HP 64797

H8/3048 Emulator Terminal Interface

User's Guide

HP Part No. 64797-97000
January 1995

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 **64797-97000, January 1995**

Using This Manual

This manual is designed to give you an introduction to the HP 64797 H8/3048 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 H8/3048 emualtor

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 H8/3048 emulator features and how they can help you in developing new hardware and software.

Chapter 2 A brief introduction to using the H8/3048 Emulator. You will load and execute a short program, and make some measurements using the emulation analyzer.

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

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

Chapter 5 How to use H8/3048 internal Flash ROM functions on the H8/3048 emulator.

Appendix A H8/3048 Emulator Specific Command Syntax and Error Message

Contents

Illustrations

Tables

6-Contents

1

Introduction

The topics in this chapter include:

- Purpose of the H8/3048 Emulator
- Features of the H8/3048 Emulator

Purpose of the H8/3048 Emulator

The H8/3048 Emulator is designed to replace the H8/3048 microprocessor in your target system so you can control operation of the microprocessor in your application hardware (usually referred to as the *target system*). The H8/3048 emulator performs just like the H8/3048 microprocessor, but is a device that allows you to control the H8/3048 microprocessor directly. These features allow you to easily debug software before any hardware is available, and ease the task of integrating hardware and software.

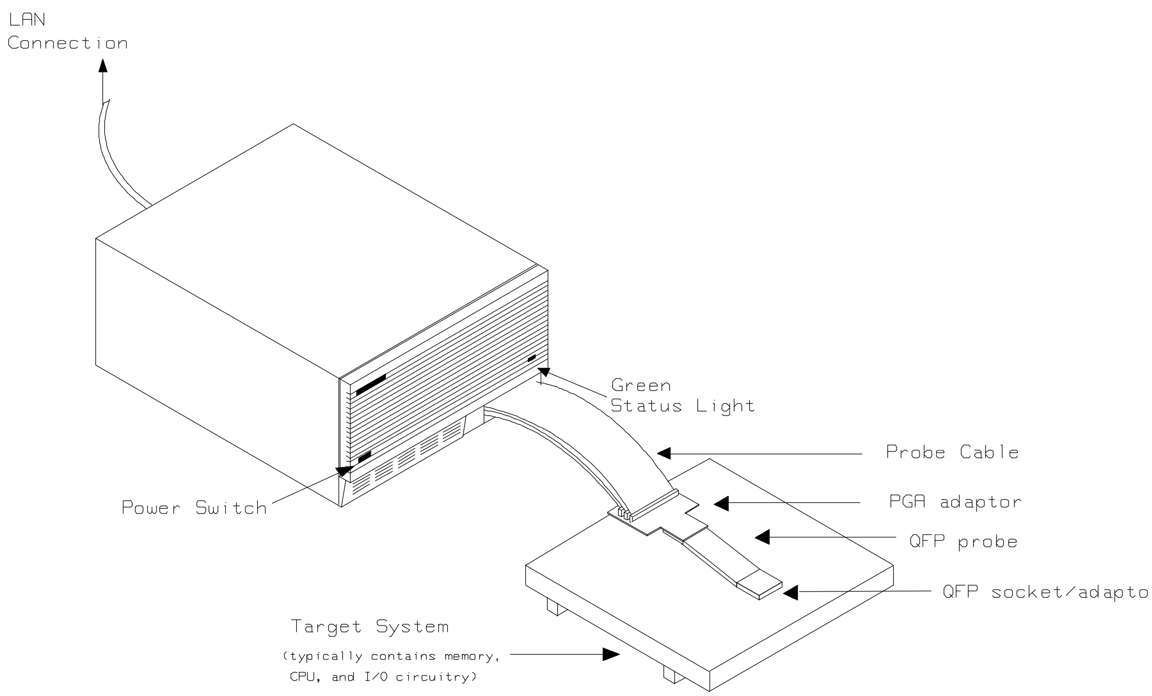


Figure 1-1. HP 64797 Emulator for the H8/3048

Features of the H8/3048 Emulator

Supported Microprocessors

The HP 64797A H8/3048 emulator supports the microprocessors listed in Table 1-1.

Table 1-1. Supported Microprocessors

Supported Microprocessors			
Type	**Package**	**On-chip ROM**	**Supply Voltage**
H8/3048	100 pin QFP	PROM	4.75 to 5.25V
			2.70 to 5.25V
		Masked ROM	4.75 to 5.25V
			2.70 to 5.25V
H8/3048F	100 pin QFP	Flash Memory	4.25 to 5.25V
			2.70 to 5.25V
H8/3047	100 pin QFP	Masked ROM	4.75 to 5.25V
			2.70 to 5.25V
H8/3044	100 pin QFP	Masked ROM	4.75 to 5.25V
			2.70 to 5.25V

The H8/3048 emulator is provided without any adaptor and probe. To emulate each processor with your target system, you need to purchase appropriate adaptor and probe. 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. You need to select a clock input conforming to the specification of Table 1-2.

Crystal ocsillator frequency of internal clock is 8MHz.

Refer to the "Configuration the Emulator" Chapter in this manual for more details.

Table 1-2. Clock Speeds

Emulation Memory	Clock Speed		
	With HP64784D	**With HP64784E**	**With HP64797B**
64726A 64727A 64728A	From 1 up to 16MHz (System Clock)	From 1 up to 16MHz (System Clock)	From 1 up to 13MHz (System Clock)
64729A	From 1 up to 18MHz (System Clock)	From 1 up to 18MHz (System Clock)	From 1 up to 13MHz (System Clock)

Emulation memory

The H8/3048 emulator is used with one of the following Emulation Memory Cards.

- HP 64726A 128K byte Emulation Memory Card
- HP 64727A 512K byte Emulation Memory Card
- HP 64728A 1M byte Emulation Memory Card
- HP 64729A 2M byte Emulation Memory Card

When you use the HP64797A emulator over 16MHz, you have to use the HP 64729A 2M byte Emulation Memory Card.

You can define up to 16 memory ranges (at 512 byte boundaries and least 512 byte in length.) The emulator occupies 6K byte, which is used for monitor program and internal RAM of microprocessor mapped as emulation RAM, leaving 122K, 506K, 1018K, 2042K 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 H8/3048 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 64703A 64-channel Emulation Bus Analyzer and 16-channel State/Timing Analyzer.
- HP 64794A/C/D Deep Emulation Bus Analyzer

The Emulation Bus Analyzer monitors the emulation processor using an internal analysis bus. The HP 64703A 64-channel Emulation Bus Analyzer and 16-channel State/Timing Analyzer allows you to probe up to 16 different lines in your target system.

Registers

You can display or modify the H8/3048 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.

Easy Product Updates

Because the HP 64700 Series development tools(emulator, analyzer, LAN board) contain programmable parts, it is possible to reprogram the firmware and some of the hardware without disassembling the HP 64700B Card Cage. This means that you'll be able to update product firmware, if desired, without having to call to HP field representative to your site.

Limitations, Restrictions

Foreground Monitor

Foreground monitor is not supported for the H8/3048 emulator.

DMA Support

Direct memory access to the emulation by external DMAC is not allowed.

Watch Dog Timer in Background

Watch dog timer is suspended count up while the emulator is running in 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 H8/3048 emulator. Hardware standby request from target system will drive the emulator into the reset state.

Interrupts in Background Cycles

The H8/3048 emulator does not accept any interrupts while in background monitor. Such interrupts are suspended while running the background monitor, and will occur when context is changed to foreground.

On-chip Flash Memory

The H8/3048 emulator uses emulation memory instead of actual on-chip flash memory. So, operation for on-chip flash memory is different from H8/3048 microprocessor. Refer to "Using the On-chip Flash Memory" chapter in this manual for more details.

Evaluation chip

Hewlett-Packard makes no warranty of the problem caused by the H8/3048 Evaluation chip in the emulator.

Notes

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 H8/3048 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
   - Transparent configuration
   - Remote configuration
   - Local Area Network 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 trouble shooting 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 - 0ff800 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 7A01000FF800                         MOV.L           #Cmd_Input,ER1
                           ;*****************************************************
                           ;* Clear previous command.
                           ;*****************************************************
00100C F800                Clear            MOV.B           #H'00,R0L
00100E 6AA8000FF800                         MOV.B           R0L,@Cmd_Input
                           ;*****************************************************
                           ;* Read command input byte.  If no command has been
                           ;* entered, continue to scan for it.
                           ;*****************************************************
001014 6A2A000FF800        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 7A05000FF804       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 7A05000FF804                       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

0FF800                                    .SECTION        Data,DATA,LOCATE=H'FF800
                          ;*****************************************************
                          ;* Command input area.
                          ;*****************************************************
0FF800 00000004           Cmd_Input       .RES.L          1
                          ;*****************************************************
                          ;* Destination of the command messages.
                          ;*****************************************************
0FF804 00000100           Msg_Dest        .RES.W          H'80
0FF904                    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 sysmbol file
  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 -o         - data received from the non-command source port
  load -s <str>   - send a character string out the other port
  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)
  load -c <file>  - data is received from the 64000. file name format is:
                    <filename>:<userid>:absolute
```

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 ba=en
cf chip=3048
cf clk=int
cf dbc=en
cf mode=7
cf nmi=en
cf rrt=dis
cf rsp=9
cf tdma=en
cf trfsh=en
cf trst=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 ba=en
cf chip=3042
cf clk=ext
cf dbc=en
cf mode=7
cf nmi=en
cf rrt=en
cf rsp=9
cf tdma=en
cf trfsh=en
cf trst=en
```

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

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

3. Now, you need to set up stack pointer.
   Type:

```
R> cf rsp=0ff904
```

4. 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 128K, 512K, 1M or 2M bytes, mappable in 512 byte blocks. The monitor occupies 2K bytes and the emulator maps 4K bytes for internal RAM as emulation RAM automatically, leaving 122K, 506K, 1018K or 2042K 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/on-chip flash memory 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.

Getting the Sample Program into Emulation Memory

This section assumes you are using the emulator in one of the following three configurations:

1. Connected only to a terminal, which is called the *standalone* configuration. In the standalone configuration, you must modify memory to load the sample program.

2. Connected between a terminal and a host computer, which is called the *transparent* configuration. In the transparent configuration, you can load the sample program by downloading from the "other" port.

3. Connected to a host computer and accessed via a terminal emulation program. This configurations is called *remote* configurations. In the remote configuration, you can load the sample program by downloading from the same port.

Standalone Configuration

If you are operating the emulator in the standalone configuration, the only way to load the sample program into emulation memory is by modifying emulation memory locations with the **m** (memory display/modification) command.

You can enter the sample program into memory with the **m** command as shown below.

(Note the hex letters must be preceded by a digit.)

```
R> m 001000..00100f=7a,07,00,0f,0f9,04,7a,01,00,0f,0f8,00,0f8,00,6a,0a8
R> m 001010..00101f=00,0f,0f8,00,6a,2a,00,0f,0f8,00,0aa,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=00,0f,0f8,04,0fe,20,68,0d8,0b,05,1a,0e,46,0f8,7a,05
R> m 001060..00106f=00,0f,0f8,04,6c,4e,68,0de,0b,05,1a,0b,46,0f6,40,9c
R> m 002000="THIS IS MESSAGE ATHIS IS MESSAGE BINVALID COMMAND"
```

After entering the opcodes and operands, you would typically display memory in mnemonic format to verify that the values entered are correct (see the example below). If any errors exist, you can modify individual locations. Also, you can use the **cp** (copy memory) command if, for example, a byte has been left out, but the locations which follow are correct.

Note

Be careful about using this method to enter programs from the listings of relocatable source files. If source files appear in relocatable sections, the address values of references to locations in other relocatable sections are not resolved until link-time. The correct values of these address operands will not appear in the assembler listing.

Transparent Configuration

If your emulator is connected between a terminal and a host computer, you can download programs into memory using the **load** command with the **-o** (from other port) option. The **load** command will accept absolute files in the following formats:

- HP absolute.
- Intel hexadecimal.
- Tektronix hexadecimal.

- Motorola S-records.

The examples which follow will show you the methods used to download HP absolute files and the other types of absolute files.

HP Absolutes

If you have a Graphical User Interface, a file format converter is provided with it. The file format converter can convert Hitachi and IAR format files to HP Absolute files (Refer to *Softkey Interface User's Guide* for more details).

Downloading HP format absolute files requires the **transfer** protocol. The example below assumes that the **transfer** utility has been installed on the host computer.

Note

Notice that the transfer command on the host computer is terminated with the <ESCAPE>**g** characters; by default, these are the characters which temporarily suspend the transparent mode to allow the emulator to receive data or commands.

```
R>load -hbo <RETURN> <RETURN>
$ transfer -rtb cmd_rds.X <ESCAPE>g
####
R>
```

Other Supported Absolute Files

The example which follows shows how to download Intel hexadecimal files by the same method (but different **load** options) can be used by load Tektronix hexadecimal and Motorola S-record files as well.

```
R>load -io <RETURN> <RETURN>
 $ cat ihexfile <ESCAPE>g
#####
Data records = 00003 Checksum error = 00000
R>
```

Remote Configuration

If the emulator is connected to a host computer, and you are accessing the emulator from the host computer via a terminal emulation program, you can also download files with the **load** command. However, in the remote configuration, files are loaded from the same port that commands are entered from. For example, if you wish to download a Tektronix hexadecimal file from a Vectra personal computer, you would enter the following commands.

```
R>load -t <RETURN>
```

After you have entered the **load** command, exit from the terminal emulation program to the MS-DOS operating system. Then, copy your hexadecimal file to the port connected to the emulator, for example:

```
C:\copy thexfile com1: <RETURN>
```

Now you can return to the terminal emulation program and verify that the file was loaded correctly.

For More Information

For more information on downloading absolute files, refer to the **load** command description in the *HP 64700 Emulators Terminal Interface: User's Reference* manual.

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 #000ff904,ER7
0001006  -                 MOV.L #000ff800,ER1
000100c  -                 MOV.B #00,R0L
000100e  -                 MOV.B R0L,@0ff800
0001014  -                 MOV.B @0ff800,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 #000ff804,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 #000ff804,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=001014 ccr=84 er0=00000000 er1=000ff800 er2=00000000 er3=00000000
reg er4=00000000 er5=00000000 er6=00000000 er7=000ff904 sp=000ff904 mdcr=c7
```

Notice that ER1 contains 0ff800 hex.

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

   Type:

   U> **m -db 0ff800**

   You will see:

```
00ff800..00ff800     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 0ff800=4**

   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 0ff804..0ff823**

   You will see:

```
00ff804..00ff813     54 48 49 53 20 49 53 20 4d 45 53 53 41 47 45 20
00ff814..00ff823     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 #000ff904,ER7
PC =0001006
reg pc=001006 ccr=80 er0=00000000 er1=000ff800 er2=00000000 er3=00000000
reg er4=00000000 er5=00000000 er6=00000000 er7=000ff904 sp=000ff904 mdcr=c7
```

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 #000ff800,ER1
PC =000100c
reg pc=00100c ccr=80 er0=00000000 er1=000ff800 er2=00000000 er3=00000000
reg er4=00000000 er5=00000000 er6=00000000 er7=000ff904 sp=000ff904 mdcr=c7
```

Tracing Program Execution

Predefined Trace Labels

Three trace labels are predefined in the H8/3048 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..57
```

Predefined Status Equates

Common values for the H8/3048 status trace signals have been predefined. You can view these predefined equates by entering the **equu** command with no options.

```
M> equ
```

```
### Equates ###
equ bg=0xxx0xxxxxxxxxxxxxxy
equ byte=0xxxxxx1xxxx1xxxx1xy
equ cpu=0xxxxxx1xxxx11xxxxxy
equ data=0xxxxxx1xxxx1x1xxxxy
equ dma=0xxxxxx1xxxx10xxxxxy
equ fetch=0xxxxxx1x1xx110xx01y
equ fg=0xxx1xxxxxxxxxxxxxxy
equ grd=0xxxx011xxxx1xx1xxxy
equ intack=0xxxxxxx0xxxxxxxxxxy
equ io=0xxxxxx1xxxx1xx0xxxy
equ mem=0xxxxxx1xxxx1xx1xxxy
equ read=0xxxxxx1xxxx1xxxxx1y
equ refresh=0xxxxxx1xxxx01xxxxxy
equ word=0xxxxxx1xxxx1xxxx0xy
equ write=0xxxxxx1xxxx1xxxxx0y
equ wrrom=0xxxx101xxxx1xx1xx0y
```

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 ff804 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=0ff804
```

To store only the accesses to the address range ff804 through ff815 hex, type :

```
M> tsto addr=0ff804..0ff815
```

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 ff804 hex). Type:

```
U> m 0ff800=41
```

To display the trace list, type :

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

You will see:

```
   Line  addr,H  data,A    count,R  seq
  -----  ------  ------  ---------  ---
      0  0ff804  ..            ---   +
      1  0ff805  ..       1.480 uS   .
      2  0ff806  ..       1.520 uS   .
      3  0ff807  ..       1.480 uS   .
      4  0ff808  ..       1.520 uS   .
      5  0ff809  ..       1.480 uS   .
      6  0ff80a  ..       1.520 uS   .
      7  0ff80b  ..       1.480 uS   .
      8  0ff80c  ..       1.520 uS   .
      9  0ff80d  ..       1.480 uS   .
     10  0ff80e  ..       1.520 uS   .
     11  0ff80f  ..       1.480 uS   .
     12  0ff810  ..       1.520 uS   .
     13  0ff811  ..       1.480 uS   .
     14  0ff812  ..       1.520 uS   .
     15  0ff813  ..       1.480 uS   .
     16  0ff814  ..       1.520 uS   .
     17  0ff815  ..       1.480 uS   .
     18  0ff804  TT       24.00 uS   .
     19  0ff805  HH       2.240 uS   .
     20  0ff806  II       2.280 uS   .
     21  0ff807  SS       2.240 uS   .
     22  0ff808  ..       2.240 uS   .
     23  0ff809  II       2.240 uS   .
     24  0ff80a  SS       2.280 uS   .
     25  0ff80b  ..       2.240 uS   .
     26  0ff80c  MM       2.240 uS   .
     27  0ff80d  EE       2.240 uS   .
     28  0ff80e  SS       2.280 uS   .
     29  0ff80f  SS       2.240 uS   .
     30  0ff810  AA       2.240 uS   .
     31  0ff811  GG       2.240 uS   .
     32  0ff812  EE       2.280 uS   .
     33  0ff813  ..       2.240 uS   .
     34
```

If you look at the last lines of the trace listing, you will notice that the analyzer seems to have stored only part of the output message, even though you specified more than the full range needed to store all of the message. The reason for this is that the analyzer has a storage pipeline, which holds states that have been acquired but not yet written to trace memory. To see all of the states, halt the analyzer by typing:

U> **th**

You will see:

```
Emulation trace halted
```

Now display the trace list:

U> **tl 0..34**

You will see:

Line	addr,H	data,A	count,R	seq
0	0ff804	..	---	+
1	0ff805	..	1.480 uS	.
2	0ff806	..	1.520 uS	.
3	0ff807	..	1.480 uS	.
4	0ff808	..	1.520 uS	.
5	0ff809	..	1.480 uS	.
6	0ff80a	..	1.520 uS	.
7	0ff80b	..	1.480 uS	.
8	0ff80c	..	1.520 uS	.
9	0ff80d	..	1.480 uS	.
10	0ff80e	..	1.520 uS	.
11	0ff80f	..	1.480 uS	.
12	0ff810	..	1.520 uS	.
13	0ff811	..	1.480 uS	.
14	0ff812	..	1.520 uS	.
15	0ff813	..	1.480 uS	.
16	0ff814	..	1.520 uS	.
17	0ff815	..	1.480 uS	.
18	0ff804	TT	24.00 uS	.
19	0ff805	HH	2.240 uS	.
20	0ff806	II	2.280 uS	.
21	0ff807	SS	2.240 uS	.
22	0ff808	..	2.240 uS	.
23	0ff809	II	2.240 uS	.
24	0ff80a	SS	2.280 uS	.
25	0ff80b	..	2.240 uS	.
26	0ff80c	MM	2.240 uS	.
27	0ff80d	EE	2.240 uS	.
28	0ff80e	SS	2.280 uS	.
29	0ff80f	SS	2.240 uS	.
30	0ff810	AA	2.240 uS	.
31	0ff811	GG	2.240 uS	.
32	0ff812	EE	2.280 uS	.
33	0ff813	..	2.240 uS	.
34	0ff814	AA	2.240 uS	.

As you can see, all of the requested states have been captured by the analyzer.

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 0ff800=41
!ASYNC_STAT  615! Software break point: 000104e
```

```
M> reg
reg pc=00104e ccr=80 er0=00000000 er1=000ff800 er2=00000041 er3=00000011
reg er4=00002000 er5=00000000 er6=00000000 er7=000ff904 sp=000ff904 mdcr=c7
```

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 0ff800=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 1000..106f=0aa
```

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

Making Program Coverage Measurements

In testing your program, you will often want to verify that all possible code segments are executed. With the sample program, we might want to verify that all of the code is executed if a command "A", command "B", and an unrecognized command are input to the program.

To make this measurement, we must first reset the coverage status.

```
M> cov -r
```

Note

You should **always** reset the coverage status before making a coverage measurement. Any emulator system command which accesses emulation memory will affect the coverage status bit, resulting in measurement errors if the coverage status is not reset.

Now, run the program and input the three commands:

```
M> r 100
M> m 0ff800=41
M> m 0ff800=42
M> m 0ff800=43
```

Make the coverage measurement:

```
U> cov 1000..106f
percentage of memory accessed:  % 100.0
```

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 triger 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 0ff800=41
```

To display the trace list, type:

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

```
 Line   addr,H  H8/3048 mnemonic,H                     count,R  seq
-----   ------  -------------------------------------  ---------  ---
    0   00101e    aa41  fetch mem                             ---   +
    1   001014  MOV.B @0ff800,R2L                     0.240 uS   .
    2   001016    000f  fetch mem                     0.240 uS   .
    3   001018    f800  fetch mem                     0.240 uS   .
    4   00101a  CMP.B #00,R2L                         0.280 uS   .
    5   0ff800    00xx  read  mem byte                0.240 uS   .
    6   00101c  BEQ 001014                            0.240 uS   .
    7   00101e    aa41  fetch mem                     0.240 uS   .
    8   001014  MOV.B @0ff800,R2L                     0.280 uS   .
    9   001016    000f  fetch mem                     0.240 uS   .
   10   001018    f800  fetch mem                     0.240 uS   .
   11   00101a  CMP.B #00,R2L                         0.240 uS   .
   12   0ff800    00xx  read  mem byte                0.280 uS   .
   13   00101c  BEQ 001014                            0.240 uS   .
   14   00101e    aa41  fetch mem                     0.240 uS   .
   15   001014  MOV.B @0ff800,R2L                     0.240 uS   .
   16   001016    000f  fetch mem                     0.280 uS   .
   17   001018    f800  fetch mem                     0.240 uS   .
   18   00101a  CMP.B #00,R2L                         0.240 uS   .
   19   0ff800    00xx  read  mem byte                0.240 uS   .
   20   00101c  BEQ 001014                            0.280 uS   .
```

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 0ff800=41
```

Now display the trace list:

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

As you can see, the analyzer captured the execution of Exe_Cmd routine which starts from line -2 of the trace list.

```
 Line   addr,H  H8/3048 mnemonic,H                         count,R  seq
-----   ------  -----------------------------------  ---------  ---
  -10   00101c  BEQ 001014                             0.240 uS   .
   -9   00101e    aa41  fetch mem                      0.240 uS   .
   -8   001014  MOV.B @0ff800,R2L                      0.240 uS   .
   -7   001016    000f  fetch mem                      0.280 uS   .
   -6   001018    f800  fetch mem                      0.240 uS   .
   -5   00101a  CMP.B #00,R2L                          0.240 uS   .
   -4   0ff800    41xx  read  mem byte                 0.240 uS   .
   -3   00101c  BEQ 001014                             0.280 uS   .
   -2   00101e  CMP.B #41,R2L                          0.240 uS   .
   -1   001014    6a2a  unused fetch mem               0.240 uS   .
    0   001020  BEQ 00102e                             0.240 uS   +
    1   001022    000a  fetch mem                      0.280 uS   .
    2   00102e  MOV.B #11,R3L                          0.480 uS   .
    3   001030  MOV.L #00002000,ER4                    0.240 uS   .
    4   001032    0000  fetch mem                      0.280 uS   .
    5   001034    2000  fetch mem                      0.240 uS   .
    6   001036  BRA 00104e                             0.240 uS   .
    7   001038    0014  fetch mem                      0.240 uS   .
    8   00104e  MOV.L #000ff804,ER5                    0.520 uS   .
    9   001050    000f  fetch mem                      0.240 uS   .
   10   001052    f804  fetch mem                      0.240 uS   .
```

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  H8/3048 mnemonic,H                   count,R  seq
-----   ------  ----------------------------------  ---------  ---
    0   001018    f800  fetch mem                          ---   +
    1   00101a  CMP.B #00,R2L                        0.240 uS   .
    2   0ff800    00xx  read  mem byte               0.280 uS   .
    3   00101c  BEQ 001014                           0.240 uS   .
    4   00101e    aa41  fetch mem                    0.240 uS   .
    5   001014  MOV.B @0ff800,R2L                    0.240 uS   .
    6   001016    000f  fetch mem                    0.280 uS   .
    7   001018    f800  fetch mem                    0.240 uS   .
    8   00101a  CMP.B #00,R2L                        0.240 uS   .
    9   0ff800    00xx  read  mem byte               0.240 uS   .
   10   00101c  BEQ 001014                           0.280 uS   .
   11   00101e    aa41  fetch mem                    0.240 uS   .
   12   001014  MOV.B @0ff800,R2L                    0.240 uS   .
   13   001016    000f  fetch mem                    0.240 uS   .
   14   001018    f800  fetch mem                    0.280 uS   .
   15   00101a  CMP.B #00,R2L                        0.240 uS   .
   16   0ff800    00xx  read  mem byte               0.240 uS   .
   17   00101c  BEQ 001014                           0.240 uS   .
   18   00101e    aa41  fetch mem                    0.280 uS   .
   19   001014  MOV.B @0ff800,R2L                    0.240 uS   .
   20   001016    000f  fetch mem                    0.240 uS   .
```

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  H8/3048 mnemonic,H                   count,R  seq
-----   ------  ----------------------------------  ---------  ---
    0   00101e    aa41  fetch mem                          ---   +
    1   001014  MOV.B @0ff800,R2L                    0.240 uS   .
    2   001016    000f  fetch mem                    0.240 uS   .
    3   001018    f800  fetch mem                    0.280 uS   .
    4   00101a    aa00  fetch mem                    0.240 uS   .
    5   00101c  BEQ 001014                           0.480 uS   .
    6   00101e    aa41  fetch mem                    0.280 uS   .
    7   001014  MOV.B @0ff800,R2L                    0.240 uS   .
    8   001016    000f  fetch mem                    0.240 uS   .
    9   001018    f800  fetch mem                    0.240 uS   .
   10   00101a    aa00  fetch mem                    0.280 uS   .
   11   00101c  BEQ 001014                           0.480 uS   .
   12   00101e    aa41  fetch mem                    0.240 uS   .
   13   001014  MOV.B @0ff800,R2L                    0.280 uS   .
   14   001016    000f  fetch mem                    0.240 uS   .
   15   001018    f800  fetch mem                    0.240 uS   .
   16   00101a    aa00  fetch mem                    0.240 uS   .
   17   00101c  BEQ 001014                           0.520 uS   .
   18   00101e    aa41  fetch mem                    0.240 uS   .
   19   001014  MOV.B @0ff800,R2L                    0.240 uS   .
   20   001016    000f  fetch mem                    0.280 uS   .
```

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 bit data bus area	byte/word	even	ddxx *1
		odd	xxdd *1
16 bit data bus area	byte	even	ddxx *1
		odd	xxdd *1
	word	even	hhll *2

*1 dd means 8 bits data
*2 hhll means 16 bits data

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 to 8 bit data bus area. 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.

Notes

3

In-Circuit Emulation

When you are ready to use the H8/3048 emulator in conjunction with actual target system hardware, there are some special considerations you should keep in mind.

- installing the emulation cable
- 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

Caution

The following precautions should be taken while using the H8/3048 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 H8/3048 emulator before attaching and detaching the adaptor and probe 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/adaptor and Pin 1 of the QFP probe are properly aligned before inserting the QFP probe the QFP socket/adaptor. Failure to do so may result in damage to the emulator circuitry.

Protect Against Static Discharge. The H8/3048 emulator and the PGA adaptor and QFP probe 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.

Protect Target System CMOS Components. If your target system includes any CMOS components, turn on the target system first, then turn on the H8/3048 emulator; when powering down, turn off the emulator first, then turn off power to the target system.

The H8/3048 emulator is provided without any PGA adaptor, QFP adaptor and QFP probe. To emulate each processor with your target system, you need to purchase appropriate adaptor and/or probe.

PGA adaptor

To emulate each processor with your target system, you can use HP 64784E 5 voltage PGA adaptor or HP 64797B low voltage PGA adaptor as shown in Figure 3-1 and 3-3. These PGA adaptors allow you to connect the emulation cable to QFP socket/adaptor on your target system using the HP 64784G QFP probe. HP 64784G QFP probe is flexible and comfortable to connect the PGA adaptor to a densely populated circuit board.

If you want to connect the PGA adaptor to your target system directly, you need to prepare PGA socket on your target system. To prepare the PGA socket, refer to Table 3-2 and Figure 3-5 to know H8/3048 pin assignment.

QFP adaptor

To emulate with your target system running with supply voltage 5V, you can also use HP 64784D QFP adaptor. The QFP adaptor allows you to connect the emulation cable to the QFP socket/adapter on your target system as shown in Figure 3-2.

QFP socket/adaptor

The QFP socket/adaptor designed for H8/3048 microprocessor is provided with the QFP adaptor and QFP probe. You must attach the QFP socket/adaptor to your target system except you connect the PGA adaptor to your target system directly.

When you use this QFP socket/adaptor, you can replace the H8/3048 emulator with actual H8/3048 microprocessor. Refer to Figure 3-4.

Note

You can order additional QFP socket/adaptor with part No. HP 64784-61612.

Installing into a 5 voltage target

You can select either of the followings to connect the H8/3048 emulator to your 5 voltage target.

- HP 64784D
- HP 64784E + HP 64784G

Note

The H8/3048 emulator can only operate rightly with supply voltage from 4.75V to 5.25V, when you use HP 64784E PGA adaptor or HP 64784D QFP adaptor.

Note

If you have a HP 64797B low voltage PGA adaptor, you can use this low voltage PGA adaptor instead of HP 64784E 5 voltage PGA adaptor. HP 64797B low voltage PGA adaptor can operate rightly with supply voltage from 2.70V to 5.25V.

Note

You must use a clock conforming to the specification of Table 4-1, when you configure the emulator to use external clock.

Installing 64784E PGA adaptor

1. Attach the QFP socket/adaptor to your target system.

2. Connect the PGA adaptor to the emulation cable.

3. Install the PGA adaptor to the QFP socket/adaptor on your target system through QFP probe as shown in Figure 3-1.

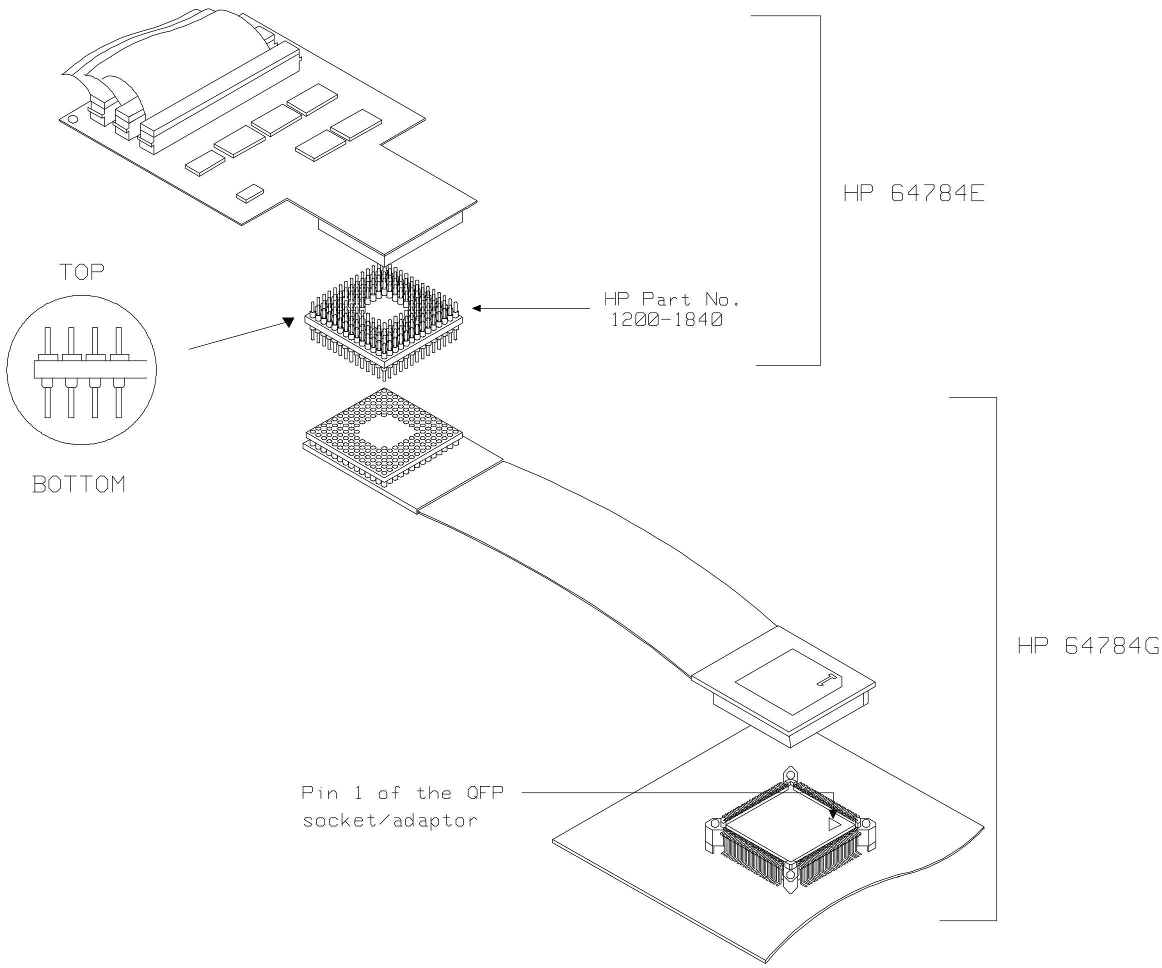


Figure 3-1 Installing HP 64784E/HP 64784G

Installing QFP adaptor

4. Attach the QFP socket/adaptor to your target system.

5. Connect the QFP adaptor to the emulation cable.

6. Install the QFP adaptor to the QFP socket/adaptor on your target system as shown in Figure 3-2.

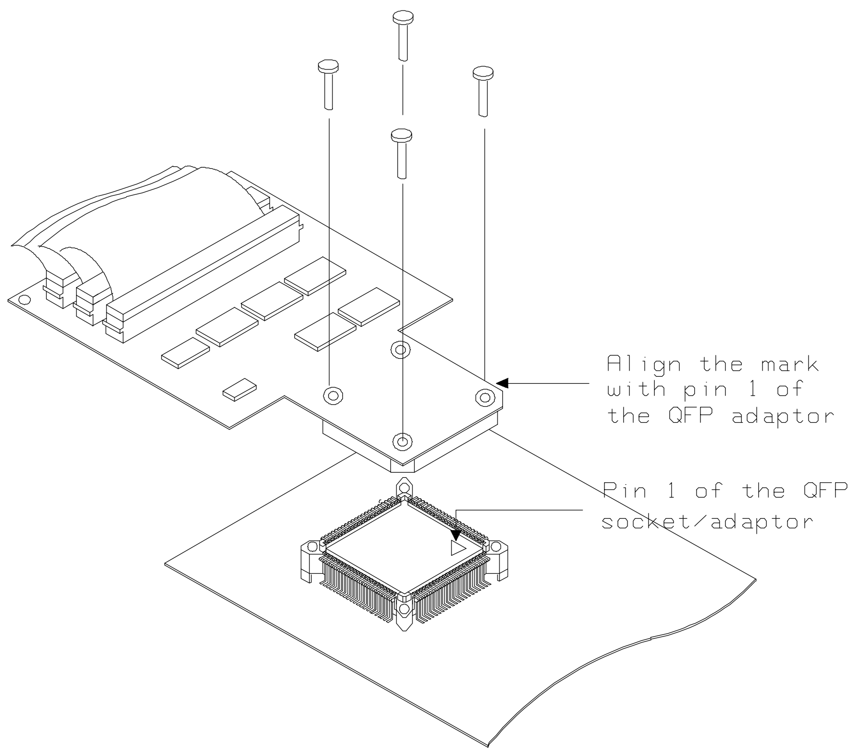


Figure 3-2 Installing HP 64784D

Installing into a low voltage target

To connect the emulator into a low voltage target, you should use HP 64797B PGA adaptor and 64784G QFP probe.

Specification

The emulator can only operate rightly with supply voltage from 2.7V up to 5.25V. You must conform input high voltage(Vih) to the specification of Table 3-1, because these DC characteristics are different from the actual processor's specification.

Table 3-1. DC Characteristics of input high voltage

Item	Minimum (V)
P1 - P5, D0 - D15	Vcc x 0.7 or 2.4 *1
Others	Vcc x 0.7 or 2.0 *1

*1 Higher of the two.

Note

You must use a clock conforming to the specification of Table 4-1, when you configure the emulator to use external clock.

Installing 64797B PGA adaptor

1. Attach the QFP socket/adaptor to your target system.

2. Connect the PGA adaptor to the emulation probe.

3. Install the PGA adaptor to the QFP socket/adaptor on your target system through QFP probe as shown in Figure 3-3.

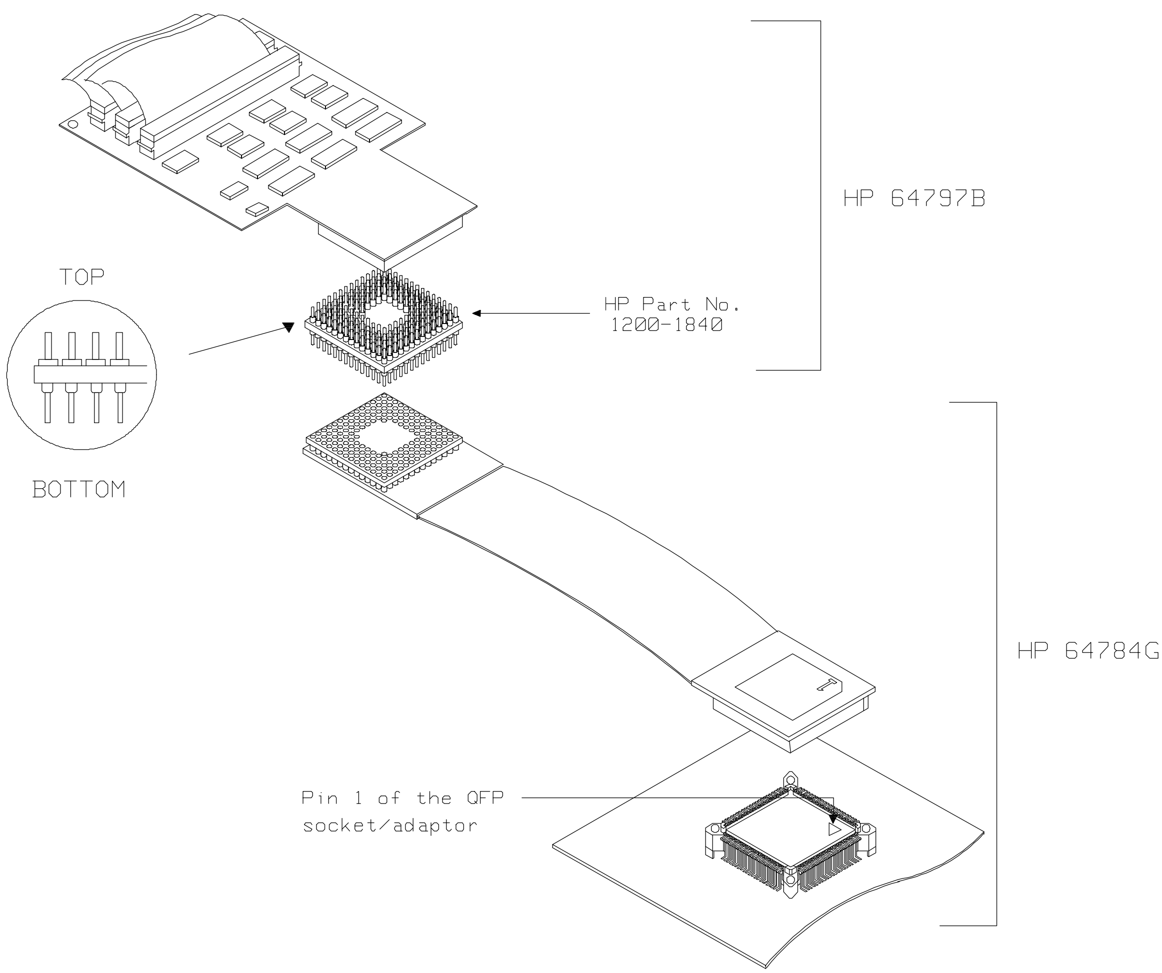


Figure 3-3 Installing HP 64797B/HP 64784G

Installing the H8/3048 microprocessor

You can replace the QFP adaptor/probe with the H8/3048 microprocessor.

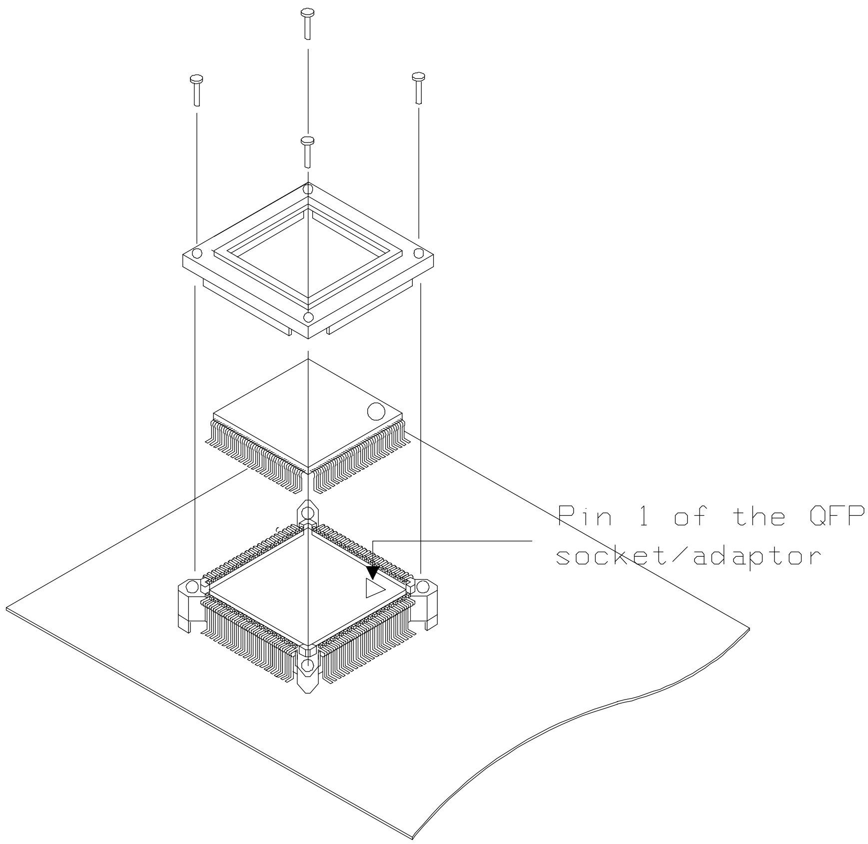


Figure 3-4 Installing the H8/3048 processor

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

Table 3-2 PGA Pin Assignment

PGA 135 pin #	**QFP 100 pin #**	**Function name**	**PGA 135 pin #**	**QFP 100 pin**	**Function name**
1	98	PA/TP5/TIOCB1/A22/$\overline{\text{CS5}}$	15	21	P43/D3
2	1	Vcc	16	25	P46/D6
3	3	PB1/TP9/TIOCB3	17	27	P30/D8
4		nc	18	30	P33/D11
5	8	PB6/TP14/$\overline{\text{DREQ0}}$/$\overline{\text{CS7}}$	19	31	P34/D12
6		nc	20	33	P36/D14
7		nc	21	34	P37/D15
8		nc	22		nc
9		nc	23	37	P11/A1
10		nc	24	40	P14/A4
11	14	P92/RxD0	25	43	P17/A7
12	17	P95/SCK1/$\overline{\text{IRQ5}}$	26	57	Vss
13	18	P40/D0	27		nc
14		nc	28	49	P24/A12

Table 3-2 PGA Pin Assignment (Cont'd)

PGA 135 pin #	QFP 100 pin #	Function name	PGA 135 pin #	QFP 100 pin #	Function name
29	52	P27/A15	47		nc
30	54	P51/A17	48	88	P81/$\overline{CS3}$/$\overline{IRQ1}$
31		nc	49	89	P82/$\overline{CS2}$/$\overline{IRQ2}$
32	58	P60/$\overline{WAIT}$	50	91	P84/$\overline{CS0}$
33	61	ø	51	95	PA2/TP2/TIOCA0/TCLKC
34	64	NMI	52	97	PA4/TP4/TIOCA1/A23/$\overline{CS6}$
35	65	Vss	53		nc
36	68	Vcc	54	2	PB0/TP8/TIOCA3
37		nc	55	5	PB3/TP11/TIOCB4
38	72	P66/$\overline{LWR}$	56	7	PB5/TP13/TOCXB4
39	75	MD2	57	11	Vss
40	76	AVcc	58		nc
41	80	P72/AN2	59		nc
42	81	P73/AN3	60		nc
43	84	P76/AN6/DA0	61	12	P90/TxD0
44		nc	62	15	P93/RxD1
45	92	Vss	63		nc
46		nc	64	19	P41/D1

Table 3-2 PGA Pin Assignment (Cont'd)

PGA 135 pin #	QFP 100 pin #	Function name	PGA 135 pin #	QFP 100 pin #	Function name
65		nc	83		nc
66	24	P45/D5	84	70	P64/$\overline{\text{RD}}$
67	44	Vss	85	73	MD0
68	28	P31/D9	86		nc
69	32	P35/D13	87	79	P71/AN1
70	35	Vcc	88	83	P75/AN5
71	36	P10/A0	89	86	AVss
72	38	P12/A2	90		nc
73	41	P15/A5	91		nc
74	45	P20/A8	92	87	P80/$\overline{\text{RFSH}}$/$\overline{\text{IRQ0}}$
75	48	P23/A11	93	90	P83/$\overline{\text{CS1}}$/$\overline{\text{IRQ3}}$
76	51	P26/A14	94	93	PA0/TP0/$\overline{\text{TEND0}}$/TCLKA
77		nc	95		nc
78	55	P52/A18	96	99	PA6/TP6/TIOCA2/A21/$\overline{\text{CS4}}$
79		nc	97		nc
80	59	P61/$\overline{\text{BREQ}}$	98	4	PB2/TP10/TIOCA4
81	63	$\overline{\text{RES}}$	99	6	PB4/TP12/TOCXA4
82	66	EXTAL	100	9	PB7/TP15/$\overline{\text{DREQ1}}$/$\overline{\text{ADTRG}}$

Table 3-2 PGA Pin Assignment (Cont'd)

PGA 135 pin #	QFP 100 pin #	Function name	PGA 135 pin #	QFP 100 pin #	Function name
101		nc	119	60	P62/$\overline{\text{BACK}}$
102		nc	120	62	$\overline{\text{STBY}}$
103	10	$\overline{\text{RESO}}$	121	67	XTAL
104	13	P91/TxD1	122	69	P63/$\overline{\text{AS}}$
105	16	P94/SCK0/$\overline{\text{IRQ4}}$	123	71	P65/$\overline{\text{HWR}}$
106	22	Vss	124	74	MD1
107	20	P42/D2	125	78	P70/AN0
108	23	P44/D4	126	82	P74/AN4
109	26	P47/D7	127	85	P77/AN7/DA1
110	29	P32/D10	128		nc
111		nc	129		nc
112	39	P13/A3	130	94	PA1/TP1/$\overline{\text{TEND1}}$/TCLKB
113	42	P16/A6	131	96	PA3/TP3/TIOCB0/TCLKD
114	46	P21/A9	132	100	PA7/TP7/TIOCB2/A20
115	50	P25/A13	133		nc
116	53	P50/A16	134	47	P22/A10
117		nc	135	77	Vref
118	56	P53/A19	-	-	-

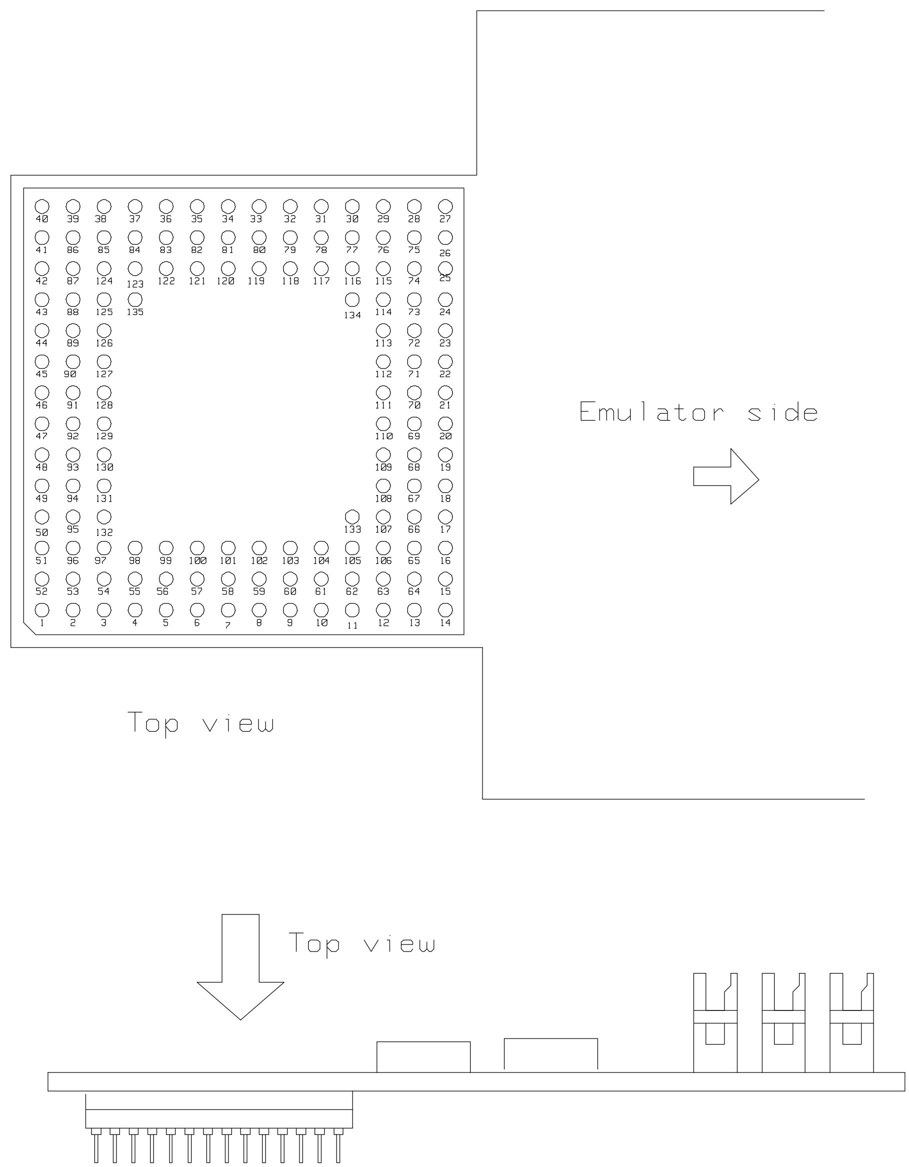


Figure 3-5 PGA Adaptor Pin Assignment

Electrical Characteristics

The AC characteristics of the HP 64797 H8/3048 emulator are listed in the following table.

Table 3-3 Bus timing (Vcc = 5.0V, f = 18MHz)

Characteristics	Symbol	H8/3048 Vcc = 5V f = 18MHz		Probe Type				Unit
				HP 64784E + HP 64784G		HP 64784D		
		min	max	typ *1	worst	typ *1	worst	
Clock cycle time	t_{cyc}	55.5	1000	-	-	-	-	ns
Clock pulse low width	t_{CL}	17	-	19.4	9.9	20.0	10.3	ns
Clock pulse high width	t_{CH}	17	-	26.8	9.9	26.8	10.3	ns
Clock rise time	t_{CR}	-	10	7.6	17.1	7.0	16.7	ns
Clock fall time	t_{CF}	-	10	7.6	17.1	7.6	16.7	ns
Address delay time	t_{AD}	-	25	19.6	53.2	19.2	51.7	ns
Address hold time	t_{AH}	10	-	36.0	-17.3	36.2	-17.8	ns
Address strobe delay time	t_{ASD}	-	25	6.4	30.8	5.6	29.7	ns
Write strobe delay time	t_{WSD}	-	25	10.0	30.8	9.6	29.7	ns
Strobe delay time	t_{SD}	-	25	4.6	30.8	4.8	29.7	ns
Write data strobe pulse width 1	t_{WSW1}	32	-	47.2	25.6	48.0	26.1	ns
Write data strobe pulse width 2	t_{WSW2}	62	-	75.2	55.6	76.0	56.1	ns
Address setup time 1	t_{AS1}	10	-	14.0	-18.8	14.8	-17.9	ns
Address setup time 2	t_{AS2}	38	-	42.8	9.2	43.6	10.1	ns

Table 3-3 Bus timing (Vcc = 5.0V, f = 18MHz) (Cont'd)

Characteristics	Symbol	H8/3048 Vcc = 5V f = 18MHz		Probe Type				Unit
				HP 64784E + HP 64784G		HP 64784D		
		min	max	typ *1	worst	typ *1	worst	
Read data setup time	t_{RDS}	15	-	23.2	50.7	17.4	48.3	ns
Read data hold time	t_{RDH}	0	-	-22.0	14.2	-16.2	17.8	ns
Write data delay time	t_{WDD}	-	55	39.6	58.7	40.4	59.9	ns
Write data setup time 1	t_{WDS1}	10	-	14.6	-1.8	13.6	-3.6	ns
Write data setup time 2	t_{WDS2}	-10	-	1.2	-29.5	0.4	-31.3	ns
Write data hold time	t_{WDH}	20	-	20.0	5.9	20.4	7.3	ns
Read data access time 1	t_{ACC1}	-	50	45.0	-9.9	53.2	-6.4	ns
Read data access time 2	t_{ACC2}	-	105	100.5	45.6	108.7	49.1	ns
Read data access time 3	t_{ACC3}	-	20	27.2	-15.3	34.2	-12.2	ns
Read data access time 4	t_{ACC4}	-	80	82.7	40.2	89.7	43.3	ns
Pre-charge time	t_{PCH}	40	-	56.0	33.6	56.0	34.1	ns
WAIT setup time	t_{WTS}	25	-	25.6	60.9	22.8	54.7	ns
WAIT set hold time	t_{WTH}	5	-	-24.8	-11.0	-22.0	-6.1	ns
BREQ setup time	t_{BRQS}	40	-	-	75.9	-	69.7	ns
BACK delay time 1	t_{BACD1}	-	30	8.8	35.8	8.4	34.7	ns

Table 3-3 Bus timing (Vcc = 5.0V, f = 18MHz) (Cont'd)

Characteristics	Symbol	H8/3048 Vcc = 5V f = 18MHz		Probe Type				Unit
				HP 64784E + HP 64784G		HP 64784D		
		min	max	typ *1	worst	typ *1	worst	
BACK delay time 2	t_{BACD2}	-	30	-1.2	35.8	-1.0	34.7	ns
Bus floating time	t_{BZD}	-	40	18.4	46.2	19.8	44.7	ns

*1 Typical outputs measured with 50pF load

Table 3-4 Refresh controller timing (Vcc = 5.0V, f = 18MHz)

Characteristics	Symbol	H8/3048 Vcc = 5V f = 18MHz		Probe Type				Unit
				HP 64784E + HP 64784G		HP 64784D		
		min	max	typ *1	worst	typ *1	worst	
$\overline{RAS}$ delay time 1	t_{RAD1}	-	30	10.8	41.6	5.0	39.0	ns
$\overline{RAS}$ delay time 2	t_{RAD2}	-	30	15.6	41.6	9.6	39.0	ns
$\overline{RAS}$ delay time 3	t_{RAD3}	-	30	6.0	41.6	4.6	39.0	ns
Row address hold time	t_{RAH}	15	-	26.0	-10.4	31.0	-9.4	ns
$\overline{RAS}$ pre-charge time	t_{RP}	40	-	56.0	30.8	56.6	31.7	ns
$\overline{CAS}$ to $\overline{RAS}$ pre-charge time	t_{CRP}	40	-	54.4	33.6	53.2	34.1	ns
$\overline{CAS}$ pulse width	t_{CAS}	35	-	46.8	28.6	47.2	29.1	ns
$\overline{RAS}$ access time	t_{RAC}	-	70	62.0	29.4	69.8	34.0	ns
Address access time	t_{AA}	-	45	34.0	-9.9	42.2	-6.4	ns
$\overline{CAS}$ access time	t_{CAC}	-	25	12.4	-15.3	19.4	-12.2	ns
Write data setup time 3	t_{WDS3}	10	-	16.4	-1.8	16.0	-3.6	ns
$\overline{CAS}$ setup time	t_{CSR}	10	-	23.2	6.6	23.2	6.0	ns
Read strobe delay time	t_{RSD}	-	30	10.0	35.8	10.6	34.7	ns

*1 Typical outputs measured with 50pF load

Table 3-5 Control signal timing (Vcc = 5.0V, f = 18MHz)

Characteristics	Symbol	H8/3048 Vcc = 5V f = 18MHz		Probe Type				Unit
				HP 64784E + HP 64784G		HP 64784D		
		min	max	typ *1	worst	typ *1	worst	
$\overline{RES}$ setup time	t_{RESS}	200	-	-	281.9	-	275.7	ns
$\overline{RES}$ pulse width	t_{RESW}	10	-	-	-	-	-	tcyc
$\overline{RESO}$ output delay time	t_{RESD}	-	100	-	109.6	-	108.4	ns
$\overline{RESO}$ output pulse width	t_{RESOW}	132	-	-	-	-	-	tcyc
NMI setup time	t_{NMIS}	150	-	-	231.9	-	225.7	ns
NMI hold time	t_{NMIH}	10	-	-	-9.0	-	-4.1	ns
Interrupt pulse width	t_{NMIW}	200	-	-	209.2	-	208.3	ns
Crystal oscillator setting time(reset)	t_{OSC1}	20	-	-	-	-	-	ms
Crystal oscillator setting time (software standby)	t_{OSC2}	7	-	-	-	-	-	ms

*1 Typical outputs measured with 50pF load

Table 3-6 Timing condition of On-chip supporting modules (Vcc = 5.0V, f = 18MHz)

Characteristics		Symbol	H8/3048 Vcc = 5V f = 18MHz		Probe Type				Unit
					HP 64784E + HP 64784G		HP 64784D		
			min	max	typ *1	worst	typ *1	worst	
DMA	$\overline{\text{DREQ}}$ setup time	t_{DRQS}	30	-	-	65.9	-	59.7	ns
	$\overline{\text{DREQ}}$ hold time	t_{DRQH}	10	-	-	-6.0	-	-1.1	ns
	$\overline{\text{TEND}}$ delay time 1	t_{TED1}	-	50	-	61.6	-	59.0	ns
	$\overline{\text{TEND}}$ delay time 2	t_{TED2}	-	50	-	61.6	-	59.0	ns
ITU	Timer input delay time	t_{TOCD}	-	100	-	111.6	-	109.0	ns
	Timer input setup time	t_{TICS}	50	-	-	85.9	-	79.7	ns
	Timer clock input setup time	t_{TCKS}	50	-	-	85.9	-	79.7	ns
	Timer clock pulse width (single edge)	t_{TCKWH}	1.5	-	-	-	-	-	tcyc
	Timer clock pulse width (both edge)	t_{TCKWL}	2.5	-	-	-	-	-	tcyc
SCI	Input clock cycle (Async)	t_{SCYC}	4	-	-	-	-	-	tcyc
	Input clock cycle (Sync)	t_{SCYC}	6	-	-	-	-	-	tcyc
	Input clock rise time	t_{SCKr}	-	1.5	-	-	-	-	tscyc
	Input clock fall time	t_{SCKf}	-	1.5	-	-	-	-	tscyc

Table 3-6 Timing condition of On-chip supporting modules (Cont'd) (Vcc = 5.0V, f = 18MHz)

Characteristics		Symbol	H8/3048 Vcc = 5V f = 18MHz		Probe Type				Unit
					HP 64784E + HP 64784G		HP 64784D		
			min	max	typ *1	worst	typ *1	worst	
SCI	Input clock pulse width	t_{SCKw}	0.4	0.6	-	-	-	-	tscyc
	Transmit data delay time	t_{TXD}	-	100	-	105.8	-	104.7	ns
	Received data setup time	t_{RXS}	100	-	-	136.8	-	128.4	ns
	Received data hold time (Clock input)	t_{RXH}	100	-	-	109.2	-	108.3	ns
	Recieved data hold time (Clock output)	t_{RXH}	0	-	-	-18.4	-	-11.8	ns
PORT TPC	Output data delay time	t_{PWD}	-	100	-	111.6	-	109.0	ns
	Input data setup time	t_{PRS}	50	-	-	90.5	-	86.2	ns
	Input data hold time	t_{PRH}	50	-	-	37.0	-	40.8	ns

*1 Typical outputs measured with 50pF load

Table 3-7 Bus timing (Vcc = 3.0V, f = 13MHz)

Characteristics	Symbol	H8/3048 Vcc = 3V f = 13MHz		Probe Type HP 64797B + HP 64784G		Unit
		min	max	typ *1	worst	
Clock cycle time	t_{cyc}	76.9	1000	-	-	ns
Clock pulse low width	t_{CL}	20	-	30.9	20.4	ns
Clock pulse high width	t_{CH}	20	-	36.5	20.4	ns
Clock rise time	t_{CR}	-	15	6.8	17.3	ns
Clock fall time	t_{CF}	-	15	8.6	17.3	ns
Address delay time	t_{AD}	-	50	18.6	53.6	ns
Address hold time	t_{AH}	20	-	37.2	-18.7	ns
Address strobe delay time	t_{ASD}	-	50	6.8	31.6	ns
Write strobe delay time	t_{WSD}	-	50	10.4	31.6	ns
Strobe delay time	t_{SD}	-	50	5.2	31.6	ns
Write data strobe pulse width 1	t_{WSW1}	40	-	66.8	45.6	ns
Write data strobe pulse width 2	t_{WSW2}	90	-	105.5	86.3	ns
Address setup time 1	t_{AS1}	15	-	25.5	-9.1	ns
Address setup time 2	t_{AS2}	45	-	65.0	29.6	ns
Read data setup time	t_{RDS}	30	-	34.0	44.8	ns
Read data hold time	t_{RDH}	0	-	-32.8	23.9	ns

Table 3-7 Bus timing (Vcc = 3.0V, f = 13MHz) (Cont'd)

Characteristics	Symbol	H8/3048 Vcc = 3V f = 13MHz		Probe Type HP 64797B + HP 64784G		Unit
		min	max	typ *1	worst	
Write data delay time	t_{WDD}	-	75	38.4	59.4	ns
Write data setup time 1	t_{WDS1}	20	-	35.6	18.3	ns
Write data setup time 2	t_{WDS2}	-10	-	13.1	-20.1	ns
Write data hold time	t_{WDH}	15	-	30.9	15.5	ns
Read data access time 1	t_{ACC1}	-	60	75.4	27.9	ns
Read data access time 2	t_{ACC2}	-	140	152.3	104.8	ns
Read data access time 3	t_{ACC3}	-	30	46.7	11.4	ns
Read data access time 4	t_{ACC4}	-	100	123.6	88.3	ns
Pre-charge time	t_{PCH}	55	-	77.4	53.6	ns
WAIT setup time	t_{WTS}	40	-	31.2	63.1	ns
WAIT set hold time	t_{WTH}	10	-	-30.4	-13.0	ns
BREQ setup time	t_{BRQS}	40	-	-	78.1	ns
BACK delay time 1	t_{BACD1}	-	50	9.2	36.6	ns
BACK delay time 2	t_{BACD2}	-	50	-1.0	36.6	ns
Bus floating time	t_{BZD}	-	70	17.2	46.6	ns

*1 Typical outputs measured with 50pF load

Table 3-8 Refresh controller timing (Vcc = 3.0V, f = 13MHz)

Characteristics	Symbol	H8/3048 Vcc = 5V f = 18MHz		Probe Type HP 64797B+ HP 64784G		Unit
		min	max	typ *1	worst	
$\overline{\text{RAS}}$ delay time 1	t_{RAD1}	-	50	8.8	41.6	ns
$\overline{\text{RAS}}$ delay time 2	t_{RAD2}	-	50	13.2	41.6	ns
$\overline{\text{RAS}}$ delay time 3	t_{RAD3}	-	50	7.0	41.6	ns
Row address hold time	t_{RAH}	20	-	38.9	-0.3	ns
$\overline{\text{RAS}}$ pre-charge time	t_{RP}	55	-	75.2	52.2	ns
$\overline{\text{CAS}}$ to $\overline{\text{RAS}}$ pre-charge time	t_{CRP}	55	-	73.8	53.6	ns
$\overline{\text{CAS}}$ pulse width	t_{CAS}	55	-	67.8	48.6	ns
$\overline{\text{RAS}}$ access time	t_{RAC}	-	80	103.5	78.3	ns
Address access time	t_{AA}	-	45	64.4	27.9	ns
$\overline{\text{CAS}}$ access time	t_{CAC}	-	30	31.9	11.4	ns
Write data setup time 3	t_{WDS3}	20	-	39.8	18.3	ns
$\overline{\text{CAS}}$ setup time	t_{CSR}	10	-	43.6	16.5	ns
Read strobe delay time	t_{RSD}	-	50	10.2	36.6	ns

*1 Typical outputs measured with 50pF load

Table 3-9 Control signal timing (Vcc = 3.0V, f = 13MHz)

Characteristics	Symbol	H8/3048 Vcc = 5V f = 18MHz		Probe Type HP 64797B + HP 64784G		Unit
		min	max	typ *1	worst	
$\overline{RES}$ setup time	t_{RESS}	200	-	-	284.1	ns
$\overline{RES}$ pulse width	t_{RESW}	10	-	-	-	tcyc
$\overline{RESO}$ output delay time	t_{RESD}	-	100	-	110.3	ns
$\overline{RESO}$ output pulse width	t_{RESOW}	132	-	-	-	tcyc
NMI setup time	t_{NMIS}	200	-	-	234.1	ns
NMI hold time	t_{NMIH}	10	-	-	-11.0	ns
Interrupt pulse width	t_{NMIW}	200	-	-	209.2	ns
Crystal oscillator setting time(reset)	t_{OSC1}	20	-	-	-	ms
Crystal oscillator setting time (software standby)	t_{OXC2}	7	-	-	-	ms

*1 Typical outputs measured with 50pF load

Table 3-10 Timing condition of On-chip supporting modules (Vcc = 3.0V, f = 13MHz)

Characteristics		Symbol	H8/3048 Vcc = 5V f = 18MHz		Probe Type HP 64797B + HP 64784G		Unit
			min	max	typ *1	worst	
DMA	$\overline{DREQ}$ setup time	t_{DRQS}	40	-	-	68.1	ns
	$\overline{DREQ}$ hold time	t_{DRQH}	10	-	-	-8.0	ns
	$\overline{TEND}$ delay time 1	t_{TED1}	-	100	-	61.6	ns
	$\overline{TEND}$ delay time 2	t_{TED2}	-	100	-	61.6	ns
ITU	Timer input delay time	t_{TOCD}	-	100	-	111.6	ns
	Timer input setup time	t_{TICS}	50	-	-	88.1	ns
	Timer clock input setup time	t_{TCKS}	50	-	-	88.1	ns
	Timer clock pulse width (single edge)	t_{TCKWH}	1.5	-	-	-	tcyc
	Timer clock pulse width (both edge)	t_{TCKWL}	2.5	-	-	-	tcyc
SCI	Input clock cycle (Async)	t_{SCYC}	4	-	-	-	tcyc
	Input clock cycle (Sync)	t_{SCYC}	6	-	-	-	tcyc
	Input clock rise time	t_{SCKr}	-	1.5	-	-	tscyc
	Input clock fall time	t_{SCKf}	-	1.5	-	-	tscyc
	Input clock pulse width	t_{SCKw}	0.4	0.6	-	-	tscyc

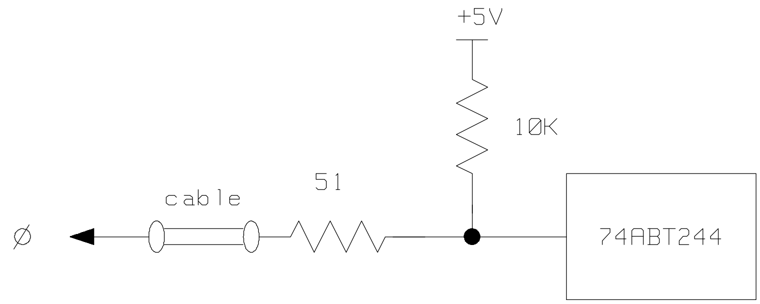
Table 3-10 Timing condition of On-chip supporting modules (Cont'd) (Vcc = 3.0V, f = 13MHz)

Characteristics		Symbol	H8/3048 Vcc = 5V f = 18MHz		Probe Type HP 64797B + HP 64784G		Unit
			min	max	typ *1	worst	
SCI	Transmit data delay time	t_{TXD}	-	100	-	106.6	ns
	Received data setup time	t_{RXS}	100	-	-	138.8	ns
	Received data hold time (Clock input)	t_{RXH}	100	-	-	109.2	ns
	Recieved data hold time (Clock output)	t_{RXH}	0	-	-	-20.4	ns
PORT TPC	Output data delay time	t_{PWD}	-	100	-	111.6	ns
	Input data setup time	t_{PRS}	50	-	-	103.1	ns
	Input data hold time	t_{PRH}	50	-	-	35.6	ns

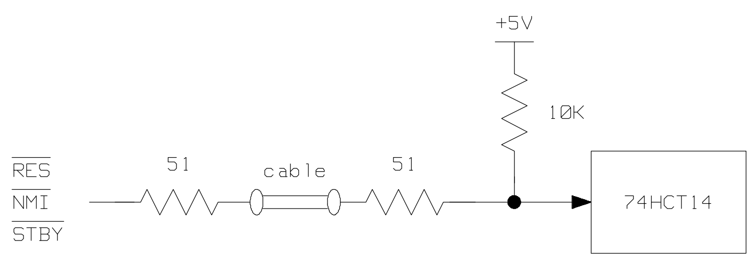
*1 Typical outputs measured with 50pF load

Target System Interface

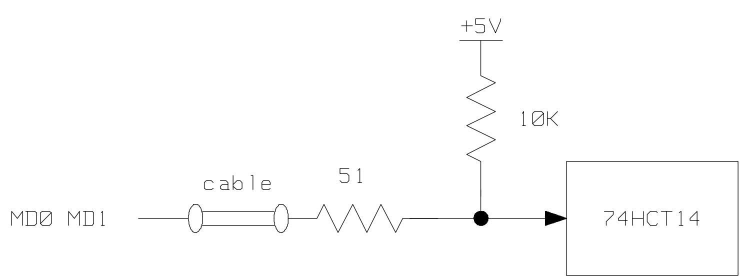
ø



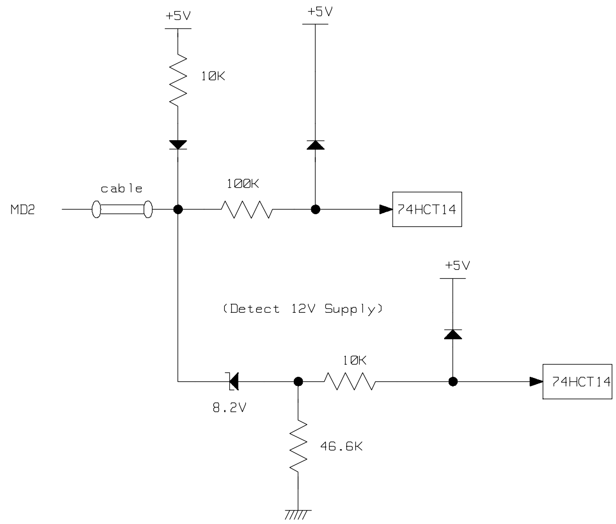
/RES, /NMI, STBY



MD0, MD1

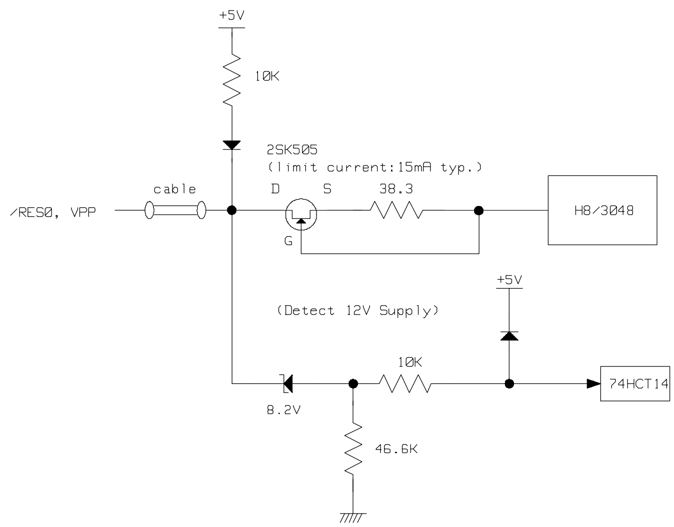


MD2



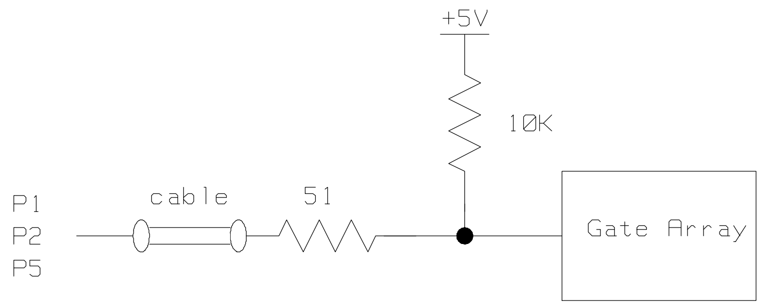
+5V
+5V
10K
cable
100K
MD2
74HCT14
+5V
(Detect 12V Supply)
10K
74HCT14
8.2V
46.6K

/RESO, VPP

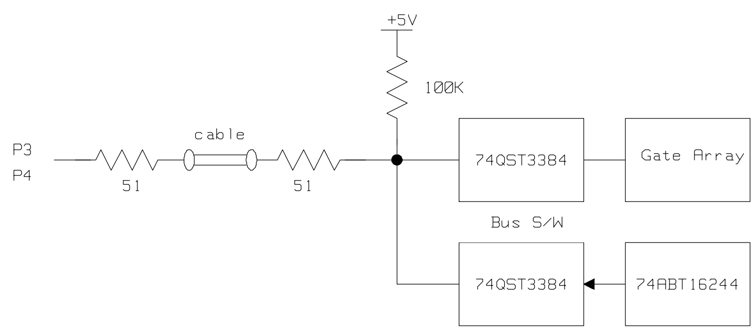


+5V
10K
2SK505
(limit current:15mA typ.)
cable
D
S
38.3
/RES0, VPP
H8/3048
G
+5V
(Detect 12V Supply)
10K
74HCT14
8.2V
46.6K

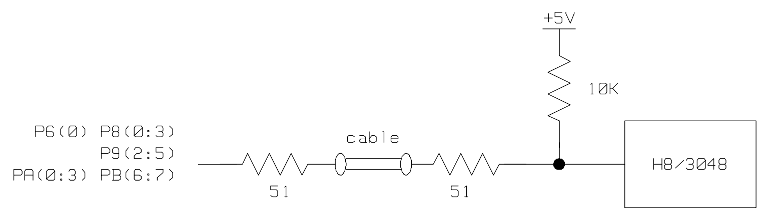
P1, P2, P5 (A0-19)



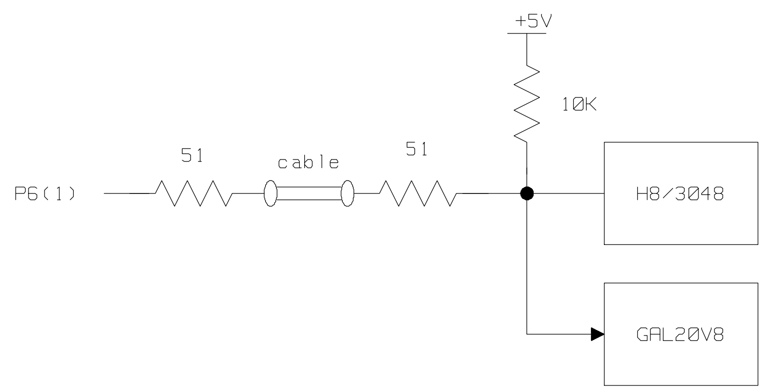
P3, P4 (D0-15)



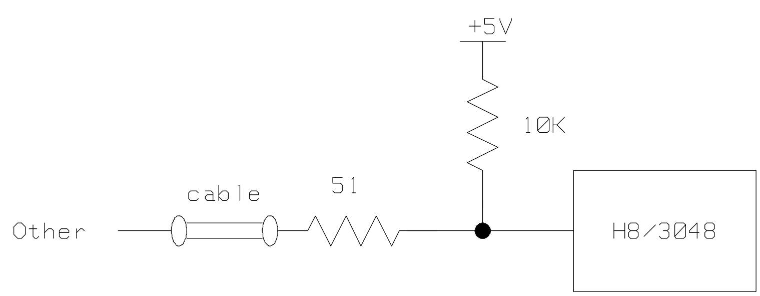
P6, P8, P9, PA, PB



+5V
10K
P6(0) P8(0:3)
P9(2:5)
PA(0:3) PB(6:7)
cable
51
51
H8/3048

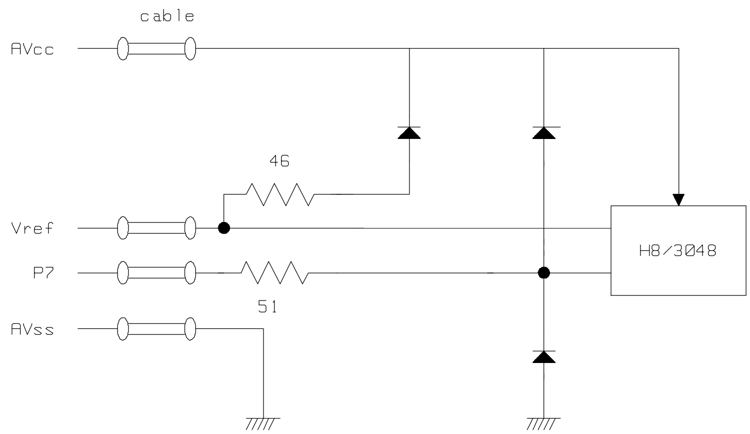


+5V
10K
P6(1)
51
cable
51
H8/3048
GAL20V8



+5V
10K
Other
cable
51
H8/3048

P7, AVcc, Vref, AVss



Notes

4

Configuring the H8/3048 Emulator

In this chapter, we will discuss:

- how to configure the HP 64700 emulator for H8/3048 microprocessor to fit your particular measurement needs.

- some restrictions of HP 64700 emulator for H8/3048 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 ba=en
cf chip=3048
cf clk=int
cf dbc=en
cf mode=7
cf nmi=en
cf rrt=dis
cf rsp=9
cf tdma=en
cf trfsh=en
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 ba

The **ba** (bus arbitration) configuration item defines how your emulator responds to bus request signals from the target system.

```
M> cf ba=en
```

When bus arbitration is enabled, the /BREQ (bus request) signal from the target system is responded to exactly as it would be if only the emulation processor was present without an emulator. In other words, if the emulation processor receives a /BREQ from the target system, it will respond by asserting /BACK and will set the various processor lines to tri-state. /BREQ is then released by the target; /BACK is negated by the processor, and the emulation processor restarts execution.

```
M> cf ba=dis
```

When you disable bus arbitration by entering the above command, the emulator ignores the /BREQ signal from the target system. The emulation processor will never drive the /BACK line true; nor will it place the address, data and control signals into the tri-state mode.

Enabling and disabling bus master arbitration can be useful to you in isolating target system problems. For example, you may have a situation where the processor never seems to execute any code. You can disable bus arbitration using **cf ba=dis** to check and see if faulty arbitration circuitry in your target system is contributing to the problem.

Note When bus arbitration is enabled, the emulator responds to /BREQ signal during both foreground and background operation.

Note The commands which cause the emulator to break to monitor are ignored during the processor releases bus cycles.

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

---

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**
3048	Emulate H8/3048 microprocessor.
3048f	Emulate H8/3048F microprocessor.
3047	Emulate H8/3047 microprocessor.
3044	Emulate H8/3044 microprocessor.

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

Emulation Memory	Clock Speed		
	With HP64784D	**With HP64784E**	**With HP64797B**
64726A 64727A 64728A	From 1 up to 16MHz (System Clock)	From 1 up to 16MHz (System Clock)	From 1 up to 13MHz (System Clock)
64729A	From 1 up to 18MHz (System Clock)	From 1 up to 18MHz (System Clock)	From 1 up to 13MHz (System Clock)

Note Crystal oscillator frequency of internal clock is 8MHz.

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

cf dbc

The **dbc** (drive background cycles) option allows you to select whether or not the emulator will drive the target system bus on background cycles.

```
M> cf dbc=en
```

You can enable background cycle drive to target system by entering the above command. Emulation processor's address and control strobes (except /LWR and /HWR) are driven during background cycles.

Background write cycles won't appear to the target system. (/LWR and /HWR signals are always "high" when the **dbc** option is enabled.)

```
M> cf dbc=dis
```

If you specify the above command, background monitor cycles are not driven to the target system except address.

You use the **dbc** option to avoid target system interaction problems. For example, your target system interaction scheme may depend on the constant repetition of bus cycles. In such case, using the **dbc** option will help avoid the problem.

Note

Refresh cycles, internal DMA cycles and target memory access are always driven to the target system regardless of this configuration.

Note

When dbc is disabled, the emulator can't respond to /WAIT signal.

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=ext
```

The emulator will work using the mode setting by the target system. The target system must supply appropriate inputs to MD0, MD1 and MD2.

```
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. (expanded 1M bytes mode without internal ROM: 8 bit data bus)
2	The emulator will operate in mode 2. (expanded 1M bytes mode without internal ROM:16 bit data bus)
3	The emulator will operate in mode 3. (expanded 16M bytes mode without internal ROM: 8 bit data bus)
4	The emulator will operate in mode 4. (expanded 16M bytes mode without internal ROM:16 bit data bus)
5	The emulator will operate in mode 5. (expanded 1M bytes mode with internal ROM: 8 bit data bus)
6	The emulator will operate in mode 6. (expanded 16M bytes mode with internal ROM: 8 bit data bus)
7	The emulator will operate in mode 7. (single chip advanced mode)

Note

It is recommended to specify operation mode number in this configuration, since the emulator does not work fine when MD0,MD1 and MD2 are not steady.

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.

Caution

While the emulator is executing the boot program of H8/3048F, the NMI must not occur. Because the emulator can not prohibit the NMI at this time.

```
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 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 tdma

The tdma (trace internal DMA cycles) configuration item defines whether or not the emulator traces internal DMA cycles.

```
M> cf tdma=en
```

When you enable this item with the above command, each time DMA performed, one emulation analyzer state will be generated to recognize the DMA cycle.

```
M> cf tdma=dis
```

When disabled, no analyzer state will be generated at the occurrence of DMA. Therefore, any DMA cycle will be ignored by the analyzer.

Note

Some internal DMA cycles may be traced regardless of this configuration in order to disassemble the trace list correctly.

cf trfsh

The **trfsh** (trace refresh cycles) configuration item defines whether or not the emulator traces refresh cycles.

```
M> cf trfsh=en
```

When you enable this item with the above command, refresh cycles are traced by the emulation analyzer.

```
M> cf trfsh=dis
```

When disabled, refresh cycles are not traced by the analyzer.

Note

Some refresh cycles may be traced regardless of this configuration in order to disassemble the trace list correctly.

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 H8/3048 emulator dose not support hardware standby mode, and /STBY input will drive the emulator into the reset sate.

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 H8/3048 emulator memory mapper allows you to define up to 16 different map terms; each map term has a minimum size of 512 bytes. If you specify a value less than 512 byte, 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 emulation memory  : 7e800h 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 H8/3048 emulator.

Note

When you use the internal ROM or on-chip flash memory, 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 properly, you cannot access that area.

Note

You don’t have to map internal RAM as emulation RAM, since the H8/3048 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 H8/3048 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).

5

Using the On-chip Flash Memory

Introduction

The H8/3048 emulator is equipped with functions for the on-chip flash memory of H8/3048F microprocessor. So you can direct the on-chip flash memory using the H8/3048 emulator.

The following pages will describe differences between actual on-chip flash memory and the H8/3048 emulator. You need to pay attention for following contents.

Memory Mapping

The H8/3048 emulator uses emulation memory instead of actual on-chip flash memory of the H8/3048F microprocessor. So you need to map this area as emulation ROM in the emulator configuraiton before using the H8/3048 emulator. And also, you need to configure the H8/3048 emulator as H8/3048F and mode 5/6/7 in the emulator configuration to use flash memory functions.

Note

When you use the on-chip flash memory on the H8/3048 emulator, you must map that area as emulation ROM. When you power on the emulator, all memory space except internal RAM is mapped as target RAM. Therefore, if you don't map this area properly, you can not use the flash memory functions.

Flash Memory Registers

You don't need to take care of FLMCR, EBR1, EBR2 and RAMCR registers, since the H8/3048 emulator uses emulation memory instead of actual on-chip flash memory and does not use these registers.

Note

You cannot direct these registers to control the flash memory functions. These register are not effctive for the on-chip flash memory operation on the H8/3048 emulator.

Note

The H8/3048 emulator can display or modify these registers except for the RAMCR register. RAMCR is always FF'H.

Note

You cannot do flash memory emulation by RAM using RAMCR register.

Programming/ Erasing Flash Memory

Programming Data

To write data onto the on-chip flash memory, you need to supply 12V to Vpp/RESO pin. When you supply 12V correctly, write to ROM break does not occur even if write to ROM break is enabled in the emulator configuration.

You need only to write data to the destination address only once. The H8/3048 emulator can program data correctly and therefore it is not necessary to repeat writing unless you prefer.

Note If you don't supply 12V to Vpp/$\overline{\text{RESO}}$ pinc correctly, the H8/3048 emulator does not write data to destination address. And if write to ROM break is enabled in the emulator configuration, write to ROM break will occur when you write date onto on-chip flash memory.

Erasing Data

To erase data onto the on-chip flash memory, you need to supply 12V to Vpp/RESO pin.

You cannot use FLMCR, EBR1 and EBR2 registers. These registers aren't effective in the emulator operation. As a result, you cannot use block-erase to erase your data of the on-chip flash memory.

Also, you cannot prewrite the data using inverted data. If you write inverted data onto on-chip flash memory, the H8/3048 emulator will write inverted data onto on-chip flash memory area (i.e data can never be 00'H, it can be inverted data).

Note It is recommended to write 00'h to destination address directly, when you want to prewrite the data of the on-chip flash memory.

Protection Mode

The H8/3048 emulator does not support the following protection modes.

- Block protection
- Emulation protection
- Error protection

Note

The H8/3048 emulator never detects errors such as read cycle to on-chip flash memory area, exceptions, and execution of "SLEEP" instruction during writing/erasing on-chip flash memory area.

Boot Mode

The H8/3048 emulator drives into the boot mode, when the emulator accepts reset signal from target system and you supply 12V to MD2 and Vpp/RESO pin. Then, command prompt becomes **U>** (Running User Program). Emulation reset does not cause the boot mode.

Note

While the boot program on internal PROM is in progress, break command is suspended and occurs after the boot program is completed. If you want to discontinue the boot program, you need to reset the emulator.

Note

The H8/3048 emulator does not trace execution of the boot program on internal PROM.

Note

While the emulator is executing the boot program on internal PROM, NMI must not occur. Because the emulator cannot prohibit the NMI at this time.

A

H8/3048 Emulator Specific Command Syntax

The following pages contain descriptions of command syntax specific to the H8/3048 emulator. The following syntax items are included (several items are part of other command syntax):

- <ACCESS_MODE>. May be specified in the **mo** (display and access mode), **m** (memory), and **ser** (search memory for data) commands. The display mode is used when memory locations are displayed or modified.

- <ADDRESS>. May be specified in emulation commands which allow addresses to be entered.

- <CONFIG_ITEMS>. May be specified in the **cf** (emulator configuration) and **help cf** commands.

- <DISPLAY_MODE>. May be specified in the **mo** (display and access mode), **m** (memory), and **ser** (search memory for data) commands. The display mode is used when memory locations are displayed or modified.

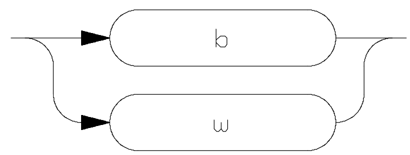
- <REG_NAME>. May be specified in the **reg** (register) command.

Command and error messages which are specific to the H8/3048 emulator are also described in this chapter.

ACCESS_MODE

Summary Specify cycles used by monitor when accessing target system memory.

Syntax



b Byte. Target memory is accessed using byte cycles.

w Word. Target Memory is accessed using word cycles

Note

When the **<ACCESS_MODE>** is **w**, modifying target memory will fail if you try to modify memory from an odd address or with data which byte count is odd. Also, you can't load file which byte count is odd. Therefore, it is recommended to use the emulator with the default **<ACCESS_MODE>** (**b**).

Defaults The **<ACCESS_MODE>** is **b** at power up initialization. Display mode specifications are saved; that is, when a command changes the display mode, the new display mode becomes the current default.

Related Information Refer to the **mo** syntax information in the *User's Reference* manual for further information on use of the mode command.

ADDRESS

Summary Address specification used in emulation commands.

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

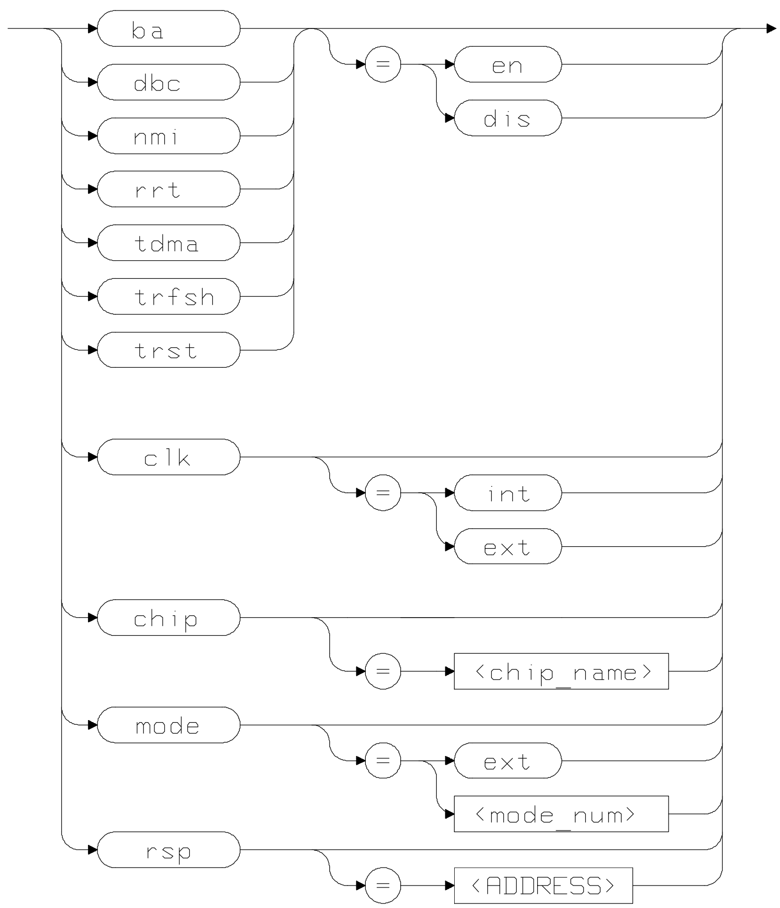
Examples **m 1000**

m 200000..2000ff

CONFIG_ITEMS

Summary H8/3048 emulator configuration items.

Syntax



Description

The H8/3048 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:

ba	Enable/disable bus arbitration with target system.
chip	Select processor to be emulated.
clk	Select internal/external clock source.
dbc	Enable/disable to drive background cycles to target system.
mode	Determine emulator processor operation mode.
nmi	Enable/disable NMI (non maskable interrupt) from target system.
rrt	Restrict emulator to real time runs.
rsp	Specify system stack pointer value to load upon each transition from emulation reset to the monitor.
tdma	Enable/disable tracing internal DMA cycles.
trfsh	Enable/disable tracing refresh cycles.
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

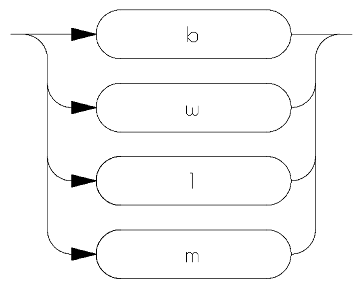
Refer to the **cf** syntax pages in the *User's Reference* manual. Also, refer to chapter 3 of this manual for complete information about each configuration item.

DISPLAY_MODE

Summary

Specify the memory display format or the size of memory locations to be modified.

Syntax



b Byte. Memory is displayed in a byte format, and when memory locations are modified, bytes are changed.

w Word. Memory is displayed in a word format, and when memory locations are modified, words are changed.

l Long word. Memory is displayed in a long word format, and when memory locations are modified, long words are changed.

m Mnemonic. Memory is displayed in mnemonic format; that is, the contents of memory locations are inverse-assembled into mnemonics and operands. When memory locations are modified, the last non-mnemonic display mode specification is used. You cannot specify this display mode in the **ser** (search memory for data) command.

Defaults

The **<DISPLAY_MODE>** is **b** at power up initialization. Display mode specifications are saved; that is, when a command changes the display mode, the new display mode becomes the current default.

Related Information

Refer to the **mo** syntax information in the **User's Reference** manual for further information on use of the mode command.

REGISTER CLASS and NAME

Summary

H8/3048 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
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
mdcr	Mode control register(Read Only)

sys (System control)

mdcr	Mode control register(Read Only)
syscr	System control register
dastcr	D/A standby control register
divcr	Clock divider control register
mstcr	Module standby control register
cscr	Chip select control register

Note

Even if PSTOP bit of the mstcr register is set to 1, the emulator cannot stop the ø clock output.

intc (Interrupt controller)

iscr	IRQ sense control register
ier	IRQ enable register
isr	IRQ status register
ipra	Interrupt priority register A
iprb	Interrupt priority register B

busc (Bus controller)

abwcr	Byte/Word area control register
astcr	2/3 state area control register
wcr	Wait control register
wcer	Wait controller enable register
brcr	Bus release control register

rfshc (Refresh controller)

rfshcr	Refresh control register
rtmcsr	Refresh timer control/status register
rtcnt	Refresh timer counter
rtcor	Refresh time constant register

dmac0 (DMA controller 0)

mar0a	Memory address register 0A
etcr0a	Transfer count register 0A
ioar0a	I/O address register 0A
dtcr0a	Data transfer control register 0A
mar0b	Memory address register 0B
etcr0b	Transfer count register 0B
ioar0b	I/O address register 0B
dtcr0b	Data transfer control register 0B

dmac1 (DMA controller 1)

mar1a	Memory address register 1A
etcr1a	Transfer count register 1A
ioar1a	I/O address register 1A
dtcr1a	Data transfer control register 1A
mar1b	Memory address register 1B
etcr1b	Transfer count register 1B
ioar1b	I/O address register 1B
dtcr1b	Data transfer control register 1B

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)
p4ddr	Port 4 data direction register(Write Only)
p5ddr	Port 5 data direction register(Write Only)
p6ddr	Port 6 data direction register(Write Only)
p8ddr	Port 8 data direction register(Write Only)
p9ddr	Port 9 data direction register(Write Only)
paddr	Port A data direction register(Write Only)
pbddr	Port B data direction register(Write Only)
p1dr	Port 1 data register
p2dr	Port 2 data register
p3dr	Port 3 data register
p4dr	Port 4 data register
p5dr	Port 5 data register
p6dr	Port 6 data register
p7dr	Port 7 data register(Read Only)
p8dr	Port 8 data register
p9dr	Port 9 data register
padr	Port A data register
pbdr	Port B data register
p2pcr	Port 2 input pull up MOS control register
p4pcr	Port 4 input pull up MOS control register
p5pcr	Port 5 input pull up MOS control register

Note

The emulator can not support input pull up MOS control function of the p2pcr, p4pcr and p5pcr.

itug (16 bit integrated timer pulse unit general)

tstr	Timer start register
tsnc	Timer synchro register
tmdr	Timer mode register
tfcr	Timer function control register
toer	Timer output master control register
tocr	Timer output control register

itu0 (16 bit integrated timer pulse unit 0)

tcr0	Timer control register 0
tior0	Timer I/O control register 0
tier0	Timer interrupt enable register 0
tsr0	Timer status register 0
tcnt0	Timer counter 0
gra0	General register A0
grb0	General register B0

itu1 (16 bit integrated timer pulse unit 1)

tcr1	Timer control register 1
tior1	Timer I/O control register 1
tier1	Timer interrupt enable register 1
tsr1	Timer status register 1
tcnt1	Timer counter 1
gra1	General register A1
grb1	General register B1

itu2 (16 bit integrated timer pulse unit 2)

tcr2	Timer control register 2
tior2	Timer I/O control register 2
tier2	Timer interrupt enable register 2
tsr2	Timer status register 2
tcnt2	Timer counter 2
gra2	General register A2
grb2	General register B2

itu3 (16 bit integrated timer pulse unit 3)

tcr3	Timer control register 3
tior3	Timer I/O control register 3
tier3	Timer interrupt enable register 3
tsr3	Timer status register 3
tcnt3	Timer counter 3
gra3	General register A3
grb3	General register B3
bra3	Buffer register A3
brb3	Buffer register B3

itu4 (16 bit integrated timer pulse unit 4)

tcr4	Timer control register 4
tior4	Timer I/O control register 4
tier4	Timer interrupt enable register 4
tsr4	Timer status register 4
tcnt4	Timer counter 4
gra4	General register A4
grb4	General register B4
bra4	Buffer register A4
brb4	Buffer register B4

tpc (Programable timing pattern controller)

tpmr	TPC output mode register
tpcr	TPC output control register
nder	Next data enable register
ndra	Next data register A (address: 0xxffa5h)
ndra0	Next data register A (address: 0xxffa7h)
ndrb	Next data register B (address: 0xxffa4h)
ndrb2	Next data register B (address: 0xxffa6h)

wdt (Watch dog timer)

wdtcsr	Timer control/status register
wdtcnt	Timer counter
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)

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)
adcsr	A/D control/status register
adcr	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

flash (flash memory)

flmcr	Flash memory control register
ebr1	Erase block appoint register 1
ebr2	Erase block appoint register 2
ramcr	RAM control register

Note

These register cannot control the flash memory. But the emulator can display or modify these register except for the ramcr register. The remcr register is always FF'H.

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 H8/3048 emulator. The cause of the errors is described, as well as the action you must take to remedy the situation.

Message

140 : Stack is in I/O registers

Cause

This error occurs when you attempt to execute user program (with **r** or **s** command) with the stack pointer set at internal I/O register area.

Action

Set up the stack pointer with **cf rsp** command. Refer to chapter 3 of this manual for more information.

Message

141 : 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

170 : Emulation memory card not found in card cage

Cause

This error occurs when you don't insert memory board in card cage, or connect memory board which is not supported.

Action

Insert correct memory board.

Index