Agilent Technologies

Dear Customer:

You have probably heard from news reports and from your sales representative that as of November 1, 1999, four of Hewlett-Packard's businesses became a new company -- Agilent Technologies. The new company includes the following former HP businesses: test and measurement, semiconductor products, healthcare solutions and chemical analysis."

We at Agilent Technologies are working diligently to make this transition as seamless as possible for you, however, we are not able to make all changes immediately. As a result, the products and related documentation may be labeled with either the Hewlett-Packard name and logo or the Agilent Technologies name and logo. Rest assured that whatever logo you see, the information, products and services come from the same reliable source.

In addition, it is our sincere intent that the transition from Hewlett Packard to Agilent Technologies should have no impact on your warranties, service levels, or purchase volume credits.

For more information about this transition, please visit our website at: http://www.agilent.com, or contact your local sales representative. It has been our pleasure to work with you for the past 60 years as part of Hewlett-Packard. We look forward to continuing to serve you as Agilent Technologies for years to come.

HP E3471

H8S/2000 Emulator Terminal Interface

User's Guide

HEWLETT PACKARD

HP Part No. E3471-97000
Printed in Japan
July 1996

Edition 1

Notice

Hewlett-Packard Company
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901-2197, U.S.A.

Printing History

New editions are complete revisions of the manual. The date on the title page changes only when a new edition is published.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual revisions.

Edition 1   **E3471-97000, July 1996**

Safety Summary

The following general safety precautions must be observed during all phases of operation, service, and repair of this instrument. Failure to comply with these precautions or with specific WARNINGS elsewhere in this manual may impair the protection provided by the equipment. In addition it violates safety standards of design, manufacture, and intended use of the instrument.
The Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Note

HP E3471A complies with INSTALLATION CATEGORY I and POLLUTION DEGREE 2 in IEC1010-1. PRODNO is INDOOR USE product.

DO NOT Operate in an Explosive Atmosphere

Do not operate the instrument in the presence of flammable gasses or fumes. Operation of any electrical instrument in such an environment constitutes a safety hazard.

DO NOT Substitute Parts or Modify Instrument

Because of the danger of introducing additional hazards, do not install substitute parts or perform unauthorized modifications to the instrument. Return the instrument to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings

Warnings , such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

Warning

Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting this instrument.

Safety Symbols

General definitions of safety symbols used in manuals are listed below.

This **Warning** sign denotes a hazard. If calls attention to a procedure, practice, condition or the like, which, if not correctly performed or adhered to, could result in injury or death to personnel.

This **Caution** sign denotes a hazard. If calls attention to a procedure, practice, condition or the like, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product.

This **Note** sign denotes important information. If calls attention to a procedure, practice, condition or the like, which is essential to highlight.

Using This Manual

This manual is designed to give you an introduction to the HP E3471 H8S/2000 Emulator. This manual will also help define how these emulators differ from other HP 64700 Emulators.

This manual will:

- give you an introduction to using the emulator
- explore various ways of applying the emulator to accomplish your tasks
- show you emulator commands which are specific to the H8S/2000 Emulator

This manual will not:

- tell you how to use each and every emulator/analyzer command (refer to the *User's Reference* manual)

Organization

Chapter 1 An introduction to the H8S/2000 emulator features and how they can help you in developing new hardware and software.

Chapter 2 A brief introduction to using the H8S/2000 Emulator. You will load and execute a short program, and make some measurements using the emulation analyzer.

Chapter 3 How to plug the emulator probe into a target system.

Chapter 4 Configuring the emulator to adapt it to your specific measurement needs.

Appendix A H8S/2000 Emulator Specific Command Syntax and Error Message

Contents

Illustrations

Tables

1

Introduction to the H8S/2000 Emulator

Introduction

The topics in this chapter include:

- Purpose of the H8S/2000 Emulator
- Features of the H8S/2000 Emulator

Purpose of the H8S/2000 Emulator

The H8S/2000 Emulator is designed to replace the H8S/2000 microprocessor in your target system so you can control operation of the microprocessor in your application hardware (usually refer to as the *target system*). The H8S/2000 emulator performs just like the H8S/2000 microprocessor, but is a device that allows you to control the H8S/2000 microprocessor directly. These features allow you to easily debug software before any hardware is available, and ease the task of integrating hardware and software.

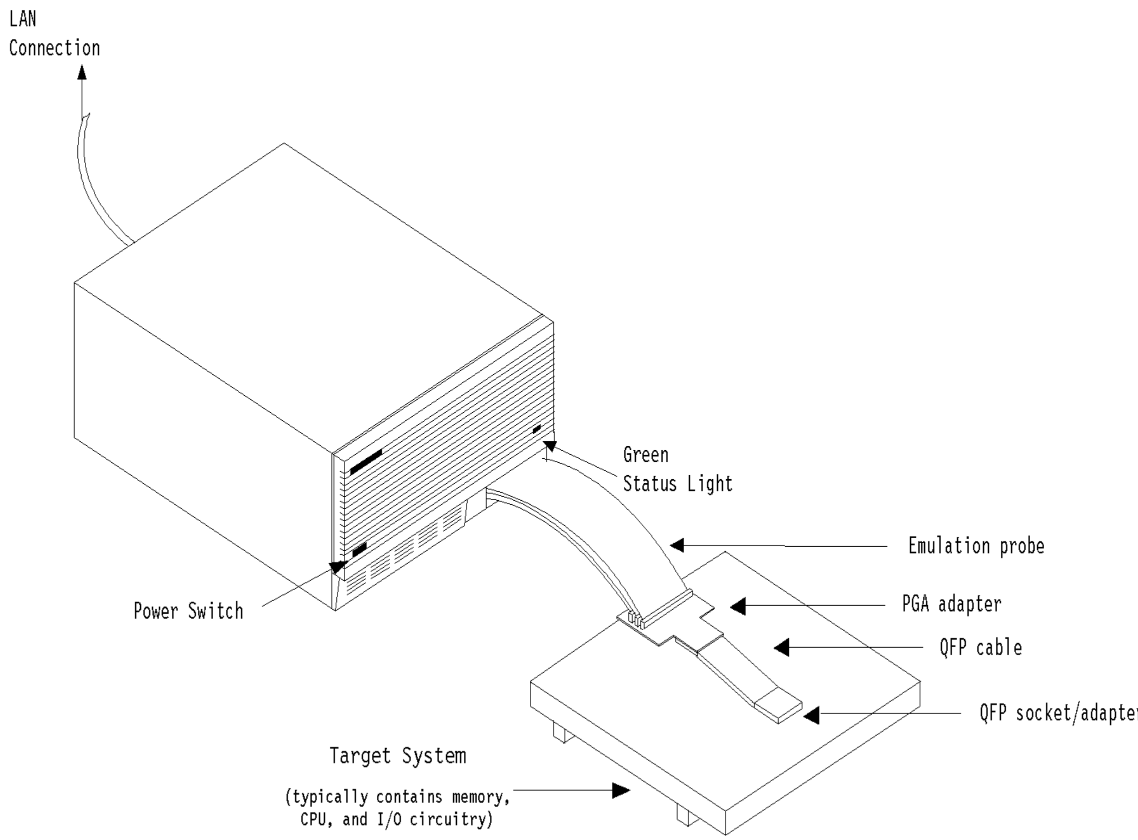


Figure 1-1. HP E3471 Emulator for the H8S/2000

Features of the H8S/2000 Emulator

Supported Microprocessors

The HP E3471 H8S/2000 emulator supports the microprocessors listed in Table 1-1.

Table 1-1. Supported Microprocessors

Supported Microprocessor		QFP Cable *1	Additional QFP Socket/Adapter
Type	Package		
H8S/2653	120pin TQFP	HP E3471B	HP E3471-61620
	128Pin QFP	HP E3471C	HP E3471-61621 *2
H8S/2655	120pin TQFP	HP E3471B	HP E3471-61620
	128pin QFP	HP E3471C	HP E3471-61621 *2
H8S/2241	100 pin TQFP	HP E3471D	HP E3471-61622
	100pin QFP	HP E3471D	HP E3471-61622
H8S/2242	100 pin TQFP	HP E3471D	HP E3471-61622
	100pin QFP	HP E3471D	HP E3471-61622
H8S/2245	100 pin TQFP	HP E3471D	HP E3471-61622
	100pin QFP	HP E3471D	HP E3471-61622
H8S/2246	100 pin TQFP	HP E3471D	HP E3471-61622
	100pin QFP	HP E3471D	HP E3471-61622

*1 The QFP cable includes one QFP socket/adapter.

*2 HP E3471-61621 QFP socket/adapter does not include the cap required for attaching a microprocessor. To attach the microprocessor, you need to purchase the cap (P/N E3471-61631), separately.

The H8S/2000 emulator is provided with a PGA adapter. To emulate each processor with your target system, you need to purchase appropriate QFP cable listed in Table 1-1. To purchase them, contact your local HP sales representative.

The list of supported microprocessors in Table 1-1 is not necessarily complete. To determine if your microprocessor is supported or not, contact Hewlett-Packard.

Clock Speeds

You can select whether the emulator will be clocked by the internal clock source or by the external clock source on your target system. When you select a clock input as external, you need to conform to the specification of Table 1-2.

Table 1-2. Clock Speeds

Clock source	With 64700A	With 64700B
Internal	10MHz (System clock)	10MHz (System clock)
External	From 2.0MHz up to 20MHz (System clock)	From 33kHz up to 20MHz (System clock)

Note

When the emulator is connected to the target system operating at low voltage (2.7 to 4.5 V), the maximum system clock is 13 MHz.

Emulation memory

The H8S/2000 emulator is used with one of the following Emulation Memory.

- HP 64172A 20ns 256K byte Emulation SIMM Memory
- HP 64172B 20ns 1M byte Emulation SIMM Memory
- HP 64173A 20ns 4M byte Emulation SIMM Memory

You can define up to 16 memory ranges (at 1K byte boundaries). The emulator occupies some emulation memory, which is used for monitor program and internal RAM of microprocessor, leaving 248K, 992K, 3968K byte of emulation memory which you may use. You can characterize memory range as emulation RAM (eram), emulation ROM (erom), target system RAM (tram), target system ROM (trom), or guarded memory (grd). The emulator generates an error message when accesses are made to guarded memory locations. You can also configure the emulator so that writes to memory defined as ROM cause emulator execution to break out of target program execution.

Analysis

The H8S/2000 emulator is used with one of the following analyzers which allows you to trace code execution and processor activity.

- HP 64704A 80-channel Emulation Bus Analyzer
- HP 64794A/C/D Deep Emulation Bus Analyzer

The Emulation Bus Analyzer monitors the emulation processor using an internal analysis bus.

Registers

You can display or modify the H8S/2000 internal register contents. This includes the ability to modify the program counter (PC) value so you can control where the emulator starts a program run.

Breakpoints

You can set the emulator/analyzer interaction so the emulator will break to the monitor program when the analyzer finds a specific state or states, allowing you to perform post-mortem analysis of the program execution. You can also set software breakpoints in your program. This feature is realized by inserting a special instruction into user program. One of undefined opcodes (5770 hex) is used as software breakpoint instruction. Refer to the "Using Software Breakpoints" section of "Getting Started" chapter for more information.

Reset Support

The emulator can be reset from the emulation system under your control; or your target system can reset the emulation processor.

Real Time Operation

Real-time signifies continuous execution of your program at full rated processor speed without interference from the emulator. (Such interference occurs when the emulator needs to break to the monitor to perform an action you requested, such as displaying target system memory.) Emulator features performed in real time include: running and analyzer tracing. Emulator features not performed in real time include: display or modification of target system memory, load/dump of target memory, display or modification of registers.

Limitations, Restrictions

DMA Support

Direct memory access to the emulation memory by the external DMAC is not allowed. Single-mode transfer to the emulation memory by the internal DMAC also is not supported.

Burst ROM

Do not map the burst ROM space with the 1-state burst cycle as the emulation memory.

Write Data Buffer Function

Do not use the write data buffer function for the emulation memory. When using the emulation memory, do not set the write data buffer enable bit in the bus control register L (BCRL).

EEPMOV

A break command, issued during the execution of the "EEPMOV" command, is suspended and occurs after the execution is completed.

Watch Dog Timer in Background

Watch dog timer is suspended count up while the emulator is running in the background monitor.

Monitor Break at Sleep/Standby Mode

When the emulator breaks into the background monitor, sleep or software standby mode is released. Then, PC indicates next address of "SLEEP" instruction.

Hardware Standby Mode

Hardware standby mode is not supported for the H8S/2000 emulator. Hardware standby request from the target system will give the reset signal to the emulator.

Interrupts in Background Cycles

The H8S/2000 emulator does not accept any interrupts while in the background monitor. Such interrupts are suspended while running the background monitor, and will occur when context is changed to foreground.

Evaluation chip

Hewlett-Packard makes no warranty of the problem caused by the H8S/2000 Evaluation chip in the emulator.

2

Getting Started

Introduction

This chapter will lead you through a basic, step by step tutorial designed to familiarize you with the use of the HP 64700 emulator for the H8S/2000 microprocessor. When you have completed this chapter, you will be able to perform these tasks:

- Set up an emulation configuration for out of circuit emulation use
- Map memory
- Transfer a small program into emulation memory
- Use run/stop controls to control operation of your program
- Use memory manipulation features to alter the program’s operation
- Use analyzer commands to view the real time execution of your program
- Use software breakpoint feature to stop program execution at specific address
- Search memory for strings or numeric expressions
- Make program coverage measurements

Before You Begin

Before beginning the tutorial presented in this chapter, you must have completed the following tasks:

1. Completed hardware installation of the HP 64700 emulator in the configuration you intend to use for your work:

   Standalone configuration

   Remote configuration

2. If you are using the Remote Configuration, you must have completed installation and configuration of a terminal emulator program which will allow your host to act as a terminal connected to the emulator. In addition, you must start the terminal emulator program before you can work the examples in this chapter.

3. If you have properly completed steps 1 and 2 above, you should be able to hit <RETURN> (or <ENTER> on some keyboards) and get one of the following command prompts on your terminal screen:

```
U>
R>
M>
```

   If you do not see one of these command prompts, retrace your steps through the hardware and software installation procedures outlined in the manuals above, verifying all connections and procedural steps. If you are still unable to get a command prompt, refer to the *HP 64700 Support Services Guide*. The guide gives basic troubleshooting procedures. If this fails, call the local HP sales and service office listed in the *Support Services Guide.*

   In any case, you **must** have a command prompt on your terminal screen before proceeding with the tutorial.

A Look at the Sample Program

The sample program "COMMAND_READER" used in this chapter is shown figure 2-1. The program emulates a primitive command interpreter.

Data Declarations

Msg_A, Msg_B and Msg_I are the messages used by the program to respond to various command inputs.

Initialization

The locations of stack and input area(Cmd_Input) are moved into address registers for use by the program. Next, the CLEAR routine clears the command byte(the first location pointed to by Cmd_Input - 0fff000 hex). Cmd_Input contains 00 hex for late use.

Scan

This routine continuously reads the byte at location of Cmd_Input until it is something other than a null character (00 hex); when this occurs, the Exe_Cmd routine is executed.

Exe_Cmd

Compares the input byte (now something other than a null) to the possible command bytes of "A" (ASCII 41 hex) and "B" (ASCII 42 hex), then jumps to the appropriate set up routine for the command message. If the input byte does not match either of these values, a branch to a set up routine for an error message is executed.

Cmd_A, Cmd_B, Cmd_I

These routines set up the proper parameters for writing the output message: the number of bytes in the message is moved to the R3L register and the base address of the message in the data area is moved to address register ER4.

Write_Msg

First the base address of the output area is copied to ER5. Then the Clear_Old routine writes nulls to 32 bytes of the output area (this serves both to initialize the area and to clear old messages written during previous program passes).

Finally, the proper message is written to the output area by the Write_Loop routine. When done, Write_Loop jumps back to Clear and the command monitoring process begins again.

Using the various features of the emulator, we will show you how to load this program into emulation memory, execute it, monitor the program’s operation with the analyzer, and simulate entry of different commands utilizing the memory access commands provided by the HP 64700 command set.

```
002000                                     .SECTION        Table,DATA,LOCATE=H'2000
002000                     Msgs
002000 5448495320495320    Msg_A           .SDATA          "THIS IS MESSAGE A"
002008 4D45535341474520
002010 41
002011 5448495320495320    Msg_B           .SDATA          "THIS IS MESSAGE B"
002019 4D45535341474520
002021 42
002022 494E56414C494420    Msg_I           .SDATA          "INVALID COMMAND"
00202A 434F4D4D414E44
002031                     End_Msgs
001000                                     .SECTION        Prog,CODE,LOCATE=H'1000
                           ;*****************************************************
                           ;* Set up the Pointers.
                           ;*****************************************************
001000 7A07000FF904        Init            MOV.L           #Stack,ER7
001006 7A0100FFF000                        MOV.L           #Cmd_Input,ER1
                           ;*****************************************************
                           ;* Clear previous command.
                           ;*****************************************************
00100C F800                Clear           MOV.B           #H'00,R0L
00100E 6AA800FFF000                        MOV.B           R0L,@Cmd_Input
                           ;*****************************************************
                           ;* Read command input byte.  If no command has been
                           ;* entered, continue to scan for it.
                           ;*****************************************************
001014 6A2A00FFF000        Scan            MOV.B           @Cmd_Input,R2L
00101A AA00                                CMP.B           #H'00,R2L
00101C 47F6                                BEQ             Scan
                           ;*****************************************************
                           ;* A command has been entered.  Check if it is
                           ;* command A, command B, or invalid command.
                           ;*****************************************************
00101E AA41                Exe_Cmd         CMP.B           #H'41,R2L
001020 5870000A                            BEQ             Cmd_A
001024 AA42                                CMP.B           #H'42,R2L
001026 58700010                            BEQ             Cmd_B
00102A 58000018                            BRA             Cmd_I
                           ;*****************************************************
                           ;* Command A is entered.  R3L = the number of bytes
                           ;* in message A.  R4 = location of the message.
                           ;* Jump to the routine which writes the message.
                           ;*****************************************************
00102E FB11                Cmd_A           MOV.B           #Msg_B-Msg_A,R3L
001030 7A0400002000                        MOV.L           #Msg_A,ER4
001036 58000014                            BRA             Write_Msg
                           ;*****************************************************
                           ;* Command B is entered.
                           ;*****************************************************
00103A FB11                Cmd_B           MOV.B           #Msg_I-Msg_B,R3L
00103C 7A0400002011                        MOV.L           #Msg_B,ER4
001042 58000008                            BRA             Write_Msg
```

Figure 2-1. Sample Program Listing

```
                          ;****************************************************
                          ;* An invalid command is entered.
                          ;****************************************************
001046 FB0F               Cmd_I           MOV.B           #End_Msgs-Msg_I,R3L
001048 7A0400002022                       MOV.L           #Msg_I,ER4
                          ;****************************************************
                          ;* The destination area is cleared.
                          ;****************************************************
00104E 7A0500FFF004       Write_Msg       MOV.L           #Msg_Dest,ER5
001054 FE20               Clear_Old       MOV.B           #H'20,R6L
001056 68D8               Clear_Loop      MOV.B           R0L,@ER5
001058 0B05                               ADDS.L          #1,ER5
00105A 1A0E                               DEC.B           R6L
00105C 46F8                               BNE             Clear_Loop
                          ;****************************************************
                          ;* Message is written to the destination.
                          ;****************************************************
00105E 7A0500FFF004                       MOV.L           #Msg_Dest,ER5
001064 6C4E               Write_Loop      MOV.B           @ER4+,R6L
001066 68DE                               MOV.B           R6L,@ER5
001068 0B05                               ADDS.L          #1,ER5
00106A 1A0B                               DEC.B           R3L
00106C 46F6                               BNE             Write_Loop
                          ;****************************************************
                          ;* Go back and scan for next command.
                          ;****************************************************
00106E 409C                               BRA             Clear

FFF000                                    .SECTION        Data,DATA,LOCATE=H'FF800
                          ;****************************************************
                          ;* Command input area.
                          ;****************************************************
FFF000 00000004           Cmd_Input       .RES.L          1
                          ;****************************************************
                          ;* Destination of the command messages.
                          ;****************************************************
FFF004 00000100           Msg_Dest        .RES.W          H'80
FFF104                    Stack
       00001000                           .END            Init
```

Figure 2-1. Sample Program Listing (Cont'd)

Using the Help Facility

If you need a quick reference to the Terminal Interface syntax, you can use the built-in help facilities. For example, to display the top level help menu, type:

R> **help**

```
help  - display help information

  help <group>         - print help for desired group
  help -s <group>      - print short help for desired group
  help <command>       - print help for desired command
  help                 - print this help screen

--- VALID <group> NAMES ---
  gram     - system grammar
  proc     - processor specific grammar

  sys      - system commands
  emul     - emulation commands
  hl       - highlevel commands (hp internal use only)
  trc      - analyzer trace commands
  *        - all command groups
```

You can type the ? symbol instead of typing help. For example, if you want a list of commands in the emul command group, type:

R> **? emul**

```
emul - emulation commands
-----------------------------------------------------------------------------
b......break to monitor  cp.....copy memory       mo.....modes
bc.....break condition   dump...dump memory       r......run user code
bp.....breakpoints       es.....emulation status  reg....registers
cf.....configuration     io.....input/output      rst....reset
cim....copy target image load...load memory       rx.....run at CMB execute
cmb....CMB interaction   m......memory            s......step
cov....coverage          map....memory mapper     ser....search memory
```

To display help information for any command, just type help (or ?) and the command name. For example:

R> **help load**

```
load - download absolute file into processor memory space

  load -i          - download intel hex format
  load -m          - download motorola S-record format
  load -t          - download extended tek hex format
  load -S          - download symbol file
  load -n          - reserved for internal hp use
  load -h          - download hp format (requires transfer protocol)
  load -a          - reserved for internal hp use
  load -e          - write only to emulation memory
  load -u          - write only to target memory
  load -b          - data sent in binary (valid with -h option)
  load -x          - data sent in hex ascii (valid with -h option)
  load -q          - quiet mode
  load -p          - record ACK/NAK protocol (valid with -imt options)
```

Initialize the Emulator to a Known State

To initialize the emulator to a known state for this tutorial:

Note

It is especially important that you perform the following step if the emulator is being operated in a standalone mode controlled by only a data terminal. The only program entry available in this mode is through memory modification; consequently, if the emulator is reinitialized, emulation memory will be cleared and a great deal of tedious work could be lost.

1. Verify that no one else is using the emulator or will have need of configuration items programmed into the emulator.

2. Initialize the emulator by typing the command:

```
R> init
```

Set Up the Proper Emulation Configuration

Set Up Emulation Conditions

To set the emulator’s configuration values to the proper state for this tutorial, do this:

1. Type:

```
R> cf
```

You should see the following configuration items displayed:

```
cf chip=2653
cf clk=int
cf mode=7
cf nmi=en
cf qbrk=dis
cf rrt=dis
cf rsp=0fffffc00
cf trfsh=en
```

Note

The individual configuration items won’t be explained in this example; refer to Chapter 4 of this manual and the *User’s Reference* manual for details.

---

2. If the configuration items displayed on your screen don’t match the ones listed above, here is how to make them agree:

For each configuration item that does not match, type:

```
R> cf <config_item>=<value>
```

For example, if you have the following configuration items displayed (those in bold indicate items different from the list above):

cf chip=2653
cf clk=ext
cf mode=7
cf nmi=en
cf qbrk=dis
cf rrt=en
cf rsp=0fffffc00
cf trfsh=en

To make these configuration values agree with the desired values, type:

```
R> cf clk=int
R> cf rrt=dis
```

3. Let’s go ahead and set up the proper break conditions. Type:

```
R> bc
```

You will see:

```
bc -d bp #disable
bc -e rom #enable
bc -d bnct #disable
bc -d cmbt #disable
bc -d trig1 #disable
bc -d trig2 #disable
```

For each break condition that does not match the one listed, use one of the following commands:

To enable break conditions that are currently disabled, type:

```
R> bc -e <breakpoint type>
```

To disable break conditions that are currently enabled, type:

```
R> bc -d <breakpoint type>
```

For example, if typing bc gives the following list of break conditions:

bc -d bp #disable
bc -d rom #disable
bc -d bnct #disable
bc -d cmbt #disable
bc -e trig1 #enable
bc -e trig2 #enable

(items in bold indicate improper values for this example)

Type the following commands to set the break conditions correctly for this example:

R> **bc -e rom**

(this enables the write to ROM break)

R> **bc -d trig1 trig2**

(this disables break on triggers from the analyzer)

Mapping Memory

Depending on the memory board, emulation memory consists of 256K, 1M or 4M bytes, mappable in 1K byte blocks. The monitor occupies some memories for internal RAM and monitor program, leaving 248K, 992K, 3968K bytes of emulation memory which you may use.

The memory mapper allows you to characterize memory locations. It allows you specify whether a certain range of memory is present in the target system or whether you will be using emulation memory for that address range. You can also specify whether the target system memory is ROM or RAM, and you can specify that emulation memory be treated as RAM or ROM.

Type:

R> **map 0..0ffff erom**

To verify that memory blocks are mapped properly, type:

R> **map**

You will see:

```
# remaining number of terms   : 15
# remaining emulation memory  : 6e800h bytes
map 0000000..000ffff     erom    # term 1
map  other tram
```

Note You must map internal ROM as emulation memory.

Note You don't have to map internal RAM, since the emulator maps internal RAM as emulation RAM. And the emulator memory system dose not introduce it in memory mapping display.

Refer to "Memory Mapping" section of "Configuring the Emulator" chapter in this manual for more details.

Transfer Code into Emulation Memory

Transferring Code from a Terminal In Standalone Configuration

To transfer code into emulation memory from a data terminal running in standalone mode, you must use the modify memory commands. This is necessary because you have no host computer transfer facilities to automatically download the code for you (as if you would if you were using the transparent configuration or the remote configuration.) To minimize the effects of typing errors, you will modify only one row of memory at a time in this example. Do the following:

1. Enter the data information for the program by typing the following commands:

```
R> m 002000..00200f=54,48,49,53,20,49,53,20,4d,45,53,53,41,47,45,20
R> m 002010..00201f=41,54,48,49,53,20,49,53,20,4d,45,53,53,41,47,45
R> m 002020..00202f=20,42,49,4e,56,41,4c,49,44,20,43,4f,4d,4d,41,4e
R> m 002030=44
```

You could also type the following line instead:

```
R> m 002000="THIS IS MESSAGE ATHIS IS MESSAGE BINVALID COMMAND"
```

2. You should now verify that the data area of the program is correct by typing:

```
R> m 002000..002030
```

You should see:

```
002000..00200f    54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45 20
002010..00201f    41 54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45
002020..00202f    20 42 49 4e 56 41 4c 49 44 20 43 4f 4d 4d 41 4e
002030..002030    44
```

If this is not correct, you can correct the errors by re-entering only the modify memory commands for the particular rows of memory that are wrong.

For example, if row 002000..00200f shows these values:

```
002000..00200f     54 48 49 53 20 20 49 53 20 4d 45 53 53 41 47 45
```

you can correct this row of memory by typing:

```
R> m 002000..00200f=54,48,49,53,20,49,53,20,4d,45,53,53,41,47,45,20
```

Or, you might need to modify only one location, as in the instance where address 00200f equals 22 hex rather than 20 hex. Type:

```
R> m 00200f=22
```

3. Enter the program information by typing the following commands:
(Note the hex letters must be preceded by a digit.)

```
R> m 001000..00100f=7a,07,00,0ff,0f1,04,7a,01,00,0ff, 0f0,00,0f8,00,6a,0a8
R> m 001010..00101f=00,0ff,0f0,00,6a,2a,00,0ff,0f,00,0aa,00,47,0f6,0aa,41
R> m 001020..00102f=58,70,00,0a,0aa,42,58,70,00,10,58,00,00,18,0fb,11
R> m 001030..00103f=7a,04,00,00,20,00,58,00,00,14,0fb,11,7a,04,00,00
R> m 001040..00104f=20,11,58,00,00,08,0fb,0f,7a,04,00,00,20,22,7a,05
R> m 001050..00105f=00,0ff,0f0,04,0fe,20,68,0d8,0b,05,1a,0e,46,0f8,7a,05
R> m 001060..00106f=00,0ff,0f,04,6c,4e,68,0de,0b,05,1a,0b,46,0f6,40,9c
```

4. You should now verify that the program area is correct by typing:

```
R> m 001000..00106f
```

You should see:

```
001000..00100f      7a 07 00 0f f9 04 7a 01 00 0f f8 00 f8 00 6a a8
001010..00101f      54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45 20
001020..00102f      41 54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45
001030..00103f      7a 04 00 00 20 00 58 00 00 14 fb 11 7a 04 00 00
001040..00104f      20 11 58 00 00 08 fb 0f 7a 04 00 00 20 22 7a 05
001050..00105f      00 ff f0 04 fe 20 68 d8 0b 05 1a 0e 46 f8 7a 05
001060..00106f      00 ff f0 04 6c 4e 68 de 0b 05 1a 0b 46 f6 40 9c
```

If this is not correct, you can correct the errors by re-entering only the modify memory commands for the particular rows of memory that are wrong.

Looking at Your Code

Now that you have loaded your code into emulation memory, you can display it in mnemonic format. Type:

R> **m -dm 1000..106f**

You will see:

```
0001000  -                  MOV.L #00fff104,ER7
0001006  -                  MOV.L #00fff000,ER1
000100c  -                  MOV.B #00,R0L
000100e  -                  MOV.B R0L,@0fff000
0001014  -                  MOV.B @fff000,R2L
000101a  -                  CMP.B #00,R2L
000101c  -                  BEQ 001014
000101e  -                  CMP.B #41,R2L
0001020  -                  BEQ 00102e
0001024  -                  CMP.B #42,R2L
0001026  -                  BEQ 00103a
000102a  -                  BRA 001046
000102e  -                  MOV.B #11,R3L
0001030  -                  MOV.L #00002000,ER4
0001036  -                  BRA 00104e
000103a  -                  MOV.B #11,R3L
000103c  -                  MOV.L #00002011,ER4
0001042  -                  BRA 00104e
0001046  -                  MOV.B #0f,R3L
0001048  -                  MOV.L #00002022,ER4
000104e  -                  MOV.L #00fff004,ER5
0001054  -                  MOV.B #20,R6L
0001056  -                  MOV.B R0L,@ER5
0001058  -                  ADDS #1,ER5
000105a  -                  DEC.B R6L
000105c  -                  BNE 001056
000105e  -                  MOV.L #00fff004,ER5
0001064  -                  MOV.B @ER4+,R6L
0001066  -                  MOV.B R6L,@ER5
0001068  -                  ADDS #1,ER5
000106a  -                  DEC.B R3L
000106c  -                  BNE 001064
000106e  -                  BRA 00100c
```

Familiarize Yourself with the System Prompts

Note

The following steps are not intended to be complete explanations of each command; the information is only provided to give you some idea of the meanings of the various command prompts you may see and reasons why the prompt changes as you execute various commands.

You should gain some familiarity with the HP 64700 emulator command prompts by doing the following:

1. Ignore the current command prompt. Type:

```
*> rst
```

You will see:

```
R>
```

The **rst** command resets the emulation processor and holds it in the reset state. The "R>" prompt indicates that the processor is reset.

2. Type:

```
R> r 1000
```

You will see:

```
U>
```

The r command runs the processor from address 1000 hex.

3. Type:

```
U> b
```

You will see:

```
M>
```

The **b** command causes the emulation processor to "break" execution of whatever it was doing and begin executing within the emulation monitor. The "M>" prompt indicates that the emulator is running in the monitor.

Note If DMA transfer is in progress with BURST transfer mode, **b** command is suspended and occurs after DMA transfer is completed.

Running the Sample Program

4. Type:

```
M> r 1000
```

The emulator changes state from background to foreground and begins running the sample program from location 1000 hex.

Note The default number base for address and data values within HP 64700 is hexadecimal. Other number bases may be specified. Refer to the Tutorials chapter of this manual or the *HP 64700 User's Reference* manual for further details.

5. Let's look at the registers to verify that the address registers were properly initialized with the pointers to the input and output areas. Type:

```
U> reg
```

You will see:

```
reg pc=00101a ccr=84 exr=7f er0=00000000 er1=00fff000 er2=00000000
reg er3=00000000 er4=00000000 er5=00000000 er6=00000000 er7=00fff104 reg sp=00fff104
mach=00000000 macl=00000000 mdcr=87
```

Notice that ER1 contains 0fff000 hex.

6. Verify that the input area command byte was cleared during initialization.

   Type:

```
U> m -db 0fff000
```

   You will see:

```
0fff000..0fff000  00
```

   The input byte location was successfully cleared.

7. Now we will use the emulator features to make the program work. Remember that the program writes specific messages to the output area depending on what the input byte location contains. Type:

```
U> m 0fff000=41
```

   This modifies the input byte location to the hex value for an ASCII "A". Now let's check the output area for a message.

```
U> m 0fff004..0fff023
```

   You will see:

```
0fff004..0fff013    54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45 20
0fff014..0fff023    41 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

   These are the ASCII values for Msg_A.

   Repeat the last two commands twice. The first time, use 42 instead of 41 at location ff800h and note that Msg_B overwrites Msg_A. Then try these again, using any number except 00, 41, or 42 and note that the Msg_I message is written to this area.

Stepping Through the Program

8. You can also direct the emulator processor to execute one instruction or number of instructions. Type:

M> **s 1 1000;reg**

This command steps 1 instruction from address 1000 hex, and displays registers. You will see:

```
0001000  -       MOV.L #00fff104,ER7
PC =0001006
reg pc=001006 ccr=80 exr=7f er0=00000000 er1=00fff000 er2=00000000
reg er3=00000000 er4=00002011 er5=00fff015 er6=00000041 er7=00fff104
reg sp=00fff104 mach=00000000 macl=00000000 mdcr=87
```

Notice that PC contains 1006 hex.

9. To step one instruction from present PC, you only need to type s at prompt. Type:

M> **s;reg**

You will see:

```
0001006  -       MOV.L #00fff000,ER1
PC =000100c
reg pc=00100c ccr=80 exr=7f er0=00000000 er1=00fff000 er2=00000000
reg er3=00000000 er4=00002011 er5=00fff015 er6=00000041 er7=00fff104
reg sp=00fff104 mach=00000000 macl=00000000 mdcr=87
```

Tracing Program Execution

Predefined Trace Labels

Three trace labels are predefined in the H8S/2000 emulator. You can view these labels by entering the tlb (trace label) command with no options.

M> **tlb**

```
#### Emulation trace labels
tlb addr 16..39
tlb data 0..15
tlb stat 40..63
```

Predefined Status Equates

Common values for the H8S/2000 status trace signals have been predefined. You can view these predefined equates by entering the equ command with no options.

```
M> equ
```

```
### Equates ###
equ bg=0xxx0xxxxxxxxxxxxxxxxxxxxy
equ byte=0xxxxxxxxxxxxx1xxxxxxxx1xy
equ cpu=0xxxxxxxxxxxxx1xx110xxxxxy
equ data=0xxxxxxxxxxxxx1x1110xxxxxy
equ dma=0xxxxxxxxxxxxx1x0000xxxxxy
equ dtc=0xxxxxxxxxxxxx1x0001xxxxxy
equ fetch=0xxxxxxxxxxxxx1x0110xxx01y
equ fg=0xxx1xxxxxxxxxxxxxxxxxxxxy
equ grd=0xxxx01xxxxxxx1xxxxxxxxxxy
equ intack=0xxxxxxx0xx0xxxxxxxxxxxxxy
equ io=0xxxxxxxxxxxxx1xxxxx01xxxy
equ read=0xxxxxxxxxxxxx1xxxxxxxxx1y
equ word=0xxxxxxxxxxxxx1xxxxxxxx0xy
equ write=0xxxxxxxxxxxxx1xxxxxxxxx0y
equ wrrom=0xxxx10xxxxxxx1xxxxxxxxx0y
```

These equates may be used to specify values for the **stat** trace label when qualifying trace conditions.

Specifying a Trigger

Now let's use the emulation analyzer to trace execution of the program. Suppose that you would like to start the trace when the analyzer begins writing data to the message output area. You can do this by specifying analyzer trigger upon encountering the address 0fff004 hex. Furthermore, you might want to store only the data written to the output area. This can be accomplished by modifying what is known as the "analyzer storage specification".

Note

For this example, you will be using the analyzer in the easy configuration, which simplifies the process of analyzer measurement setup. The complex configuration allows more powerful measurements, but requires more interaction from you to set up those measurements. For more information on easy and complex analyzer configurations and the analyzer, refer to the *HP 64700 Analyzer User's Guide* and the *User's Reference*.

Now, let’s set the trigger specification. Type:

```
M> tg addr=0fff004
```

To store only the accesses to the address range 0fff04 through 0fff015 hex, type:

```
M> tsto addr=0fff004..0fff015
```

Let’s change the data format of the trace display so that you will see the output message writes displayed in ASCII format:

```
M> tf addr,h data,A count,R seq
```

Start the trace by typing:

```
M> t
```

You will see:

```
Emulation trace started
```

To start the emulation run, type:

```
M> r 1000
```

Now, you need to have a "command" input to the program so that the program will jump to the output routines (otherwise the trigger will not be found, since the program will never access address 0fff004 hex). Type:

```
U> m 0fff000=41
```

To display the trace list, type:

```
U> tl 0..34
```

You will see:

```
 Line   addr,H  data,A    count,R  seq
------- ------ ------ ------------- ---
      0 fff004 ..     -------------  +
      1 fff005 ..           0.60uS   .
      2 fff006 ..           0.60uS   .
      3 fff007 ..           0.60uS   .
      4 fff008 ..           0.60uS   .
      5 fff009 ..           0.60uS   .
      6 fff00a ..           0.60uS   .
      7 fff00b ..           0.60uS   .
      8 fff00c ..           0.60uS   .
      9 fff00d ..           0.60uS   .
     10 fff00e ..           0.60uS   .
     11 fff00f ..           0.60uS   .
     12 fff010 ..           0.60uS   .
     13 fff011 ..           0.60uS   .
     14 fff012 ..           0.60uS   .
     15 fff013 ..           0.60uS   .
     16 fff014 ..           0.60uS   .
     17 fff015 ..           0.60uS   .
     18 fff004 T.           9.60uS   .
     19 fff005 .H           0.90uS   .
     20 fff006 I.           0.90uS   .
     21 fff007 .S           0.90uS   .
     22 fff008 ..           0.90uS   .
     23 fff009 .I           0.90uS   .
     24 fff00a S.           0.90uS   .
     25 fff00b ..           0.90uS   .
     26 fff00c M.           0.90uS   .
     27 fff00d .E           0.90uS   .
     28 fff00e S.           0.90uS   .
     29 fff00f .S           0.90uS   .
     30 fff010 A.           0.90uS   .
     31 fff011 .G           0.90uS   .
     32 fff012 E.           0.90uS   .
     33 fff013 ..           0.90uS   .
     34 fff014 A.           0.90uS   .
```

Using Software Breakpoints

You can stop program execution at specific address by using **bp** (software breakpoint) command. When you define a software breakpoint to a certain address, the emulator will replace the opcode with one of undefined opcode (5770 hex) as software breakpoint instruction. When the emulator detects the special instruction, user program breaks to the monitor, and the original opcode will be placed at the breakpoint address. A subsequent run or step command will execute from this address.

If the special instruction was not inserted as the result of **bp** command (in other words, it is part of the user program), the "Undefined software breakpoint" message is displayed.

Note

You can set software breakpoints only at memory locations which contain instruction opcodes (not operands or data). If a software breakpoint is set at a memory location which is not an instruction opcode, the software breakpoint instruction will never be executed and the break will never occur.

Note

Because software breakpoints are implemented by replacing opcodes with the software breakpoint instruction, you cannot define software breakpoints in target ROM. You can, however, copy target ROM into emulation memory by **cim** command. (Refer to *HP 64700 Terminal Interface User's Reference* manual.)

Displaying and Modifying the Break Conditions

Before you can define software breakpoints, you must enable software breakpoints with the **bc** (break conditions) command. To view the default break conditions and change the software breakpoint condition, enter the following commands.

M> **bc**

```
bc -d bp #disable
bc -e rom #enable
bc -d bnct #disable
bc -d cmbt #disable
```

```
bc -d trig1 #disable
bc -d trig2 #disable
```

M> **bc -e bp**

Defining a Software Breakpoint

Now that the software breakpoint is enabled, you can define software breakpoints. Enter the following command to break on the address of the Write_Msg label.

M> **bp 104e**

Run the program and verify that execution broke at the appropriate address.

M> **r 1000**
U> **m 0fff000=41**

```
!ASYNC_STAT  615! Software break point: 000104e
```

M> **reg**

```
reg pc=00104e ccr=80 exr=7f er0=00000000 er1=00fff000 er2=00000041
reg er3=00000011 er4=00002000 er5=00fff015 er6=00000041 er7=00fff104
reg sp=00fff104 mach=00000000 macl=00000000 mdcr=87
```

Notice that PC contains 104e.

When a breakpoint is hit, it becomes disabled. You can use the -e option to the bp command to re-enable the software breakpoint.

M> **bp**

```
###BREAKPOINT FEATURE IS ENABLED###
bp 000104e #disabled
```

M> **bp -e 104e**
M> **bp**

```
###BREAKPOINT FEATURE IS ENABLED###
bp 000104e #enabled
```

M> **r 1000**
U> **m 0fff000=41**

```
!ASYNC_STAT  615! Software breakpoint: 000104e
```

M> **bp**

```
###BREAKPOINT FEATURE IS ENABLED###
bp 000104e #disabled
```

Searching Memory for Strings or Numeric Expressions

The HP 64700 Emulator provides you with tools that allow you to search memory for data strings or numeric expressions. For example, you might want to know exactly where a string is loaded. To locate the position of the string "THIS IS MESSAGE A" in the sample program. Type:

```
M> ser 2000..2fff="THIS IS MESSAGE A"
```

```
pattern match at address:  0002000
```

You can also find numeric expressions. For example, you might want to find all of the **CMP.B** instructions in the sample program. Since a **CMP.B** instruction begins with aa hex, you can search for that value by typing:

```
M> ser -db 10000..106f=0aa
```

```
pattern match at address:  000101a
pattern match at address:  000101e
pattern match at address:  0001024
```

Trace Analysis Considerations

There are some points you need to attend to in using the emulation analyzer. The following section describes such points.

How to Specify the Trigger Condition

Suppose that you would like to start the trace when the program begins executing Exe_Cmd routine.

To initialize the emulation analyzer, type:

```
U> tinit
```

To set the trigger condition, type:

```
U> tg addr=101e
```

Start the trace and modify memory so that the program will jump to the Exe_Cmd routine:

```
U> t
U> m 0fff000=41
```

To display the trace list, type:

```
U> tl 0..20
```

```
 Line   addr,H  H8S/2653 mnemonic,H                    count,R  seq
-------  ------ ---------------------------------- ------------- ---
      0  00101e   aa41  fetch mem                  -------------  +
      1  001014 MOV.B @fff000,R2L                         0.10uS  .
      2  001016   00ff  fetch mem                         0.10uS  .
      3  001018   f000  fetch mem                         0.10uS  .
      4  00101a CMP.B #00,R2L                             0.08uS  .
      5  fff000   00xx  read  mem byte                    0.12uS  .
      6  00101c BEQ 001014                                0.10uS  .
      7  00101e   aa41  fetch mem                         0.10uS  .
      8  001014 MOV.B @fff000,R2L                         0.10uS  .
      9  001016   00ff  fetch mem                         0.08uS  .
     10  001018   f000  fetch mem                         0.12uS  .
     11  00101a CMP.B #00,R2L                             0.10uS  .
     12  fff000   00xx  read  mem byte                    0.10uS  .
     13  00101c BEQ 001014                                0.10uS  .
     14  00101e   aa41  fetch mem                         0.08uS  .
     15  001014 MOV.B @fff000,R2L                         0.12uS  .
     16  001016   00ff  fetch mem                         0.10uS  .
     17  001018   f000  fetch mem                         0.10uS  .
     18  00101a CMP.B #00,R2L                             0.10uS  .
     19  fff000   00xx  read  mem byte                    0.10uS  .
     20  00101c BEQ 001014                                0.08uS  .
```

This is not what we were expecting to see. (We expected to see the program executed Exe_Cmd routine which starts from 101e hex.) As you can see at the fist line of the trace list, address 101e hex appears on the address bus during the program executing Scan loop. This triggered the emulation analyzer before EXE_Cmd routine was executed. To avoid mis-trigger by this cause, set the trigger condition to the second instruction of the routine you want to trace. Type:

```
U> tg addr=1020
```

To change the trigger position so that 10 states appear before the trigger in the trace list, type:

```
U> tp -b 10
```

Start the trace again and modify memory:

```
U> t
U> m 0fff000=41
```

Now display the trace list:

```
U> tl -10..10
```

As you can see, the analyzer captured the execution of Exe_Cmd

```
 Line   addr,H  H8S/2653 mnemonic,H                  count,R  seq
-----   ------  ---------------------------------  ---------  ---
   -10  00101c BEQ 001014                        -------------  .
    -9  00101e   aa41  fetch mem                        0.10uS  .
    -8  001014 MOV.B @fff000,R2L                        0.10uS  .
    -7  001016   00ff  fetch mem                        0.10uS  .
    -6  001018   f000  fetch mem                        0.10uS  .
    -5  00101a CMP.B #00,R2L                            0.10uS  .
    -4  fff000   41xx  read  mem byte                   0.10uS  .
    -3  00101c BEQ 001014                               0.10uS  .
    -2  00101e CMP.B #41,R2L                            0.10uS  .
    -1  001014   6a2a  unused fetch mem                 0.12uS  .
     0  001020 BEQ 00102e                               0.08uS  +
     1  001022   000a  fetch mem                        0.10uS  .
     2  00102e MOV.B #11,R3L                            0.20uS  .
     3  001030 MOV.L #00002000,ER4                      0.12uS  .
     4  001032   0000  fetch mem                        0.08uS  .
     5  001034   2000  fetch mem                        0.10uS  .
     6  001036 BRA 00104e                               0.10uS  .
     7  001038   0014  fetch mem                        0.10uS  .
     8  00104e MOV.L #00fff004,ER5                      0.20uS  .
     9  001050   00ff  fetch mem                        0.10uS  .
    10  001052   f004  fetch mem                        0.10uS  .
```

routine which starts from line -2 of the trace list.

Store Condition and Disassembling

When you specify store condition with tsto command, disassembling of program execution may not be accurate.

Type:

```
U> tinit
U> t
U> tl 0..20
```

```
 Line   addr,H  H8S/2653 mnemonic,H                     count,R  seq
-------  ------ ---------------------------------- ------------- ---
      0  001016   00ff  fetch mem                  -------------  +
      1  001018   f000  fetch mem                         0.12uS  .
      2  00101a CMP.B #00,R2L                             0.10uS  .
      3  fff000   00xx  read  mem byte                    0.10uS  .
      4  00101c BEQ 001014                                0.08uS  .
      5  00101e   aa41  fetch mem                         0.10uS  .
      6  001014 MOV.B @fff000,R2L                         0.12uS  .
      7  001016   00ff  fetch mem                         0.10uS  .
      8  001018   f000  fetch mem                         0.10uS  .
      9  00101a CMP.B #00,R2L                             0.10uS  .
     10  fff000   00xx  read  mem byte                    0.08uS  .
     11  00101c BEQ 001014                                0.10uS  .
     12  00101e   aa41  fetch mem                         0.12uS  .
     13  001014 MOV.B @fff000,R2L                         0.10uS  .
     14  001016   00ff  fetch mem                         0.10uS  .
     15  001018   f000  fetch mem                         0.08uS  .
     16  00101a CMP.B #00,R2L                             0.10uS  .
     17  fff000   00xx  read  mem byte                    0.12uS  .
     18  00101c BEQ 001014                                0.10uS  .
     19  00101e   aa41  fetch mem                         0.10uS  .
     20  001014 MOV.B @fff000,R2L                         0.08uS  .
```

The program is executing Scan loop.

Now, specify the store condition so that only accesses to the address range 1000 hex through 10ff hex will be stored:

U> **tsto addr=1000..10ff**

Start the trace and display the trace list:

U> **t**
U> **tl 0..20**

```
 Line   addr,H  H8S/2653 mnemonic,H                     count,R  seq
-------  ------ ---------------------------------- ------------- ---
      0  001016   00ff  fetch mem                  -------------  +
      1  001018   f000  fetch mem                         0.12uS  .
      2  00101a   aa00  fetch mem                         0.10uS  .
      3  00101c BEQ 001014                                0.20uS  .
      4  00101e   aa41  fetch mem                         0.08uS  .
      5  001014 MOV.B @fff000,R2L                         0.10uS  .
      6  001016   00ff  fetch mem                         0.12uS  .
      7  001018   f000  fetch mem                         0.10uS  .
      8  00101a   aa00  fetch mem                         0.10uS  .
      9  00101c BEQ 001014                                0.18uS  .
     10  00101e   aa41  fetch mem                         0.12uS  .
     11  001014 MOV.B @fff000,R2L                         0.10uS  .
     12  001016   00ff  fetch mem                         0.10uS  .
     13  001018   f000  fetch mem                         0.08uS  .
     14  00101a   aa00  fetch mem                         0.10uS  .
     15  00101c BEQ 001014                                0.22uS  .
     16  00101e   aa41  fetch mem                         0.10uS  .
     17  001014 MOV.B @fff000,R2L                         0.10uS  .
     18  001016   00ff  fetch mem                         0.08uS  .
     19  001018   f000  fetch mem                         0.10uS  .
     20  00101a   aa00  fetch mem                         0.12uS  .
```

As you can see, the executions of CMP.B instruction are not disassembled. This occurs when the analyzer cannot get necessary information for disassembling because of the store condition. Be careful when you use the store condition.

Triggering the Analyzer by Data

You may want to trigger the emulation analyzer when specific data appears on the data bus. You can accomplish this with the following command.

```
U> tg data=<data>
```

There are some points to be noticed when you trigger the analyzer in this way. You always need to specify the <data> with 16 bits value even when access to the data is performed by byte access. This is because the analyzer is designed so that it can capture data on internal data bus (which has 16 bits width). The following table shows the way to specify the trigger condition by data.

Location of data	Access size	Address value	Available <data> Specification		
8/16 bit data bus area	byte	even		ddxx	*1
		odd		xxdd	*1
	word	even		hhll	*2
	long	even	1st	hhhl	*3
			2nd	lhll	*3

*1 dd means 8 bits data
*2 hhll means 16 bits data
*3 long word access always stores 32bit as two word accesses

For example, to trigger the analyzer when the processor performs word access to data 1234 hex in 16 bit bus area, you can specify the following:

```
U> tg data=1234
```

To trigger the analyzer when the processor accesses data 12 hex to the even address located in 8 bit data bus area:

```
U> tg data=12xx
```

On the other hand, to trigger 12 hex to the odd address located 8 bit data bus.

```
U> tg data=0xx12
```

Notice that you always need to specify "xx" value to capture byte access. Be careful to trigger the analyzer by data.

You’re now finished with the "Getting Started" example. You can proceed on with using the emulator and use this manual and the *Terminal Interface Reference* manual as needed to answer your questions.

3

In-Circuit Emulation

When you are ready to use the H8S/2000 emulator in conjunction with actual target system hardware, there are some special considerations you should keep in mind.

- installing the emulator probe
- properly configure the emulator

We will cover the first topic in this chapter. For complete details on in-circuit emulation configuration, refer to Chapter 4.

Installing the Target System Probe

Warning

The following precautions should be taken while using the H8S/2000 emulator. Damage to the emulator circuitry may result if these precautions are not observed.

Power Down Target System. Turn off power to the user target system and to the H8S/2000 emulator before attaching and detaching the PGA adapter to the emulator or target system to avoid circuit damage resulting from voltage transients or mis-insertion

Verify User Plug Orientation. Make certain that Pin 1 of the QFP socket/adapter and Pin 1 of the QFP cable are properly aligned before inserting the QFP cable into the QFP socket/adapter. Failure to do so may result in damage to the emulator circuitry.

Protect Against Static Discharge. The H8S/2000 emulator and the PGA adapter contain devices which are susceptible to damage by static discharge. Therefore, operators should take precautionary measures before handling the user plug to avoid emulator damage.

Compatibility of VOLTAGE/CURRENCY. Please be sure to check that the voltage/currency of the emulator and target system being connected are compatible. If there is a discrepancy, damage may result.

Protect Target System CMOS Components. If your target system includes any CMOS components, turn on the target system first, then turn on the H8S/2000 emulator; when powering down, turn off the emulator first, then turn off power to the target system.

The H8S/2000 emulator is provided without any QFP cable. To emulate each processor with your target system, you need to purchase appropriate QFP cable, separately.

QFP cable

To emulate each processor with your target system, you can use the QFP cable as shown in Figure 3-1. The QFP cable allows you to connect the emulation probe to QFP socket/adapter on your target system using with the PGA adapter. Refer to the Table 1-1 to know appropriate QFP cable.

Caution

Do not apply strong force to the QFP cable, as that might damage the QFP cable.

QFP socket/adapter

To do in-circuit emulation, you must attach the QFP socket/adapter to your target system and connect with the PGA adapter through the QFP cable. One QFP socket/adapter is provided with the QFP cable.

Note

You can order additional QFP socket/adapter with part number listed in Table 1-1.

Installing the PGA adapter

You can use the PGA adapter to connect the emulator to your target system. This PGA adapter gives you a feature to emulate your target system running with supply voltage from 2.7V up to 5.25V.

Note

You must also use a clock conforming to the specification of Table 1-2, when you do in-circuit emulation and configure the emulator to use external clock.

---

1. Attach the QFP socket/adapter to your target system.
2. Connect the PGA adapter to the emulation probe.
3. Install the QFP cable to the QFP socket/adapter as shown in Figure 3-1.

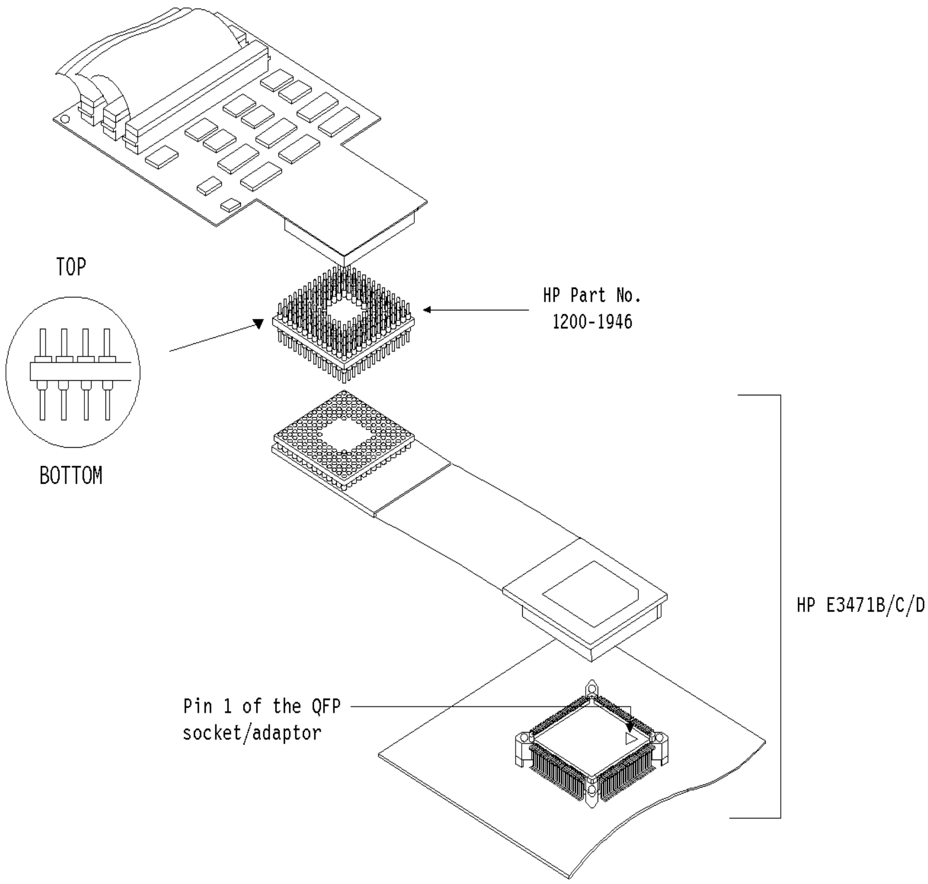


Figure 3-1 Installing the PGA adapter

Installing the H8S/2000 microprocessor

You can replace the QFP cable with H8S/2000 microprocessor. Refer to the Figure 3-2.

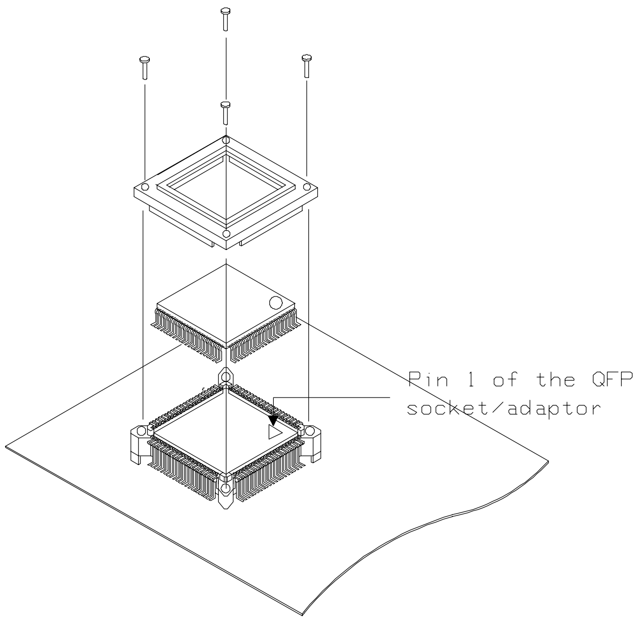


Figure 3-2 Installing the H8S/2000 microprocessor

Note

The QFP socket/adapter (E3471-61621) does not have the cap required for attaching a microprocessor. To attach a microprocessor, you need to purchase the cap (P/N E3471-61631), separately.

Run from Target System Reset

You can use "**r rst**" command to execute program from target system reset. You will see **T**> system prompt when you enter "r rst". In this status, the emulator accept target system reset. Then program starts if reset signal from target system is released.

Note

In the "Awaiting target reset" status(T>), you can not break into the monitor. If you enter "r rst" in out-of-circuit or in the configuration that emulator does not accept target system reset (cf trst=dis), you must reset the emulator.

PGA Pin Assignments

When you connect the PGA adapter to your target system directly, pin assignment of your target PGA socket must be compatible with the PGA adapter pin assignment. The following table and figure show you the pin assignment of the PGA adapter.

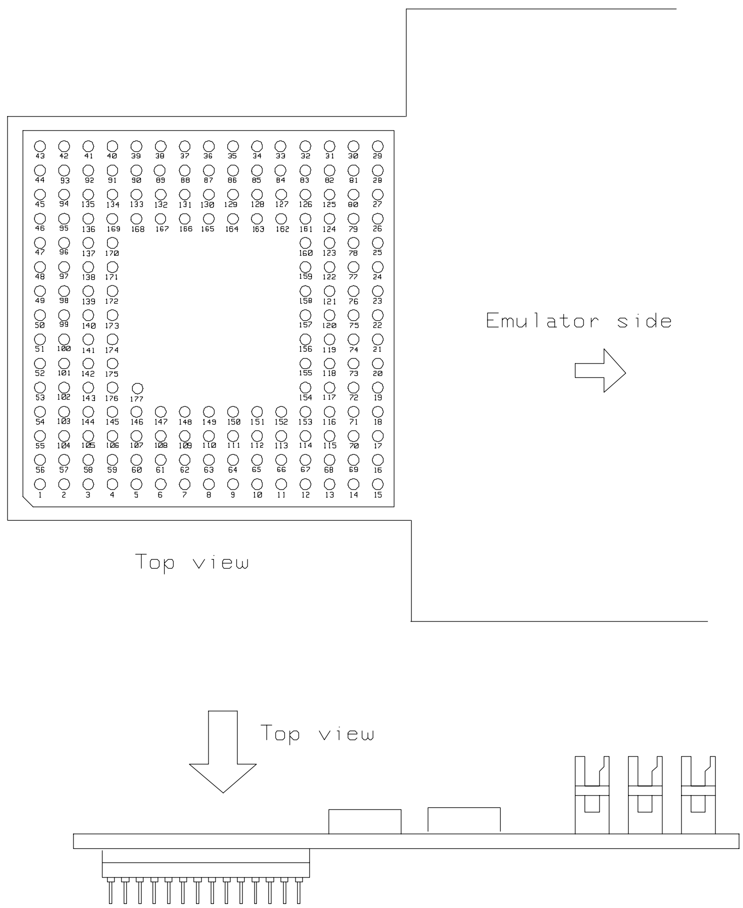


Figure 3-3 PGA adapter pin assignment

Table 3-1 PGA177 to QFP120 Pin Assignment (E3471-61610)

PGA 177 pin #	QFP 120 pin #	Function name	PGA 177 pin #	QFP 120 pin	Function name
1		nc	15		nc
2		nc	16		nc
3	119	PG3	17	29	P67
4		nc	18		nc
5	2	PC0	19	32	P64
6	5	PC3	20	35	PE1
7	8	PC5	21	38	Vss
8	11	PB0	22	41	PE6
9	14	PB3	23	44	PD1
10	17	PB5	24	47	Vss
11	20	PA0	25	50	PD6
12	23	PA3	26	53	P30
13	26	PA5	27	56	P33
14	28	PA7	28	58	P35

Table 3-1 PGA177 to QFP120 Pin Assignment (E3471-61610)(Cont'd)

PGA 177 pin #	QFP 120 pin #	Function name	PGA 177 pin #	QFP 120 pin #	Function name
29		nc	43		nc
30		nc	44		nc
31	59	Vss	45	89	P50
32		nc	46		nc
33	62	P62	47	92	P53
34	65	P26	48	95	P40
35	68	P23	49	98	P43
36	71	P20	50	101	P46
37	74	NMI	51	104	Vss
38	77	XTAL	52	107	P15
39	80	PF7	53	110	P12
40	83	PF5	54	113	MD0
41	86	PF2	55	116	PG0
42	88	PF0	56	118	PG2

Table 3-1 PGA177 to QFP120 Pin Assignment (E3471-61610)(Cont'd)

PGA 177 pin #	QFP 120 pin #	Function name	PGA 177 pin #	QFP 120 pin #	Function name
57		nc	71		nc
58	120	PG4	72	33	Vcc
59		nc	73	36	PE2
60	3	PC1	74	39	PE4
61	6	Vss	75	42	PE7
62	9	PC6	76	45	PD2
63	12	PB1	77	48	PD4
64	15	Vss	78	51	PD7
65	18	PB6	79	54	P31
66	21	PA1	80	57	P34
67	24	Vss	81		nc
68	27	PA6	82	60	P60
69		nc	83		nc
70	30	P66	84	63	P63

Table 3-1 E3471-61610 PGA to QFP120 Adaptor Pin Assignment (Cont'd)

PGA 177 pin #	QFP 120 pin #	Function name	PGA 177 pin #	QFP 120 pin #	Function name
85	66	P25	99	102	P47
86	69	P22	100	105	P17
87	72	WDTOVF	101	108	P14
88	75	STBY	102	111	P11
89	78	EXTAL	103	114	MD1
90	81	Vcc	104	117	PG1
91	84	PF4	105		GND
92	87	PF1	106	1	Vcc
93		nc	107	4	PC2
94	90	P51	108	7	PC4
95		nc	109	10	PC7
96	93	AVcc	110	13	PB2
97	96	P41	111	16	PB4
98	99	P44	112	19	PB7

Table 3-1 PGA177 to QFP120 Pin Assignment (E3471-61610)(Cont'd)

PGA 177 pin #	QFP 120 pin #	Function name	PGA 177 pin #	QFP 120 pin #	Function name
113	22	PA2	127	64	P27
114	25	PA4	128	67	P24
115		nc	129	70	P21
116	31	P65	130	73	RES
117	34	PE0	131	76	Vcc
118	37	PE3	132	79	Vss
119	40	PE5	133	82	PF6
120	43	PD0	134	85	PF3
121	46	PD3	135		nc
122	49	PD5	136	91	P52
123	52	Vcc	137	94	Vref
124	55	P32	138	97	P42
125		GND	139	100	P45
126	61	P61	140	103	AVss

Table 3-1 PGA177 to QFP120 Pin Assignment (E3471-61610)(Cont'd)

PGA 177 pin #	QFP 120 pin #	Function name	PGA 177 pin #	QFP 120 pin #	Function name
141	106	P16	155		nc
142	109	P13	156		nc
143	112	P10	157		nc
144	115	MD2	158		nc
145		GND	159		nc
146		nc	160		nc
147		nc	161		GND
148		nc	162		nc
149		nc	163		nc
150		nc	164		nc
151		nc	165		nc
152		nc	166		nc
153		nc	167		nc
154		nc	168		nc

Table 3-1 PGA177 to QFP120 Pin Assignment (E3471-61610)(Cont'd)

PGA 177 pin #	QFP 120 pin #	Function name	PGA 177 pin #	QFP 120 pin #	Function name
169		nc	174		nc
170		nc	175		nc
171		nc	176		nc
172		nc	177		GND
173		nc			

Table 3-2 PGA177 to QFP128 Pin Assignment (E3471-61611)

PGA 177 pin #	QFP 128 pin #	Function name	PGA 177 pin #	QFP 128 pin	Function name
1		nc	15		nc
2		nc	16		nc
3	1	PG3	17	33	P67
4	3	Vss	18	35	Vss
5	6	PC0	19	38	P64
6	9	PC3	20	41	PE1
7	12	PC5	21	44	Vss
8	15	PB0	22	47	PE6
9	18	PB3	23	50	PD1
10	21	PB5	24	53	Vss
11	24	PA0	25	56	PD6
12	27	PA3	26	59	P30
13	30	PA5	27	62	P33
14	32	PA7	28	64	P35

Table 3-2 PGA177 to QFP128 Pin Assignment (E3471-61611)(Cont'd)

PGA 177 pin #	QFP 128 pin #	Function name	PGA 177 pin #	QFP 128 pin #	Function name
29		nc	43		nc
30		nc	44		nc
31	65	Vss	45	97	P50
32	67	Vss	46	99	Vss
33	70	P62	47	102	P53
34	73	P26	48	105	P40
35	76	P23	49	108	P43
36	79	P20	50	111	P46
37	82	NMI	51	114	Vss
38	85	XTAL	52	117	P15
39	88	PF7	53	120	P12
40	91	PF5	54	123	MD0
41	94	PF2	55	126	PG0
42	96	PF0	56	128	PG2

Table 3-2 PGA177 to QFP128 Pin Assignment (E3471-61611)(Cont'd)

PGA 177 pin #	QFP 128 pin #	Function name	PGA 177 pin #	QFP 128 pin #	Function name
57		nc	71	36	Vss
58	2	PG4	72	39	Vcc
59	4	Vss	73	42	PE2
60	7	PC1	74	45	PE4
61	10	Vss	75	48	PE7
62	13	PC6	76	51	PD2
63	16	PB1	77	54	PD4
64	19	Vss	78	57	PD7
65	22	PB6	79	60	P31
66	25	PA1	80	63	P34
67	28	Vss	81		nc
68	31	PA6	82	66	P60
69		nc	83	68	Vss
70	34	P66	84	71	P63

Table 3-2 PGA177 to QFP128 Pin Assignment (E3471-61611)(Cont'd)

PGA 177 pin #	QFP 128 pin #	Function name	PGA 177 pin #	QFP 128 pin #	Function name
85	74	P25	99	112	P47
86	77	P22	100	115	P17
87	80	WDTOVF	101	118	P14
88	83	STBY	102	121	P11
89	86	EXTAL	103	124	MD1
90	89	Vcc	104	127	PG1
91	92	PF4	105		GND
92	95	PF1	106	5	Vcc
93		nc	107	8	PC2
94	98	P51	108	11	PC4
95	100	Vss	109	14	PC7
96	103	AVcc	110	17	PB2
97	106	P41	111	20	PB4
98	109	P44	112	23	PB7

Table 3-2 PGA177 to QFP128 Pin Assignment (E3471-61611)(Cont'd)

PGA 177 pin #	QFP 128 pin #	Function name	PGA 177 pin #	QFP 128 pin #	Function name
113	26	PA2	127	72	P27
114	29	PA4	128	75	P24
115		nc	129	78	P21
116	37	P65	130	81	RES
117	40	PE0	131	84	Vcc
118	43	PE3	132	87	Vss
119	46	PE5	133	90	PF6
120	49	PD0	134	93	PF3
121	52	PD3	135		nc
122	55	PD5	136	101	P52
123	58	Vcc	137	104	Vref
124	61	P32	138	107	P42
125		GND	139	110	P45
126	69	P61	140	113	AVss

Table 3-2 PGA177 to QFP128 Pin Assignment (E3471-61611)(Cont'd)

PGA 177 pin #	QFP 128 pin #	Function name	PGA 177 pin #	QFP 128 pin #	Function name
141	116	P16	155		nc
142	119	P13	156		nc
143	122	P10	157		nc
144	125	MD2	158		nc
145		GND	159		nc
146		nc	160		nc
147		nc	161		GND
148		nc	162		nc
149		nc	163		nc
150		nc	164		nc
151		nc	165		nc
152		nc	166		nc
153		nc	167		nc
154		nc	168		nc

Table 3-2 PGA177 to QFP128 Pin Assignment (E3471-61611)(Cont'd)

PGA 177 pin #	QFP 128 pin #	Function name	PGA 177 pin #	QFP 128 pin #	Function name
169		nc	174		nc
170		nc	175		nc
171		nc	176		nc
172		nc	177		GND
173		nc			

Table 3-3 PGA177 to QFP100 Pin Assignment (E3471-61612)

PGA 177 pin #	QFP 100 pin #	Function name	PGA 177 pin #	QFP 100 pin #	Function name
1		nc	15		nc
2		nc	16		nc
3	96	PG3	17		nc
4	7	Vss	18		nc
5	32	PC0	19		nc
6	35	PC3	20	15	PE1
7	37	PC5	21	18	Vss
8	41	PB0	22	21	PE6
9	44	PB3	23	24	PD1
10	46	PB5	24		nc
11	50	PA0	25	29	PD6
12	53	PA3	26	8	P30
13		nc	27	11	P33
14		nc	28	13	P35

Table 3-3 PGA177 to QFP100 Pin Assignment (E3471-61612)(Cont'd)

PGA 177 pin #	QFP 100 pin #	Function name	PGA 177 pin #	QFP 100 pin #	Function name
29		nc	43		nc
30		nc	44		nc
31		nc	45	54	P50
32		nc	46		nc
33		nc	47	59	P53
34	91	P26	48	79	P40
35	88	P23	49	82	P43
36	85	P20	50		nc
37	63	NMI	51	84	Vss
38	66	XTAL	52	4	P15
39	69	PF7	53	1	P12
40	71	PF5	54	57	MD0
41	74	PF2	55	93	PG0
42	76	PF0	56	95	PG2

Table 3-3 PGA177 to QFP100 Pin Assignment (E3471-61612)(Cont'd)

PGA 177 pin #	QFP 100 pin #	Function name	PGA 177 pin #	QFP 100 pin #	Function name
57		nc	71		nc
58	97	PG4	72		nc
59		nc	73	16	PE2
60	33	PC1	74	19	PE4
61	31	Vss	75	22	PE7
62	38	PC6	76	25	PD2
63	42	PB1	77	27	PD4
64	49	Vss	78	30	PD7
65	47	PB6	79	9	P31
66	51	PA1	80	12	P34
67		nc	81		nc
68		nc	82		nc
69		nc	83		nc
70		nc	84		nc

Table 3-3 PGA177 to QFP100 Pin Assignment (E3471-61612)(Cont'd)

PGA 177 pin #	QFP 100 pin #	Function name	PGA 177 pin #	QFP 100 pin #	Function name
85	90	P25	99		nc
86	87	P22	100	6	P17
87	60	WDTOVF	101	3	P14
88	64	STBY	102	100	P11
89	67	EXTAL	103	58	MD1
90		nc	104	94	PG1
91	72	PF4	105		GND
92	75	PF1	106	98	Vcc
93		nc	107	34	PC2
94	55	P51	108	36	PC4
95		nc	109	39	PC7
96	77	AVcc	110	43	PB2
97	80	P41	111	45	PB4
98		nc	112	48	PB7

Table 3-3 PGA177 to QFP100 Pin Assignment (E3471-61612)(Cont'd)

PGA 177 pin #	QFP 100 pin #	Function name	PGA 177 pin #	QFP 100 pin #	Function name
113	52	PA2	127	92	P27
114		nc	128	89	P24
115		nc	129	86	P21
116		nc	130	62	RES
117	14	PE0	131	65	Vcc
118	17	PE3	132	68	Vss
119	20	PE5	133	70	PF6
120	23	PD0	134	73	PF3
121	26	PD3	135		nc
122	28	PD5	136	56	P52
123	40	Vcc	137	78	Vref
124	10	P32	138	81	P42
125		GND	139		nc
126		nc	140	83	AVss

Table 3-3 PGA177 to QFP100 Pin Assignment (E3471-61612)(Cont'd)

PGA 177 pin #	QFP 100 pin #	Function name	PGA 177 pin #	QFP 100 pin #	Function name
141	5	P16	155		nc
142	2	P13	156		nc
143	99	P10	157		nc
144	61	MD2	158		nc
145		GND	159		nc
146		nc	160		nc
147		nc	161		GND
148		nc	162		nc
149		nc	163		nc
150		nc	164		nc
151		nc	165		nc
152		nc	166		nc
153		nc	167		nc
154		nc	168		nc

Table 3-3 PGA177 to QFP100 Pin Assignment (E3471-61612)(Cont'd)

PGA 177 pin #	QFP 100 pin #	Function name	PGA 177 pin #	QFP 100 pin #	Function name
169		nc	174		nc
170		nc	175		nc
171		nc	176		nc
172		nc	177		GND
173		nc			

Electrical Characteristics

The AC characteristics of the HP E3471 H8S/2000 emulator are listed in the following table.

Table 3-4. Clock timing (Vcc = 5.0V, f = 20MHz)

Characteristics	Symbol	H8S/2655 Vcc = 5V f = 20MHz		HP E3471A		Unit
		min	max	Typical *1	Worst Case	
Clock cycle time	t_{cyc}	50	500	-	-	ns
Clock pulse high width	t_{CH}	20	-	24	10	ns
Clock pulse low width	t_{CL}	20	-	21	10	ns
Clock rise time	t_{Cr}	-	5	2	15	ns
Clock fall time	t_{Cf}	-	5	3	15	ns
Crystal oscillator setting time(reset)	t_{OSC1}	10	-	10	10	ms
Crystal oscillator setting time (software standby)	t_{OSC2}	10	-	10	10	ms
External clock output setting delay time	t_{DEXT}	500	-	500	500	us

*1 Typical outputs measured with 50pF load

Table 3-5. Control signal timing (Vcc = 5.0V, f = 20MHz)

Characteristics	Symbol	H8S/2655 Vcc = 5V f = 20MHz		HP E3471A	Unit
		min	max	Worst Case	
$\overline{RES}$ setup time	t_{RESS}	200	-	275	ns
$\overline{RES}$ pulse width	t_{RESW}	20	-	20	tcyc
NMI reset setup time	t_{NMIRS}	200	-	260	ns
NMI reset hold time	t_{NMIRH}	200	-	200	ns
NMI setup time	t_{NMIS}	150	-	225	ns
NMI hold time	t_{NMIH}	10	-	10	ns
Interrupt pulse width	t_{NMIW}	200	-	235	ns
IRQ setup time	t_{IRQS}	150	-	180	ns
IRQ hold time	t_{IRQH}	10	-	10	ns
IRQ pulse width	t_{IRQW}	200	-	200	ns

Table 3-6. Bus timing (Vcc = 5.0V, f = 20MHz)

Characteristics	Symbol	H8S/2655 Vcc = 5V f = 20MHz		HP E3471A		Unit
		min	max	Typical *1	Worst Case	
Address delay time	t_{AD}	-	20	12	35	ns
Address setup time	t_{AS}	10	-	18	-5	ns
Address hold time	t_{AH}	15	-	22	0	ns
Pre-charge time	t_{PCH}	55	-	75	45	ns
CS delay time 1	t_{CSD1}	-	20	11	35	ns
CS delay time 2	t_{CSD2}	-	20	12	35	ns
CS pulse width	t_{CSW}	105	-	119	95	ns
Address strobe delay time	t_{ASD}	-	30	12	45	ns
Read strobe delay time 1	t_{RSD1}	-	30	10	45	ns
Read strobe delay time 2	t_{RSD2}	-	30	9	45	ns
CAS delay time	t_{CASD}	-	20	11	35	ns
Read data setup time	t_{RDS}	15	-	15*	45	ns
Read data hold time	t_{RDH}	0	-	0*	0	ns

Table 3-6. Bus timing (Vcc = 5.0V, f = 20MHz)(Cont'd)

Characteristics	Symbol	H8S/2655 Vcc = 5V f = 20MHz		HP E3471A		Unit
		min	max	Typical *1	Worst Case	
Read data access time 1	t_{ACC1}	-	25	25	-5	ns
Read data access time 2	t_{ACC2}	-	75	75	45	ns
Read data access time 3	t_{ACC3}	-	125	125	95	ns
Read data access time 4	t_{ACC4}	-	175	175	145	ns
Read data access time 5	t_{ACC5}	-	225	225	195	ns
WR delay time 1	t_{WRD1}	-	30	12	45	ns
WR delay time 2	t_{WRD2}	-	30	9	45	ns
Write data strobe pulse width 1	t_{WSW1}	30	-	42	20	ns
Write data strobe pulse width 2	t_{WSW2}	55	-	68	45	ns
Write data delay time	t_{WDD}	-	30	21	45	ns
Write data setup time	t_{WDS}	0	-	12	-15	ns
Write data hold time	t_{WDH}	10	-	10	-5	ns
WR setup time	t_{WCS}	15	-	18	0	ns
WR hold time	t_{WCH}	15	-	17	0	ns

Table 3-6. Bus timing (Vcc = 5.0V, f = 20MHz)(Cont'd)

Characteristics	Symbol	H8S/2655 Vcc = 5V f = 20MHz		HP E3471A		Unit
		min	max	Typical *1	Worst Case	
$\overline{\text{CAS}}$ setup time	t_{CSR}	15	-	20	0	ns
WAIT setup time	t_{WTS}	30	-	30	60	ns
WAIT set hold time	t_{WTH}	5	-	5	5	ns
BREQ setup time	t_{BRQS}	30	-	30	60	ns
BACK delay time	t_{BACD}	-	30	11	45	ns
Bus floating time	t_{BZD}	-	50	50	65	ns
BREQO delay time	t_{BRQOD}	-	30	15	45	ns

*1 Typical outputs measured with 50pF load

Table 3-7. DMAC timing (Vcc = 5.0V, f = 20MHz)

Characteristics	Symbol	H8S/2655 Vcc = 5V f = 20MHz		HP E3471A	Unit
		min	max	Worst Case	
$\overline{\text{DREQ}}$ setup time	t_{DRQS}	30	-	60	ns
$\overline{\text{DREQ}}$ hold time	t_{DRQH}	10	-	10	ns
$\overline{\text{TEND}}$ delay time	t_{TED}	-	30	45	ns
DACK delay time 1	t_{DACD1}	-	30	45	ns
DACK delay time 2	t_{DACD2}	-	30	45	ns

Table 3-8. Clock timing (Vcc = 3.0V, f = 10MHz)

Characteristics	Symbol	H8S/2655 Vcc = 3V f = 10MHz		HP E3471A		Unit
		min	max	Typical *1	Worst Case	
Clock cycle time	t_{cyc}	100	500	-	-	ns
Clock pulse high width	t_{CH}	35	-	46	35	ns
Clock pulse low width	t_{CL}	35	-	47	35	ns
Clock rise time	t_{Cr}	-	15	4	15	ns
Clock fall time	t_{Cf}	-	15	3	15	ns
Crystal oscillator setting time(reset)	t_{OSC1}	20	-	20	20	ms
Crystal oscillator setting time (software standby)	t_{OSC2}	20	-	20	20	ms
External clock output setting delay time	t_{DEXT}	500	-	500	500	us

*1 Typical outputs measured with 50pF load

Table 3-9. Control signal timing (Vcc = 3.0V, f = 10MHz)

Characteristics	Symbol	H8S/2655 Vcc = 3V f = 10MHz		HP E3471A	Unit
		min	max	Worst Case	
$\overline{RES}$ setup time	t_{RESS}	200	-	275	ns
$\overline{RES}$ pulse width	t_{RESW}	20	-	20	tcyc
NMI reset setup time	t_{NMIRS}	200	-	260	ns
NMI reset hold time	t_{NMIRH}	200	-	200	ns
NMI setup time	t_{NMIS}	150	-	225	ns
NMI hold time	t_{NMIH}	10	-	10	ns
Interrupt pulse width	t_{NMIW}	200	-	235	ns
IRQ setup time	t_{IRQS}	150	-	180	ns
IRQ hold time	t_{IRQH}	10	-	10	ns
IRQ pulse width	t_{IRQW}	200	-	200	ns

Table 3-10. Bus timing (Vcc = 3.0V, f = 10MHz)

Characteristics	Symbol	H8S/2655 Vcc = 3V f = 10MHz		HP E3471A		Unit
		min	max	Typical *1	Worst Case	
Address delay time	t_{AD}	-	40	12	40	ns
Address setup time	t_{AS}	20	-	43	20	ns
Address hold time	t_{AH}	30	-	46	25	ns
Pre-charge time	t_{PCH}	110	-	147	110	ns
CS delay time 1	t_{CSD1}	-	40	12	40	ns
CS delay time 2	t_{CSD2}	-	40	11	40	ns
CS pulse width	t_{CSW}	210	-	247	210	ns
Address strobe delay time	t_{ASD}	-	60	10	60	ns
Read strobe delay time 1	t_{RSD1}	-	60	9	60	ns
Read strobe delay time 2	t_{RSD2}	-	60	10	60	ns
CAS delay time	t_{CASD}	-	40	11	40	ns
Read data setup time	t_{RDS}	30	-	30	45	ns
Read data hold time	t_{RDH}	0	-	0	0	ns

Table 3-10. Bus timing (Vcc = 3.0V, f = 10MHz)(Cont'd)

Characteristics	Symbol	H8S/2655 Vcc = 3V f = 10MHz		HP E3471A		Unit
		min	max	Typical *1	Worst Case	
Read data access time 1	t_{ACC1}	-	50	50	45	ns
Read data access time 2	t_{ACC2}	-	100	100	95	ns
Read data access time 3	t_{ACC3}	-	150	150	145	ns
Read data access time 4	t_{ACC4}	-	200	200	195	ns
Read data access time 5	t_{ACC5}	-	250	250	245	ns
WR delay time 1	t_{WRD1}	-	60	11	60	ns
WR delay time 2	t_{WRD2}	-	60	11	60	ns
Write data strobe pulse width 1	t_{WSW1}	60	-	94	60	ns
Write data strobe pulse width 2	t_{WSW2}	100	-	144	100	ns
Write data delay time	t_{WDD}	-	60	18	60	ns
Write data setup time	t_{WDS}	0	-	37	0	ns
Write data hold time	t_{WDH}	20	-	20	20	ns
WR setup time	t_{WCS}	30	-	44	25	ns
WR hold time	t_{WCH}	30	-	43	25	ns

Table 3-10. Bus timing (Vcc = 3.0V, f = 10MHz)(Cont'd)

Characteristics	Symbol	H8S/2655 Vcc = 3V f = 10MHz		HP E3471A		Unit
		min	max	Typical *1	Worst Case	
$\overline{CAS}$ setup time	t_{CSR}	30	-	44	25	ns
WAIT setup time	t_{WTS}	60	-	60	60	ns
WAIT set hold time	t_{WTH}	10	-	10	10	ns
BREQ setup time	t_{BRQS}	60	-	60	60	ns
BACK delay time	t_{BACD}	-	60	9	60	ns
Bus floating time	t_{BZD}	-	100	100	100	ns
BREQO delay time	t_{BRQOD}	-	60	13	60	ns

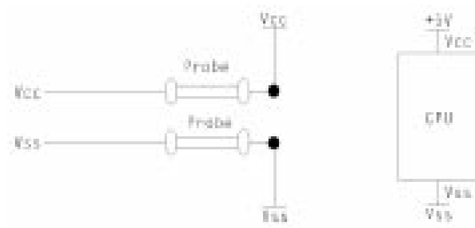
*1 Typical outputs measured with 50pF load

Table 3-11. DMAC timing (Vcc = 3.0V, f = 10MHz)

Characteristics	Symbol	H8S/2655 Vcc = 3V f = 10MHz		HP E3471A	Unit
		min	max	Worst Case	
$\overline{\text{DREQ}}$ setup time	t_{DRQS}	40	-	60	ns
$\overline{\text{DREQ}}$ hold time	t_{DRQH}	10	-	10	ns
$\overline{\text{TEND}}$ delay time	t_{TED}	-	60	60	ns
DACK delay time 1	t_{DACD1}	-	60	60	ns
DACK delay time 2	t_{DACD2}	-	60	60	ns

Target System Interface

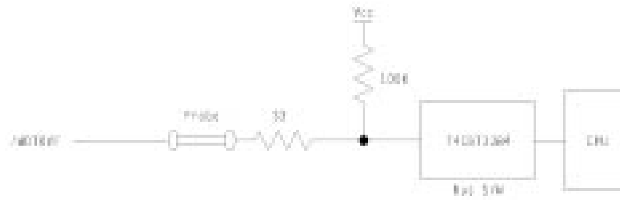
Vcc, Vss



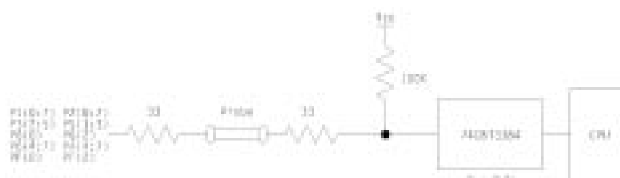
/RES, NMI, /STBY

MD0-2

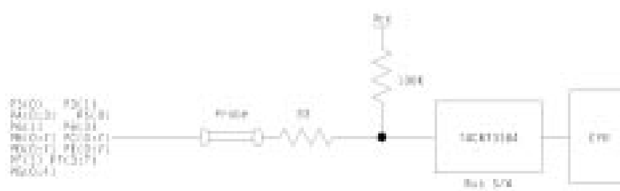
/WDTOVF



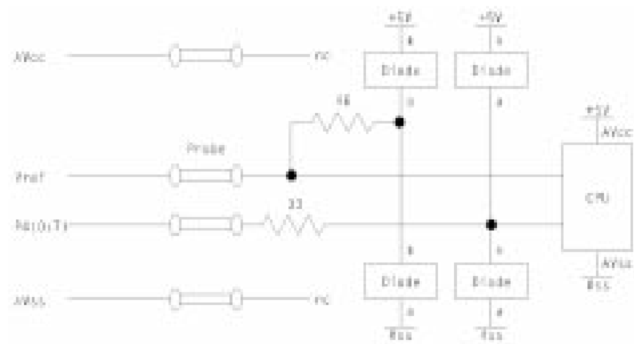
P1, P2, P3, P5, P6, PA, PF



P3, PA,   P5, P6, PB, PC,
PD, PE, PF, PG



P4, AVcc, Vref, AVss



EXTAL, XTAL Connect the circuits equivalent to those specified for H8/2000 series.

4

Configuring the H8S/2000 Emulator

In this chapter, we will discuss:

- how to configure the HP 64700 emulator for H8S/2000 microprocessor to fit your particular measurement needs.
- some restrictions of HP 64700 emulator for H8S/2000 microprocessor.

Types of Emulator Configuration

Emulation Processor to Emulator/Target System

These are the commands which are generally thought of as "configuration" items in the context of other HP 64700 emulator systems. The commands in this group set up the relationships between the emulation processor and the target system, such as determining how the emulator responds to requests for the processor bus. Also, these commands determine how the emulation processor interacts with the emulator itself; memory mapping and the emulator's response to certain processor actions are some of the items which can be configured.

These commands are the ones which are covered in this chapter.

Commands Which Perform an Action or Measurement

Several of the emulator commands do not configure the emulator; they simply start an emulator program run or other measurement, begin or halt an analyzer measurement, or allow you to display the results of such measurements.

These commands are covered in the examples presented in earlier manual chapters; they are also covered in the *HP 64700 Terminal Interface Reference* manual.

Coordinated Measurements

These commands determine how the emulator interacts with other measurement instruments, such as external analyzers, or other HP 64700 emulators connected via the CMB (Coordinated Measurement Bus).

These commands are covered in the *HP 64700 CMB User's Guide* and in the *HP 64700 Terminal Interface Reference* Manual.

Analyzer

The analyzer configuration commands are those commands which actually specify what type of measurement the analyzer is to make.

Some of the analyzer commands are covered earlier in this manual. You can also refer to the *HP 64700 Terminal Interface: Analyzer User's Guide* and the *HP 64700 Terminal Interface Reference* manual.

System

This last group of commands is used by you to set the emulator's data communications protocol, load or dump contents of emulation memory, set up command macros, and so on.

These commands are covered earlier in this manual and in the manual titled *HP 64700 Terminal Interface: User's Reference.*

Emulation Processor to Emulator/Target System

As noted before, these commands determine how the emulation processor will interact with the emulator's memory and the target system during an emulation measurement.

cf

The **cf** command defines how the emulation processor will respond to certain target system signals.

To see the default configuration settings defined by the **cf** command, type:

```
M> cf
```

You will see:

```
cf chip=2653
cf clk=int
cf mode=7
cf nmi=en
cf qbrk=dis
cf rrt=dis
cf rsp=0fffffc00
cf trst=en
```

Let's examine each of these emulator configuration options, with a view towards how they affect the processor's interaction with the emulator.

cf chip

The chip configuration item defines the microprocessor you emulate.

```
M> cf chip=<chip_name>
```

Valid <chip_name> are the following:

<chip_name>	**Description**
2653	Emulate H8S/2653 microprocessor.
2655	Emulate H8S/2655 microprocessor.
2241	Emulate H8S/2241 microprocessor.
2242	Emulate H8S/2242 microprocessor.
2245	Emulate H8S/2245 microprocessor.
2246	Emulate H8S/2246 microprocessor.

Note

When you use the H8S/2655 in mode 6 and map the address range of 010000h to 01ffffh as external address space, specify "2653" for the <chip_name>. When you map the range as internal ROM, specify "2655".

Note

The emulator does not configure the EAE bit in the system control register (SYSCR) automatically. Be sure to configure it manually.

Note

Executing this command will drive the emulator into the reset state.

cf clk

The **clk** (clock) option allows you to select whether the emulation processor's clock will be sourced by your target system or by the emulator.

```
M> cf clk=int
```

You can select the emulator's internal system clock using the above command.

```
M> cf clk=ext
```

You can specify that the emulator should use the clock input to the emulator probe from the target system. You must use a clock input conforming to the specifications of Table 4-1.

Table 4-1. Clock Speeds

Clock source	With 64700A	With 64700B
Internal	10MHz (System clock)	10MHz (System clock)
External	From 2.0MHz up to 20MHz (System clock)	From 33kHz up to 20MHz (System clock)

Note

When the emulator is connected to the target system operating at low voltage (2.7 to 4.5 V), the maximum system clock is 13 MHz.

Note

Executing this command will drive the emulator into the reset state.

cf mode

The **mode** (cpu operation mode) configuration item defines operation mode in which the emulator works.

```
M> cf mode=<mode_num>
```

When <mode_num> is selected, the emulator will operate in selected mode regardless of the mode setting by the target system.

Valid <mode_num> are following:

<mode_num>	**Description**
1	The emulator will operate in mode 1. (normal expanded mode: 8bit data bus)
2	The emulator will operate in mode 2. (normal expanded mode with on-chip ROM)
3	The emulator will operate in mode 3. (normal single-chip mode)
4	The emulator will operate in mode 4. (advanced expanded mode: 16bit data bus)
5	The emulator will operate in mode 5. (advanced expanded mode: 8bit data bus)
6	The emulator will operate in mode 6. (advanced expanded mode with on-chip ROM)
7	The emulator will operate in mode 7. (advanced single-chip mode)

Note

If mode ’2’, ’3’, ’6’ or ’7’ is selected and the emulation processor is configured no on-chip ROM type using the ’cf chip’ command, the emulator will ignore this mode configuration option and the emulation processor will be operated in mode ’1’.

Note Executing this command will drive the emulator into the reset state.

cf nmi

The **nmi** (non maskable interrupt) configuration item determines whether or not the emulator responds to NMI signal from the target system during foreground operation.

```
M> cf nmi=en
```

Using the above command, you can specify that the emulator will respond to NMI from the target system.

```
M> cf nmi=dis
```

The emulator won't respond to NMI from the target system.

The emulator does not accept any interrupt while in background monitor. Such interrupts are suspended while running the background monitor, and will occur when context is changed to foreground.

Note Executing this command will drive the emulator into the reset state.

cf qbrk

The **qbrk**(quick temporary break) configuration item specifies to use quick temporary break or not.

```
M> cf qbrk=en
```

Setting qbrk equal to en specifies that a temporary break to the monitor for an operation such as display registers will spend a very small amount of time in the monitor. The CMB does not work in this setting.

```
M> cf qbrk=dis
```

Setting qbrk equal to dis specifies that a temporary break to the monitor will spend more time in the monitor.

Note

Execution of this configuration option will drive the emulator into a reset state.

cf rrt

The **rrt** (restrict to real time) option lets you configure the emulator so that commands which cause the emulator to break to monitor and return to the user program will be rejected by the emulator command interpreter.

```
M> cf rrt=en
```

You can restrict the emulator to accepting only commands which don't cause temporary breaks to the monitor by entering the above command. Only the following emulator run/stop commands will be accepted:

rst (resets emulation processor)

b (breaks processor to background monitor until you enter another command)

r (runs the emulation processor from a given location)

s (steps the processor through a piece of code **--** returns to monitor after each step)

Commands which cause the emulator to break to the monitor and return, such as **reg**, **m** (for target memory display), and others will be rejected by the emulator.

Caution

If your target system circuitry is dependent on constant execution of program code, you should set this option to **cf rrt=en**. This will help insure that target system damage doesn't occur. However, remember that you can still execute the **rst**, **b** and **s** commands; you should use caution in executing these commands.

```
M> cf rrt=dis
```

When you use this command, all commands, regardless of whether or not they require a break to the emulation monitor, are accepted by the emulator.

cf rsp

The **rsp** (reset stack pointer) configuration item allows you to specify a value to which the stack pointer will be set upon the transition from emulation reset into the emulation monitor.

```
R> cf rsp=XXXXXXXX
```

where **XXXXXXXX** is a 32-bit even address, will set the stack pointer to that value upon entry to the emulation monitor after an emulation reset. You **cannot** set **rsp** at the following location.

- Odd address
- Internal I/O register area

For example, to set the stack pointer to 0ff00 hex, type:

```
R> cf rsp=0ff00
```

Now, if you break the emulator to monitor using the **b** command, the stack pointer will be modified to the value 0ff00 hex.

Note

Without a stack pointer, the emulator is unable to make the transition to the run state, step, or perform many other emulation functions. However, using this option **does not** preclude you from changing the stack pointer value or location within your program; it just sets the initial conditions to allow a run to begin.

---

cf trst

The **trst** (target reset) configuration item allows you to specify whether or not the emulator responds to /RES and /STBY signals from the target system during foreground operation. When running the background monitor, the emulator ignores such signals.

```
M> cf trst=en
```

When you enable target system reset with the above command, the emulator will respond to /RES input during foreground operation.

```
M> cf trst=dis
```

When disabled, the emulator won't respond to /RES and /STBY inputs from the target system.

Note

/RES and /STBY signals are always ignored during background operation regardless of this configuration.

Note

The H8S/2000 emulator dose not support hardware standby mode, and /STBY input will be given the emulator /RES input.

Note

Executing this command will drive the emulator into the reset state.

Memory Mapping

Before you begin an emulator session, you must specify the location and type of various memory regions used by your programs and your target system (whether or not it exists). You do this for several reasons:

- the emulator must know whether a given memory location resides in emulation memory or in target system memory. This allows the emulator to properly orient buffers for the given data transfer.
- the emulator needs to know the size of any emulation memory blocks so it can properly reserve emulation memory space for those blocks.
- the emulator must know if a given space is RAM (read/write), ROM (read only), or doesn't exist. This allows the emulator to determine if certain actions taken by the emulation processor are proper for the memory type being accessed. For example, if the processor tries to write to a emulation memory location mapped as ROM, the emulator will not permit the write (even if the memory at the given location is actually RAM). (You can optionally configure the emulator to break to the monitor upon such occurrence with the **bc -e rom** command.) Also, if the emulation processor attempts to access a non existent location (known as "guarded"), the emulator will break to the monitor.

You use the **map** command to define memory ranges and types for the emulator. The H8S/2000 emulator memory mapper allows you to define up to 16 different map terms; each map term has a minimum size of 1K bytes. If you specify a value less than 1K bytes, the emulator will automatically allocate an entire block. You can specify one of five different memory types (**erom, eram, trom, tram, grd**).

For example, you might be developing a system with the following characteristics:

- input port at 0f000 hex
- output port at 0f100 hex
- program and data from 1000 through 3fff hex

Suppose that the only thing that exists in your target system at this time are input and output ports and some control logic; no memory is available. You can reflect this by mapping the I/O ports to target system memory space and the rest of memory to emulation memory space. Type the following commands:

```
R> map 0f000..0f100 tram
R> map 1000..3fff eram
R> map
```

```
# remaining number of terms   : 14
# remaining emulation memory  : 3a000h bytes
map  001000..003fff    eram    # term 1
map  00f000..00f1ff     tram    # term 2
map  other tram
```

As you can see, the mapper rounded up the second term to 512 bytes block, since those are minimum size blocks supported by the H8S/2000 emulator.

Note

When you use the internal ROM, you **must** map that area to emulation memory. When you power on the emulator, all memory space except internal RAM is mapped to target RAM. Therefore, if you don't map internal ROM properly, you cannot access that area.

Note

You don't have to map internal RAM as emulation RAM, since the H8S/2000 emulator automatically maps internal RAM as emulation RAM and this area is behaved like internal RAM. However emulation memory system does not introduce internal RAM area in memory mapping display.

Note

If you map internal RAM area as emulation memory, this area is behaved like external memory overlapped with internal RAM. However the H8S/2000 emulator is always accessed internal RAM area mapped by the emulator. And if you map internal RAM as guarded memory, the emulator prohibits to access to this area by m commands.

Note

You should map all memory ranges except internal RAM used by your programs **before** loading programs into memory. This helps safeguard against loads which accidentally overwrite earlier loads if you follow a **map/load** procedure for each memory range.

Note

Executing this command will drive the emulator into the reset state.

For further information on mapping, refer to the examples in earlier chapters of this manual and to the *HP 64700 Terminal Interface User's Reference* manual.

Break Conditions

The bc command lets you configure the emulator's response to various emulation system and external events.

Write to ROM

If you want the emulator to break into the emulation monitor whenever the user program attempts to write to a memory region mapped as ROM, enter:

```
M> bc -e rom
```

You can disable this function by entering:

```
M> bc -d rom
```

When disabled, the emulator will not break to the monitor upon a write to ROM.

Note

If emulator writes to the memory mapped as ROM or guarded area in internal DMA cycles, the emulator will not break to the monitor regardless of this configuration.

Software Breakpoints

The **bp** command allows you to insert software traps in your code which will cause a break to the emulation monitor when encountered during program execution. If you want to enable the insertion and use of software breakpoints by the **bp** command, enter:

```
M> bc -e bp
```

To disable use of software breakpoints, type:

```
M> bc -d bp
```

Any breakpoints which previously existed in memory are disabled, but are not removed from the breakpoint table.

Trigger Signals

The HP 64700 emulator provides four different trigger signals which allow you to selectively start or stop measurements depending on the signal state. These are the **bnct** (rear panel BNC input), **cmbt** (CMB trigger input), **trig1** and **trig2** signals (provided by the analyzer).

You can configure the emulator to break to the monitor upon receipt of any of these signals. Simply type:

```
M> bc -e <signal>
```

For example, to have the emulator break to monitor upon receipt of the trig1 signal from the analyzer, type:

```
M> bc -e trig1
```

(Note: in this situation, you must also configure the analyzer to drive the **trig1** signal upon finding its trigger by entering **tgout trig1**).

Where to Find More Information

Due to the architecture of the HP 64700 emulators, there are a wide variety of items that affect how the emulator interacts with your system, controller, and other measuring instruments. If you need more configuration information, we suggest the following strategy:

If you need tutorial information --

- Emulator: look at this manual.
- Analyzer: look at the *Analyzer User's Guide* and this manual.
- CMB: look at the *CMB User's Guide*.

If you need reference information --

- Look at the *Terminal Interface User's Reference* manual (also contains some examples).

Notes

A

H8S/2000 Emulator Specific Command Syntax

The following pages contain descriptions of command syntax specific to the H8S/2000 emulator. The following syntax items are included (several items are part of other command syntax):

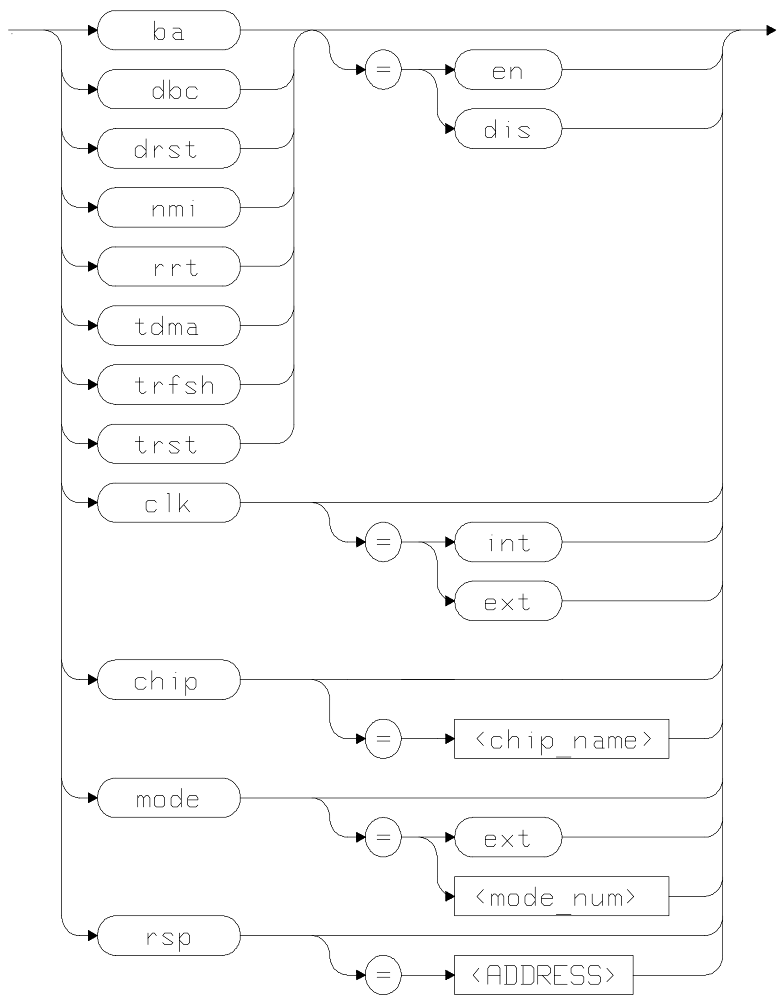
- <CONFIG_ITEMS>. May be specified in the **cf** (emulator configuration) and **help cf** commands.
- <ADDRESS>. May be specified in emulation commands which allow addresses to be entered.
- <REG_NAME>. May be specified in the **reg** (register) command.

Command and error messages which are specific to the H8S/2000 emulator are also described in this chapter.

CONFIG_ITEMS

Summary H8S/2000 emulator configuration items.

Syntax



Description

The H8S/2000 emulator has several dedicated configuration items which allow you to specify the emulator's interaction with the target system and the rest of the emulation system. These items are:

chip	Select processor to be emulated.
clk	Select internal/external clock source.
mode	Determine emulator processor operation mode.
nmi	Enable/disable NMI (non maskable interrupt) from target system.
qbrk	Enable/disable quick temporary break.
rrt	Restrict emulator to real time runs.
rsp	Specify system stack pointer value to load upon each transition from emulation reset to the monitor.
trst	Enable/disable target system reset.

Complete explanations of all configuration items are given in chapter 4 of this manual.

Examples To select an external clock, type:

```
M> cf clk=ext
```

You can obtain the status of configuration items by typing the item name without a value. You can also specify multiple configuration items on the same line. Type:

```
M> cf nmi=dis rrt=dis clk
```

cf clk=ext

Related information Refer to the cf syntax pages in the *User's Reference* manual. Also, refer to chapter 3 of this manual for complete information about each configuration item.

ADDRESS

Summary Address specification used in emulation commands.

Description The **<ADDRESS>** parameter used in emulation commands is specified in 24 bits address information.

Examples **m 1000**

m 200000..2000ff

REGISTER CLASS and NAME

Summary H8S/2000 register designators. All available register class names and register names are listed below.

<REG_CLASS>

<REG_NAME> Description

* (All basic registers)

pc	Program counter
ccr	Condition code register
exr	Extended register
er0	Register ER0
er1	Register ER1
er2	Register ER2
er3	Register ER3
er4	Register ER4
er5	Register ER5
er6	Register ER6
er7	Register ER7
sp	Stack pointer
mach	Multiply and accumulate register H
macl	Multiply and accumulate register L
m dcr	Mode control register(Read Only)

sys (System control)

sbycr	Stand-by control register
syscr	System control register
sckcr	System clock control register
mdcr	Mode control register(Read Only)
mstpcr	Module stop control register
lpwcr	Low power control register

intc (Interrupt controller)

syscr	System control register
iscr	IRQ sense control register
ier	IRQ enable register
isr	IRQ status register
icra	Interrupt control register A
icrb	Interrupt control register B
icrc	Interrupt control register C
ipra	Interrupt priority register A
iprb	Interrupt priority register B
iprc	Interrupt priority register C
iprd	Interrupt priority register D
ipre	Interrupt priority register E
iprf	Interrupt priority register F
iprg	Interrupt priority register G
iprh	Interrupt priority register H
ipri	Interrupt priority register I
iprj	Interrupt priority register J
iprk	Interrupt priority register K

busc (Bus controller)

abwcr	Byte/Word area control register
astcr	2/3 state area control register
wcr	Wait control register
bcrh	Bud control register H
bcrl	Bud control register L
mcr	Memory control register
dramcr	DRAM control register
rtcnt	Refresh timer counter register
rtcor	Refresh timer constant register

dmacg (DMA controller general)

dmawer	DMA write enable register
dmatcr	DMA terminal control register
dmacr0a	DMA control register 0A
dmacr0b	DMA control register 0B
dmacr1a	DMA control register 1A
dmacr1b	DMA control register 1B
dmabcr	DMA band control register

dmac0 (DMA controller 0)

mar0a	Memory address register 0A
ioar0a	I/O address register 0A
etcr0a	Transfer count register 0A
mar0b	Memory address register 0B
ioar0b	Transfer count register 0B
etcr0b	I/O address register 0B

dmac1 (DMA controller 1)

mar1a	Memory address register 1A
ioar1a	I/O address register 1A
etcr1a	Transfer count register 1A
mar1b	Memory address register 1B
ioar1b	Transfer count register 1B
etcr1b	I/O address register 1B

dtc (Data transfer controller)

dtcera	DTC enable register A
dtcerb	DTC enable register B
dtcerc	DTC enable register C
dtcerd	DTC enable register D
dtcere	DTC enable register E
dtcerf	DTC enable register F
dtvect	DTC vector register

port (I/O port)

p1ddr	Port 1 data direction register(Write Only)
p2ddr	Port 2 data direction register(Write Only)
p3ddr	Port 3 data direction register(Write Only)
p5ddr	Port 5 data direction register(Write Only)
p6ddr	Port 6 data direction register(Write Only)
paddr	Port A data direction register(Write Only)
pbddr	Port B data direction register(Write Only)
pcddr	Port C data direction register(Write Only)
pdddr	Port D data direction register(Write Only)
peddr	Port E data direction register(Write Only)
pfddr	Port F data direction register(Write Only)
pgddr	Port G data direction register(Write Only)
p1dr	Port 1 data register
p2dr	Port 2 data register
p3dr	Port 3 data register
p5dr	Port 5 data register
p6dr	Port 6 data register
padr	Port A data register
pbdr	Port B data register
pcdr	Port C data register
pddr	Port D data register
pedr	Port E data register
pfdr	Port F data register
pgdr	Port G data register
port1	Port 1 register(Read Only)
port2	Port 2 register(Read Only)
port3	Port 3 register(Read Only)
port4	Port 4 register(Read Only)
port5	Port 5 register(Read Only)
port6	Port 6 register(Read Only)
porta	Port A register(Read Only)
portb	Port B register(Read Only)
portc	Port C register(Read Only)
portd	Port D register(Read Only)
porte	Port E register(Read Only)
portf	Port F register(Read Only)
portg	Port G register(Read Only)

papcr	Port A pull-up MOS control register
pbpcr	Port B pull-up MOS control register
pcpcr	Port C pull-up MOS control register
pdpcr	Port D pull-up MOS control register
pepcr	Port E pull-up MOS control register
p3odr	Port 3 open drain control register
paodr	Port A open drain control register

ipug (16 bit integrated timer pulse unit general)

tstr	Timer start register
tsyr	Timer synchro register

ipu0 (16 bit integrated timer pulse unit 0)

tcr0	Timer control register 0
tmdr0	Timer mode register 0
tior0	Timer I/O control register 0
tier0	Timer interrupt enable register 0
tsr0	Timer status register 0
tcnt0	Timer counter 0
tgr0a	Timer general register 0A
tgr0b	Timer general register 0B
tgr0c	Timer general register 0C
tgr0d	Timer general register 0D

ipu1 (16 bit integrated timer pulse unit 1)

tcr1	Timer control register 1
tmdr1	Timer mode register 1
tior1	Timer I/O control register 1
tier1	Timer interrupt enable register 1
tsr1	Timer status register 1
tcnt1	Timer counter 1
tgr1a	Timer general register 1A
tgr1b	Timer general register 1B
tgr1c	Timer general register 1C
tgr1d	Timer general register 1D

ipu2 (16 bit integrated timer pulse unit 2)

tcr2	Timer control register 2
tmdr2	Timer mode register 2
tior2	Timer I/O control register 2
tier2	Timer interrupt enable register 2
tsr2	Timer status register 2
tcnt2	Timer counter 2
tgr2a	Timer general register 2A
tgr2b	Timer general register 2B
tgr2c	Timer general register 2C
tgr2d	Timer general register 2D

ipu3 (16 bit integrated timer pulse unit 3)

tcr3	Timer control register 3
tmdr3	Timer mode register 3
tior3	Timer I/O control register 3
tier3	Timer interrupt enable register 3
tsr3	Timer status register 3
tcnt3	Timer counter 3
tgr3a	Timer general register 3A
tgr3b	Timer general register 3B
tgr3c	Timer general register 3C
tgr3d	Timer general register 3D

ipu4 (16 bit integrated timer pulse unit 4)

tcr4	Timer control register 4
tmdr4	Timer mode register 4
tior4	Timer I/O control register 4
tier4	Timer interrupt enable register 4
tsr4	Timer status register 4
tcnt4	Timer counter 4
tgr4a	Timer general register 4A
tgr4b	Timer general register 4B
tgr4c	Timer general register 4C
tgr4d	Timer general register 4D

ipu5 (16 bit integrated timer pulse unit 5)

tcr5	Timer control register 5
tmdr5	Timer mode register 5
tior5	Timer I/O control register 5
tier5	Timer interrupt enable register 5
tsr5	Timer status register 5
tcnt5	Timer counter 5
tgr5a	Timer general register 5A
tgr5b	Timer general register 5B
tgr5c	Timer general register 5C
tgr5d	Timer general register 5D

ppc (Programable pulse generator)

pcr	TPC output control register
pmr	TPC output mode register
nder	Next data enable register
podr	Output data register
ndrh	Next data register H (address: 0xxff4ch)
ndrl	Next data register L (address: 0xxff4dh)
ndrh2	Next data register H (address: 0xxff4eh)
ndrl0	Next data register L (address: 0xxff4fh)

tmr0 (8 bit timer 0)

ttcr0	Timer control register 0
ttcsr0	Timer control/status register 0
ttcora0	Timer constant register A0
ttcorb0	Timer constant register B0
ttcnt0	Timer counter register 0

tmr1 (8 bit timer 1)

ttcr1	Timer control register 1
ttcsr1	Timer control/status register 1
ttcora1	Timer constant register A1
ttcorb1	Timer constant register B1
ttcnt1	Timer counter register 1

wdt (Watch dog timer)

wdtcsr	Timer control/status register
wdtcnt	Timer counter register
rstcsr	Reset control/status register

sci0 (Serial communication interface 0)

smr0	Serial mode register 0
brr0	Bit rate register 0
scr0	Serial control register 0
tdr0	Transmit data register 0
ssr0	Serial status register 0
rdr0	Receive data register 0 (Read Only)
scmr0	Smart card mode register 0

sci1 (Serial communication interface 1)

smr1	Serial mode register 1
brr1	Bit rate register 1
scr1	Serial control register 1
tdr1	Transmit data register 1
ssr1	Serial status register 1
rdr1	Receive data register 1 (Read Only)
scmr1	Smart card mode register 1

sci2 (Serial communication interface 2)

smr2	Serial mode register 2
brr2	Bit rate register 2
scr2	Serial control register 2
tdr2	Transmit data register 2
ssr2	Serial status register 2
rdr2	Receive data register 2 (Read Only)
scmr2	Smart card mode register 2

adc (A/D converter)

addra	A/D data register A(Read Only)
addrb	A/D data register B(Read Only)
addrc	A/D data register C(Read Only)
addrd	A/D data register D(Read Only)
addre	A/D data register E(Read Only)
addrf	A/D data register F(Read Only)
addrg	A/D data register G(Read Only)
addrh	A/D data register D(Read Only)
adcsr	A/D data register D(Read Only)
adcr	A/D control/status register
	A/D control register

dac (D/A converter)

dadr0	D/A data register 0
dadr1	D/A data register 1
dacr	D/A control register

NOCLASS

The following register names are not included in any register class.

r0	Register R0
r1	Register R1
r2	Register R2
r3	Register R3
r4	Register R4
r5	Register R5
r6	Register R6
r7	Register R7
e0	Register E0
e1	Register E1
e2	Register E2
e3	Register E3
e4	Register E4
e5	Register E5
e6	Register E6
e7	Register E7
r0h	Register R0H
r0l	Register R0L
r1h	Register R1H
r1l	Register R1L
r2h	Register R2H
r2l	Register R2L
r3h	Register R3H
r3l	Register R3L
r4h	Register R4H
r4l	Register R4L
r5h	Register R5H
r5l	Register R5L
r6h	Register R6H
r6l	Register R6L
r7h	Register R7H
r7l	Register R7L

Emulator Specific Error Messages

The following is the error messages which are specific to the H8S/2000 emulator. The cause of the errors is described, as well as the action you must take to remedy the situation.

Message

140 : Invalid address for run or step in current mode

Cause

This error occurs when you attempt to execute user program (with **r** or **s** command) from address over area of current mode.

Message

141 : Use register command to modify I/O registers

Cause

This error occurs when you attempt to modify the internal I/O register using the **m** or **load** command.

Message

170 : Copy target image not supported

Cause

This error occurs when you attempt to execute the **cim** command.

Message

178 : Update HP64700 system firmware to A.04.00 or newer

Cause

This error occurs when the version of the controller firmware you use is earlier than A.04.00.

Message 179 : Memory module not found

Cause

This error occurs when no memory module is connected or when a memory module not supported is connected.

Index