program.

0 Pre-trigger Cycles
8191 Post-trigger Cycles
P-F9EE Y-FD92 X-F929 A-04 B-00 C-84 S...Z.. S-10FC
>raw 5<CR>                       View first 5 cycles captured.

Offset   Address   Data   TTTTTTMR
654321KW
0000:    103D    10    11111110   Write to INIT register: remap MCU RAM to
$1000,                                     registers to $0000.
+0001:    10FF    0A    11111110
+0002:    10FE    F8    11111110
+0003:    1037    30    11111110   First variable initialization.
+0004:    1038    30    11111110

>find a 1001<CR>                 Locate first write to variable at $1001.

Trigger Offset Moved From : +0005
To : +0055

>raw<CR>                         View data.

Offset   Address   Data   TTTTTTMR
654321KW
+0055:    1001    00    11111110

>

```

**Figure 4-11. Trace Writes to Variables**

#### 4.5.9.6 Trace Faults

If the user program runs away during user-program execution and the reset switch (SW3) must be pressed to regain control of the MCU, the **FAULT** command may provide useful information for isolating the error (Figure 4-12). User-program runaway is usually caused by an incorrect stacking operation within either a subroutine or an interrupt service routine. The **FAULT** command monitors CPU execution and triggers if an instruction is executed below the upper 16K address range (\$C000 – \$FFFF). **FAULT** also triggers due to a write cycle into the upper 16K address range.

In some cases when user-program runaway occurs, the first instructions executed are still within the \$C000 – \$FFFF range but are not instructions in the user program. The CPU may eventually jump below \$C000, or a write instruction be executed which writes to the \$C000 – \$FFFF range and trigger the **FAULT** trace. In this case the trigger offset is pointing to irrelevant data in the capture buffer, and the capture buffer must be searched to locate the last instruction of the user program.

Anytime user-program runaway occurs, the user program may become corrupted by spurious writes into its address range. Reload user programs into the EVB2 anytime user-program runaway occurs.

>fault<CR>	Setup to trigger on a fault
LA Set-up	
>run<CR>	Execute the user program
Fault occurrence: instruction fetch out-of-range	
Trigger offset = previous instruction	
515 Pre-trigger Instructions	
0 Post-trigger Instructions	
P-0607 Y-0405 X-0203 A-01 B-00 C-11 ...I...C S-1107	
>dis x<CR>	View last instruction executed before fault
0000: F823 RTS	
High: #06 From \$10FD	
Low : #F8 From \$10FE	Note program counter loaded with \$06F8
End Of Capture Buffer	
>skip -4<CR>	Skip backwards 4 instructions
Trigger Offset Moved From : 0000	
To : -0004	
>dis x 5<CR>	Disassemble next 5 instructions
-0004: F817 JSR \$F81C	
Low : #1A Into \$10FF	
High: #F8 Into \$10FE	
-0003: F81C PSHA	
Data: #06 Into \$10FD	
-0002: F81D LDAA \$1001	
Data: #00 From \$1001	
-0001: F820 STAA \$1000	
Data: #00 Into \$1000	
0000: F823 RTS	Note incorrect stacking operation: stack push
with-	
High: #06 From \$10FD	in subroutine without a corresponding stack pull.
Low : #F8 From \$10FE	
End Of Capture Buffer	
>	

**Figure 4-12. Trace Faults**

---

#### 4.5.10 Logic Analyzer Self-Test Error Messages

When the logic analyzer self-test program is executed, error messages are displayed after a power-on reset. If you enter a logic analyzer command and a logic analyzer error condition exists, an error message is displayed. You cannot execute logic analyzer commands until: the error condition is fixed, a power-on reset is performed, and the self-test program executes successfully. Use the **RESET T** command to reset the MCU and execute self-tests.

The logic analyzer self-test error messages are as follows:

- EVB2LA Logic Analyzer Not Present — All self-tests failed
- EVB2LA Logic Analyzer Not Functioning XXXXXXXX — At least one self-test failed, consult Figure 2-13
- EVB2LA Not Available- EEPROM or Internal ROM mapped over EVB2 RAM — The ROMON bit in the CONFIG register is on, or internal EEPROM is mapped over the EVB2 RAM via CONFIG[7:4]

#### 4.5.11 Internal Read Visibility

Normally, data transfers on the internal MCU data bus from the control registers, internal RAM, and internal EEPROM to the CPU are invisible outside of the MCU. In order to capture these transfers, the logic analyzer makes use of the MC68HC11 E-Family internal read visibility (IRV) feature (via the HPRIO register). The monitor program automatically sets the IRV bit to 1 so that internal-MCU-read-cycle data is present on the MCU-external-data bus. The data is then captured by the logic analyzer.

The IRV feature is only available in MC68HC11E-series parts. When IRV is disabled or an MCU is used which does not contain the IRV feature, irrelevant data bus values are captured by the logic analyzer. Because the IRV feature is automatically disabled when using the **USER** command, irrelevant-data captures occur. To circumvent erroneous data captures, include these instructions at the beginning of the user program:

```
LDAA #35
STAA HPRIO;set IRV bit on
```

The label HPRIO must be defined for the current HPRIO control register address.

---

## 4.6 SOFTWARE PREPARATION

This section provides some hints for preparing user programs for downloading to the EVB2. The EVB2 emulates an M68HC11 MCU operating in single-chip mode to user software. However, several important exceptions are discussed regarding the EVB2.

### 4.6.1 RAM/Control Register Mapping

Following a normal reset into the monitor program, the locations of the internal MCU RAM and control registers are set by the monitor program. The monitor positions these resources independently starting at either address \$0000 or \$1000 (refer to paragraph 2.3.2.2). The user must set these jumpers because writes to the INIT register, which controls the locations of these resources, have no effect 64 cycles after reset. If you want the control registers and RAM locations to be different than the default locations, instructions which include the new location must occur early in the user program to write to the INIT register (located at address \$103D). These instructions will have no effect on the EVB2, but are required when the program is installed in the MCU in the application hardware.

### 4.6.2 Stack

In normal MCU applications, one of the first instructions in the user program should set the stack pointer to the top of the RAM area. For single-chip mode applications, this can only be the MCU internal RAM. User programs running on the EVB2 should follow this rule. However, to assist users wishing to test only a subroutine without running their program from the beginning, the monitor program contains a default user stack area at \$C000 – C014. This stack area is within the EVB2 RAM which is external to the MCU. This RAM will not exist in a single-chip mode application; therefore it should be used with caution.

### 4.6.3 Emulation ROM

The external EVB2 RAM provides a user-map at addresses \$D000 – FFFF which is used to emulate the ROM memory of an MCU in an application. User programs are typically downloaded into this area using an S-record format. User programs should be configured so that they originate at the same address as in their intended application. Programs intended for an MC68HC11A8 typically originate at \$E000, while programs intended for the larger MC68HC11E9 with 12K bytes of ROM originate at \$D000. If the program is designed to reside in the internal 2K byte EEPROM memory of the MC68HC811E2, it should originate at \$F800, and so on.

Users must remember that while the memory at addresses \$D000 – FFFF is RAM memory in the EVB2, this range will be ROM memory or undefined in an MCU application. Only constants and program instructions should be downloaded into the user-map.



#### **4.6.4 Interrupt Vectors**

An interrupt vector table including the reset vector should be included in all user programs. This table is expected by the MCU and is fixed at the upper 64 bytes of the memory map (\$FFC0 – FFFF). For each interrupt that is enabled by the user program, the corresponding two-byte interrupt vector should contain the address of a routine in the user program designed to service the interrupt. The reset vector should also contain the starting address of the user program.

#### **4.6.5 User Program Preparation**

User programs must be assembled into an S-record format to be downloaded into the EVB2. Numerous cross-assemblers are available for generating S-record files. The MCU Toolbox book (included in EVB2 shipping carton) provides a list of third party cross-assembler suppliers.

Note that freeware cross-assemblers are available from Motorola for the IBM-PC (or compatible), the Apple Macintosh family of computers, and the Unix operating system that generate the S-Record output file. Additional EVB/EVM/EVS software information can be obtained by calling the EVM hotline (512-891-EVMS).



## CHAPTER 5

### MONITOR COMMANDS

#### 5.1 INTRODUCTION

This section provides a detailed description of each BUFFALO/GATECH monitor command with examples of its use. For a complete listing of the BUFFALO/GATECH monitor program refer to the BUFFALO Monitor Program Listing Reference Manual, BUGATECH/AD1. Standard monitor commands are addressed first followed by the additional commands available with the M68HC11EVB2LA Logic Analyzer. Table 5-1 summarizes the standard BUFFALO command set while Table 5-2 lists the EVB2LA commands.

The following are valid BUFFALO command line inputs:

- Commands are shown in their full form, however only the minimum number of characters required to uniquely identify the command may be entered
- Commands and parameters are shown in both upper- and lower-case, and may be entered in any combination of upper- or lower-case letters
- Command parameters shown in brackets ([ ]) are optional
- Values shown in angle brackets (< >) represent variables
- Parameters preceding three periods (...) may be repeated
- Parameters separated by commas (,) are mutually exclusive

**Table 5-1. Standard BUFFALO Command Set**

Command	Description
ASM [<address>]	Assembler/disassembler (interactive)
BF <addr1> <addr2> [<data>]	Block fill memory with data
BREAK [-][<address>]...	Breakpoint set
BULK	Bulk erase internal EEPROM
BULKALL	Bulk erase EEPROM and CONFIG register
CALL [<address>]	Execute user subroutine
COPY <addr1> <addr2> [<dest>]	Move memory contents (synonym for MOVE)
ERASE	Bulk erase internal EEPROM (synonym for BULK)
GO [<address>]	Execute user program
HELP	Display BUFFALO commands
LOAD [T] [<host download cmnd>]	Load S-records into memory
MD [<addr1> [<addr2>]]	Display memory
MEM	Display memory command menu
MM [<address>]	Modify memory
MOVE <addr1> <addr2> [<dest>]	Move memory contents
NOBR	Remove all breakpoints
OFFSET [[-]<offset>]	Specify signed hex value for LOAD and VERIFY
PROCEED	Continue past breakpoint
RD	Display contents of user registers
RESET [T]	Perform MCU hardware reset
RM [P,Y,X,A,B,C,S [<data>]]	Modify user register
RUN	Execute user program from address in reset vector

**Table 5-1. Standard BUFFALO Command Set (continued)**

<b>Command</b>	<b>Description</b>
SB, DB <address> [\$, &, %]	Single-byte/double-byte real-time trace
SP, PC	Stack pointer/program counter real-time trace
SPEED [<baud>]	Set HOST port baud rate
STOPAT [<address>]	Single-step trace until stop address
TEST1,TEST2,TEST3	Execute external command
TM	Enter transparent mode
TRACE [<n>]	Single-step trace instructions
USER	Prepare for user-reset
VERIFY [T] [<host download cmd>]	Verify S-records against memory
XBOOT [<addr1> [<addr2>]]	Send data to another MCU in special bootstrap mode
[<address>]/ (Slash mark)	Modify memory (synonym for MM)
Question mark (?)	Display BUFFALO commands (synonym for HELP)

**Table 5-2. M68HC11EVB2LA Logic Analyzer Command Set**

Command	Description
<, >, ^	Move trigger offset to beginning, end, or trigger in buffer
ABOUT {address list} <sup>1</sup>	Setup to trace before and after trigger at any address in list
AFTER {address list} <sup>1</sup>	Trace after trigger at any address in list
BEFORE {address list} <sup>1</sup> [N <n>]	Trace before execution of trigger at any address in list
DIS [X] [<n>]	Disassemble instructions from buffer
FAULT	Setup to trace before a fault condition
FIND {event} <sup>2</sup> [-] [<n>]	Move trigger offset to <n>th occurrence of event in buffer
FIND O [-] <offset>	Move trigger offset
LA	Display EVB2LA commands help menu
ONLY [R,W,A,E] {address list} <sup>1</sup>	Trace only read, write, access, or EEPROM write cycles to trigger at any address in list
ONLY V [[-] <addr1> [<addr2>...]]	Trace only fetches of interrupt vectors
RAW [<n>]	Display instructions or cycles of bus-cycle data
SKIP [-] [<n>]	Move trigger offset forward or backward
TIME {event1} <sup>2</sup> [-] [<n1>] [: {event2} <sup>2</sup> [-] [<n2>]	Display cycle-count between two events in buffer
TIME O [-] <offset1> [: [-] <offset2>]	Display cycle-count between two offsets in buffer

<sup>1</sup>{address list} = <addr1> [[-] [<addr2>]]...

“<addr1> – <addr2>” represents a range of addresses

“<addr1> <addr2> <addr3>” represents distinct addresses

<sup>2</sup>{event} = [A, I [%] <address>] [D [%] <data>] [X [%] <aux>]

% = parameter in binary

## 5.2 MONITOR COMMANDS

This section describes the standard BUFFALO commands. For each command, the command line format is given with a description of the command and examples of its use. Within the example boxes, **bold** text represents text entered by the user. A carriage return is represented by a <CR>. Program monitor text examples are in this distinctive typeface. Explanatory comments appear to the right. Error messages along with possible explanations are also included for the appropriate command.

---

---

## ASM

## Assembler/Disassembler

### 5.2.1 Assembler/Disassembler

ASM [<address>]

Parameters:

<address > Assembler operation starting address. When <address> is not specified, the starting address defaults to \$E000.

The assembler/disassembler subsystem is an interactive line assembler and editor. Each source line is converted into the proper machine language code and stored into memory overwriting previous data on a line-by-line basis at the time of 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 terminal as *ILLOP* (illegal operation).

The syntax rules for the assembler are as follows:

- All numerical values must be in hexadecimal. No base designators (e.g. \$ = hex, % = binary, etc.) are allowed.
- One or more space or tab characters must separate the opcode and each operand.
- Any characters after a valid opcode and associated operands are ignored.

Addressing modes are designated as follows:

- Immediate addressing is designated by a number sign (#) preceding the address.
- Indexed addressing is designated by a one-byte offset followed by a comma followed by an X or Y.
- Direct versus extended addressing is determined by the length of the address operand (1 or 2 digits specifies direct, 3 or 4 specifies extended). Extended addressing may be forced by padding the address operand with leading zeros.
- Relative offsets for branch instructions are computed by the assembler. The valid operand for any branch instruction is an absolute address.

The following M68HC11 instruction pairs have the same opcode and disassembly displays the same mnemonic:

ASL and LSL display LSL

BCC and BHS display BCC

BCS and BLO display BCS



## ASM

## Assembler/Disassembler

The assembler accepts the following keypress sequence subcommands:

Subcommand	Description
control-A (CNTL-A) or period (.)	Exit assembler/disassembler subsystem
carriage return <CR>	Accept <data> & disassemble next instruction
control-J (CNTL-J) or plus (+)	Accept <data> & disassemble next address
- or caret (^)	Accept <data> & disassemble previous address
= or slash (/)	Accept <data> & disassemble same address

### EXAMPLES

<b>&gt;asm e000&lt;CR&gt;</b>		
E000 NOP CE 10 00	<b>&gt;ldx #1000&lt;CR&gt;</b>	Immediate addressing
E003 NOP A6 00	<b>&gt;ldaa 0,x&lt;CR&gt;</b>	Indexed addressing
E005 NOP 97 04	<b>&gt;staa 04&lt;CR&gt;</b>	Direct addressing
E007 NOP 09	<b>&gt;dex&lt;CR&gt;</b>	Inherent addressing
E008 NOP	<b>&gt;-&lt;CR&gt;</b>	Subcommand: previous address
E007 DEX 08	<b>&gt;inx&lt;CR&gt;</b>	
E008 NOP 8C 10 10	<b>&gt;cpx #1010&lt;CR&gt;</b>	
E00B NOP 26 F6	<b>&gt;bne e003&lt;CR&gt;</b>	Relative addressing
E00D NOP 39	<b>&gt;rts&lt;CR&gt;</b>	
E00E NOP	<b>&gt;.</b>	Subcommand: exit assembler
<b>&gt;</b>		

---

---

**ASM****Assembler/Disassembler**

<b>Error Messages</b>	<b>Possible Cause</b>
Bad argument	On entry, the parameter must be a hexadecimal address in assembler, the correct number of operands must be used.
Branch out of range	The branch must be within $\pm 128$ bytes of branch instruction.
Immed mode illegal	Immediate addressing mode not allowed for specified instruction.
Mnemonic not found	The specified opcode is not M68HC11.
rom-xxxx	Only RAM and EEPROM destination addresses are allowed.

# BF

# Block Fill

## 5.2.2 Block Fill

BF <addr1> <addr2> [<data>]

Parameters:

- <address1> Lower limit for fill operation.
- <address2> Upper limit for fill operation.
- <data> Fill pattern hexadecimal value. If <data> is not given, the default value is \$FF.

The **BF** command may only be used on RAM and EEPROM addresses. **BF** writes data one byte at a time. If the monitor program encounters an error while writing a data byte to a new location, the command is immediately aborted and an error message is displayed.

### EXAMPLES

>bf 0100 01ff 5a<CR>	Write value \$5A to addresses \$0100 – \$01FF.
>bf 8000 8040 ff<CR>	Attempt to write \$FF to addresses \$8000 – \$8040.
rom-8000 >	Error at address \$8000, command aborted.

Error Messages	Possible Cause
Bad argument	At least two addresses must be specified. No more than one <data> value may be specified. Only hexadecimal values are allowed.
rom-xxxx	Only RAM and EEPROM addresses may be specified.

## BREAK

## Breakpoint Set

### 5.2.3 Breakpoint Set

**BREAK** [-][<address>]...

Parameters:

- <address> Breakpoint is set at the parameter defined by <address>
- <address> <address> is removed from the table.

The **BREAK** command sets or removes breakpoints in the breakpoint table. If no parameters are specified, the breakpoint table contents are displayed. Several set or remove operations may be specified, separated by one or more spaces or tab characters. A maximum of four breakpoints may be set. The contents of the table are displayed after all table changes.

Define breakpoints before user program execution. When the **GO**, **CALL**, **RUN**, or **USER** commands are entered, the monitor program installs the breakpoint values into the user program. Encountering a breakpoint causes the user program to halt, and control to return to the monitor program. The instruction at the breakpoint address is not executed. When the EVB2LA is set-up for a trace capture, breakpoint values are not loaded into the user program.

Breakpoints place a software interrupt instruction (SWI) at the addresses specified in the breakpoint table. Therefore SWI instructions cannot be used in user programs because the software interrupt vector at address \$FFF6 is used by the monitor program. Breakpoints can only be set at locations in RAM or EEPROM.

### EXAMPLES

>break e005<CR>	Install breakpoint at address \$E005
E005 0000 0000 0000	
>break -e005 e017<CR>	Remove breakpoint at \$E005 and set one at \$E017
E017 0000 0000 0000	
>break -<CR>	Clear all breakpoints
0000 0000 0000 0000	
>	

---

---

**BREAK****Breakpoint Set**

<b>Error Messages</b>	<b>Possible Cause</b>
Bad argument	Parameters must be hexadecimal addresses or minus sign (-).
Full	Only four breakpoints may be set in the breakpoint table.
rom-xxxx	Breakpoints can only be set in RAM or EEPROM.

---

---

# BULK

# Bulk

## 5.2.4 Bulk

### BULK

The **BULK** command lets the user erase all MCU internal EEPROM locations. All EEPROM cells are written with the value \$FF, in effect erasing the cells. **BULK** works for all M68HC11 family members with internal EEPROM (both 512 byte and 2K byte EEPROM versions). The synonym for the **BULK** command is **ERASE**.

### NOTE

No erase verification message is displayed upon completion of the bulk EEPROM erase operation. User must verify erase operation by examining EEPROM locations using the **MM** or **MD** command.

### EXAMPLE

```
>bulk<CR>  
>
```

---

---

# BULKALL

# Bulkall

## 5.2.5 Bulkall

### BULKALL

The **BULKALL** command followed by an MCU reset lets the user erase all MCU internal EEPROM locations. All EEPROM cells are written with the value \$FF, in effect erasing the cells. To erase the CONFIG register, first enter the **BULKALL** command, then either enter a **RESET** command or press the RESET switch (SW3). Alternately, if the **BULKALL** and **RESET T** commands are executed sequentially, all EEPROM cells are erased but not the CONFIG register.

### NOTE

No erase verification message is displayed upon completion of the bulk EEPROM erase operation. User must verify erase operation by examining EEPROM locations using the **MM** or **MD** command.

### EXAMPLES

```
>bulkall<CR>
CONFIG programmed, reset to implement
>
```

## CALL

## Execute User Subroutine

### 5.2.6 Execute User Subroutine

CALL [<address>]

Parameter:

<address> Execute a user subroutine at <address>. If <address> is not defined, the current user P-register (program counter) value is used. When writing the user subroutine, the subroutine must end with a return from subroutine (RTS) instruction.

Prior to user program execution, the current values of the user registers are loaded into the CPU registers and breakpoints in the breakpoint table are set (if the EVB2LA is not set-up). The X bit of the user C-register (condition code register) is cleared prior to execution of the user program. The monitor program halts execution and the user program executes until one of these conditions occurs:

1. An RTS instruction is executed returning to the monitor
2. A key is pressed on the terminal device
3. A breakpoint is encountered
4. The EVB2LA terminates a trace capture

When the monitor program regains control, the current CPU registers are saved in the user registers and any breakpoints are removed from the user program. The user register values are displayed and the EVB2LA status is displayed if it is present.

### EXAMPLES

```

>call f819<CR>                                Call subroutine at $F819 and return to monitor
P-F819 Y-FD92 X-105B A-00 B-87 C-94 S..I.Z.. S-C014
>call e000<CR>                                Call program at $E000 and press key to return
                                           To monitor
P-EA1D Y-FD92 X-0064 A-00 B-76 C-80 S..... S-00FB
>

```

Error Messages	Possible Cause
Bad argument	Only one hexadecimal address parameter is allowed.
rom-xxxx	The EVB2 cannot be used to directly debug programs in ROM (refer to paragraph 4.4.6).



# GO

# Execute User Program

## 5.2.7 Execute User Program

GO [<address>]

Parameter:

<address> Execute a user program at <address>. If <address> is not defined, the current user P-register (program counter) value is used.

Prior to user program execution, the current values of the user registers are loaded into the CPU registers and breakpoints in the breakpoint table are set (if the EVB2LA is not set-up). The X bit of the user C-register (condition code register) is cleared prior to execution of the user program. The monitor program halts execution and the user program executes until one of these conditions occurs:

1. A key is pressed on the terminal device
2. A breakpoint is encountered
3. The EVB2LA terminates a trace capture

When the monitor program regains control, the current CPU registers are saved in the user registers and any breakpoints are removed from the user program. The user register values are displayed and the EVB2LA status is displayed if it is present.

### EXAMPLES

```

>go e000<CR>           Execute program at $E000, press key to return
                        to monitor
P-E017 Y-FD92 X-F929 A-FC B-00 C-80 S..... S-00FF
>go e000<CR>           Execute program at $E000 and hit a breakpoint
                        at $F819
P-F819 Y-FD92 X-F929 A-10 B-00 C-80 S..... S-00FD
>
    
```

Error Messages	Possible Cause
Bad argument	Only one hexadecimal address parameter is allowed.
rom-xxxx	The EVB2 cannot be used to directly debug programs in ROM (refer to paragraph 4.4.6).

## HELP

## Primary Command Menu

### 5.2.8 Primary Command Menu

#### HELP

Show the primary BUFFALO command help menu.

Refer also to the **MEM** command to display commands related to memory operations and the **LA** command for the EVB2LA. The synonym for the **HELP** command is a question mark (?).

#### EXAMPLE

```
>help<CR>
***** PRIMARY HELP MENU *****
ASM [<addr>]      :line asm/disasm, [CR] :next instr, [CTLA,.] :quit
                  [/,=] :same addr, [^,-] :prev addr, [+ ,CTLJ] :next addr
BR [-][<addr>]    :set breakpoints, [-] : remove breakpoints
CALL [<addr>]     :call subroutine
GO [<addr>]       :execute code at addr
LOAD T or LOAD <host dwnld command>: load or verify S-record
OFFSET [-]<val>   :offset for download
PROCEED          :continue past breakpoint
RESET [T]        :reset processor, [T] :self-test enabled
RUN              :execute code starting at reset vector address
SPEED [<baud>]   :set host port baud rate
STOPAT <addr>    :single-step trace until addr
TM               :transparent mode (CTLA = exit, CTLB = send brk)
TRACE [<n>]      :single-step trace n instructions
USER             :wait for user reset
VERIFY T or VERIFY <host dwnld command>: verify S-record
HELP            :this screen
LA               :logic analyzer help menu
MEM             :memory options help menu

[CTLS,CTLW]:wait          [CTLX,DEL]:abort          [CR]:repeat last cmd
*****
>
```

---

---

## LOAD

## Load S-Records

### 5.2.9 Load S-Records into Memory

LOAD [<host download command>]

LOAD T

Parameters:

<host download command> Load S-records into memory from the HOST port (P5). The <host download command> parameter defines a download command to be executed by the host computer. The LOAD [<host download command>] is issued from the terminal device and tells the host computer to download S-record files. If no parameter is given, the **LOAD** command performs as a **TM** command.

T Load S-records into memory from the TERMINAL port (P6)

The monitor program waits for S-record data from the specified port (refer to Appendix A for S-record information). If all S-records are correctly received during **LOAD**, the message done is displayed on the terminal device. If an error is encountered during **LOAD**, the monitor program quits writing additional S-record information into memory, and waits for the S9 termination record. An error message is then displayed indicating the specific error encountered.

The **LOAD** command can only be used for downloading S-records into RAM or EEPROM. The current offset value as specified by the **OFFSET** command is added to all S-record addresses before the data is written into memory. The **LOAD** command will not write to destination address range \$C000 – \$C0FF, which is reserved for monitor variables.

Pressing **CNTL-A** or **CNTL-X** aborts the **LOAD** command. The character sequence <host download command> may not contain the slash (/) character because it is a BUFFALO command.

### EXAMPLES

>load cat program.s19<CR> cat program.s19	Load from HOST port command HOST's download command is "cat program.s19"
done >load t<CR>	Load from TERMINAL port command S-record file sent from terminal device
done >	

---

---

**LOAD****Load S-Records**

<b>Error Messages</b>	<b>Possible Cause</b>
BUFFALO memory conflict	The destination address range \$C000 – \$C0FF is reserved.
chksum error	At least one S-record checksum test failed (indicates an attempt to load data into EEPROM without using hardware handshaking, which is not possible above 300 baud).
xxxx-does not verify	Only RAM and EEPROM destination addresses are allowed.

# MD

# Memory Display

## 5.2.10 Memory Display

MD [<addr1> [<addr2>]]

Parameters:

- <addr1> Display memory contents beginning at address <addr1> in lines of 16 bytes. If no address is given, memory display continues from the end of the last **MD** command or the address of the last **MM** command.
- <addr2> Display memory contents ending at address <addr2>. If <addr2> is not given, eight lines of 16 bytes are displayed.

Each memory display is formatted with column headings designating the address of each of the 16 bytes per line. The starting address of each line is displayed in the left column, and the ASCII character equivalent for each memory byte is displayed in a block on the right. Bytes corresponding to non-printable characters are displayed as a period.

### EXAMPLES

```

>md 00 3f<CR>
      x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 xA xB xC xD xE xF 0123456789ABCDEF
0000 00 FF 03 00 00 80 06 00 20 00 00 00 00 00 CA B7 .....m.
0010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
0020 00 00 00 F8 00 C0 00 00 05 00 00 07 00 00 C0 00 .....
0030 80 00 00 00 00 00 FF FF FF 93 FF 00 35 10 00 1F .....5...
>md<CR>
      x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 xA xB xC xD xE xF 0123456789ABCDEF
0040 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F @ABCDEFGHJKLMNO
0050 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F PQRSTUVWXYZ[\]^_
0060 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F `abcdefghijklmnop
0070 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F pqrstuvwxyz.....
0080 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F .....
0090 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F .....
00A0 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF .....
00B0 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF .....
>
    
```

Error Message	Possible Cause
Bad argument	Only hexadecimal addresses are allowed. No more than two parameters are allowed.

## MEM

## Memory Command Menu

### 5.2.11 Memory Command Menu

#### MEM

Show the BUFFALO memory commands help menu.

Refer also to the **HELP** command to display primary commands and the **LA** command for commands associated with the EVB2LA.

#### EXAMPLE

```

>mem<CR>

***** MEMORY OPTIONS MENU *****
BF <addr1> <addr2> [<data>] :block fill memory
MD [<addr1> [<addr2>]]      :memory display
MM [<addr>] or [<addr>]/    :memory modify, [CR,.] :quit, [/,=] :same addr,
                           [^,-,CTLH] :prev addr, [+ ,CTLJ,SPACE] :nxt addr
                           <addr>O :compute offset to addr
MOVE <s1> <s2> [<d>]       :block move memory
RM [P,Y,X,A,B,C,S] [<val>] :register modify
RD                          :register display
SB,DB <addr> [$,&,%]       :single- or double-byte trace $=hex &=dec %=bin
PC,SP                      :program counter trace, stack pointer trace
XBOOT [<addr1> [<addr2>]]  :talk out SCI to hcl1 in boot mode

BULK                        :erase EEPROM
BULKALL                    :erase EEPROM and CONFIG
*****
>

```

# MM

# Memory Modify

## 5.2.12 Memory Modify

MM [<address>]  
 [<address>]/

Parameters:

<address>    Modify the memory contents at <address>. If <address> is not defined, the address of the last **MM** command or the end of the last **MD** command is used. The synonym for the **MM** command is a slash mark (/).

The memory contents at <address> is displayed with a prompt for a new value. The format for the new data entry and the subcommands accepted are:

Data Field	Subcommand	Description
[<data>]	control-X (CNTL-X) or period (.)	Abort
[<data>]	carriage return (<CR>)	Accept <data> and exit
[<data>]	control-J (CNTL-J), spacebar, or plus (+)	Accept <data> & modify next address
[<data>]	control-H (CNTL-H), minus (-), or caret (^)	Accept <data> & modify previous address
[<data>]	slash (/) or =	Accept <data> & modify same address
[<address>]	O	Compute <address> offset

Only one-byte hexadecimal values (two ASCII digits) are allowed for <data>. If more than two digits are entered, only the last two are taken as <data>. For the **O** subcommand, four digits may be entered to calculate the offset between the current address and <address>. If <address> is more than ±128 bytes away from the current address, an error message is displayed.

**MM** may only be used to change the contents of RAM and internal EEPROM cells.

**MM****Memory Modify****EXAMPLES**

<code>&gt;mm 10&lt;CR&gt;</code>	Modify address \$10
<code>0010 FF 33 (SPACE) FF 5a&lt;CR&gt;</code>	Write \$33 to address \$10 and \$5A to address \$11
<code>&gt;10/</code>	Modify address \$10
<code>0010 33 34/</code>	Write \$34 to address \$10 and modify same address
<code>0010 34 35&lt;CR&gt;</code>	Write \$35 to address \$10
<code>&gt;</code>	

<b>Error Messages</b>	<b>Possible Cause</b>
Bad argument	Only one hexadecimal address parameter is allowed.
Command?	An incorrect subcommand character was entered.
Too Long	With the O parameter, the offset to <address> is greater than $\pm 128$ bytes
rom-xxxx	Only RAM and EEPROM destination addresses are allowed.



# MOVE

# Move Memory

## 5.2.13 Move Memory

MOVE <addr1> <addr2> [<dest>]

Parameters:

- <addr1><addr2> Copy the memory contents between addresses <addr1> and <addr2> to the destination address range starting at <dest>.
- <dest> First address of the destination address. If <dest> is not defined, the address <addr1>+1 is used.

The **MOVE** command copies data from any type of memory, but can only write to RAM and internal EEPROM. **MOVE** copies data one byte at a time. If the monitor program encounters an error while writing a data byte to a new location, the command is immediately aborted and an error message is displayed. The synonym for the **MOVE** command is **COPY**.

### EXAMPLES

```

>move e000 e100 f000<CR>          Copy data at $E000 – $E100 to $F000 – $F100
>move e000 e100 8000<CR>          Copy data at $E000 – $E100 to $8000 – $8100
rom-8000                            Error writing to $8000, command aborted
>
    
```

Error Messages	Possible Cause
Bad argument	Only three parameters are allowed. Only hexadecimal addresses are allowed.
rom-xxxx	The EVB2 cannot be used to directly debug programs in ROM (refer to paragraph 4.4.6).

---

---

## NOBR

## Remove Breakpoints

### 5.2.14 Remove Breakpoints

NOBR

The **NOBR** command removes all breakpoints from the breakpoint table.

Same as **BREAK -**.

### EXAMPLE

```
>nobr<CR>
0000 0000 0000 0000
>
```

## OFFSET

## Offset Value

### 5.2.15 Offset Value

OFFSET [[-]<offset>]

Parameters:

- <offset> Specify a signed hexadecimal value <offset> to add to all S-record address values when using the **LOAD** and **VERIFY** commands. If no <offset> is entered, the current offset value is displayed.
- The minus sign preceding the offset subtracts the signed hexadecimal value of <offset> from all S-Record addresses.

Every S-record contains a 16-bit address field which specifies the destination address where the S-record will begin loading data in the 64K byte memory map. The **OFFSET** command is used to alter the destination address of all S-records as received via the **LOAD** and **VERIFY** commands. The offset value is maintained until the next MCU reset.

The **OFFSET** command is useful for loading S-Records containing assembled code destined for internal EEPROM at \$B600 – \$B7FF at normal baud rates when hardware handshaking is not being performed. An offset of \$4000 may be specified to load the data into RAM at \$F600 – \$F7FF, and then the command **MOVE F600 F7FF B600** places the data at \$B600 – \$B7FF.

### EXAMPLES

>offset<CR>	Display current offset value
0000	
>offset 2000<CR>	Add \$2000 to all S-Record addresses
2000	
>offset -2000<CR>	Subtract \$2000 from all S-Record addresses
E000	
>	

Error Message	Possible Cause
Bad argument	Only one hexadecimal parameter is allowed (with or without a minus sign preceding it).

## PROCEED

## Continue Past Breakpoint

### 5.2.16 Continue Past Breakpoint

#### PROCEED

Continue user program execution past a breakpoint. The current user P-register (program counter) is used as the starting address of execution.

Prior to user program execution, the current values of the user registers are loaded into the CPU registers. A single-step trace executes one instruction in the user program (refer to paragraph 4.4.1.3). Breakpoints in the breakpoint table are set in the user program after control returns to the monitor program. The X bit of the user C-register (condition code register) is cleared before execution of the user program. The monitor program halts execution and the user program executes until one of these conditions occurs:

1. A key is pressed on the terminal device
2. A breakpoint is encountered

When the monitor program regains control of the MCU, the current CPU registers are saved in the user registers and breakpoints are removed from the user program. The user register values are displayed and the EVB2LA status is displayed if it is present.

**PROCEED** is similar to **GO** except that the single-step trace is used to execute the first instruction (past the breakpoint) and then a **GO** command is issued. **PROCEED** cannot be used if the EVB2LA is set-up because the single-step trace method is used.

#### EXAMPLE

	A breakpoint has been encountered at \$F816
>proceed<CR>	Continue past breakpoint
P-F91B Y-FD92 X-F929 A-FC B-00 C-90 S..I.... S-C014	
>	Another breakpoint encountered at \$F91B

---

---

**PROCEED****Continue Past Breakpoint**

<b>Error Messages</b>	<b>Possible Cause</b>
ILLOP	The instruction to be executed is not a legal M68HC11 opcode.
LA Set-up	The EVB2LA is set-up for a trace (entering <b>RESET</b> clears the logic analyzer trace).
rom-xxxx	The EVB2 cannot be used to directly debug programs in ROM (refer to paragraph 4.4.6).

---



---

**RD**
**Register Display****5.2.17 Register Display**

## RD

Display the contents of the user registers. The contents of each user register is displayed with the corresponding register name. Each user register corresponds to one of the CPU registers as shown below.

User Register	CPU Register
P	program counter
Y	index register Y
X	index register X
A	accumulator A
B	accumulator B
C	condition code register
S	stack pointer

The user registers are loaded into the CPU registers prior to execution of a user program via the **CALL**, **GO**, **PROCEED**, **TRACE**, and **STOPAT** commands. The X bit in the user C-register (condition code register) is cleared before execution of the user program. Upon returning to the monitor program, the new CPU register values are stored in the user registers.

Additionally, the C-register is displayed bitwise. The single-letter abbreviation for each bit of the condition code register is displayed for bits with value 1, while a period is displayed for bits with value 0.

**EXAMPLE**

```
>rd<CR>
P-F9EE Y-FD92 X-F929 A-00 B-00 C-9C S..INZ.. S-10FC
>
```

---

---

## RESET

## MCU Hardware Reset

### 5.2.18 MCU Hardware Reset

RESET [T]

Parameter:

T EVB2 hardware self-tests are performed after reset.

The **RESET** command performs a hardware reset of the MCU. The **RESET** command serves the same function as pressing the RESET switch (SW3), driving the MCU RESET pin low. This occurs approximately 2 milliseconds after you issue the command. After a reset, all MCU internal registers are changed to their pre-defined reset values as specified in the MCU Data Sheet. If the USER switch (SW2) is not pressed, the monitor program gains control of the MCU and displays the BUFFALO startup screen.

If the T parameter is specified, the EVB2 power-on RAM test is performed, and if present, the EVB2LA self-test is executed. Additionally, the breakpoint table is cleared and the host port baud rate is reset to 9600 baud. The T parameter is useful for aborting unwanted CONFIG register modifications which require a **RESET** to implement.

### EXAMPLES

```
>reset<CR>
BUFFALO/GATECH 1.0 - Bit User's Fast Friendly Aid to Logical Operation

Register Start (J15): 1000
MCU RAM Range (J16): 0000 01FF
EEPROM Range:      B600 B7FF
User Program Range: D000 FFFF
Internal ROM Disabled
COP System Disabled
>reset t<CR>
BUFFALO/GATECH 1.0 - Bit User's Fast Friendly Aid to Logical Operation

Register Start (J15): 1000
MCU RAM Range (J16): 0000 01FF
EEPROM Range:      B600 B7FF
User Program Range: D000 FFFF
Internal ROM Disabled
COP System Disabled
EVB2 RAM test passed
EVB2LA Logic Analyzer Functioning
>
```

---

---

**RESET****MCU Hardware Reset**

<b>Error Message</b>	<b>Possible Cause</b>
Bad argument	T is the only parameter allowed.



# RM

# Register Modify

## 5.2.19 Register Modify

RM [p,y,x,a,b,c,s [<data>]]

Parameters:

p,y,x,a,b,c,s Each user register corresponds to one of these CPU registers:

user register	CPU register
P	program counter
Y	index register Y
X	index register X
A	accumulator A
B	accumulator B
C	condition code register
S	stack pointer

<data> Modify the specified user register to the hexadecimal value <data>. If <data> is not given, the user is prompted for a new value. If a register name is not given, the user is prompted for the P-register value.

The user registers are loaded into the CPU registers before user program execution using the **CALL**, **GO**, **PROCEED**, **TRACE**, and **STOPAT** commands. The X bit in the user C-register (condition code register) is cleared prior to user program execution. Upon returning to the monitor program, the new CPU register values are stored in the user registers.

If the <data> value is specified on the **RM** command line, the specified register is modified and the new user register contents are displayed. If <data> is not given, the current value of the specified register is displayed and a prompt is provided for a new value. These subcommands are accepted:

Subcommand	Description
carriage return (<CR>)	Accept new register data and exit
spacebar (SPACE)	Accept new register data and open the next register

**RM****Register Modify****EXAMPLES**

<pre>&gt;rm c 99&lt;CR&gt; P-F9EE Y-FFFF X-F929 A-00 B-00 C-99 S..IN..C S-C014  &gt;rm x&lt;CR&gt; P-F9EE Y-FFFF X-F929 A-00 B-00 C-99 S..IN..C S-C014 X-F929 4444&lt;CR&gt; &gt;</pre>	<pre>Modify the C-register to \$99 Modify the X-register Change X-register value to \$4444</pre>
---	--

Error Message	Possible Cause
Bad argument	The only valid register names are P, Y, X, A, B, C, or S. Only one hexadecimal <data> value is allowed.

# RUN                      Execute User Program from Reset Vector

## 5.2.20    Execute User Program from Reset Vector

### RUN

Execute a user program from the address in the reset vector (\$FFFE – \$FFFF). The RUN command executes identically to the GO command except that the user program start address is loaded from the user reset vector.

Prior to user program execution, the current values of the user registers are loaded into the CPU registers and breakpoints in the breakpoint table are set (if the EVB2LA is not set-up). The X bit of the user C-register (condition code register) is cleared before user program execution. The monitor program halts execution and the user program executes until one of these conditions occurs:

1. A key is pressed on the terminal device
2. A breakpoint is encountered
3. The EVB2LA terminates a trace capture

When the monitor program regains control, the current CPU registers are saved in the user registers and the breakpoints are removed from the user program. The user register values are displayed and the EVB2LA status is displayed if it is present.

### EXAMPLE

```

>run<CR>                               Execute program, press key to return to monitor
P-E017 Y-FD92 X-F929 A-FC B-00 C-80 S..... S-00FF
>
```

Error Message	Possible Cause
rom-xxxx	The EVB2 cannot be used to directly debug programs in ROM (refer to paragraph 4.4.6).

---



---

## SB/DB      Single-Byte/Double-Byte Real-Time Trace

### 5.2.21 Single-Byte/Double-Byte Real-Time Trace

SB <addr> [\$, &, %]

DB <addr> [\$, &, %]

Parameters:

- <addr>      Execute a user program from the address in the reset vector (\$FFFE – \$FFFF) and begin single-byte or double-byte real-time tracing of the contents at <addr>.
- \$, &, %      Specifies the numeric base for the display (\$ = hexadecimal, & = decimal, and % = binary). The default display format is hexadecimal.

Real-time tracing interrupts the user program with a non-maskable interrupt (XIRQ) once every 0.25 seconds and displays the requested data on the terminal device. The displayed data may be the contents of any address in the MCU memory map; however, reading the MCU control registers PIOC, SPSR, or SPCR may affect execution of the user program. The length of the user program interruption needed to display the data value has been minimized, but this process in itself may affect execution of the user program.

Prior to user program execution, breakpoints in the breakpoint table are set. The X bit of the condition code register is cleared before user program execution. The monitor program halts execution and the user program executes until one of these conditions occurs:

1. A key is pressed on the terminal device
2. A breakpoint is encountered

When the monitor program regains control of the MCU, the current CPU registers are saved in the user registers and the breakpoints are removed from the user program.

### EXAMPLE

```
>db fe<CR>
```

```
Tracing address 00FE --> $ F817
```

```
Start real-time double-byte trace at $00FE with
hexadecimal display
Terminal screen is updated every 0.25 sec.
```

---

---

## SB/DB      Single-Byte/Double-Byte Real-Time Trace

Error Messages	Possible Cause
Bad argument	Only one hex. address parameter is allowed followed by %, &, or \$.
rom-xxxx	The EVB2 cannot be used to directly debug programs in ROM (refer to paragraph 4.4.6).
WARNING: reading this address may affect execution of your program	PIOC, SPSR, or SPCR register specified as <addr>.

## SP/PC

## Stack Pointer/Program Counter Real-Time Trace

### 5.2.22 Stack Pointer/Program Counter Real-Time Trace

SP  
PC

Execute a user program from the address in the reset vector (\$FFFE – \$FFFF) and begin real-time tracing of the user stack pointer or program counter. The display format is hexadecimal.

Real-time tracing interrupts the user program with a non-maskable interrupt (XIRQ) once every 0.25 seconds and displays the requested data on the terminal device. The length of the user program interruption needed to display the data value has been minimized, but this process may affect user program execution.

Prior to user program execution, breakpoints in the breakpoint table are set in the user program. The X bit of the condition code register is cleared before user program execution. The monitor program halts execution and the user program executes until one of these conditions occurs:

1. A key is pressed on the terminal device
2. A breakpoint is encountered

When the monitor program regains control, the current CPU registers are saved in the user registers and the breakpoints are removed from the user program.

### **EXAMPLE**

<code>&gt;sp&lt;CR&gt;</code>	Start real-time stack pointer trace
Tracing stack pointer --> \$ 00FD	Terminal screen is updated every 0.25 sec.

Error Messages	Possible Cause
Bad argument	No additional parameters are permitted.
rom-xxxx	The EVB2 cannot be used to directly debug programs in ROM (refer to paragraph 4.4.6).

## SPEED

## Set Host Port Baud Rate

### 5.2.23 Set Host Port Baud Rate

SPEED [<baud>]

Parameters:

<baud> Set the HOST port baud rate to <baud>. Valid <baud> values are 300, 1200, 2400, 4800, 9600, and 384 (The <baud> entry 384 specifies the baud rate 38,400). If <baud> is not given, the current HOST port baud rate is displayed.

The monitor program sets the HOST port baud rate to 9600 baud at power-on. When a new value is specified, it is maintained until the next EVB2 power-on reset or a **RESET T** command is issued.

### EXAMPLES

>speed<CR>	Display current HOST port baud rate
host port speed 9600	
>speed 1200<CR>	Set HOST port baud rate to 1200 baud
host port speed 1200	
>	

Error Message	Possible Cause
Bad argument	Only the valid baud rate values 300, 1200, 2400, 4800, 9600, and 38,400 are accepted.

## STOPAT

## Single-Step Trace

### 5.2.24 Single-Step Trace

STOPAT [<address>]

Parameters:

<address> Single-step trace from the current user P-register (program counter) address through the instruction at <address>. If <address> is zero or undefined, one step is executed.

The address and disassembled user instruction are displayed prior to execution. Each user instruction is executed individually and CPU control returns to the monitor program after each user instruction. The new values of the user registers are displayed after the instruction is executed.

The single-step trace places a software interrupt (SWI) instruction in the user program. The monitor program places an SWI after the instruction to be executed. The SWI instruction returns MCU control to the monitor program (refer to paragraph 4.4.1.3). This requires that all user programs reside in RAM or internal EEPROM to use the **PROCEED**, **STOPAT**, and **TRACE** commands. **STOPAT** execution is not in real time which may affect the operation of some user programs.

#### NOTE

Do not include SWI instructions in user programs. An SWI may return control to the monitor program at an inopportune time.

If <address> is not an instruction opcode address in the user program, the **STOPAT** command may continue indefinitely. Single-step tracing can be aborted by pressing **CNTL-A** or **CNTL-X**.

If **STOPAT** attempts to execute an illegal M68HC11 opcode, an error message is displayed and further execution is aborted. Additionally, single-step tracing is not allowed when the EVB2LA is set-up to capture a trace.

#### EXAMPLE

<b>&gt;stopat f805&lt;CR&gt;</b>		User P-register value is \$F800	
F800 LDS #10FF	P-F803	Y-FFFF X-4444	A-90 B-00 C-90 S..I.... S-10FF
F803 LDAA #10	P-F805	Y-FFFF X-4444	A-10 B-00 C-90 S..I.... S-10FF
F805 STAA \$103D	P-F808	Y-FFFF X-4444	A-10 B-00 C-90 S..I.... S-10FF
<b>&gt;</b>			



---

---

**STOPAT****Single-Step Trace**

<b>Error Messages</b>	<b>Possible Cause</b>
Bad argument	Only one hexadecimal address parameter is allowed.
ILLOP	The instruction to be executed is not a legal M68HC11 opcode.
LA Set-up	The EVB2LA is set-up for a trace (entering RESET clears the logic analyzer trace).
rom-xxxx	The EVB2 cannot be used to directly debug programs in ROM (refer to paragraph 4.4.6).

---

---

## TEST1/TEST2/TEST3

## Execute External Command

### 5.2.25 Execute External Command

TEST1 [<parameters>]

TEST2 [<parameters>]

TEST3 [<parameters>]

Parameters:

<parameters> User-defined parameters.

The **TEST1**, **TEST2**, and **TEST3** commands give you the ability to write and test new BUFFALO commands without having to re-assemble the entire monitor program.

The external commands are treated identically to internal commands except that the command's address is found in RAM as opposed to the monitor command table. The addresses of the external commands are found at these addresses:

**TEST1**    \$C100

**TEST2**    \$C102

**TEST3**    \$C104

Individual commands are treated as subroutines of the monitor program. To return to the monitor command line, a return from subroutine (RTS) instruction must be executed at the end of the external command.

The RAM locations for the **TEST1**, **TEST2**, and **TEST3** command addresses are not protected from the **LOAD** command as monitor variables. An external command may be written and downloaded into EVB2 RAM along with the address of the command at the proper RAM location as shown above.

### EXAMPLE

```
>test1<CR>
This output is from an external command
>
```

### Error Messages

Various, depending on the user-defined external command.

---

---

**TM****Transparent Mode****5.2.26 Transparent Mode****TM**

In transparent mode, the EVB2 TERMINAL port and HOST port are effectively connected for direct communication between the terminal device and the HOST device. All data sent by either device is ignored by the EVB2 except for the following subcommands from the TERMINAL port:

CNTL-A Exit transparent mode

CNTL-B Send an RS-232 BREAK signal to the HOST

In order for characters sent from the terminal device to be displayed in transparent mode, the HOST device must echo the data sent to it back to the EVB2.

**EXAMPLES**

>tm<CR>	Enter transparent mode
<CR>	Data sent to HOST device
\$ xasm test.asm -l > test.lst	Data sent to HOST device
	...
\$ CNTL-A	Exit transparent mode
>	

## TRACE

## Trace

### 5.2.27 Trace

TRACE [<n>]

Parameters:

- <n> The number of single-step trace instructions from the current user P-register (program counter) address. If <n> is not given, the value 1 is assumed. The parameter <n> is a decimal number between 1 and 255.

The address and disassembled user instruction are displayed prior to execution. Each user instruction is executed individually and MCU control returns to the monitor program after each user instruction. The new values of the user registers are displayed after the instruction is executed.

If **TRACE** attempts to execute an illegal M68HC11 opcode, an error message is displayed and further execution is aborted. Single-step tracing can be manually aborted by pressing **CNTL-A** or **CNTL-X**. Single-step tracing is not allowed when the EVB2LA is set-up to capture a trace.

The single-step trace places a software interrupt (SWI) instruction in the user program after the instruction to be executed. If interrupts are enabled to the CPU (i.e., if the user's I bit is cleared), then the service routine for all pending interrupts will be executed as well as each traced instruction. The SWI instruction returns MCU control to the monitor program (refer to paragraph 4.4.1.3). This requires that all user programs reside in RAM or internal EEPROM to use the **PROCEED**, **STOPAT**, and **TRACE** commands. **TRACE** execution is not in real time which may affect the operation of some user programs.

### NOTE

Do not include SWI instructions in user programs. An SWI may return control to the monitor program at an inopportune time.

### EXAMPLES

```

>trace 3<CR>                                User P-register value is $F800
F800 LDS  #$10FF          P-F803 Y-FCFD X-FAFB A-20 B-F8 C-90 S..I.... S-10FF
F803 LDAA #$10           P-F805 Y-FCFD X-FAFB A-10 B-F8 C-90 S..I.... S-10FF
F805 STAA $103D         P-F808 Y-FCFD X-FAFB A-10 B-F8 C-90 S..I.... S-10FF
>trace<CR>
F808 BSR  $F819          P-F819 Y-FCFD X-FAFB A-10 B-F8 C-90 S..I.... S-10FD
>

```

---

---

**TRACE****Trace**

<b>Error Messages</b>	<b>Possible Cause</b>
Bad argument	Only one parameter is allowed. The parameter must be decimal, greater than 0, and less than 256.
ILLOP	The instruction to be executed is not a legal M68HC11 opcode.
LA Set-up	The EVB2LA is set-up for a trace (entering RESET clears the logic analyzer trace).
Parameter Range Error	Parameter <n> must be less than 256.
rom-xxxx	The EVB2 cannot be used to directly debug programs in ROM (refer to paragraph 4.4.6).

---

---

## USER

## User-Reset Preparation

### 5.2.28 User-Reset Preparation

#### USER

A user-reset is accomplished by holding down the USER switch (SW2) while pressing and releasing the RESET switch (SW3). This switch combination resets the MCU and causes it to come out of reset in expanded multiplexed operating mode, fetching the reset vector from \$FFFE – \$FFFF. A user program then has immediate control of the MCU and its operation may be fully evaluated.

Use of the USER command prior to a user-reset permits returning to the monitor program when any of these events occurs:

1. A key is pressed on the terminal device
2. A breakpoint is encountered
3. The EVB2LA terminates a trace capture

When the monitor program regains control, the current CPU registers are saved in the user registers and any breakpoints are removed from the user program. The user register values are displayed and the EVB2LA status is displayed if it is present.

In order to return control to the monitor program, the USER command alters the user program so that three new instructions are executed immediately after the user-reset. These three instructions are used to clear the X bit in the CPU condition code register and require seven E-clock cycles to execute. These seven cycles encroach on the 64 cycle time-out limitation on changing some MCU control registers.

The USER command may be aborted by pressing any key on the terminal device prior to initiating a user-reset.

---

---

**USER****User-Reset Preparation****EXAMPLE**

<pre>&gt;user&lt;CR&gt; Do USER RESET now, any key to abort Press USER and RESET switch (SW2 &amp; SW3) P-F80A Y-FD92 X-105B A-00 B-87 C-94 S..I.Z.. S-10FF &gt;</pre>	<pre>Prepare for user-reset Breakpoint encountered at \$F80A</pre>
--	--

<b>Error Message</b>	<b>Possible Cause</b>
rom-xxxx	The EVB2 cannot be used to directly debug programs in ROM (refer to paragraph 4.4.6).

## VERIFY

## Verify S-Records

### 5.2.29 Verify S-Records

VERIFY T

VERIFY [<host download command>]

Parameters:

T Verify S-record from the TERMINAL port (P6) against memory contents.

<host download command> Verify S-record from the HOST port (P5) against memory contents. The <host download command> parameter defines a command to be executed by the host computer. The **VERIFY** [<host download command>] is issued from the terminal device and tells the host computer to download the S-record file to be checked against memory contents. If no parameter is given, the **VERIFY** command performs as a **TM** command.

The monitor program waits for S-record data from the specified port (refer to Appendix A for S-record information). If all S-records are correctly received during **VERIFY** and all S-record data agrees with the corresponding memory contents, the message `done` is displayed on the terminal device. If an error is encountered during **VERIFY** or a mismatch is encountered, the monitor program ceases verifying additional S-record information and waits for the S9 termination record. An error message is then displayed.

The **VERIFY** command does not write any data into memory and can be used for verifying S-record data against any type of memory. The current offset value as specified by the **OFFSET** command is added to all S-record addresses before the verification.

Pressing **CNTL-A** or **CNTL-X** aborts the **VERIFY** command. The <host download command> may not contain the slash (/) character because it is a BUFFALO command.

### EXAMPLES

<code>&gt;verify t&lt;CR&gt;</code>	Verify from TERMINAL port command
<code>done</code>	S-record file sent from terminal device
<code>&gt;verify cat program.s19&lt;CR&gt;</code>	Verify from HOST port command
<code>cat program.s19</code>	HOST's download command is "cat program.s19"
<code>E005-does not verify</code>	Data at address \$E005 does not match
<code>&gt;</code>	



---

---

**VERIFY****Verify S-Records**

<b>Error Messages</b>	<b>Possible Cause</b>
chksum error	At least one S-record checksum test failed.
xxxx-does not verify	The data at address \$xxxx does not match the S-record data.

## XBOOT

## Send Data to Another MCU

### 5.2.30 Send Data to Another MCU

XBOOT [<addr1> [<addr2>]]

#### Parameters:

<addr1> <addr2> Transmit the data between addresses <addr1> and <addr2> through the MCU SCI to another M68HC11 MCU in special bootstrap mode. If <addr2> is not given, 256 bytes are transmitted from <addr1>. If <addr1> is not given, the default starting address is \$E000.

The purpose of the **XBOOT** command is to download a user-written program to another M68HC11 MCU operating in special bootstrap mode. The target MCU must be configured to operate in special bootstrap mode and its SCI RXD line must be tied to the MCU TXD line. The target MCU should be reset immediately prior to issuing the **XBOOT** command.

These EVB2 MCU control registers are modified:

PORTD[bit-1]	→	1
DDRD[bit-1]	→	1
BAUD	→	\$22
SCCR2	→	\$0C

The resident bootloader program input requirements have minor differences among the various M68HC11 family members. For instance, the MC68HC11A1 and A8 members require the bootstrap program to be 256 bytes in length, while the MC68HC11E9 accepts variable length downloads from 1 to 512 bytes.

Refer to the M68HC11 Reference Manual and Motorola Semiconductor Application Note AN1060 – MC68HC11 Bootstrap Mode for additional information on special bootstrap mode downloading.

### **EXAMPLE**

<pre>&gt;xboot e000 e1ff&lt;CR&gt; &gt;</pre>	Send program at \$E000 – \$E1FF to MC68HC11E9
---	---

---

---

**XBOOT****Send Data to Another MCU**

<b>Error Message</b>	<b>Possible Cause</b>
Bad argument	Only two address parameters are allowed Only hexadecimal values are allowed

---

### 5.3 EVB2LA COMMANDS

This section describes the commands that are added to the BUFFALO command set for use with the EVB2LA. For each command, the command line format is given with a description of the command and examples of its use. Within the example boxes, **bold** text represents text entered by the user. A carriage return is represented by a <CR>. Program monitor text examples are in this distinctive typeface. Explanatory comments appear to the right. Error messages along with possible explanations are also included for the each command. Refer to paragraph 4.5 for a description of the EVB2LA.

This list defines the terms used to describe the EVB2LA operating characteristics:

- Trace instruction execution mode — captures every cycle of user program execution.
- Trigger — in trace instruction execution mode, an address in the user program that causes the EVB2LA to start or stop capturing user-program-execution cycles. In trace only mode, an address at which a read or write cycle is captured in the capture buffer.
- Trace only mode — captures only selected cycles during user program execution.
- Capture buffer — 32-bit wide by 8K RAM memory. Each 32-bit row of the buffer holds data concerning one cycle of user program execution. This data includes the 16-bit address bus, 8-bit data bus, and an 8-bit auxiliary byte.
- Auxiliary byte — 8 bits in the capture buffer. The byte is comprised of six user-defined test points, the MARK bit, and the MCU R/W signal.
- Test points — six user-defined TTL compatible inputs on the EVB2LA that are captured in the capture buffer.
- MARK bit — control signal generated by the EVB2LA control circuitry and stored in the capture buffer. In trace instruction execution mode, MARK is asserted during the first cycle of each instruction executed by the MCU. In trace only mode, MARK is asserted for all cycles captured in the capture buffer.
- Trigger offset — pointer used to indicate the current position in the capture buffer from which data is displayed. The trigger offset value is relative to the trigger point and is in units of instructions for data captured from trace instruction execution mode, and units of cycles for trace only mode.
- Trace set-up command — BUFFALO command used to specify trigger addresses and prepare the EVB2LA to capture a particular type of trace the next time the user program is executed.
- Trace display command — BUFFALO command used to display data in the capture buffer from a previous user program trace.

**</>/^**

## Move Capture Buffer Trigger Offset

### 5.3.1 Move Capture Buffer Trigger Offset

- < Move trigger offset to the beginning of the capture buffer
- > Move trigger offset to the end of the capture buffer
- ^ Move trigger offset to the capture buffer trigger point

### EXAMPLES

```

><<CR>                                Move trigger offset to beginning of capture buffer
Trigger Offset Moved From :  0000
To End Of Capture Buffer : -0513

>^<CR>                                Move trigger offset to the trigger instruction
Trigger Offset Moved From : -0513
To :  0000
>
    
```

Error Message	Possible Cause
Capture Buffer Empty	Capture buffer was cleared by RESET or EVB2LA setup command.

---

---

## ABOUT

## Set-Up Trace About Trigger

### 5.3.2 Set-Up Trace About Trigger

ABOUT <addr1> [[-] <addr2>]...

Parameters:

- <addr1> Set-up the EVB2LA to trace before and after the instruction at the trigger address defined by the value <addr1>. All addresses must be within \$D000 – \$FFFF.
- <addr2> Set-up the EVB2LA to trace before and after the instruction at a second trigger address defined by the value <addr2>. Any number of addresses may be specified, separated by one or more space or tab characters.
  - <addr1> and <addr2> separated by a hyphen (-) denotes a range of trigger addresses.

The EVB2LA trace set-up commands prepare the EVB2LA to capture a trace during the next user program execution via the **GO**, **CALL**, **RUN**, or **USER** commands. The message LA Set-up indicates that the EVB2LA is prepared for the trace. A subsequent issuance of a new trace set-up command aborts any previous trace set-up.

During user program execution, the **ABOUT** command captures as many as 4096 bus-cycles of data before and after the trigger point. Once the capture buffer fills with data, a non-maskable interrupt is generated, returning control to the monitor program. The trigger offset is set at the trigger instruction.

If the capture buffer does not fill with data before execution returns to the monitor program, all captured data is visible in the capture buffer. If a trigger instruction has not been reached, the trigger offset is at the last instruction executed.

Normally, breakpoints in the breakpoint table are installed in the user program when the user program executes. However, when an EVB2LA trace is set-up, breakpoints are not installed. Additionally, once an EVB2LA trace is set-up, executing a single-step trace via the **PROCEED**, **STOPAT**, or **TRACE** commands is prohibited.

## ABOUT

## Set-Up Trace About Trigger

### EXAMPLES

<pre>&gt;about e0a0&lt;CR&gt; LA Set-up</pre>	Set-up trace about trigger at address \$E0A0
<pre>&gt;about e106-e112&lt;CR&gt; LA Set-up</pre>	Abort previous set-up, and set-up trace about trigger at any address between \$E106 and \$E112
<pre>&gt;</pre>	

Error Messages	Possible Cause
Improper Parameter Specification	At least one address parameter is required. Only hexadecimal address parameters or a hyphen are allowed.
Parameter Range Error	Addresses must be within \$D000 – \$FFFF. The second address of a range must be greater than the first address.

---

---

## AFTER

## Set-up Trace After Trigger

### 5.3.3 Set-up Trace After Trigger

AFTER <addr1> [[-] <addr2>]...

Parameters:

- <addr1> Set-up the EVB2LA to trace after the instruction at the trigger address defined by the value <addr1>. All addresses must be within \$D000 – \$FFFF.
- <addr2> Set-up the EVB2LA to trace after the instruction at a second trigger address defined by the value <addr2>. Any number of addresses may be specified, separated by one or more space or tab characters.
  - <addr1> and <addr2> separated by a hyphen (–) denotes a range of trigger addresses.

The EVB2LA trace set-up commands prepare the EVB2LA to capture a trace during the next user program execution via the **GO**, **CALL**, **RUN**, or **USER** commands. The message LA Set-up indicates that the EVB2LA is prepared for the trace. A subsequent issuance of a new trace set-up command aborts any previous trace set-up.

During user program execution, the **AFTER** command captures as many as 8192 bus-cycles of data after the trigger. Once the capture buffer fills with data, a non-maskable interrupt is generated, returning control to the monitor program. The trigger offset is set at the trigger instruction.

If the capture buffer does not fill with data before execution returns to the monitor program by some other means, all captured data are visible in the capture buffer. If a trigger instruction has not been reached, the capture buffer will be empty.

Normally, breakpoints in the breakpoint table are installed in the user program when the user program executes. However, when an EVB2LA trace is set-up, breakpoints are not installed. Additionally, once an EVB2LA trace is set-up, executing a single-step trace via the **PROCEED**, **STOPAT**, or **TRACE** commands is prohibited.



## AFTER

## Set-up Trace After Trigger

### EXAMPLES

<pre>&gt;after e0a0&lt;CR&gt; LA Set-up</pre>	Set-up trace after trigger at address \$E0A0
<pre>&gt;after e106-e112&lt;CR&gt; LA Set-up</pre>	Abort previous set-up, and set-up trace after trigger at any address between \$E106 and \$E112
<pre>&gt;</pre>	

Error Messages	Possible Cause
Improper Parameter Specification	At least one address parameter is required. Only hexadecimal address parameters or a hyphen are allowed.
Parameter Range Error	Addresses must be within \$D000 – \$FFFF. The second address of a range must be greater than the first address.

## BEFORE

## Set-up Trace Before <n>th Execution of Trigger

### 5.3.4 Set-up Trace Before <n>th Execution of Trigger

BEFORE <addr1> [[-] <addr2>]...[N <n>]

Parameters:

- <addr1> Set-up the EVB2LA to trace before the <n>th execution of an instruction at the trigger address defined by the value <addr1>. All addresses must be within \$D000 – \$FFFF.
- <addr2> Set-up the EVB2LA to trace before the <n>th execution of an instruction at a second trigger address defined by the value <addr2>. Any number of addresses may be specified, separated by one or more space or tab characters.
  - Two addresses separated by a hyphen (-) denote a range of trigger addresses. All addresses must be within \$D000 – \$FFFF.
- N <n> If the <n> parameter is not defined, the value 1 is assumed. <n> is a decimal value between 1 and 4096.

The EVB2LA trace set-up commands prepare the EVB2LA to capture a trace during the next user program execution via the **GO**, **CALL**, **RUN**, or **USER** commands. The message **LA Set-up** indicates that the EVB2LA is prepared for the trace. A subsequent issuance of a new trace set-up command aborts any previous trace set-up.

During user program execution, the **BEFORE** command captures as many as 8192 bus-cycles of data before the <n>th execution of any trigger instruction, at which time a non-maskable interrupt is generated, returning control to the monitor program. The trigger offset is set at the trigger instruction.

If the <n>th trigger instruction has not been executed before control returns to the monitor program by some other means, all captured data will be visible in the capture buffer and the trigger offset will be set at the end of the buffer.

Normally, breakpoints in the breakpoint table are installed in the user program when the user program executes. However, when an EVB2LA trace is set-up, breakpoints are not installed. Additionally, once an EVB2LA trace is set-up, executing a single-step trace via the **PROCEED**, **STOPAT**, or **TRACE** commands is prohibited.

## BEFORE

## Set-up Trace Before <n>th Execution of Trigger

### EXAMPLES

<pre>&gt;before e0a0 N 5&lt;CR&gt; LA Set-up</pre>	Set-up trace before 5th execution of instruction at address \$E0A0
<pre>&gt;BEFORE e106-e112&lt;CR&gt; LA Set-up</pre>	Abort previous set-up, and set-up trace before first trigger at any address between \$E106 and \$E112
<pre>&gt;</pre>	

Error Messages	Possible Cause
Improper Parameter Specification	At least one address parameter is required. Only hexadecimal address parameters or a hyphen are allowed.
Parameter Range Error	Addresses must be within \$D000 – \$FFFF. The second address of a range must be greater than the first address. <n> must be decimal, greater than 0, and less than 4096.

---

## **DIS** **Disassemble Instructions in Buffer**

### **5.3.5 Disassemble Instructions in Buffer**

DIS [X] [<n>]

Parameters:

- X Specifies extended disassembly and displays bus transactions.
- <n> Number of instructions to disassemble from the capture buffer. <n> is a decimal number between 1 and 8192. The default is 1.

The **DIS** command begins disassembly at the current trigger offset position and continues for <n> instructions or until the end of the capture buffer is reached. At the completion of the command, the trigger offset is positioned at the next instruction in the capture buffer.

The left column of the **DIS** output displays the offset of each instruction from the trigger instruction. Disassembly follows the same format as the **ASM** command disassembly, rebuilding the required data from the capture buffer. The **DIS** command can only be used to display data captured from the trace instruction execution mode commands: **ABOUT**, **AFTER**, **BEFORE**, and **FAULT**.

The additional information available from extended disassembly follows each disassembled instruction. This information summarizes memory transactions that take place during the instruction. Additionally, interrupt occurrences are indicated and the stacking of CPU registers is displayed. Refer to Table 5-3 for special extended disassembly mnemonic codes and their meaning:

**DIS****Disassemble Instructions in Buffer****Table 5-3. Extended Disassembly Mnemonic Codes**

<b>Mnemonic</b>	<b>Meaning</b>
From	read cycle
Into	write cycle
Data	1-byte transaction
Low	low byte of 2-byte transaction
High	high byte of 2-byte transaction
Orig	original (read cycle) of read-modify-write instruction
Rslt	result (write cycle) of read-modify-write instruction
INTERRUPT	indicates beginning of interrupt stacking
PCL	Program Counter low byte
PCH	Program Counter high byte
IYL	Index Register Y low byte
IYH	Index Register Y high byte
IXL	Index Register X low byte
IXH	Index Register X high byte
ACCA	Accumulator A
ACCB	Accumulator B
CCR	Condition Code Register
V_HI	interrupt vector high byte
V_LO	interrupt vector low byte

## DIS Disassemble Instructions from Buffer

### Examples

```

0 Pre-trigger Instructions           Capture buffer contains data from AFTER
1889 Post-trigger Instructions
P-F8E1 Y-FD6C X-1000 A-FC B-00 C-80 S..... S-00FD
>dis 3<CR>                          Disassemble 3 instructions
 0000: F800  LDS  #$00FF
+0001: F803  LDAA #$10
+0002: F805  STAA $1000

>dis x 4<CR>                        Extended disassemble 4 instructions
+0003: F808  BSR  $F819
        Low  :  #$0A Into $00FF
        High:  #$F8 Into $00FE
+0004: F819  CLI
+0005: F81A  STAA $1002
        Data:  #$10 Into $1002
        INTERRUPT
        PCL  :  #$1D Into $00FD           Interrupt stacking of CPU registers
        PCH  :  #$F8 Into $00FC
        IYL  :  #$92 Into $00FB
        IYH  :  #$FD Into $00FA
        IXL  :  #$5B Into $00F9
        IXH  :  #$10 Into $00F8
        ACCA:  #$10 Into $00F7
        ACCB:  #$87 Into $00F6
        CCR  :  #$80 Into $00F5
        V_HI:  #$FA From $FFF0
        V_LO:  #$5A From $FFF1
+0006: FA5A  LDAA #$40                   First instruction of interrupt subroutine
>

```

Error Messages	Possible Cause
Capture Buffer Empty	The capture buffer was cleared by Reset or an EVB2LA setup command.
No TIME or DIS after ONLY	The <b>RAW</b> command must be used to view data after <b>ONLY</b> command.
Parameter Range Error	Only decimal values between 1 and 8192 are allowed.
Syntax Error	Only an X or a decimal value is allowed.

---

---

# FAULT

# Fault

## 5.3.6 Fault

### FAULT

The EVB2LA identifies a fault condition as the occurrence of one of two events during user program execution:

A write to user-program address fault — a write cycle at an address in the \$C000 – \$FFFF range

An instruction fetch out-of-range fault — the fetch of an opcode from any address outside of the \$C000 – \$FFFF range

The EVB2LA trace set-up commands prepare the EVB2LA to capture a trace during the next user program execution via the **GO**, **CALL**, **RUN**, or **USER** commands. The message LA Set-up indicates that the EVB2LA is prepared for the trace. A subsequent issuance of a new trace set-up command aborts any previous trace set-up.

During user program execution, the **FAULT** command captures as many as 8192 bus-cycles of data before a fault condition occurs, at which time a non-maskable interrupt is generated, returning control to the monitor program. The trigger offset is set at the instruction attempting to write to a user program address, or the instruction preceding an instruction fetched out-of-range.

If a fault condition does not occur before control returns to the monitor program, the capture buffer will be empty.

Before a stack-write operation is performed, redefine the user S-register (stack pointer) so it does not point to the user stack area: \$C000 – \$C014. Redefining the user S-register, either manually or under program control, avoids an erroneous fault condition trigger.

Normally, breakpoints in the breakpoint table are installed in the user program when the user program executes. However, when an EVB2LA trace is set-up, breakpoints are not installed. Additionally, once an EVB2LA trace is set-up, executing a single-step trace via the **PROCEED**, **STOPAT**, or **TRACE** commands is prohibited.

### EXAMPLE

<pre>&gt;fault&lt;CR&gt; LA Set-up &gt;</pre>	Set-up trace before a fault condition
---	---------------------------------------

## FIND

## Find Data in Capture Buffer

### 5.3.7 Find Data in Capture Buffer

FIND [A,I [%] <address>] [D [%] <data>] [X [%] <aux>] [-] [<n>]

FIND O [-] <offset>

Parameters:

- A, I % <address>** Find a specific 16-bit address in the capture buffer. The A parameter specifies a search for an address, the I parameter specifies a search for an address with the MARK bit in the auxiliary byte equal to 1. If the percent sign (%) precedes an <address> value, the value is interpreted bitwise, in which case only 0, 1, and X characters may be included in the value (X = “don’t care” bit). Otherwise, the value is interpreted in hexadecimal.
- D % <data>** Find a specific 8-bit data value in the capture buffer. If the percent sign (%) precedes a <data> value, the value is interpreted bitwise, in which case only 0, 1, and X characters may be included in the value (X = “don’t care” bit). Otherwise, the value is interpreted in hexadecimal.
- X % <aux>** Find a specific value in the capture buffer which matches a value in the auxiliary byte. If the percent sign (%) precedes a <data> value, the value is interpreted bitwise, in which case only 0, 1, and X characters may be included in the value (X = “don’t care” bit). Otherwise, the value is interpreted in hexadecimal.

T6	T5	T4	T3	T2	T1	RW	MK
----	----	----	----	----	----	----	----

T6-T1      User test points

RW           $\overline{R/W}$

MK          MARK Bit

- Searched backward in the capture buffer.
- <n> The <n> parameter is a decimal value between -8192 and 8192. The default is 1.
- O Find an offset value in the capture buffer defined by the value <offset>.

At least one of the parameters A, I, D, X, or O must be given. One of each parameter A or I, D, and X may be included in one command (in the order shown).

The A and I parameters are identical except that the I search is automatically qualified to find only the first cycle of an instruction (with the MARK bit set).



## FIND

## Find Data in Capture Buffer

If the capture buffer contains data from one of the trace instruction execution commands (**ABOUT**, **AFTER**, **BEFORE**, or **FAULT**), the trigger offset is placed at the instruction which contains the <n>th cycle which matches the specified criteria.

As **FIND** moves through the capture buffer, if either end of the buffer is encountered before the specified criteria is found, **FIND** halts, displays an error message, and returns the trigger offset to its previous position.

**FIND** displays the trigger offset location before and after execution of the command.

### EXAMPLES

312 Pre-trigger Instructions 921 Post-trigger Instructions P-F80C Y-FD6C X-0C0C A-04 B-00 C-84 S...Z.. S-00FF	Capture buffer contains data from <b>ABOUT</b>
>find o -312<CR>	Move trigger offset to offset -312
Trigger Offset Moved From : 0000 To : -0312	
>find a 1 x %xxxxxxxx0<CR>	Find first address bus=\$0001, with $R/\overline{W}=0$ (i.e. next write to address \$0001)
Trigger Offset Moved From : -0312 To : -0207	
>dis x<CR>	Disassemble instruction found
-0207: F8A4 STAA \$01 Data: #\$00 Into \$0001	
>find i f805<CR>	Find next instruction starting at address \$F805
Trigger Offset Moved From : -0206 To : 0000	
>dis x<CR>	Disassemble instruction found
0000: F805 CLI	

## FIND

## Find Data in Capture Buffer

Error Messages	Possible Cause
Capture Buffer Empty	The capture buffer was cleared by Reset or an EVB2LA setup command.
Currently At That Position	The trigger offset is currently at the specified <offset>.
Improper Parameter Specification	With <b>FIND O</b> , only decimal parameters are allowed. With %, only 0, 1 or X values are allowed otherwise, only hexadecimal parameters are allowed.
Parameter Range Error	Valid ranges are <address>: \$0000 – \$FFFF <data>: \$00 – \$FF <aux>: \$00 – \$FF <n>: 1 – 8192
Syntax Error	Only designators A, I, D, X, and O are allowed. Parameters must be defined in this order: A or I, D, and X. <offset> must be a decimal number between -8192 and 8192.
Trigger offset restored to previous position	Search criteria not found in buffer.

---

## LA Display Logic Analyzer Commands Help Menu

### 5.3.8 Display Logic Analyzer Commands Help Menu

LA

Refer to the **HELP** command to display primary commands and the **MEM** command for the memory commands help menu.

#### EXAMPLE

```
>la<CR>

***** LOGIC ANALYZER OPTIONS MENU *****
*** CAPTURE ***
AFTER <addr1> [-] [...<addrn>]           :trace after
ABOUT <addr1> [-] [...<addrn>]         :trace before & after
BEFORE <addr1> [-] [...<addrn>] [N <n>] :trace before nth execution
ONLY [R,W,A,E] <addr1> [-] [...<addrn>] :capture Rd, Wrt or Acc to RAM/REG
ONLY V [[-] <addr>...]                 :capture reads of interrupt Vectors
FAULT :trace before instr fetch out-of-range or write to user pgm address
*** RETRIEVAL ***
SKIP [-] [<n>] :move trigger offset n instructions
FIND [A,I [%] <addr1>] [D [%] <val1>] [X [%] <val2>] [-] [<n>]
      :move trigger offset to instr containing nth occurrence of srch pattern
FIND O [-] <offst> :load trigger offset with the value specified
TIME [A,I [%] <addr1>] [D [%] <val1>] [X [%] <val2>] [-] [<n1>]
      [: [A,I [%] <addr2>] [D [%] <val3>] [X [%] <val4>] [-] [<n2>]]
TIME O [-] <offst1> [: [-] <offst2>]
      :calculate time between events
RAW [<n>]           :dump n instructions / cycles from trigger offset
DIS [<n>]           :disassemble n instructions from trigger offset
DIS X [<n>]         :extended dissassembly - displays bus activity by cycle

< : skip to top           ^ : skip to trigger           > : skip to bottom
*****
>
```

## ONLY                      Set-up Trace of Only Trigger Addresses

### 5.3.9    Set-up Trace of Only Trigger Addresses

ONLY [R,W,A,E] <addr1> [[-] <addr2>]...

ONLY V [[-] <addr1> [<addr2>...]]

Parameters:

- R    Set-up the EVB2LA to capture only reads of any trigger address in the list given
- W    Set-up the EVB2LA to capture only writes to any trigger address in the list given
- A    Set-up the EVB2LA to capture only accesses (reads or writes) to any trigger address in the list given
- E    Set-up the EVB2LA to capture only EEPROM write cycles to any trigger address in the list given.
- <addr1>    Set-up the EVB2LA to trace the instruction at the trigger address defined by the value <addr1>. All addresses must be within \$D000 – \$FFFF.
- <addr2>    Set-up the EVB2LA to trace the instruction at a second trigger address defined by the value <addr2>. Any number of addresses may be specified, separated by one or more space or tab characters.
  - Two addresses separated by a hyphen denote a range of trigger addresses. Refer to the table below for the allowable address ranges (R, W, A, and E options).
- V    If no additional parameters follow the V option, a trigger is set at all interrupt vector addresses (except the XIRQ and SWI interrupt vectors, which are used by the monitor program, refer to paragraph 4.3). If a minus sign precedes the address list, a trigger is set at all interrupt vector addresses except those in the list. If no minus sign is present, a trigger is set only at those addresses in the list given.

---

---

## **ONLY**                      **Set-up Trace of Only Trigger Addresses**

Only one of the parameters R, W, A, E, or V may be specified. Any number of addresses may be specified separated by one or more space or tab characters.

<b>Designator</b>	<b>Capture</b>	<b>Allowable addresses</b>
R	Read cycles	\$0000-1FFF
W	Write cycles	\$0000-1FFF
A	Read or write cycles	\$0000-1FFF
E	EEPROM cell writes	\$B600-B7FF
V	Interrupt vector fetches	\$FFC0-FFFE (even addresses)

The EVB2LA trace set-up commands prepare the EVB2LA to capture a trace during the next user program execution via the **GO**, **CALL**, **RUN**, or **USER** commands. The message LA Set-up indicates that the EVB2LA is properly prepared for the trace. A subsequent issuance of a new trace set-up command aborts any previous trace set-up.

During user program execution, the **ONLY** command captures as many as 8192 bus-cycles of data. Once the capture buffer fills with data, a non-maskable interrupt is generated, returning control to the monitor program. The trigger offset is set at the beginning of the capture buffer.

If the capture buffer does not fill with data before execution returns to the monitor program, all captured data is visible in the capture buffer. If a trigger instruction has not been reached, the capture buffer is empty.

Because the product of the **ONLY** command is bus-cycle data captured over an arbitrarily long period of time, the **DIS** and **TIME** commands are not allowed following **ONLY**.

The product of the V option is a list of the even addresses of interrupt vectors in the order in which they were fetched. This data is useful for monitoring interrupt sequencing, although no timing data is available.

The E option is not allowed if the EVB2 MCU does not contain 512 bytes of internal EEPROM at \$B600 – \$B7FF.

Normally, breakpoints in the breakpoint table are installed in the user program when the user program executes. However, when an EVB2LA trace is set-up, breakpoints are not installed. Additionally, once an EVB2LA trace is set-up, executing a single-step trace via the **PROCEED**, **STOPAT**, or **TRACE** commands is prohibited.

## ONLY                      Set-up Trace of Only Trigger Addresses

### EXAMPLES

<pre>&gt;only a 4&lt;CR&gt; LA Set-up</pre>	Set-up to trace read and write cycles to address \$0004
<pre>&gt;only e b7f0&lt;CR&gt; LA Set-up</pre>	Abort previous set-up, and set-up to trace write cycles to address \$B7F0 (EEPROM cell)
<pre>&gt;only v - fff0&lt;CR&gt; LA Set-up</pre>	Abort previous set-up, and set-up to trace all interrupt vectors except the real time interrupt
<pre>&gt;only v&lt;CR&gt; LA Set-up</pre>	Abort previous set-up, and set-up to trace all interrupt vector fetches
<pre>&gt;</pre>	

Error Messages	Possible Cause
Improper Parameter Specification	<p>At least one parameter, R, W, A, E, or V, and one address parameter is required.</p> <p>Only hexadecimal address parameters or a hyphen are allowed.</p> <p>The E option cannot be used with the MC68HC811E2 MCU.</p>
Parameter Range Error	<p>Addresses must be within \$0000 – \$1FFF when using the R, W, and A options.</p> <p>When using the E option, addresses must be within \$B600 – \$B7FF.</p> <p>When using the V option, addresses must be even addresses between \$FFC0 – \$FFFE.</p> <p>The second address of a range must be greater than the first.</p>

---

---

## **RAW** **Display Bus-Cycle Data from Buffer**

### **5.3.10 Display Bus-Cycle Data from Buffer**

RAW [*<n>*]

Parameter:

- <n>* Number of instructions to display (for data captured **ABOUT**, **AFTER**, **BEFORE**, or **FAULT** command) or number of cycles (for data captured from **ONLY**) in bus-cycle format from the capture buffer. The parameter *<n>* is a decimal number and defaults to 1.

The **RAW** command begins at the current trigger offset position and continues for *<n>* instructions (or cycles) or until the end of the capture buffer is reached. At the completion of the command, the trigger offset is left at the next position in the capture buffer.

For each offset displayed from the buffer, **RAW** shows a column header, the offset value, and the cycle-count for instructions (see the example). For each bus-cycle displayed, the address bus, data bus, and auxiliary byte values are shown. The auxiliary byte is displayed in binary format. The column heading may be used to determine the bit definitions as follows:

<i>T6-T1</i>	User test points (same as EVB2LA connector P1)
<i>MK</i>	MARK Bit (refer to paragraph 4.5.1)
<i>RW</i>	R $\overline{W}$ line

# RAW Display Bus-Cycle Data from Buffer

## EXAMPLE

```

312 Pre-trigger Instructions          Capture buffer contains data from ABOUT
924 Post-trigger Instructions
P-FA2F Y-FD6C X-0A0A A-01 B-00 C-80 S..... S-00FC
>raw 2<CR>                          Display 2 instructions

Offset  Address  Data  TTTTTTMR
0000:   F805   0E   11111111
        F806   BD   11111101
Cycles:  2

Offset  Address  Data  TTTTTTMR
+0001:   F806   BD   11111111  Interrupt generated during this instruction
        F807   F8   11111101
        F808   DE   11111101
        F8DE   CE   11111101
        00FF   09   11111100
        00FE   F8   11111100  End-cycle of instruction
        F8DE   CE   11111101  Beginning of interrupt
        F8DF   10   11111101
        00FD   DE   11111100
        00FC   F8   11111100
        00FB   6C   11111100
        00FA   FD   11111100
        00F9   5B   11111100
        00F8   00   11111100
        00F7   00   11111100
        00F6   6D   11111100
        00F5   84   11111100
        00F5   84   11111101
        FFF0   FA   11111101
        FFF1   46   11111101
Cycles: 20
>
    
```

Error Messages	Possible Cause
Capture Buffer Empty	The capture buffer was cleared by RESET or an EVB2LA setup command.
Parameter Range Error	Valid parameter values are 1 – 8192.
Improper Parameter Specification	Only one decimal parameter is allowed.



## SKIP Move the Trigger Offset in Buffer

### 5.3.11 Move the Trigger Offset in Buffer

SKIP [-] [<n>]

Parameters:

- <n> Number of instructions to move the trigger offset (for data captured from **ABOUT**, **AFTER**, **BEFORE**, or **FAULT**), or number of cycles (for data captured from **ONLY**) in the capture buffer. The parameter <n> is a decimal number and defaults to 1.
- Defines direction of the move (forward or backward) in the capture buffer.

If the end of the capture buffer is encountered before the specified distance has been skipped, the trigger offset is set at the end of the capture buffer.

### EXAMPLES

```

312 Pre-trigger Instructions           Capture buffer contains data from ABOUT
924 Post-trigger Instructions
P-FA2F Y-FD6C X-0A0A A-01 B-00 C-80 S..... S-00FC
>skip -312<CR>                       Move trigger offset to beginning of buffer

Trigger Offset Moved From : 0000
To : -0312

>dis<CR>                             Disassemble instruction
-0312: F800 LDS #$00FF

>skip 10<CR>                          Move trigger offset forward 10 instructions

Trigger Offset Moved From : -0311
To : -0301

>
    
```

Error Messages	Possible Cause
Capture Buffer Empty	The capture buffer was cleared by RESET or an EVB2LA setup command.
Parameter Range Error	Only values between -8192 and 8192 are allowed.
Syntax Error	Only one decimal parameter is allowed.

## TIME Display Cycle Count Between Events in Buffer

### 5.3.12 Display Cycle Count Between Events in Buffer

```
TIME [A,I [%] <addr1>] [D [%] <data1>] [X [%] <aux1>] [-] [<n1>] :
      [A,I [%] <addr2>] [D [%] <data2>] [X [%] <aux2>] [-] [<n2>]
```

```
TIME O [-] <offset1> : [[-] <offset2>]
```

#### Parameters:

**A, I % <addr1> : A, I % <addr2>** Find a specific 16-bit address in the capture buffer. The A parameter specifies a search for an address, the I parameter specifies a search for an address with the MARK bit in the auxiliary byte equal to 1. If the percent sign (%) precedes an <address> value, the value is interpreted bitwise, in which case only 0, 1, and X characters may be included in the value (X = “don’t care” bit). Otherwise, the value is interpreted in hexadecimal.

**D % <data1> : D % <data2>** Find a specific 8-bit data value in the capture buffer. If the percent sign (%) precedes a <data> value, the value is interpreted bitwise, in which case only 0, 1, and X characters may be included in the value (X = “don’t care” bit). Otherwise, the value is interpreted in hexadecimal.

**X % <aux1> : X % <aux1>** Find a specific value in the capture buffer which matches a value in the auxiliary byte. If the percent sign (%) precedes a <data> value, the value is interpreted bitwise, in which case only 0, 1, and X characters may be included in the value (X = “don’t care” bit). Otherwise, the value is interpreted in hexadecimal.

T6	T5	T4	T3	T2	T1	RW	MK
----	----	----	----	----	----	----	----

T6-T1 User test points

RW  $\overline{R/W}$

MK MARK Bit

- Searched backward in the capture buffer.
- <n> The <n> parameter is a decimal value between -8192 and 8192. The default is 1.
- O Find an offset value in the capture buffer defined by the value <offset>.
- <n> The <n> parameter is a decimal value between -8192 and 8192 and defaults to 1.
- O Find an offset value in the capture buffer defined by the values <offset1> and <offset2>.

---

## **TIME**    Display Cycle Count Between Events in Buffer

Display the number of cycles between two events in the capture buffer. If the second event is not specified, the number of cycles between the current trigger offset and the first event is displayed. The rules for parameter entry are the same as for the **FIND** command, with a colon separating the two events.

The **TIME** command effectively performs two **FIND** commands, counting the number of cycles between each event found. The trigger offset is moved as the **TIME** command executes and is left at the last event found. If either end of the capture buffer is encountered as **TIME** executes, the command is aborted and the trigger offset is positioned at the original trigger offset position.

### **EXAMPLES**

312 Pre-trigger Instructions	Capture buffer contains data from <b>ABOUT</b>
924 Post-trigger Instructions	
P-FA2F Y-FD6C X-0A0A A-01 B-00 C-80 S..... S-00FC	
>time i fff0 : i fff0 2<CR>	Display cycle-count between the next 2 RTI interrupts
Trigger Offset Moved From : 0000	
To : +0001	
To : +0497	
Cycles Between Events: 2237	1.1185 ms @ 2 MHz between interrupts
>	

---



---

## TIME    Display Cycle-Count Between Events in Buffer

Error Messages	Possible Cause
Capture Buffer Empty	The capture buffer was cleared by RESET or an EVB2LA setup command.
Improper Parameter Specification	With TIME O, only decimal parameters are allowed. With %, only 0, 1 or X values are allowed otherwise, only hexadecimal parameters are allowed.
Parameter Range Error	Valid ranges are <addr1,2>: 0000 through FFFF <data1,2>: 00 through FF <aux1,2>: 00 through FF <n1,2>: 1 through 8192
Syntax Error	Only designators A, I, D, X, and O are allowed.  A or I, D, and X parameters must be given in that order.  <offset> must be a decimal number between -8192 and 8192.
TIME display aborted	One of the events or offsets was not found in the capture buffer.
Trigger offset restored to previous position	One of the events or offsets was not found in the capture buffer.

## CHAPTER 6

### SUPPORT INFORMATION

#### 6.1 INTRODUCTION

This chapter provides the connector signal descriptions, parts lists, and schematic diagrams of the M68HC11EVB2 Evaluation Board components (EVB2CPU and EVB2LA).

#### 6.2 I/O CONNECTOR SIGNAL DESCRIPTIONS

The tables in this section list each EVB2CPU and EVB2LA input/output connector with the signal descriptions for each pin.

**Table 6-1. MCU I/O Port Connector (EVB2CPU P1) Pin Assignments**

Pin Number	Signal Mnemonic	Signal Name and Description
1	GND	GROUND
2, 3	—	Not connected.
4	STRA	STROBE A — Bi-directional control line used to latch data into ports B and C.
5	E	EXTERNAL CLOCK — Internally generated output clock signal used as a timing reference. The frequency of E clock is 1/2 the input frequency of the signal on the OSC1 pin.
6	STRB	STROBE B — An output control line used to control data into ports B and C.
7	EXTAL	EXTAL — MCU clock input line.
8	—	Not connected.
9 – 16	PC0 – PC7	PORT C (bits 0-7) — General purpose input/output (I/O) lines.
17	$\overline{\text{RESET}}$	RESET — An active low bi-directional control line used to initialize the MCU.
18	$\overline{\text{XIRQ}}$	X INTERRUPT REQUEST — An active low input line used to request MCU asynchronous non-maskable interrupts.

**Table 6-1. MCU I/O Port Connector (EVB2CPU P1) Pin Assignments (continued)**

Pin Number	Signal Mnemonic	Signal Name and Description
19	$\overline{\text{IRQ}}$	INTERRUPT REQUEST — An active low input line used to request MCU asynchronous interrupts.
20 – 25	PD0 – PD5	PORT D (bits 0-5) — General purpose input/output (I/O) lines.
26	VCC	SYSTEM POWER SUPPLY – +5 Vdc power source used by the EVB2 logic circuits. The signal on this pin is determined by the setting of jumper header J8 (refer to paragraph 2.3.2.2)
27 – 34	PA7 – PA0	PORT A (bits 0-7) — General purpose input/output (I/O) lines.
35 – 42	PB7 – PB0	PORT B (bits 0-7) — General purpose output lines.
43	PE0/AN0	PORT E (bit 0) — General purpose input or A/D channel input line.
44	PE4/AN4	PORT E (bit 4) — General purpose input or A/D channel input line.
45	PE1/AN1	PORT E (bit 1) — General purpose input or A/D channel input line.
46	PE5/AN5	PORT E (bit 5) — General purpose input or A/D channel input line.
47	PE2/AN2	PORT E (bit 2) — General purpose input or A/D channel input line.
48	PE6/AN6	PORT E (bit 6) — General purpose input or A/D channel input line.
49	PE3/AN3	PORT E (bit 3) — General purpose input or A/D channel input line.
50	PE7/AN7	PORT E (bit 7) — General purpose input or A/D channel input line.
51	VRL	VOLTAGE REFERENCE LOW — Input reference supply voltage (low) line for the MCU analog-to-digital (A/D) converter. Increases accuracy of the A/D conversion.

**Table 6-1. MCU I/O Port Connector (EVB2CPU P1) Pin Assignments (continued)**

Pin Number	Signal Mnemonic	Signal Name and Description
52	VRH	VOLTAGE REFERENCE HIGH — Input reference supply voltage (high) line. Increases accuracy of the A/D conversion.
53 – 59	—	Not connected.
60	MODB	EVB2 control circuitry output of MCU MODB signal. (not for use as input to control MCU)

**Table 6-2. EVB2CPU/EVB2LA Connector P2 Pin Assignments**

Pin Number	Signal Mnemonic	Signal Name and Description
1,2	GND	GROUND
3,4	VCC	SYSTEM POWER SUPPLY – +5 Vdc power source, EVB2LA input signal
5 – 20	A0 – A15	ADDRESS BUS A0 – A15 — MCU address bus, EVB2LA input signals
21	$\overline{\text{RTEST}}$	READ TEST BYTE — EVB2LA input signal
22	$\overline{\text{WTRIG}}$	WRITE TO TRIGGER RAM — EVB2LA input signal
23	$\overline{\text{WNIB}}$	NIBBLE SHIFT OUTPUT — EVB2LA input signal
24	$\overline{\text{RLA}}$	READ LOGIC ANALYZER — EVB2LA input signal
25	PTRESET	PT COUNTER RESET — EVB2LA input signal
26	$\overline{\text{LAMODE}}$	LOGIC ANALYZER MODE — EVB2LA input signal
27 – 32	T1 – T6	TEST POINTS 1 – 6 — User-selectable EVB2LA input lines. These test points are not connected to the EVB2.
33	S16	STATE_16 — EVB2LA input signal
34	$\overline{\text{XIRQ}}$	X INTERRUPT REQUEST — An active low EVB2LA output line used to request MCU asynchronous non-maskable interrupts.
35	AS	ADDRESS STROBE – Active-high output signal which indicates that a valid address is on the address bus. EVB2LA input signal.

**Table 6-2. EVB2CPU/EVB2LA Connector P2 Pin Assignments (continued)**

Pin Number	Signal Mnemonic	Signal Name and Description
36	$\overline{\text{LIR}}$	LOAD INSTRUCTION REGISTER — An active low output line used to signify the first E-clock cycle of each instruction cycle and remains low for the duration of cycle (opcode fetch).— EVB2LA input signal
37	E	MCU E-CLOCK SIGNAL — EVB2LA input signal
38	$\text{R}/\overline{\text{W}}$	READ/ $\overline{\text{WRITE}}$ — EVB2LA input signal from the MCU
39 – 46	D0 – D7	DATA/ADDRESS BUS (A0/D0 – A7/D7) — EVB2LA input signals from the MCU
47 – 48	VCC	SYSTEM POWER SUPPLY – +5 Vdc power source, EVB2LA input signal
49 – 50	GND	GROUND

**Table 6-3. RS-232 HOST Port EVB2CPU P5 Pin Assignments**

Pin Number	Signal Mnemonic	Signal Name and Description
1,4,6 – 9	—	Not connected.
2	RXD	RECEIVE DATA — Serial data input line.
3	TXD	TRANSMIT DATA — Serial data output line.
5	SIG-GND	SIGNAL GROUND — This line provides signal ground or common return connection between the EVM and RS-232 compatible terminal. This line establishes the common ground reference potential between the EVM and RS-232 compatible terminal circuitry.



**Table 6-4. RS-232 TERMINAL Port EVB2CPU P6 Pin Assignments**

Pin Number	Signal Mnemonic	Signal Name and Description
1	GND	FRAME GROUND
2	TXD	TRANSMIT DATA — Serial data output line.
3	RXD	RECEIVE DATA — Serial data input line.
4	RTS	REQUEST TO SEND — An input signal used to request permission to transfer data.
5	CTS	CLEAR TO SEND — An output signal used to indicate a ready-to-transfer data status.
6	DSR	DATA SET READY — An output signal (held high) used to indicate an on-line/in-service/active status.
7	SIG-GND	SIGNAL GROUND — This line provides signal ground or common return connection between the EVM and RS-232 compatible terminal. This line establishes the common ground reference potential between the EVM and RS-232 compatible terminal circuitry.
8	DCD	DATA CARRIER DETECT — An output signal (held high) used to indicate an acceptable carrier signal has been detected.
20	DTR	DATA TERMINAL READY — An output line (held high) used to indicate an on-line/in-service/active status.
9 – 19, 21 – 25	—	Not connected.

**Table 6-5. EVB2LA Connector P1 Test Point Pin Assignments**

Pin Number	Signal Mnemonic	Signal Name and Description
T1 – T6	T1 – T6	TEST POINTS 1 – 6 — User-selectable EVB2LA input lines. These test points are not connected to the EVB2.
GND	GND	Ground

## 6.3 PARTS LISTS

Tables 6-11 and 6-12 are parts lists for the EVB2CPU and EVB2LA.

**Table 6-6. EVB2CPU Parts List**

Reference Designation	Component Description
C1, C2	Capacitor, tantalum, 22 $\mu$ F, $\pm$ 10%, @ 25 Vdc
C3, C4	Capacitor, 22pF, $\pm$ 10%, @50 Vdc
C5, C8, C9, C10, C11, C16, C17, C18, C19, C20, C21	Capacitor, 0.1 $\mu$ F, $\pm$ 10%, @25 Vdc
C6	Capacitor, .01 $\mu$ F, $\pm$ 10%, @ 25 Vdc
C7, C101	Capacitor, 1.0 $\mu$ F, $\pm$ 10%, @ 50 Vdc
C12, C13, C14, C15	Capacitor, tantalum, 10 $\mu$ F, $\pm$ 10%, @ 16 Vdc
C22	Capacitor, 10pF, $\pm$ 10%, @ 50 Vdc
GG1	Ground grabber (1" bare solid tinned #18 wire)
J1, J4, J11, J12, J13, J14, J15, J16	Header, 3-pin, single row
J2, J3, J5, J6, J7, J9, J10, J18, J19	Header 2-pin, single row
L1	Low current red LED lamp, T-1 package
P1	Header, double row post, 60 pin
P2	Header, double row post, 50 pin
P3(1)	3.5 mm PCB-mount phone jack, Mouser #16PJ528
P3A(1)	AC-to-DC adaptor, 9V @ 200 mA, Digi-Key #T401-ND
P4	Power terminal, Augat RDI 2SV-02, 25 Volts, with red and black levers
P5	DB9S connector surface mount, female, ultra short body, AMP #745781-1 or -2, or AMP #745131-1 or -2
P6	DB25S connector surface mount, female, AMP # 206584-2

**Table 6-6. EVB2CPU Parts List (continued)**

Reference Designation	Component Description
Q1	MC34064 low-voltage detection circuit
R1, R6	Resistor, 1K ohms, $\pm 5\%$ , @1/4W
R2, R3, R4	Resistor, 4.7K ohms, $\pm 5\%$ , @1/4W
R5	Resistor, 10M ohms, $\pm 5\%$ , @1/4W
R7, R8, R10, R11, R12, R13	Resistor, 10K ohms, $\pm 5\%$ , @1/4W
R9	Resistor, 3.3K ohms, $\pm 5\%$ , @1/4W
SW1	Switch, SPDT, miniature slide , C&K #1101
SW2, SW3	Switch, SPDT, momentary, C&K 8125-R2/7527
U1	I.C. MC68HC24FN Port Replacement Unit
U2	I.C. MC68HC11E1FN Microcontroller
U3, U4	I.C. 74HC138 decoder
U5	I.C. MC145407P RS-232 chip
U6	I.C. 74HC30 NAND gate
U7	I.C. 74HC04 inverter
U8	I.C. 74HC373 latch
U9	I.C. 27C128 (250 ns)
U10	I.C. 27C256 (250 ns)
U11	I.C. MCM60256 CMOS static 32K x 8 RAM chip, Motorola
U12	I.C. MC2681P
VR1 <sup>(1)</sup>	AN7805, 5 Vdc, 1 Amp voltage regulator
Y1	8 MHz ceramic resonator
Y2	3.6864 MHz crystal
	Circuit board supports, four, SPC Tech #TPCS-4
	Threaded spacers, four, 1 inch, 8-32 tapped
1. User-supplied optional parts	

**Table 6-6. EVB2CPU Parts List (continued)**

Reference Designation	Component Description
	Fabricated jumper, Aptronics # 929955-00 (use with jumper headers J15 and J16)
	I.C. socket, 44-pin PLCC , AMP 641747-2 (for U1)
	I.C. socket, 52-pin PLCC , AMP 641748-2 (for U2)
	I.C. socket, 20-pin DIP (for U5)
	I.C. socket, 28-pin DIP (for U9, U10, and U11)
	I.C. socket, 40-pin DIP (for U12)
	Heat sink <sup>(1)</sup> , AAVID #5772B (for VR1)
	Screw <sup>(1)</sup> , 4-40 x 1/4" (for VR1)
	Hex nut <sup>(1)</sup> , 4-40 (for VR1)
1. User-supplied optional parts	

**Table 6-7. EVB2LA Parts List**

Reference Designation	Component Description
C1, C2, C3, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16, C17, C18, C19, C20, C21, C22, C23	Capacitor, 0.1 $\mu$ F, $\pm$ 10%, @25 Vdc
C4	Capacitor, 4.7 $\mu$ F, $\pm$ 10%, @ 10 Vdc
C24	Capacitor, 22pF, $\pm$ 10%, @ 50 Vdc
GG1	Ground grabber (1" bare solid tinned #18 wire)
P1	Header, pin-7, 100 mil. center, right-angle
P2	Header, double row post, 50 pin
R2	Resistor, 2K ohms, $\pm$ 5%, @ 1/4W
RN1, RN4	Resistor network, 8-pin SIP, 10K ohms
RN2, RN3	Resistor network, 6-pin SIP, 10K ohms
RN5	Resistor network, 10-pin SIP, 3.3K ohms
R100	Resistor, 10K ohms, $\pm$ 5%, @ 1/4W
U1, U2, U3, U4, U10, U11, U12	74HC541 three-state buffer
U5, U6, U7, U8	6264AP-12 8Kx8 CMOS RAM
U9	6267P-45 16Kx1 CMOS RAM
U13	74HC139 Decoder
U14	74HC00 NAND gate
U15	74HC09 Open collector AND gate
U16	74HC4020 13-bit counter
U17, U19, U20, U21, U22	MC74AC169P 4-bit counter, Motorola
U18	Gould/AMI PEEL273 (30ns) CMOS FPLA
	Cable Assembly: AMP 50-pin socket connector 3" long 50-conductor ribbon cable
	I.C. socket, 28-pin DIP, (for U5, U6, U7, and U8)
	I.C. socket, 24-pin DIP, 300 mil, (for U18)
	I.C. socket, 20-pin DIP (for U4 and U9)

## **6.4 SCHEMATIC DIAGRAMS**

Figure 6-1 is the EVB2CPU schematic and Figure 6-2 is the EVB2LA schematic.

**Figure 6-1. M68HC11EVB2CPU Schematic Diagrams (1 of 6)**





**Figure 6-1. M68HC11EVB2CPU Schematic Diagrams (2 of 6)**



**Figure 6-1. M68HC11EVB2CPU Schematic Diagrams (3 of 6)**



**Figure 6-1. M68HC11EVB2CPU Schematic Diagrams (4 of 6)**



**Figure 6-1. M68HC11EVB2CPU Schematic Diagrams (5 of 6)**





**Figure 6-1. M68HC11EVB2CPU Schematic Diagrams (6 of 6)**



**Figure 6-2. M68HC11EVB2LA Schematic Diagrams (1 of 6)**



**Figure 6-2. M68HC11EVB2LA Schematic Diagrams (2 of 6)**



**Figure 6-2. M68HC11EVB2LA Schematic Diagrams (3 of 6)**





**Figure 6-2. M68HC11EVB2LA Schematic Diagrams (4 of 6)**



**Figure 6-2. M68HC11EVB2LA Schematic Diagrams (5 of 6)**



**Figure 6-2. M68HC11EVB2LA Schematic Diagrams (6 of 6)**



## APPENDIX A

### S-RECORD INFORMATION

#### A.1 INTRODUCTION

The S-record format for output 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 strings 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:

<b>TYPE</b>	<b>RECORD LENGTH</b>	<b>ADDRESS</b>	<b>CODE/DATA</b>	<b>CHECKSUM</b>
-------------	----------------------	----------------	------------------	-----------------

where the fields are composed as follows:

Field	Printable Characters	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-records 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 EVB2 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 types:

- |       |  |
|-------|--|
| 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 EVB2.  |
| 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.



## A.4 S-RECORD CREATION

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.5 S-RECORD EXAMPLE

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

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

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

The S0 header record is comprised of 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	Four-character 2-byte address field, zeroes.
00	
48	
44	ASCII H, D, and R - "HDR".
52	
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, indicates 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:

<u>OPCODE</u>	<u>INSTRUCTION</u>
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

. (Balance of this code is continued in the 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 S1 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.

Each printable character in an S-record is encoded in hexadecimal (ASCII in this example) representation of the binary bits which are actually transmitted. For example, the first S1 record above is sent as shown below.

TYPE		LENGTH			ADDRESS				CODE/DATA						CHECKSUM													
S	1	1	3		0	0	0	0	2	8	5	F	...	2	A													
5	3	3	1	3	1	3	3	3	0	3	0	3	0	3	0	3	2	3	8	3	5	4	6	...	3	2	4	1
0101	0011	0011	0001	0011	0001	0011	0011	0011	0000	0011	0000	0011	0000	0011	0000	0011	0010	0011	1000	0011	0101	0100	0110	...	0011	0010	0100	0001

## APPENDIX B

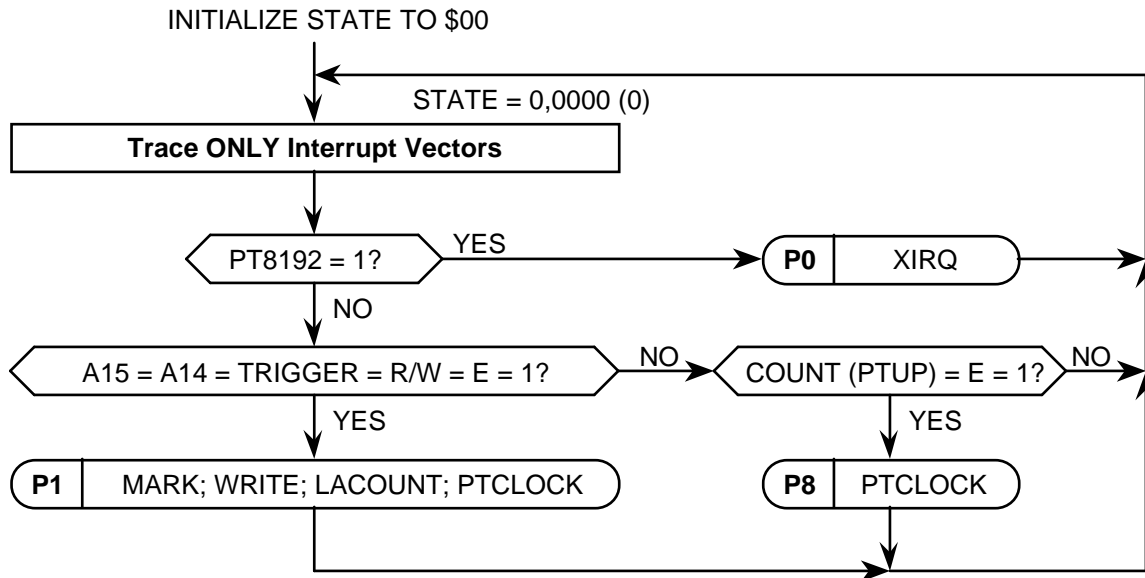
### PROGRAMMABLE ARRAY LOGIC FLOW DIAGRAMS

#### B.1 TRACE ONLY INTERRUPT VECTORS

Before calling...

Initialize MARK bits, Reset LA and PT counters,

Load Trigger RAM to detect even addresses between \$FFDE – \$FFFE



P0 - Causes return to monitor when capture RAM full

P1 - Causes capture of one bus state

P8 - Used to initialize post-trigger counter

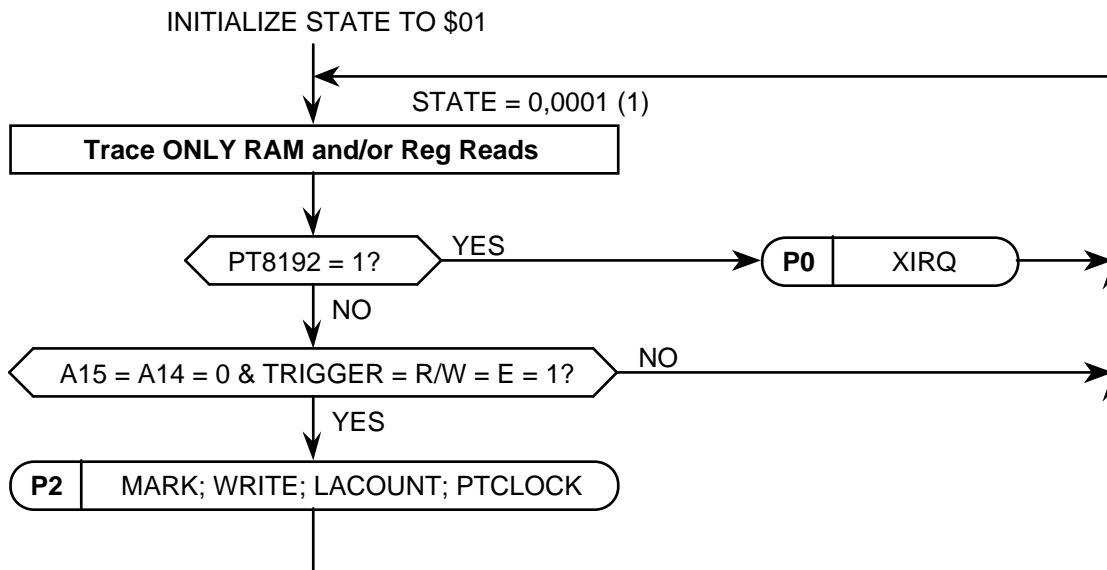
## B.2 TRACE ONLY READS OF RAM OR REGISTERS

Before calling...

Initialize MARK bits, Reset LA and PT counters,

Load Trigger RAM to detect addresses in the range...

\$0000 – \$0FFF or \$1000 – \$1FFF or \$0000 – \$1FFF



P0 - Causes return to monitor when capture RAM full

P2 - Causes capture of one bus state

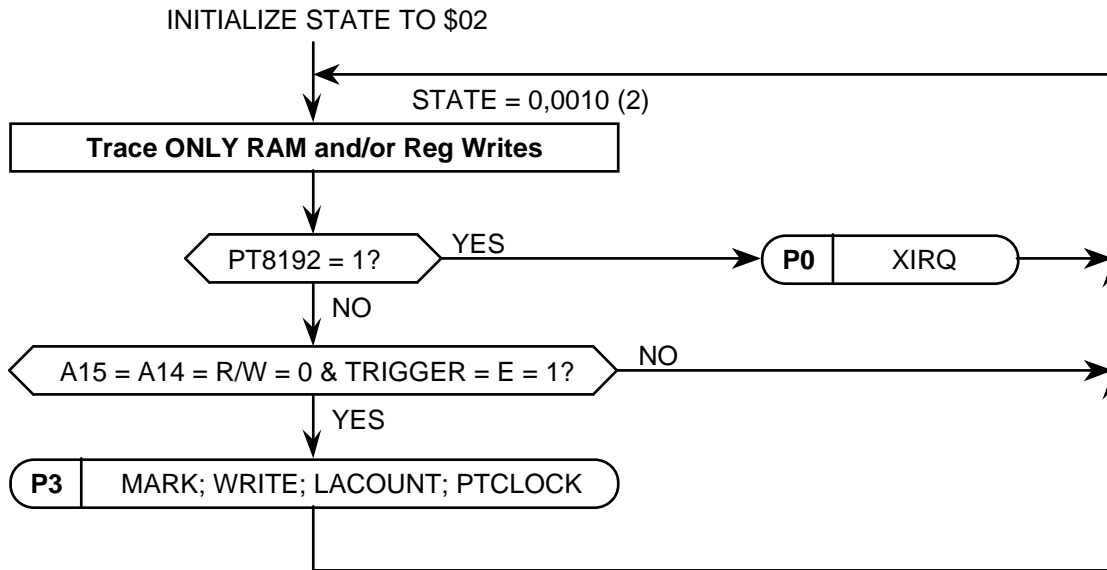
### B.3 TRACE ONLY WRITES TO RAM OR REGISTERS

Before calling...

Initialize MARK bits, Reset LA and PT counters,

Load Trigger RAM to detect addresses in the range...

\$0000 – \$0FFF or \$1000 – \$1FFF or \$0000 – \$1FFF



P0 - Causes return to monitor when capture RAM full

P3 - Causes capture of one bus state

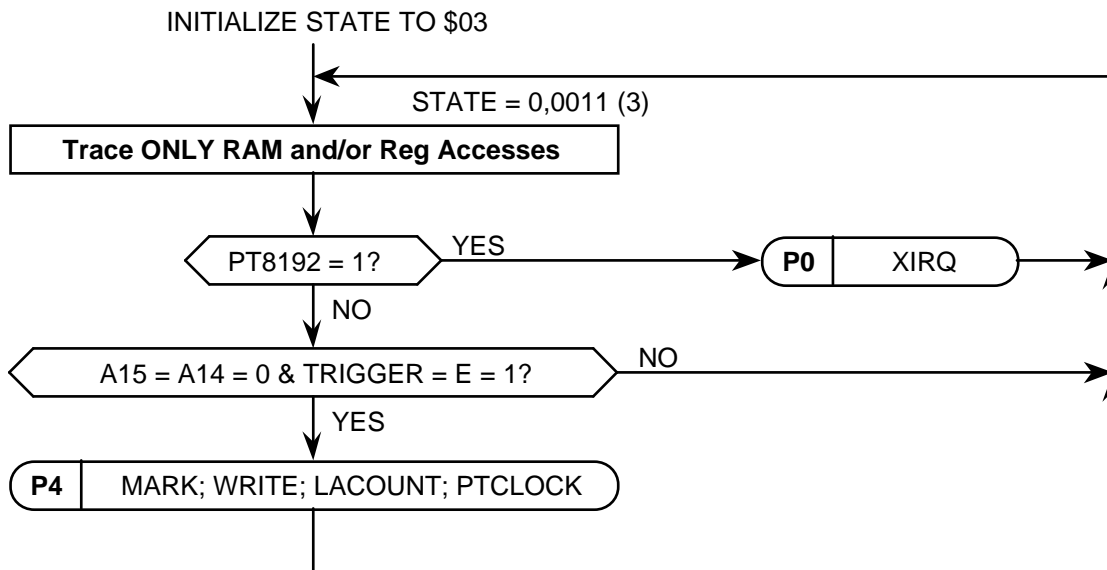
## B.4 TRACE ONLY ACCESSES OF RAM OR REGISTERS

Before calling...

Initialize MARK bits, Reset LA and PT counters,

Load Trigger RAM to detect addresses in the range...

\$0000 – \$0FFF or \$1000 – \$1FFF or \$0000 – \$1FFF



P0 - Causes return to monitor when capture RAM full

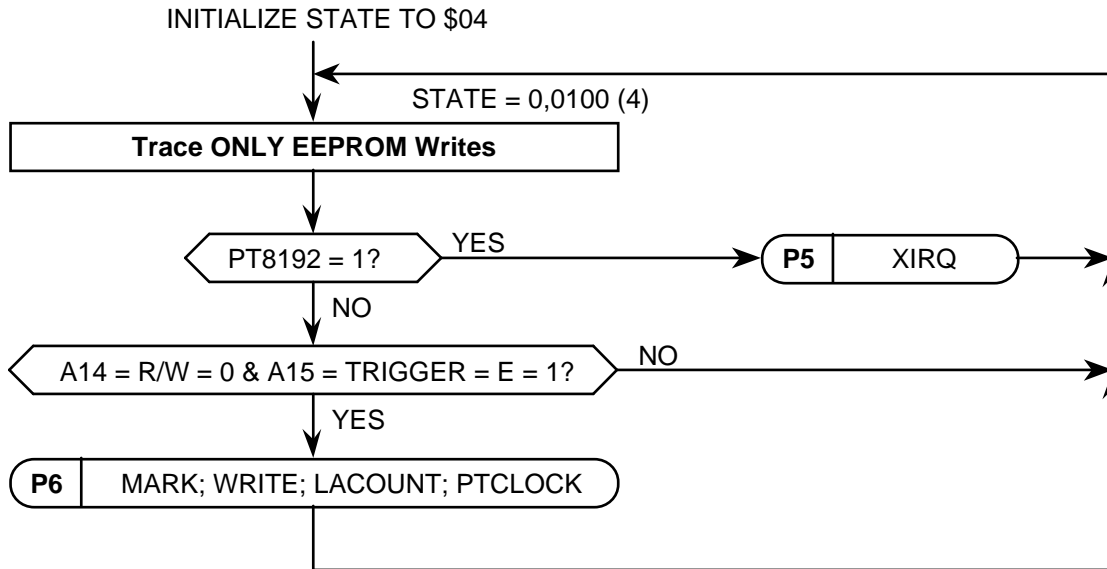
P4 - Causes capture of one bus state

## B.5 TRACE ONLY WRITES TO EEPROM

Before calling...

Initialize MARK bits, Reset LA and PT counters,

Load Trigger RAM to detect addresses in the range \$B600 – \$B7FF



P5 - Causes return to monitor when capture RAM full

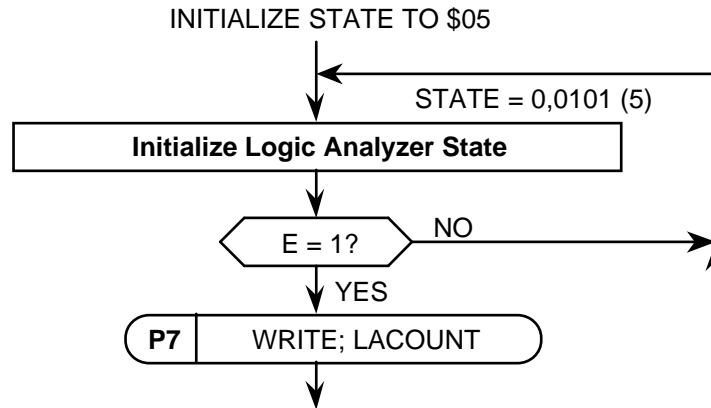
P6 - Causes capture of one bus state

## B.6 INITIALIZE LOGIC ANALYZER STATE

Write xx to "PTRESET" to reset post-trigger counter.

Write \$00 or \$01 to "TRRAM" through "TRRAM + 16383" to load trigger RAM (1=Trigger). 16K trigger RAM mirrors due to A14 and A15 not connected.

By remaining in this state for more than 4.1 milliseconds, all 8192 "MARK" bits will be written to zero.



P7 - Performs writes to capture buffer with "MARK" = 0



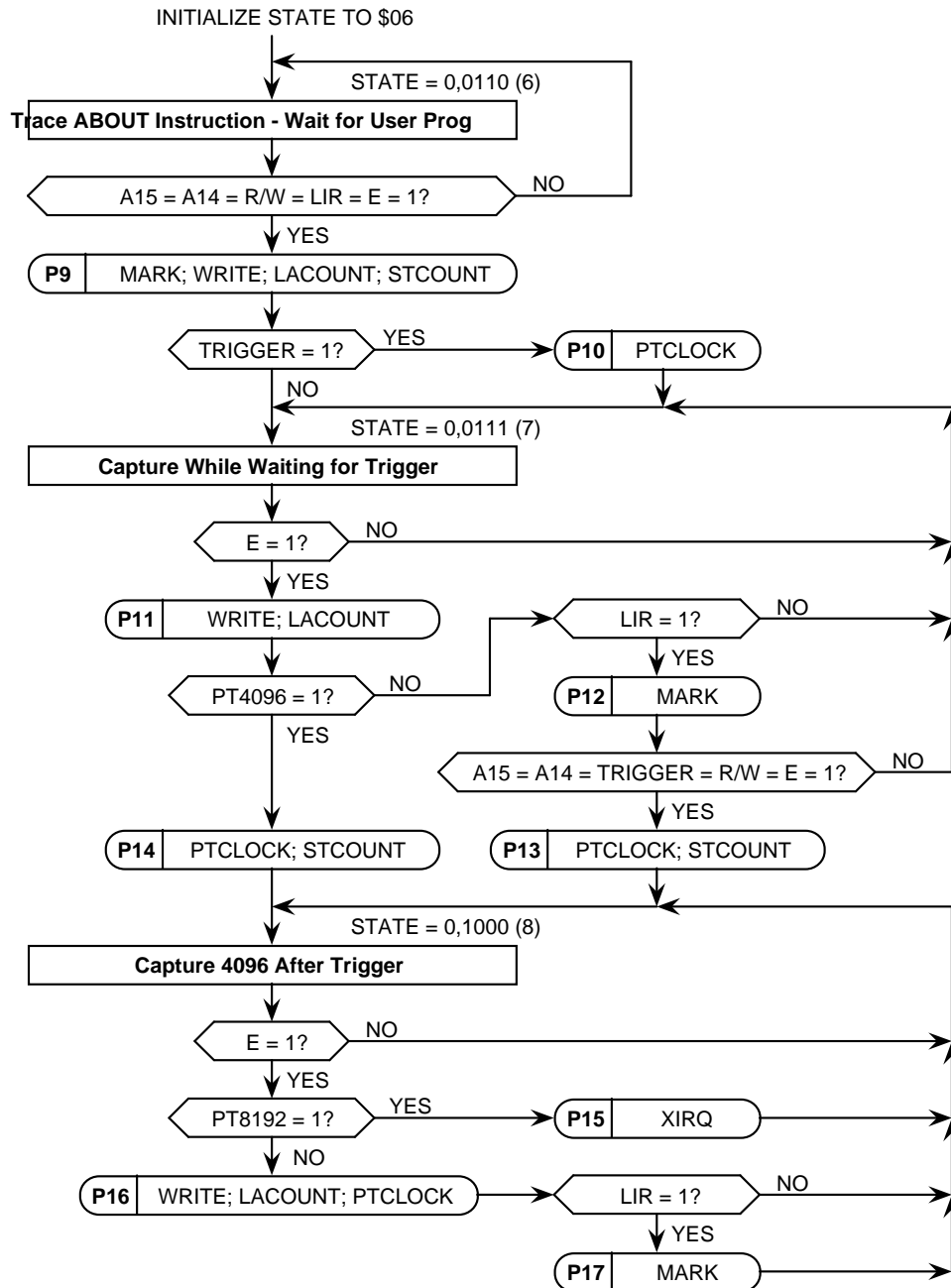
## B.7 TRACE ABOUT EXECUTION OF SELECTED USER INSTRUCTION

Before calling...

Initialize MARK bits, Reset LA and PT counters,

Load Trigger RAM to detect address of selected User Instruction

Initialize post-trigger counter to 4095

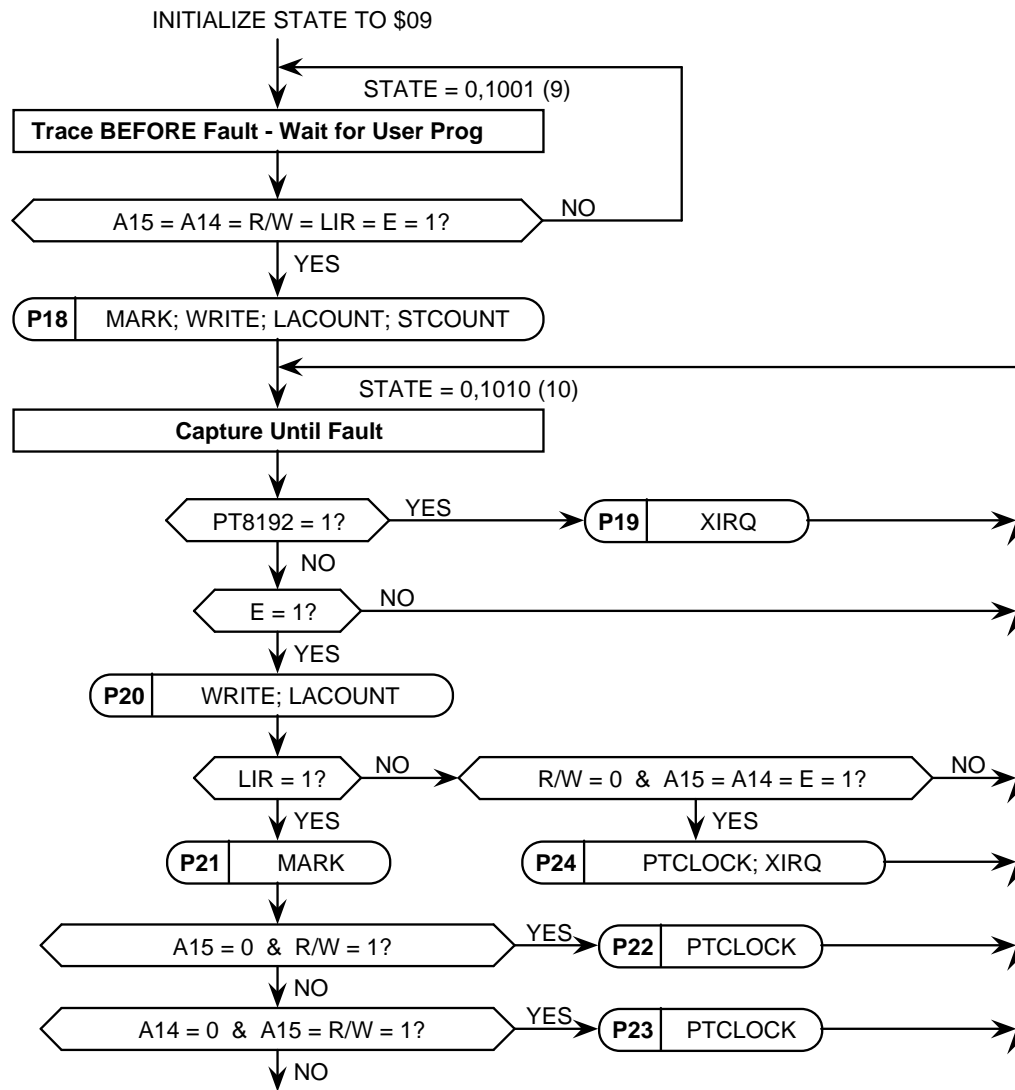


## B.7 TRACE BEFORE "INSTRUCTION FETCH OUT-OF-RANGE" FAULT OR "WRITE TO USER PROGRAM ADDRESS" FAULT

Before calling...

Initialize MARK bits, Reset LA and PT counters,

Advance post-trigger counter to 8191



P19 - Causes return to monitor when a failure is detected

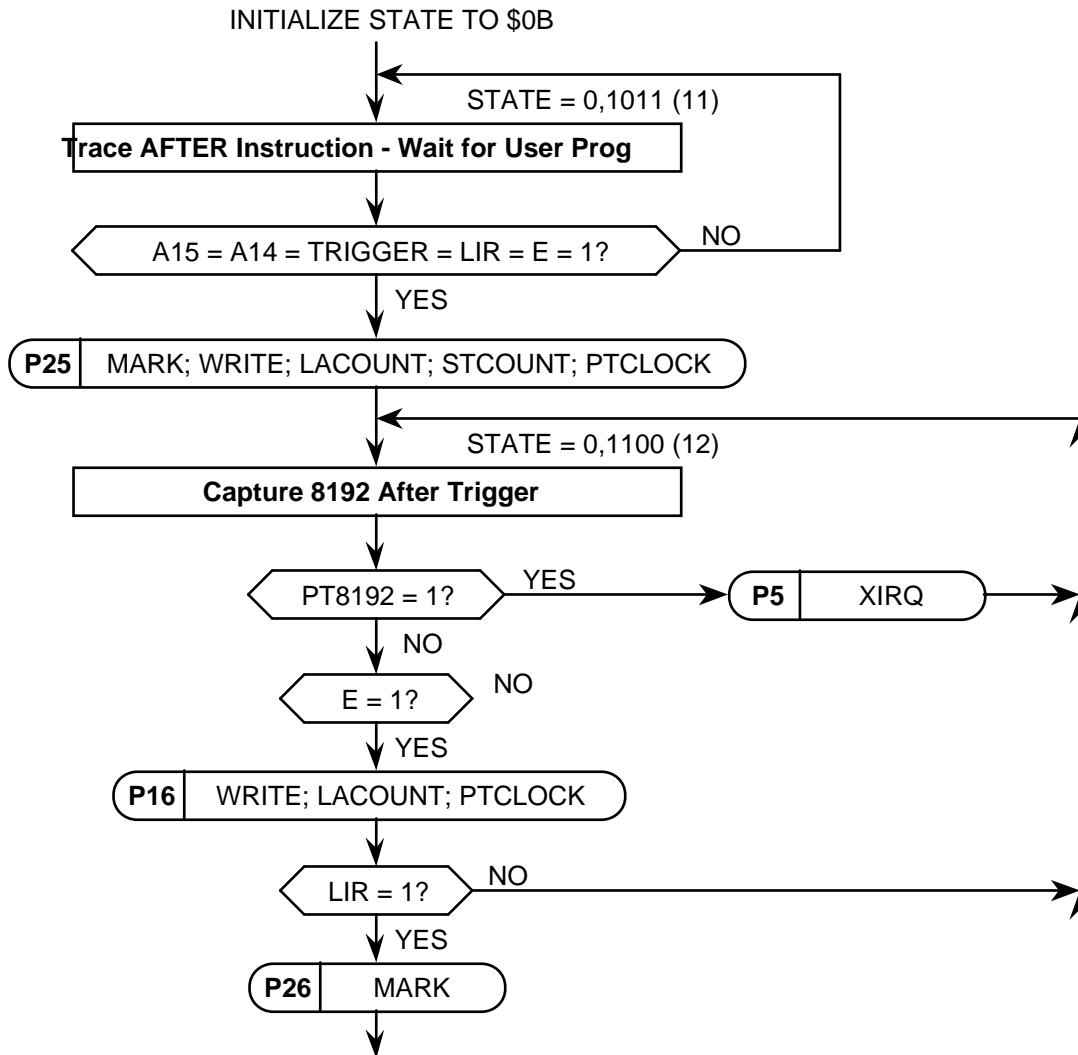
P22 - Detects attempt to fetch an opcode from below 8000

P23 - Detects attempt to fetch an opcode from 8000 – BFFF

P24 - Detects an attempt to write to program memory space and forces return to monitor

## B.8 TRACE AFTER EXECUTION OF SELECTED USER INSTRUCTION

Before calling...  
 Initialize MARK bits, Reset LA and PT counters,  
 Load trigger to detect selected user instruction



P5 - Causes return to monitor when capture buffer full

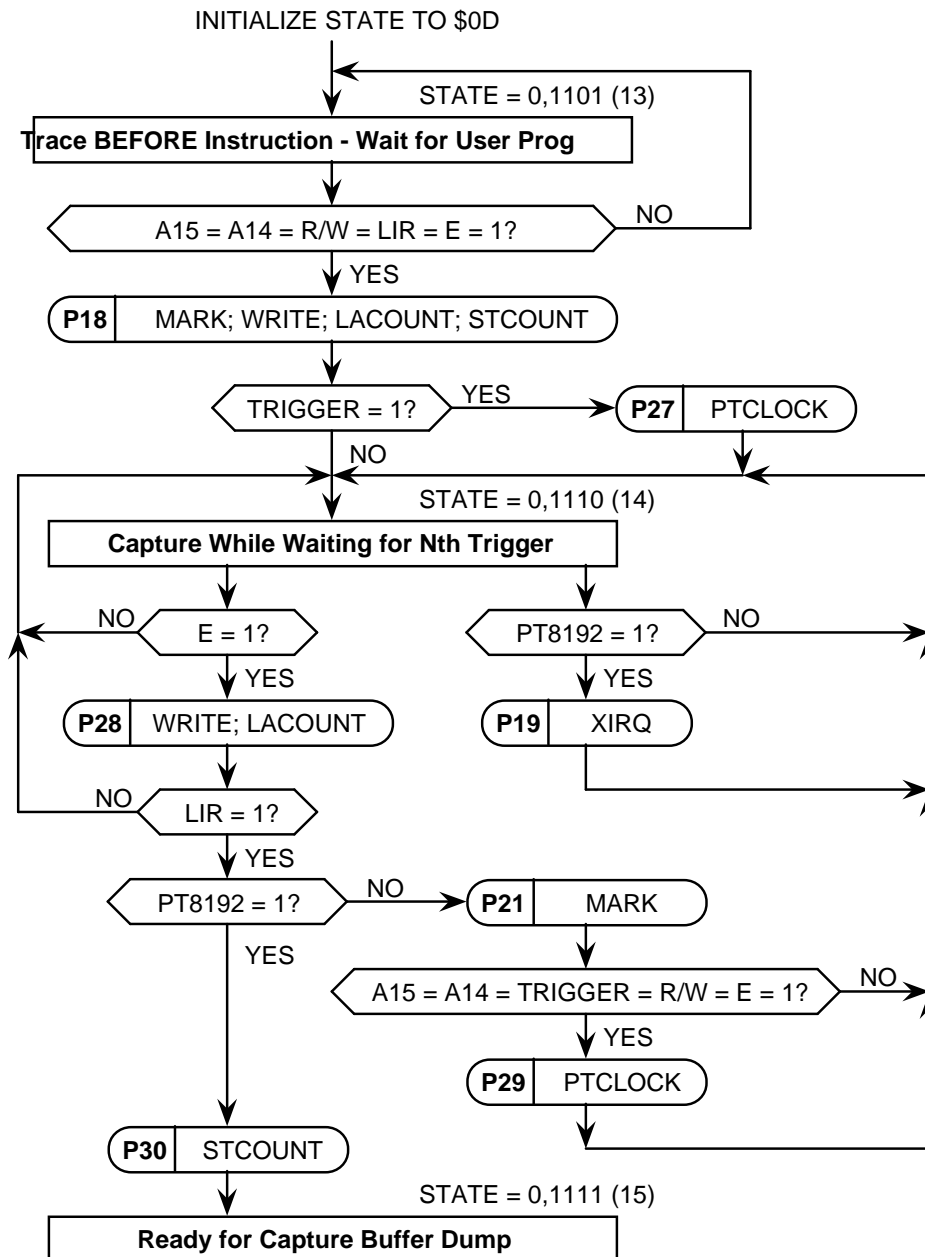
## B.9 TRACE BEFORE <n>TH EXECUTION OF SELECTED USER INSTRUCTION

Before calling...

Initialize MARK bits, Reset LA and PT counters,

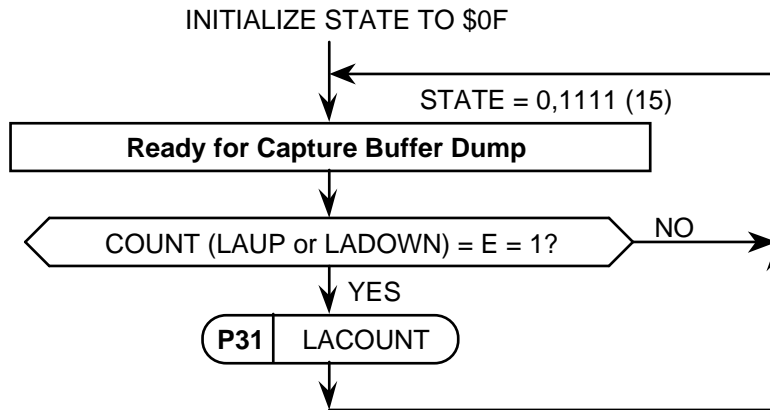
Load Trigger RAM to detect address of selected User Instruction

Initialize post-trigger counter to (8192 - N)



P19 - Causes return to monitor after N occurrences of user instruction

## B.10 DUMP CAPTURED DATA



Read "LADATA" to read data bus byte pointed to by LA counter  
 Read "LALOADD" to read low address byte pointed to by LA counter  
 Read "LAHIADD" to read high address byte pointed to by LA counter  
 Read "LAAUX" to read auxiliary byte pointed to by LA counter  
 Read "LADOWN" to decrement LA counter  
 Read "LAUP" to increment LA counter

Write to "PTRESET" to reset post-trigger counter

Enter and exit the "nibble serial mode" from this state (0F)

Write \$80 to "NIBMODE"

Read "SHIFT" to read TRIGGER, PT8192, PT4096, PTCLOCK; S8, S4, S2, S1

Write new initial state to "SHIFT"; shifts in one nibble

Read "SHIFT" to read LA count bits 3, 2, 1, 0

Write new LACOUNT bits 3, 2, 1, 0 to "SHIFT"

Read LACOUNT bits 7, 6, 5, 4 from "SHIFT"

Write new LACOUNT bits 7, 6, 5, 4 to "SHIFT"

Read LACOUNT bits 11, 10, 9, 8 from "SHIFT"

Write new LACOUNT bits 11, 10, 9, 8 to "SHIFT"

Read LACOUNT bit 12 from "SHIFT"

Write new (ENABLE T, ENABLE A, 0, LACOUNT bit 12) to "SHIFT"

Write \$80 to "NIBMODE"

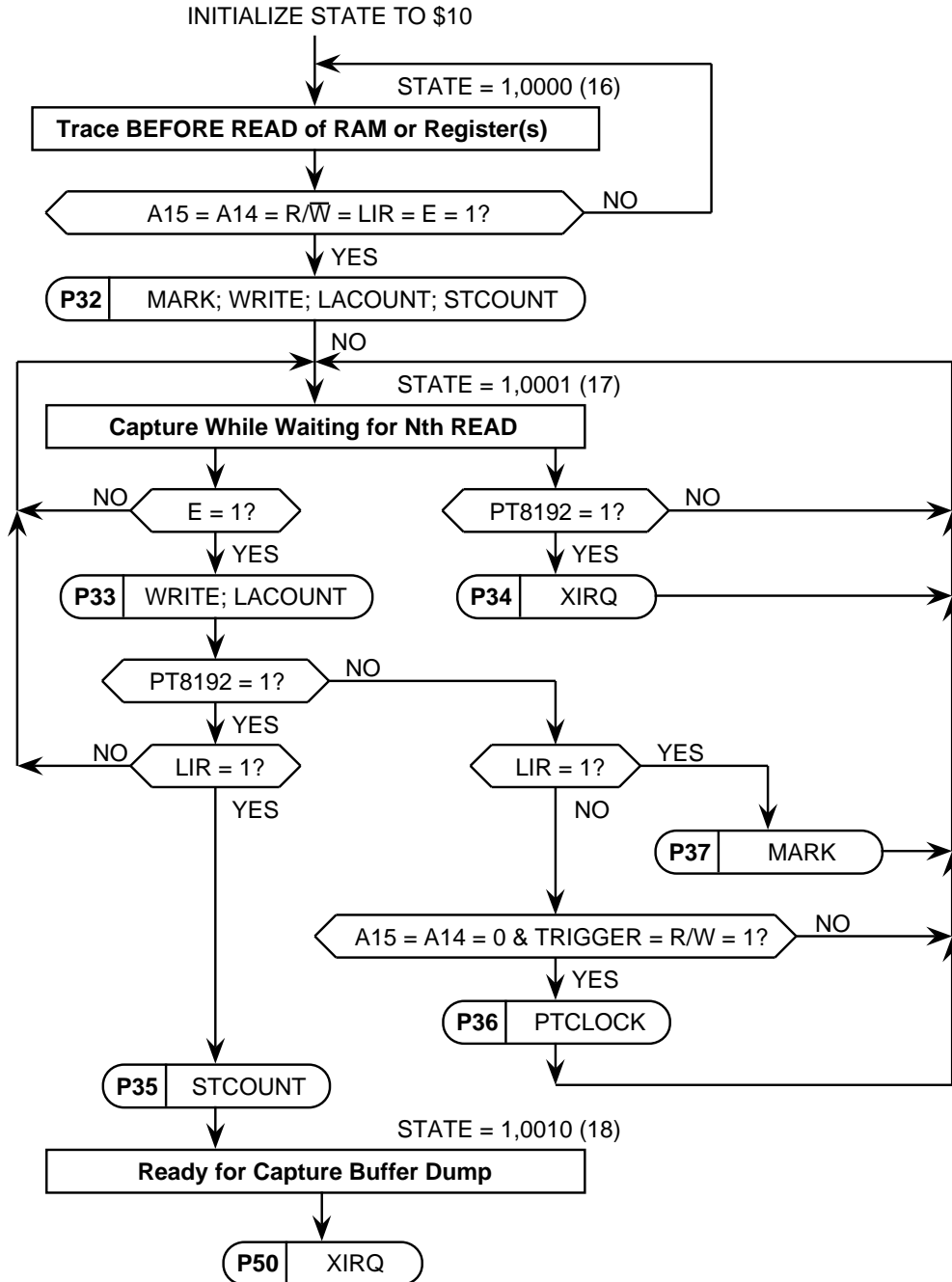
### B.11 TRACE BEFORE <n>TH READ OF SELECTED RAM OR REGISTER(S)

Before calling...

Initialize MARK bits, Reset LA and PT counters,

Load Trigger RAM to detect address(s) in the range \$0000-\$1FFF

Initialize post-trigger counter to (8192 - N)



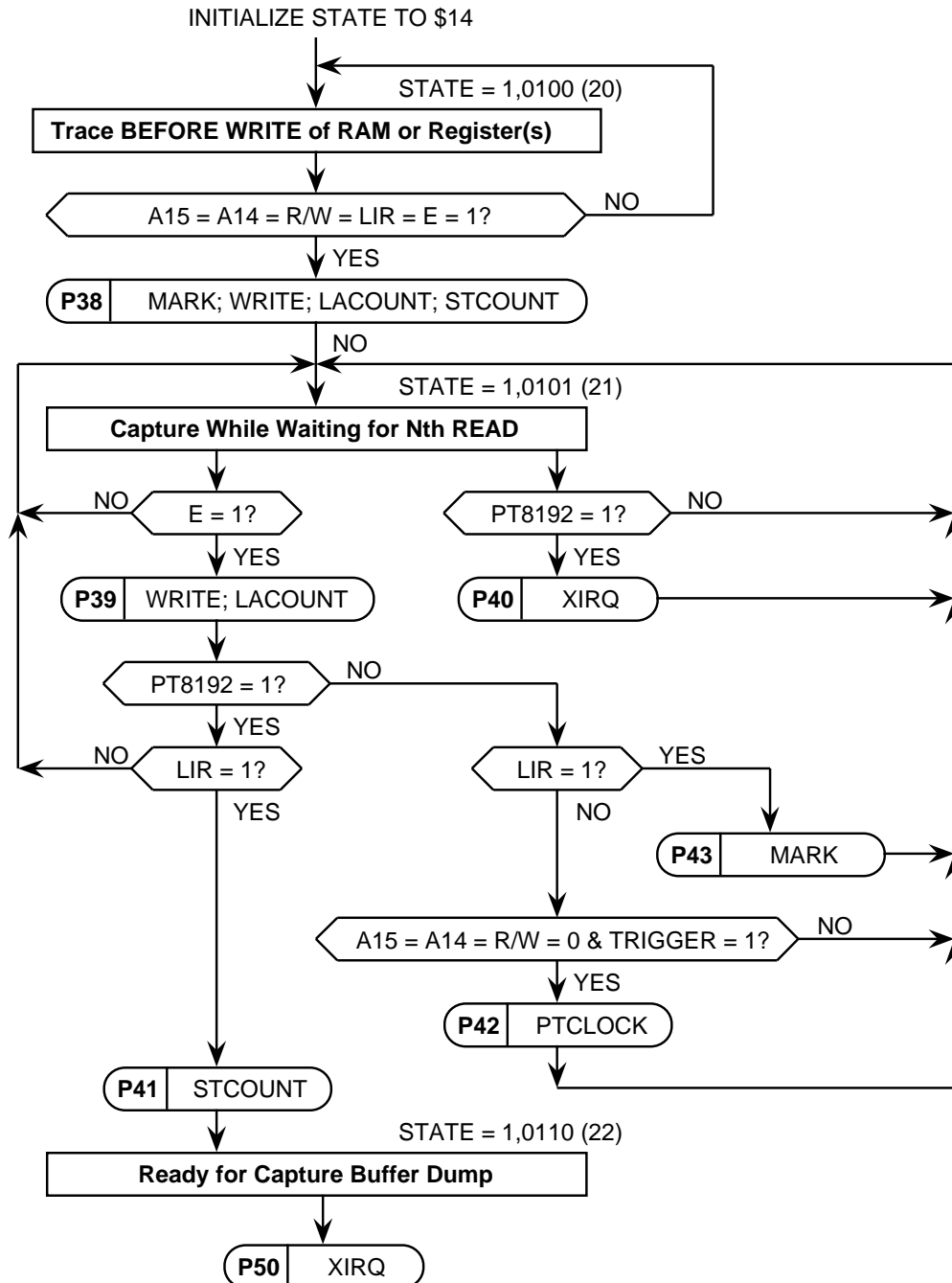
## B.12 TRACE BEFORE <n>TH WRITE OF SELECTED RAM OR REGISTER(S)

Before calling...

Initialize MARK bits, Reset LA and PT counters,

Load Trigger RAM to detect address(s) in the range \$0000-\$1FFF

Initialize post-trigger counter to (8192 - N)



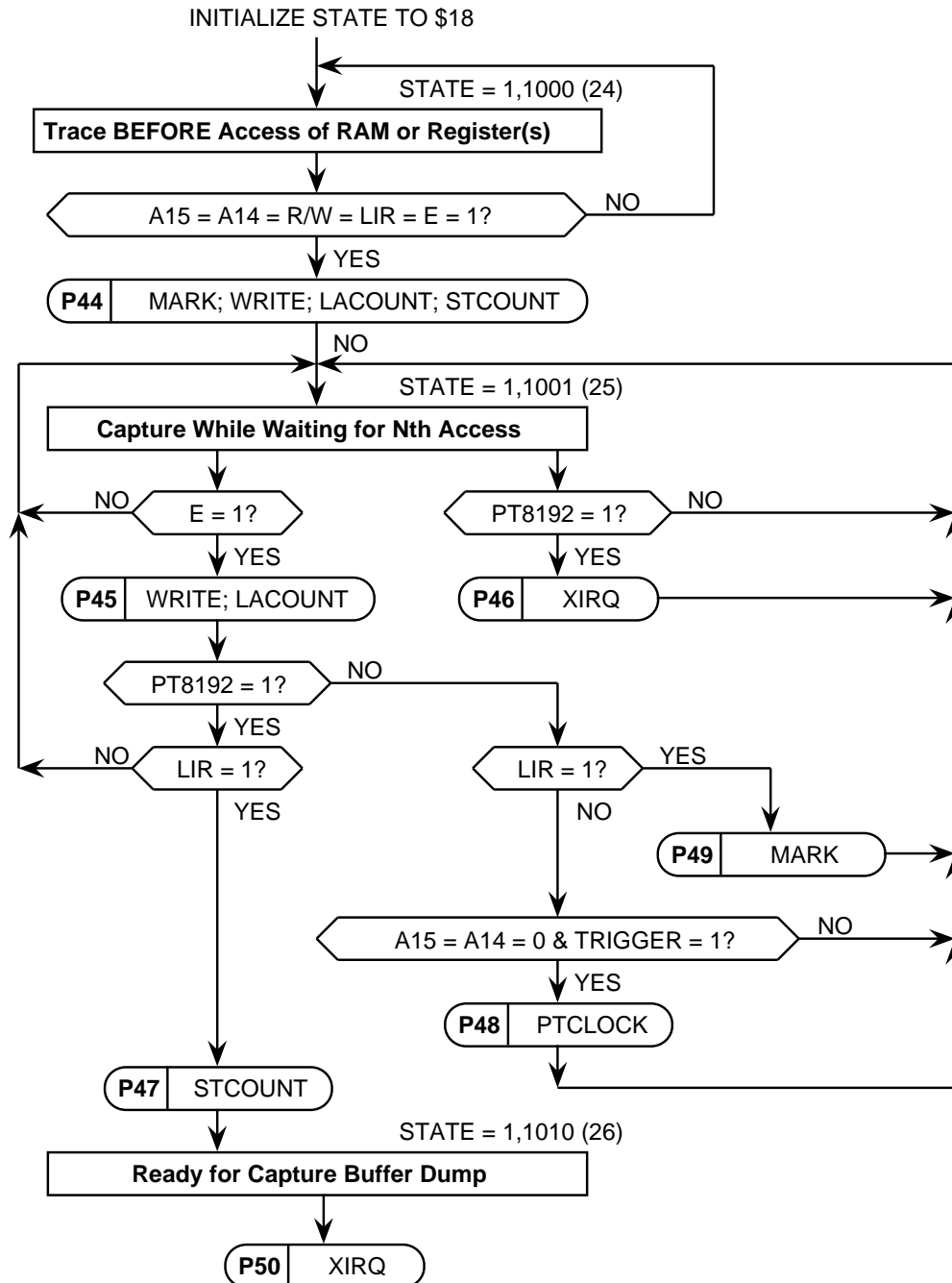
### B.13 TRACE BEFORE <n>TH ACCESS OF SELECTED RAM OR REGISTER(S)

Before calling...

Initialize MARK bits, Reset LA and PT counters,

Load Trigger RAM to detect address(s) in the range \$0000-\$1FFF

Initialize post-trigger counter to (8192 - N)





---

---

## APPENDIX C

### USING AN UNREGULATED POWER SUPPLY

#### C.1 INTRODUCTION

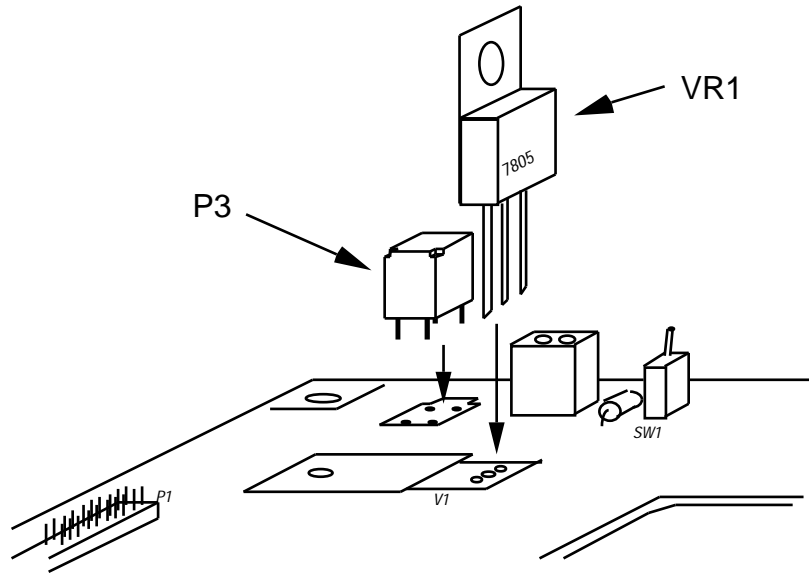
This appendix is the procedure for using an unregulated power supply with EVB2. An unregulated power supply lets you cut cost of your evaluation system by offering a low-cost alternative to a regulated power supply. The optional parts covered in this appendix, e.g. VR1, heat sink, phone jack, and unregulated power supply, are user-supplied. Refer to Table C-1 for the unregulated power supply specifications.

**Table C-1. Unregulated Power Supply Specifications**

<b>Equipment</b>	<b>Supplier</b>	<b>Connector</b>	<b>Voltage</b>	<b>Amps</b>
Unregulated supply	DigiKey #T401-ND	P3	+9 Vdc	200 mA
Voltage regulator Heat Sink 3.5mm jack	MC7805CT AAVID #5772B Mouser #16PJ528		+5 Vdc	1A

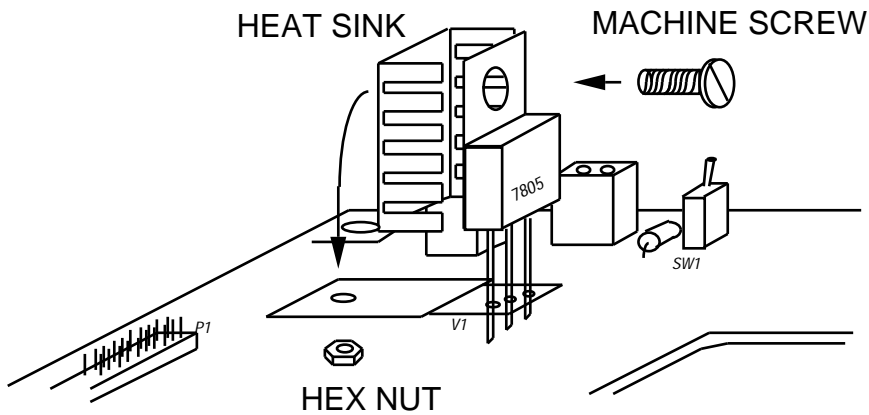
#### C.2 VOLTAGE REGULATOR INSTALLATION PROCEDURE

1. Solder the 3.5mm phone jack to the board at location P3 with the plug facing off the top of the board as shown in Figure C-1. Solder the MC7805CT voltage regulator at location VR1. Care should be taken to properly orient the voltage regulator as shown in Figure C-1.



**Figure C-1. Voltage Regulator and Phone Jack Installation**

2. Bend the voltage regulator down against the board. Install the heat sink under it and secure the two with a 4-40x1/4" machine screw and hex nut as shown in Figure C-2.



**Figure C-2. Heat Sink Installation**

After the preceding changes have been made, connect an unregulated power supply to the 3.5mm phone jack connector P3.