29K™ Family Simulators Reference Manual

29K™ Family Simulators Reference Manual, Release 3.3

The text pages of this document have been printed on recycled paper consisting of 50% recycled fiber and 50% virgin fiber; the post-consumer waste content is 10%. These pages are recyclable.

Advanced Micro Devices, Inc.
5204 E. Ben White Blvd.
Austin, TX 78741–7399

Contents

About the 29K Family Simulators

Chapter 1

Using the Simulators

Chapter 2

ISSTIP Example

Chapter 3

29K Family Three-Bus Processor Architecture Simulation

Chapter 4

29K Family Two-Bus Processor Architecture Simulation

Chapter 5

29K Family Microcontroller Architecture Simulation

Chapter 6

SIM29 OS Interface

Appendix A

Error Messages

Index

Figures and Tables

Figures

Tables

About the 29K Family Simulators

The AMD® 29K™ Family simulators, **isstip** and **sim29**, enable the user to execute 29K Family processor programs on a host system, without any 29K Family processor hardware. The following pages list and describe each chapter in this product manual, and then discuss the standards and conventions.

Product Documentation

This documentation is written for the experienced program developer, who is assumed to have a working knowledge of the C language and the specific 29K Family microprocessor or microcontroller being used. For further information on a particular 29K Family microprocessor or microcontroller, see the appropriate user's manual.

About This Manual

Chapter 1: "Using the Simulators" describes the syntax for invoking the instruction set simulator **isstip** and the architectural simulator **sim29**.

Chapter 2: "ISSTIP Example" provides examples of using the 29K Family instruction set simulator, **isstip**, with the MiniMON29K™ user interface, **mondfe**.

Chapter 3: "29K Family Three-Bus Processor Architecture Simulation" describes how to use **sim29** to simulate the Am29000®, Am29005™, and Am29050™ microprocessors.

Chapter 4: "29K Family Two-Bus Processor Architecture Simulation" describes how to use **sim29** to simulate the Am29030™, Am29035™, and Am29040™ microprocessors.

Chapter 5: "29K Family Microcontroller Architecture Simulation" describes how to use **sim29** to simulate the Am29200™, Am29205™, Am29240™, Am29243™, and Am29245™ microcontrollers.

Chapter 6: "SIM29 OS Interface" discusses the simulator's general interface to the **osboot** and trap code.

Appendix A: "Error Messages" lists and explains the simulator error messages.

Suggested Reference Material

The following additional reference documents may be of interest to the user:

- *Am29000® and Am29005™ User's Manual and Data Sheet*
  Advanced Micro Devices, order number 16914.
- *Am29030™ and Am29035™ Microprocessors User's Manual and Data Sheet*
  Advanced Micro Devices, order number 15723.
- *Am29040™ Microprocessor Data Sheet*
  Advanced Micro Devices, order number 18459.
- *Am29040™ Microprocessor User's Manual*
  Advanced Micro Devices, order number 18458.
- *Am29050™ Microprocessor Data Sheet*
  Advanced Micro Devices, order number 15039.
- *Am29050™ Microprocessor User's Manual*
  Advanced Micro Devices, order number 14778.
- *Am29200™ and Am29205™ RISC Microcontrollers Data Sheet*
  Advanced Micro Devices, order number 16361.
- *Am29200™ and Am29205™ RISC Microcontrollers User's Manual*
  Advanced Micro Devices, order number 16362.
- *Am29240™, Am29245™, and Am29243™ RISC Microcontrollers Data Sheet*
  Advanced Micro Devices, order number 17787.
- *Am29240™, Am29245™, and Am29243™ RISC Microcontrollers User's Manual*
  Advanced Micro Devices, order number 17741.
- Harbison, Samuel P. and Guy L. Steele, Jr.: *C: A Reference Manual, Second Edition*, Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1987.
- *Host Interface (HIF) Specification*
  Advanced Micro Devices, order number 11539.
- Kernighan, Brian W. and Dennis M. Ritchie: *The C Programming Language, First Edition*. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1978.

- Kernighan, Brian W. and Dennis M. Ritchie: *The C Programming Language, Second Edition*. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1988.
- *Programming Language C*, American National Standards Institute, 311 First St. NW, Suite 500, Washington, DC, 20001. ANSI document X3.159, 1989. Available from Global Engineering Documents, telephone 1–800–854–7179.
- *Programming the 29K™ RISC Family* by Daniel Mann, P T R Prentice-Hall, Inc. 1994

Standards and Conventions

Standards

This product complies with the following standards:

- COFF: AMD Common Object File Format

  Conforms to the AMD-augmented version of AT&T COFF, as described in the AMD *Common Object File Format (COFF) Specification*.

- HIF: AMD Host Interface

  Conforms to the AMD *Host Interface (HIF) Specification*.

- IEEE 754, 1985

  Conforms to the IEEE-approved standard for binary floating-point arithmetic.

- UDI: AMD Universal Debugger Interface

  Conforms to the AMD *Universal Debugger Interface (UDI) Specification*.

Conventions

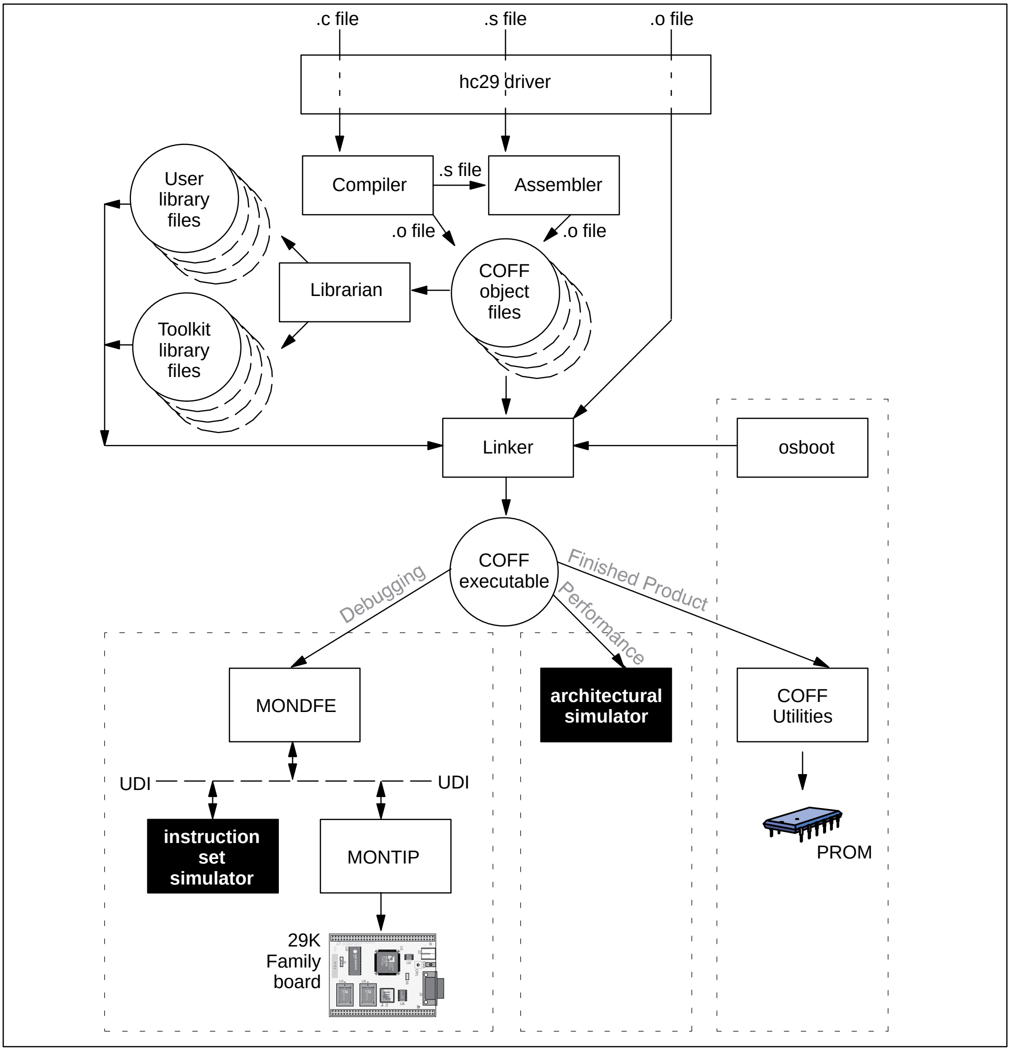
- UNIX pathnames use a forward slash (/) to separate directories, while MS-DOS pathnames use a backslash (\). For brevity, only the DOS backslash is used when specifying pathnames. In some cases, code examples are specified as either for UNIX or MS-DOS environments and the correct slash is used.
- The following abbreviations may be used in this manual:
  - LSB least significant bit
  - LSW least significant word
  - MSB most significant bit
  - MSW most significant word
  - NaN not a number
  - QNaN quiet not a number
- In this manual, a data word signifies a 32-bit entity; a data halfword signifies a 16-bit entity.
- This manual uses the notational conventions shown in Table 0-1 (unless otherwise noted). These same conventions are used in all the 29K Family support products manuals.

Table 0-1. Notational Conventions

Symbol	Usage
Boldface	Indicates that characters must be entered exactly as shown. The alphabetic case is significant only when indicated.
Italic	Indicates a descriptive term to be replaced with a user-specified term.
`Typewriter face`	Indicates computer text input or output in an example or listing.
[ ]	Encloses an optional argument. To include the information described within the brackets, type only the arguments, not the brackets themselves.
{ }	Encloses a required argument. To include the information described within the braces, type only the arguments, not the braces themselves.
..	Indicates an inclusive range.
...	Indicates that a term can be repeated.
|	Separates alternate choices in a list—only one of the choices can be entered.
:=	Indicates that the terms on either side of the sign are equivalent.

Chapter 1

Using the Simulators



0395

This chapter provides a brief overview of the available simulators and then describes how to invoke each on UNIX and MS-DOS systems. (The commands are the same for all systems.)

If the user types an incorrect command, the resulting error message is written to the standard error device.

The return codes are as follows:

0 = success
not 0 = failure

0395

Using the 29K Family Simulators

There are two simulators available for executing 29K Family application code: the instruction set simulator, **isstip**, and the architectural simulator, **sim29**.

Both simulators support all current 29K Family microprocessors and microcontrollers: the Am29000, Am29005, Am29030, Am29035, Am29040, and Am29050 microprocessors, and the Am29200, Am29205, Am29240, Am29243, and Am29245 microcontrollers.

The intended users of **isstip** are:

- Software engineers developing and/or debugging 29K Family applications.

The intended users of **sim29** are:

- Engineers evaluating the performance of the 29K Family of microprocessors for possible application designs.
- Board designers evaluating different memory configurations to achieve a desired performance.
- Software developers using the profiling capabilities of **sim29** to optimize their code.

0395

isstip

isstip is useful as an introductory evaluation platform and as a debugging tool for many applications. Because it is hardware free, it is low cost and convenient. Being entirely virtual can also aid significantly in debugging critical kernel or interrupt level code, since breakpoints can be set even if interrupts are disabled. The disadvantage is that it is not as fast as hardware, but performance is improving, making the simulator practical for a large set of applications.

isstip is compliant with the AMD Universal Debugger Interface (UDI), so it can be invoked from multiple debugger front ends (DFEs). Although most Target Interface Processes (TIPs) use UDI to provide the communication between the target and the DFE, in the case of **isstip**, there is no distinction between the TIP and the target, which is the Instruction Set Simulator. All are contained in one executable called **isstip**. See the UDI online documentation for more information on UDI.

sim29

The architectural simulator program, **sim29**, can execute the same application programs as **isstip** but rather than being used as a debugging tool, **sim29** is designed for performance estimation. **sim29** models the processor pipeline, cache, and memory latencies in order to accurately report the elapsed time for a benchmark or application. In addition to simple cycles, several profiling statistics are available, making the simulator useful for optimizing the performance of applications. However, the extra modelling done by the simulator slows its performance, making it impractical for large scale debugging. In addition, its debugging features are limited.

sim29 is not currently UDI-based but UDI support is planned for a future release.

Simulator options vary according to the processor being simulated. Option variations occur due to architectural differences between processors, and because some processors are simulated with entirely separate implementations than others.

The simulator is implemented by several different programs. The program invoked by the **sim29** "driver" depends on which microprocessor or microcontroller is specified. One program implements the Am29000, Am29005, and Am29050 microprocessors. The second program implements the Am29030 and Am29035 microprocessors. The third program implements the Am29040 microprocessor, and the fourth program implements the Am29200, Am29205, Am29240, Am29243, and Am29245 microcontrollers.

0395

The Am29000, Am29005, and Am29050 processor program mimics the microprocessor's actual implementation in considerable detail, beyond the cycle-by-cycle state defined by the microprocessor's architecture. As such, the performance estimates are quite accurate, but the programs execute somewhat slower and are more prone to failure due to the increased complexity of the program. The remaining programs (those that simulate the Am29030, Am29035, and Am29040 microprocessors, and the Am29200, Am29205, Am29240, Am29243, and Am29245 microcontrollers) model only those parts of the implementation relevant to performance, namely the pipeline and memory channel. As such, they run considerably faster than the Am29000, Am29005, and Am29050 processor simulators, which is an important feature when hundreds of millions of cycles are being simulated.

Invoking isstip

NOTE: The **isstip.exe** for MS-DOS hosts supplied with this product uses extended memory on a PC. This facilitates running larger programs on the simulator, the sizes of which are not limited by the amount of conventional memory.

isstip is a UDI-conformant Target Interface Process (TIP) for the 29K Family Instruction Set Simulator. It implements the Instruction Set Simulator with an interface defined by the UDI Specification. **isstip** can be used with any UDI-conformant Debugger Front End, such as **mondfe** or the XRAY29K™ product's **xray29u**.

The **isstip** command and its options should be provided in the UDI Configuration File with its entry. The format of the UDI Configuration for UNIX (**udi_soc** file) and MS-DOS (**udiconfs.txt** file) hosts is explained in the UDI online documentation.

The UDI Configuration File entry's TIP ID (first field of the entry) acts as the link between the DFE being used and **isstip**. The TIP ID is provided as a command-line argument to the DFE being used. For example, the command-line option **–TIP** of **mondfe** can be used to specify the TIP you want to use:

```
mondfe –D –TIP tip_id
```

This makes the DFEs independent of TIP-specific features and options. Please refer to the appropriate DFE documentation for more information on DFE command-line syntax and options.

0395

Syntax: isstip *processor* [–id {0|1}] [–le] [–n] [–p |–v]
[–r *boot_prog*] [–sp {0|1}] [–st *hexaddr*] [–t] [–tm] [–ww]

NOTE: These options are case sensitive.

where:

processor Is one of: **–29000**, **–29005**, **–29030**, **–29035**, **–29040**, **–29050**, **–29200**, **–29205**, **–29240**, **–29243** or **–29245**. These options specify which 29K Family microprocessor to simulate. Based on the option specified, **isstip** downloads the default ROM file, which can be overridden using the **–r** option (see the **–r** option for the default files). The **–n** option can be used to prevent downloading of any ROM file. Note that the *processor* is required and must be the first argument specified.

–id {0|1} Specifies whether to simulate a separate Instruction and Data address space. The default is 0, which means that the Instruction and Data spaces are the same. This option only applies to the Am29000, Am29005, and Am29050 processor simulators.

–le Specifies to simulate a little endian system. The default is big endian.

–n Can be used to bring up **isstip** without downloading a ROM file. This may be used if an application has its boot code already linked into the application.

–p | –v When the **–p** option is specified, the execution mode of the programs is physical mode. When **–v** is specified, virtual mode execution is simulated. The TLB traps provided implement a one-to-one mapping of physical addresses to virtual addresses. The default mode is physical.

–r *boot_prog* Specifies the **osboot** or ROM program. The *boot_prog* is a full pathname to the file. By default, the simulator will attempt to load the appropriate boot code needed by the compiler, based on the processor specified on the command line.

0395

The default *boot_prog* depends on the *processor* specified. The defaults are:

osb00x	For **–29000** and **–29005**
osb03x	For **–29030**, **–29035**, and **–29040**
osb050	For **–29050**
osb20x	For **–29200** and **–29205**
osb24x	For **–29240**, **–29243**, and **–29245**

When the **–r** option is specified, **isstip** first looks for *boot_prog* in the current working directory. If the file is not found, the directories specified in the **PATH** environment variable are searched after replacing the last branch with **lib**. For example, if **PATH** is:

```
C:\29k\bin;c:\29k\lib;d:\c600\bin;
```

Then the directories searched by **isstip** are:

`C:\29k\lib` and `c:\29k\lib` and `d:\c600\lib`

The code contained in this "ROM" file is downloaded and executed by **isstip** before executing the application program. Therefore, if the user's application has the startup code in it, do not use this option.

–sp {0|1} Specifies whether to simulate a separate ROM and Instruction RAM space. The default is 1, which means that the ROM space is different from Instruction RAM space. By using 0, the Instruction RAM and ROM can be made to refer to the same address space. This is useful when using **isstip** with the XRAY29K debugger.

–st *hexaddr* For special programs, this can be used to specify the address to be used by the operating system as the highest addressable data address. The default value of 0 is ignored by the operating system.

–t When specified, enables "trap"ping on certain instructions (like floating-point operations), thus using the trapware installed. When disabled, the simulator performs the necessary operation and produces the result. The default is trap disabled, which may improve the simulation speed for some applications.

–tm When specified, enables the Timer during simulation. The default is Timer disabled, which provides a slight boost to simulation speed.

–ww Specifies to simulate word-write only. The default is byte-write allowed.

Example

A sample entry in **udiconfs.txt** for executing on MS-DOS hosts would be:

```
iss_id isstip.exe –29000
```

In the above entry, **iss_id** is any ID name of the user's choice.

To use **isstip**, invoke the debugger front end, giving it the TIP ID defined in the configuration file. Make sure that the **udiconfs.tx**t file is either in the current working directory, or is defined by the environment variable **UDICONF**. Also, make sure that the **PATH** environment variable is set up so that **isstip.exe** and the *boot_prog* for the **–29000** option, **osb00x**, can be found. After completing the above steps, the **hello** program can be run using **mondfe** on an MS-DOS host by typing:

```
mondfe –TIP iss_id hello
```

The **–TIP** option of **mondfe** is used to specify which TIP ID to use from the UDI configuration file. By specifying the TIP ID as **iss_id**, **isstip.exe** is invoked and is passed **–29000** as an argument.

0395

Invoking sim29

Syntax: sim29 *processor* [*processor_options*] [–d] [–e *event_file*] [–f *frequency*] [–h *heap_size*] [–o *output_file*] [–r *boot_prog*] [–v] [*app_prog* [*prog_args*]]

NOTE: These options are case sensitive.

where:

processor

Is one of: **–29000**, **–29005**, **–29030**, **–29035**, **–29040**, **–29050**, **–29200**, **–29205**, **–29240**, **–29243** or **–29245**. These options specify which 29K Family microprocessor to simulate. Note that the *processor* is required and must be the first argument specified.

processor_options

Are options specific to a processor and can be either *3-bus_options*, *2-bus_options*, or *controller_options*, as described on the following pages, and in more detail in Chapters 3, 4, and 5.

3-bus_options are for the 29K Family three-bus microprocessors, which currently include the Am29000, Am29005, and Am29050 processors; *2-bus_options* are for the 29K Family two-bus microprocessors, which currently include the Am29030, Am29035, and Am29040 processors; and *controller_options* are for the 29K Family microcontrollers, which currently include the Am29200, Am29205, Am29240, Am29243, and Am29245 microcontrollers.

3-bus_options

Are the options for the Am29000, Am29005, and Am29050 microprocessors:

–cfg=*xx*

Specifies the setting of the configuration register, where *xx* is a 1- to 5-digit hexadecimal number. This setting overrides the default setting. No run-time modification of the configuration register is permitted if this option is specified.

0395

–n	Does not allocate two extra words at the end of data sections. The default is to put two extra words at the end of data sections so that the read-ahead library routines, **str*( )** and **mem*( )**, will not try to read beyond the end of memory.
–p *range*	Specifies code profiling to take place in the specified *range* of RAM memory space. The *range* parameter is required, and it specifies a range of hexadecimal values in the form: 1000–2ACE.
–t *max_syscalls*	Specifies the maximum number of system call types that will be used during the simulation. This option is used to allocate the array for storing the system call count for different calls. The default is 256 types.
–x[*error_code*]	Specifies that the simulator exits if an error occurs for one of the enabled *error_code* values. The *error_code* parameter is optional, and if not given, all codes are enabled. By default, no error codes are enabled. When specifying codes, each must be entered as an uppercase letter, and multiple codes must not be separated by spaces or tabs. Possible values for *error_code* are: A Address error occurred (e.g., out of bounds). K Kernel error occurred (i.e., an error in supervisor mode). O Illegal opcode error occurred. F An arithmetic trap occurred (e.g., divide by zero). P A protection violation occurred. S An error in the event file occurred.

0395

2-bus_options
controller_options

Are the options for the Am29030, Am29035, and Am29040 microprocessors, and the Am29200, Am29205, Am29240, Am29243, and Am29245 microcontrollers:

–dcacheoff	Disables the data cache (applies only to the Am29040, Am29240, and Am29243 processors).
–dynmem {0|1}	Dynamically allocates memory for address references not defined by the application Common Object File Format (COFF) file. 1 enables; 0 disables.
–help	Outputs ASCII text to standard output that briefly describes all command-line and event-file options.
–icacheoff	Disables the instruction cache.
–p	Profiles opcode, PC, Load, Store, and trap usage.
–sv	Passes a parameter to **osboot** indicating that the application is to run in supervisor mode.
–u	Configures memory wait states and enable caches via application software instead of simulator options.

–d Dumps the contents of the registers at the end of simulation.

–e *event_file* Specifies the simulator event input file, which is used for other miscellaneous commands. The *event_file* is a full pathname to the file.

–f *frequency* Specifies the CPU frequency in MHz.

–h *heap_size* Specifies the size of the system heap (in kilobytes). The *heap_size* parameter is a decimal value. The default system heap size is 32 Kbytes, or **–h 32**.

–o *output_file* Specifies the simulation summary file (**sim.out** is the default name). The *output_file* is a full pathname to the file.

–r *boot_prog* Specifies the **osboot** or ROM program. The *boot_prog* is a full pathname to the file. By default, the simulator will attempt to load the appropriate boot code needed by the compiler, based on the processor specified on the command line.

The default *boot_prog* depends on the *processor* specified. The defaults are:

osb00x	For **–29000** and **–29005** options
osb03x	For **–29030**, **–29035**, and **–29040** options
osb050	For **–29050** option
osb20x	For **–29200** and **–29205** options
osb24x	For **–29240**, **–29243**, and **–29245** options

When the **–r** option is specified, **sim29** first looks for *boot_prog* in the current working directory. If the file is not found, the directories specified in the **PATH** environment variable are searched after replacing the last branch with **lib**. For example, if **PATH** is:

```
C:\29k\bin;c:\29k\lib;d:\c600\bin;
```

Then the directories searched by **sim29** are:

`C:\29k\lib` and `c:\29k\lib` and `d:\c600\lib`

The code contained in this "ROM" file is downloaded and executed by **sim29** before executing the application program. Therefore, if the user's application has the startup code in it, specify that application file as the *boot_prog*.

–v
: Passes a parameter to **osboot** which will turn on instruction and data address virtual memory translation. This option does not apply to the Am29200 or Am29205 microcontrollers. 0395

app_prog
: Specifies the filename of the program to be simulated. The *app_prog* parameter is not required if the user's application has boot code linked in, and the **–r** option is used. Otherwise, the *app_prog* must be provided, and is a full pathname to the program object file.

prog_args
: Specifies command-line options for the program to be simulated. This argument is optional. Programs need not have command-line options to execute properly.

The output generated by the simulator includes the output generated by the program being simulated and the performance statistics, such as the number of processor cycles simulated, and the MIPS (millions of instructions per second).

For a more extensive discussion of the options for each processor type, see Chapters 3, 4, and 5.

Example

```
sim29 –29000 –e mysim.evt –o mysim.out myprog –Wcf
```

The **–29000** option specifies the Am29000 microprocessor is being simulated.

The **–e** option specifies an event input file. The parameter following the option, **mysim.evt** in this case, is the name of a file that is opened by the simulator at the beginning of run. Events specified within this file are used to control the actions of the simulator.

The **–o** option specifies that the following filename, **mysim.out**, be assigned as the output file to contain a transcript of the current simulation's results. If the **–o** option is not specified, the simulator will write the transcript on the file **sim.out**.

The **–o** option is the last option in this example, and the following command-line argument (**myapp**) is treated as the filename of the program whose execution is to be simulated. The program file must be in object form, already linked to execute in the simulator's execution environment.

The **–Wcf** argument at the end of the command line represents a hypothetical command-line argument to the **myapp** program. Command-line arguments for programs whose execution is to be simulated are specified last, in the form the program requires.

0395

Example

```
sim29 –29000 –e dhry2.evt –o dhry2.out dhry2 < dhry2.in
```

This example illustrates how the command line would be entered for a program that requires input from the standard input device (**stdin**). The angled bracket indicates (for both MS-DOS and UNIX systems) that the standard input is to be taken from a file whose name immediately follows (**dhry2.in** in this case). Simulator options are specified first, followed by the filename of the program to be simulated (**dhry2** in this case).

Example

```
sim29 –29035 –r myboot –sv my_program
```

This example illustrates the command line entered to simulate in Am29035 processor mode using the ROM file **myboot**. The **–sv** option causes the program to run in supervisor mode.

Example

```
sim29 –29050 –r myboot –cfg=b0 my_program
```

This example illustrates the command line entered to simulate in Am29050 microprocessor mode using the ROM file **myboot**. The **–cfg** option prevents the simulator from setting the configuration register to the default value at start-up and also prevents any instruction in the ROM or RAM file from setting the configuration register. In this example, the simulator will set the configuration register to the value 0xb0, which sets the register's early load enable (EE), data width (DW), and vector fetch (VF) bits.

Example

```
sim29 –29200 hello.lap
```

This example illustrates the command line to simulate the **hello.lap** program in the Am29200 microcontroller. Because the **–r** option is not specified, the default boot file **osb20x** is used.

Example

```
sim29 –29240 –e sim.evt test.lap
```

This example illustrates simulating the execution of program **test.lap** on the Am29240 microcontroller using the default boot file **osb24x** with the memory configuration as specified in the **sim.evt** file.

0395

Chapter 2

ISSTIP Example

This chapter provides examples of using the 29K Family instruction set simulator, **isstip**, with the MiniMON29K user interface, **mondfe**. **isstip** is a UDI-conformant Target Interface Process (TIP) that can be used with any UDI-conformant Debugger Front End (DFE). The TIP and the DFE run as separate processes and communicate with each other according to AMD's *Universal Debugger Interface (UDI) Specification*.

There are three items needed before simulating a program using **isstip** and **mondfe**:

- **isstip** – The 29K Family Instruction Set Simulator. (Note: Unlike other TIPs, in the case of **isstip**, there is no distinction between the Target Interface Process (TIP) and the target, which is the Instruction Set Simulator. All are contained in one executable called **isstip**.)
- **mondfe** – A UDI-conformant Debugger Front End (DFE) to issue commands. For more information, see the **mondfe** documentation.
- UDI Configuration File – The communication link for **mondfe** and **isstip**.

There are two environment variables that are used if they are defined: **PATH** and **UDICONF**. See the UDI online documentation for more information on these environment variables.

This chapter first provides an example tutorial for MS-DOS hosts (page 2-2), and then a tutorial for UNIX hosts (page 2-5).

Example Tutorial For MS-DOS Hosts

This tutorial assumes that High C® 29K™ compiler and libraries have been installed on the C:\ drive. Complete the following steps to set up the environment.

1. Create a file with the following contents:

   ```
   set PATH=C:\29K\BIN;%PATH%
   set UDICONF=C:\29K\LIB\UDICONFS.TXT
   ```

   Name this file **udisetup.bat**.

2. Run the batch file created above:

   ```
   udisetup
   ```

   Now the **PATH** variable is set to pick up the newly installed **mondfe** and **isstip** from the **C:\29K\BIN** directory. The **UDICONF** variable is pointing to the UDI configuration file, **udiconfs.txt**, in the **C:\29K\LIB** directory.

   Take a few minutes to go through the **udiconfs.txt** file and the various line-entries. Remember that the first column gives the ID, which is given as a command-line argument to **mondfe**.

3. The line-entry from **udiconfs.txt** that will be used in this example is for the ID, **iss000**. Here is the complete line:

   ```
   iss000 isstip.exe –29000
   ```

   The first column, **iss000**, gives the unique ID for the rest of that line. This ID is used as a command-line argument to **mondfe**. The second column gives the executable name of the TIP to use. In this case, it is **isstip.exe**, which is the Instruction Set Simulator TIP. If there is no **isstip.exe** in the current working directory, the **PATH** environment variable is searched to locate the executable. Since the **PATH** has already been set up above, the **isstip.exe** from the **C:\29K\BIN** directory will be used. (NOTE: Remove any files named **isstip.exe** from the current working directory if it is *not* **C:\29K\BIN**.) The rest of the line following “isstip.exe” gives the string of options that will be passed to **isstip.exe** when it is invoked.

The **–29000** option above specifies the “ROM” file that **isstip** must download (to ROM space) before executing any user commands. **isstip** will search the **PATH** variable to locate the osboot file to download (**osb00x** is the default file for the **–29000** option). As per the **PATH** environment variable, **isstip** will download the osboot file found under the **C:\29K\LIB** directory, unless the osboot file exists in the current working directory. The default settings of the rest of the options apply. Refer to page 1-6 for more information on the different options to **isstip**, their meanings, and their default values.

4. To use **mondfe** to invoke **isstip** according to the above entry, specify the entry ID, **iss000,** to **mondfe**. The **–TIP** option of **mondfe** allows you to do exactly that. This example uses the **–D** command-line option of **mondfe** to start an interactive debug session. Thus, the complete command-line to use is:

   ```
   mondfe –D –TIP iss000
   ```

   The default settings for the remaining command-line options apply. Refer to the **mondfe** manual for more information on the various **mondfe** options, their meanings, and their default values.

5. When **mondfe** is invoked using the above command-line, a sign-on message similar to the following appears:

   ```
   >AMD MONDFE Version: 4.3.5 IPC Version: 1.4.0 UDI Rev. 1.4.0 <
   >TIP Version: 4.0.2 IPC Version: 1.3.0 UDI Rev. 1.3.0<
   UDI 1.3 ISSTIP for 29K Family: ROM file: c:\29k\lib\osb00x
   MONDFE.EXE>
   ```

   The first line is printed by **mondfe**, and it gives the version number, the UDI revision of the IPC implementation **mondfe** is using, and the version of the IPC implementation itself. The second line is a formatted display of the version numbers of the different components of **isstip**. The third line is a descriptive string returned by **isstip** about itself. The last line displayed is the **mondfe** prompt, which is **MONDFE.EXE**>.

6. From here on, there are two processes actually running that are transparent to the user: the DFE (**mondfe**) and the instruction set simulator (**isstip**). The **mondfe Y** (**Yank**) command can be used to download the program as shown below:

```
MONDFE.EXE> Y hello.out
loading hello.out
Loaded TEXT section from 0x00010000 to 0x00013fb4
Loaded DATA section from 0x80003000 to 0x800033a0
Loaded LIT section from 0x800033a0 to 0x80003694
Cleared BSS section from 0x80003698 to 0x80003874
Ignoring COMMENT section (32 bytes) ...
MONDFE.EXE>
```

Then, type a **G** (**Go**) command to execute the program as shown below:

```
MONDFE.EXE> G
MONDFE.EXE>Hello, world!
Hello World Stderr!
Process exited with 0x0
MONDFE.EXE>
```

Use the **Q** (**Quit**) command to terminate the debug session as shown below:

```
MONDFE.EXE> Q
MONDFE.EXE>
Goodbye.
```

When exiting from the front end (using the **Q** command), the TIP also is killed.

Example Tutorial For UNIX Hosts

This tutorial assumes that High C® 29K™ compiler and libraries have been installed under the **/usr** directory. Complete the following steps to set up the environment.

1. Set the **PATH** and **UDICONF** environment variables using the following commands:

   ```
   setenv PATH   /usr/29k/bin:‘echo $PATH‘
   setenv UDICONF  /usr/29k/lib/udi_soc
   ```

   Now the **PATH** environment variable is set to pick up the newly installed **mondfe** and **isstip** from the **/usr/29k/bin** directory. The **UDICONF** variable is pointing to the UDI configuration file, **udi_soc**, in the **/usr/29k/lib** directory.

   Take a few minutes to go through the **udi_soc** file and the various line-entries. Remember that the first column gives the ID, which is given as a command-line argument to **mondfe**.

2. The line-entry from **udi_soc** that will be used in this example is for the ID, **isstip_unix**. Here is the complete line:

   ```
   isstip_unix AF_UNIX sockiss -ux isstip –29000
   ```

   The first column, **isstip_unix**, gives the unique ID for the rest of that line. This ID is used as a command-line argument to **mondfe**. The second column gives the socket address family, which is **AF_UNIX**. This means that both the DFE, **mondfe**, and the TIP, **isstip**, are executing on the same machine. The third field is the name of the socket itself. The fourth field gives the executable name of the TIP to use. In this case, it is **isstip**, which is the Instruction Set Simulator TIP. If there is no **isstip** in the current working directory, the **PATH** environment variable is searched to locate the executable.

   Since the **PATH** has already been set up above, the **isstip** from the **/usr/29k/bin** directory will be used. (NOTE: Remove any file named **isstip** from the current working directory if it is *not* **/usr/29k/bin**.) The rest of the line following “isstip” gives the string of options that will be passed to **isstip** when it is invoked.

The **–29000** option above specifies the ROM file that **isstip** must download (to ROM space) before executing any user commands. The **isstip** will search the **PATH** variable to locate the osboot file to download (**osb00x** is the default file for the **–29000** option). As per the **PATH** environment variable, **isstip** will download the osboot file found under the **/usr/29k/lib** directory, unless the osboot file exists in the current working directory. The default settings of the rest of the options apply. Refer to page 1-6 for more information on the different options to **isstip**, their meanings, and their default values.

3. To use **mondfe** to invoke **isstip** according to the above entry, specify the entry ID, **isstip_unix**, to **mondfe**. The **–TIP** option of **mondfe** allows you to do exactly that. This example uses the **–D** command-line option of **mondfe** to start an interactive debug session. Thus, the complete command-line to use is:

   ```
   mondfe –D –TIP isstip_unix
   ```

   The default settings for the remaining command-line options apply. Refer to the **mondfe** manual for more information on the various **mondfe** options, their meanings, and their default values.

4. When **mondfe** is invoked using the above command-line, a sign-on message similar to the following appears:

   ```
   >AMD MONDFE Version: 4.3.5 IPC Version: 1.4.0 UDI Rev. 1.4.0 <
   >TIP Version: 4.0.2 IPC Version: 1.3.1 UDI Rev. 1.3.0<
   UDI 1.3 ISSTIP for 29K Family: ROM File: /usr/29k/lib/osb00x
   mondfe>
   ```

   The first line is printed by **mondfe**, and it gives the version number, the UDI revision of the IPC implementation **mondfe** is using, and the version of the IPC implementation itself. The second line is a formatted display of the version numbers of the different components of **isstip**. The third line is a descriptive string returned by **isstip** about itself. The last line displayed is the **mondfe** prompt, which is **mondfe**>.

5. From here, there are two processes actually running that are transparent to the user: the DFE (**mondfe**) and the instruction set simulator (**isstip**). The **mondfe Y** (**Yank**) command can be used to download the program as shown below:

```
mondfe> y hello.out
loading hello.out
Loaded TEXT section from 0x00010000 to 0x00013fb4
Loaded DATA section from 0x80003000 to 0x800033a0
Loaded LIT section from 0x800033a0 to 0x80003694
Cleared BSS section from 0x80003698 to 0x80003874
Ignoring COMMENT section (32 bytes) ...
mondfe>
```

Then, type a **G** (**Go**) command to execute the program as shown below:

```
mondfe> g
mondfe>Hello, world!
Hello World Stderr!
Process exited with 0x0
mondfe>
```

Use the **Q** (**Quit**) command to terminate the debug session as shown below:

```
MONDFE.EXE> q
MONDFE.EXE>
Goodbye.
```

When exiting from the front end (using the **Q** command), the TIP also is killed.

Chapter 3

29K Family Three-Bus Processor Architecture Simulation

This chapter describes how to use **sim29** to simulate the Am29000, Am29005, and Am29050 microprocessors. The following topics are discussed:

Simulator Command-Line Syntax

Syntax: sim29 *processor* [–cfg=*xx*] [–d][–e *event_file*] [–f *frequency*]
[–h *heap_size*] [–n] [–o *output_file*] [–p *range*]
[–r *boot_prog*] [–t *max_syscalls*] [–v] [–x [*error_code*]]
[*app_prog* [*prog_args*]]

where:

processor	Is one of: **–29000**, **–29005**, or **–29050**. These options specify which 29K Family three-bus microprocessor to simulate. Note that the *processor* is required and must be the first argument specified.
–cfg=*xx*	Specifies the setting of the configuration register, where *xx* is a 1- to 5-digit hexadecimal number. This setting overrides the default setting. No run-time modification of the configuration register is permitted if this option is specified.
–d	Dumps the contents of the registers at the end of simulation.
–e *event_file*	Specifies the simulator event input file, which is used for other miscellaneous commands. The *event_file* is a full pathname to the file.
–f *frequency*	Specifies the CPU frequency in MHz. The default values are 25 for the Am29000 processor, 16 for the Am29005 processor, and 40 for the Am29050 processor.
–h *heap_size*	Specifies the size of the system heap (in kilobytes). The *heap_size* parameter is a decimal value. The default system heap size is 32 Kbytes, or **–h 32**.
–n	Does not allocate two extra words at the end of data sections. The default is to put two extra words at the end of data sections so that the read-ahead library routines, **str*( )** and **mem*( )**, will not try to read beyond the end of memory.
–o *output_file*	Specifies the simulation summary file (**sim.out** is the default name). The *output_file* is a full pathname to the file.

–p *range*	Specifies code profiling to take place in the specified *range* of RAM memory space. The *range* parameter is required, and it specifies a range of hexadecimal values in the form: 1000–2ACE.
–r *boot_prog*	Specifies the **osboot** or ROM program. (Default is **osb00x** for the Am29000 and Am29005 microprocessors, and **osb050** for the Am29050 microprocessor.) The *boot_prog* is a full pathname to the file. By default, the simulator will attempt to load the appropriate boot code needed by the compiler, based on the processor specified on the command line. If the user's application has its boot code linked in, specify that application file as the *boot_prog*.
–t *max_syscalls*	Specifies the maximum number of system call types that will be used during the simulation. This option is used to allocate the array for storing the system call count for different calls. The default is 256 types.
–v	Passes a parameter to **osboot** which will turn on instruction and data address virtual memory translation.
–x[*error_code*]	Specifies that the simulator exits if an error occurs for one of the enabled *error_code* values. The *error_code* parameter is optional, and if not given, all codes are enabled. By default, no error codes are enabled. Codes are entered in uppercase letters, immediately following the **–x** option (e.g., **–xAP**). Possible values for *error_code* are: A Address error occurred (e.g., out of bounds). K Kernel error occurred (i.e., an error in supervisor mode). O Illegal opcode error occurred. F An arithmetic trap occurred (e.g., divide by zero). P A protection violation occurred. S An error in the event file occurred.
app_prog	Specifies the filename of the program to be simulated. The *app_prog* parameter is not required if the user's application has boot code linked in, and the **–r** option is used. Otherwise, the *app_prog* must be provided, and is a full pathname to the program object file.

prog_args	Specifies command-line options for the program to be simulated. This argument is optional. Programs need not have command-line options to execute properly.

The output generated by the simulator includes the output generated by the program being simulated and the performance statistics. Some of the performance statistics are as follows: the number of processor cycles simulated; the MIPS (millions of instructions per second); the percentage time the pipeline was held for various reasons, such as instruction fetch wait, data fetch wait, or LOADM/STOREM wait; register spill and fill count; histogram of different instruction opcodes executed; and memory parameters.

Example

```
sim29 –29000 –e mysim.evt –o mysim.out myprog –Wcf
```

The **–29000** option specifies the Am29000 microprocessor is being simulated.

The **–e** option specifies an event input file. The parameter following the option, **mysim.evt** in this case, is the name of a file that is opened by the simulator at the beginning of run. Events specified within this file are used to control the actions of the simulator.

The **–o** option specifies that the following filename, **mysim.out**, be assigned as the output file to contain a transcript of the current simulation's results. If the **–o** option is not specified, the simulator will write the transcript on the file **sim.out**.

The **–o** option is the last to be specified, and the following command-line argument (**myprog**) is treated as the filename of the program whose execution is to be simulated. The program file must be in object form, already linked to execute in the simulator's execution environment.

The **–Wcf** argument at the end of the command line represents a hypothetical command-line argument to the **myprog** program. Command-line arguments for programs whose execution is to be simulated are specified last, in the form the program requires.

Example

```
sim29 –29000 –e dhry2.evt –o dhry2.out dhry2 < dhry2.in
```

This example illustrates how the command line would be entered for a program that requires input from the standard input device (**stdin**). The angled bracket indicates (for both MS-DOS and UNIX systems) that the standard input is to be taken from a file whose name immediately follows (**dhry2.in** in this case). Simulator options are specified first, followed by the filename of the program to be simulated (**dhry2** in this case).

Example

```
sim29 –29050 –r myboot –cfg=b0 my_program
```

This example illustrates the command line entered to simulate in Am29050 microprocessor mode using the ROM file **myboot**. The **–cfg** option prevents the simulator from setting the configuration register to the default value at start-up and also prevents any instruction in the ROM or RAM file from setting the configuration register. In this example, the simulator will set the configuration register to the value 0xb0, which sets the register's early load enable (EE), data width (DW), and vector fetch (VF) bits.

Default Configuration

The default settings of the simulator are specified by the Configuration (CFG) register and the Current Processor Status (CPS) register. The contents of these registers on simulator start-up, simulating different 29K Family microprocessors, are described in Table 3-1 and Table 3-2.

If the **–cfg** command-line option is used, the CFG register setting is as specified by the option, except the PRL field, which is set by the simulator according to the mode of simulation.

Table 3-1. CFG Register Default Settings for the Three-Bus Microprocessor Simulators

Field	Setting	Description
BO	0	Bytes are numbered from left-to-right, big endian
CD[1]	0	Branch Target Cache enabled
CO[2]	0	Branch Target Cache organized as 64 entries of 4 words each
CP	0	No coprocessor
DW	0	Byte and half-word access not permitted
EE[2]	1	Early loads permitted
PRL	2	Processor release level for Am29000 and Am29005 microprocessors
	32	Processor release level for Am29050 processor
VF	1	Vector area is a block of 256 vector addresses
RV	0	ROM Vector Area (setting is irrelevant since VF=1)

NOTES:

[1]Not applicable to the Am29005 microprocessor.

[2]Not applicable to the Am29000 and Am29005 microprocessors.

Table 3-2. CPS Register Default Settings for the Three-Bus Microprocessor Simulators

Field	Setting	Description
MM[1]	0	Not monitor mode
CA	0	Coprocessor inactive
IP	0	No external interrupts pending
TE	0	Trace enabled
TP	0	No trace pending
TU	0	Unaligned access trap disabled
FZ	1	In freeze mode
LK	0	LOCK controlled by LOADSET, LOADL, and STOREL
RE	1	Instruction access from ROM
WM	0	Wait mode disabled
PD	1	Data address translation disabled
PI	1	Instruction address translation disabled
SM	1	Supervisor mode
IM	0	Interrupt mask cleared
DI	1	Interrupts enabled
DA	1	Interrupt and traps enabled

NOTE:
[1]Not applicable to the Am29000 and Am29005 microprocessors.

The Event File

The event file is a command file used to specify simulation control parameters, most of which are not handled by the command line. These options primarily consist of memory configuration and wait states.

The simulator executes a program by a combination of direct execution and interpretation. The simulation control program is in charge at all times. A simulated CPU clock drives the simulation. Both rising and falling edges are simulated, and register contents become valid outputs on rising edges, while new contents are clocked in on falling edges, in general.

In addition to reading signals from the processor clock—on which the simulator depends to fetch, decode and execute instructions successively in the target program—the simulator optionally will read an event file that contains 1 or more single-line entries, each consisting of a processor clock value and a command. The processor clock value tells the simulator when the event is to occur, and the command specifies a task for the simulator to do at that time. Commands are provided for entering comments into the output file, stopping the simulation at a particular time, setting simulator parameters, showing current state information, and the like.

Specifying Constants in the Event File

The Am29000, Am29005, and Am29050 processor simulators will accept constant values encoded in decimal, hexadecimal, octal, and binary format, as well as character constants. The rules to express constants in these various radices are straightforward. In general, they conform to similar rules for expressing constant values in the C language, with a few exceptions.

Decimal Constants

The simulator assumes that constant values are expressed in decimal form, unless explicitly declared otherwise. A decimal constant can be expressed either as a contiguous series of digits in the range 0–9, or as a group of decimal digits preceded by the prefix **D** or **d**, and enclosed within single-quote characters. In the case where digits are enclosed within single quotes, embedded space characters are permitted. When expressed as simple decimal integers, space characters serve to separate values. Leading zero digits are ignored in any case. Examples of legal decimal constants are shown below:

```
0  19  d'27'  d'1 3'  d'025'  0189
```

Hexadecimal Constants

Traditionally, hexadecimal constants are entered by prefixing the value with the characters **0X** or **0x**; however, the simulator also allows them to be entered as a group, prefixed by **H**, **h**, **X** or **x**, and enclosed within single-quote characters. Hexadecimal values are restricted to the characters 0–9 and A–F or a–f. In the case where hexadecimal values are enclosed within single-quote characters, embedded space characters are permitted. When expressed as simple hexadecimal integers, space characters serve to separate values. Leading zero digits are ignored in any case. Examples of legal hexadecimal constants are shown below:

```
0xbE  0X  H'FF'  h'7 8'  x'2a'  0xee123
```

Octal Constants

Octal constants are entered by prefixing the value with the letter **O** or **o**, and enclosing the octal digits within single-quote characters. Octal values are restricted to sequences of digits in the range 0–7. Embedded spaces within the octal integer value are allowed. Examples of legal octal integer values are shown below:

```
O'13'  o'17 7'  O'06352'
```

Binary Constants

Binary constants are entered by prefixing the value with the letter **B** or **b**, and enclosing the binary digits within single-quote characters. Binary values are restricted to sequences of the digits 0 or 1. Embedded spaces within the binary integer value are allowed. Alternatively, binary constant values can be expressed by prefixing the sequence of digits with **0b** or **0B**, in which case, embedded spaces are not allowed. Examples of legal binary integer values are shown below:

```
B’1010  0000  1001  0101’  0b10111000
```

Character Constants

Character constants consist of single-byte values, which are enclosed by single-quote characters. The value of a character constant is taken to be its ASCII equivalent numeric value. Each character occupies a single 8-bit byte. Only one character can be entered in any of the legal representations; character “strings” are not allowed.

A character may be expressed by its keyboard equivalent (e.g., **A**), or for control or unprintable characters in the ASCII set, a backslash (\) character, followed by its octal numeric equivalent is allowed. For example, the character constant **\014** is assumed to represent a decimal value of 12, which is the ASCII form-feed character.

Examples of common character constants are shown below:

```
’A’  ’a’  ’+’  ’*’  ’=’  ’9’  ’@’
```

Examples of character constants expressed using the leading backslash provision are shown on the following line:

```
’\014’  ’\177’  ’\376’  ’\0’  ’\256’
```

An extension to the leading backslash representation includes commonly used control and quotation characters expressed as shown below:

`\t`	Horizontal tab
`\n`	Newline = linefeed
`\0`	ASCII NULL character
`\\`	Backslash
`\'`	Single-quote character
`\"`	Double-quote character
`\f`	Form-feed
`\v`	Vertical tab
`\r`	Return
`\b`	Backspace

Event File Syntax

Syntax: *cyclenum command*

where:

cyclenum Specifies a time in processor clock cycles when the event is to be executed by the simulator, or a **+** sign can be used before the time to indicate relative time (+n cycles after the last command). All values are integral numbers of cycles, as would be reported by the CPU clock.

command Specifies a simulator command. These are discussed in detail in the following pages. Each command must be completely specified on a single line. The commands are case insensitive.

Multiple events with the same *cyclenum* can be specified; however, the events must be in ascending time order in the file. That is, each entry must have an equal or greater time value than the previous entry. Events with equal time values are “executed” by the simulator all at once, one following the other, in the order they occur in the file.

When the processor clock value equals an event’s *cyclenum*, the event command is executed. Events that the user wishes to be executed at the beginning of the simulation can be entered with a time value of 0. This will guarantee that they are acted on immediately when the simulation begins.

Commands vary in composition, but each command begins with a keyword that designates the function it performs. A command must fit on a single line in the event file, as there is no way to continue a command from one line to another. This does not present a problem, because all simulator commands are short.

The simulator processes events on each pass through its main loop. In this way, the simulator can faithfully reproduce the effect of an external stimulus, or delay, and report the resulting performance effects in the output log.

Table 3-3 shows the list of available simulator commands, and includes a short description of each. The individual commands are fully described in the pages that follow.

Table 3-3. Three-Bus Microprocessor Simulator Events

Command	Description	Page
COM	Places a comment in the simulator output file	3-14
DELTA	Selects addresses and registers to monitor for changed values	3-15
DUMP	Permits dumping segment, data, instruction, TLB, BTC, and register file contents	3-17
LOG	Selects logging of instructions, floating-point unit, and channel activity	3-19
ONERROR	Specifies action to perform if an error occurs	3-22
SET	Sets simulator parameters and configuration	3-24
STOP	Stops the simulator	3-33
TITLE	Changes the title on simulator output	3-34

COM — Simulator Comment

Syntax: *cyclenum* COM *string*

where:

cyclenum Specifies the time at which the comment should be written into the simulator output file.

string Specifies a string of characters, up to an end of line, that is to be written to the simulator output file.

The **COM** command writes a message, specified by the *string* parameter, to the simulator's output file at the designated time. This facility is useful to indicate that a particular portion of a program has been reached, or to title succeeding output.

Example

```
105233  COM =====================================
105233  COM  SIMULATOR OUTPUT IN VICINITY OF BUG
105233  COM =====================================
```

In the example above, three comments are to be written to the simulator output file when the simulator clock value is equal to or greater than **105233** cycles.

DELTA — Enable Delta Monitor

Syntax: *cyclenum* DELTA SHOW *class format address*

where:

cyclenum	Specifies when the **DELTA** command is to take place.
SHOW	Specifies that the contents of the changed location are to be written to the simulator's output file.
class	Specifies the class of data to be monitored. The class can be one of the following:

REG	Specifies a general purpose register.
DATA	Specifies a data memory location.
IROM	Specifies an instruction ROM location.
IRAM	Specifies an instruction RAM location.
SPECIAL	Specifies a special register.
PERIPHERAL	Specifies a peripheral register.

format Specifies the format of the data. The format is one of the following:

BYTE	Indicates one 8-bit byte, shown in decimal.
CHAR	Indicates one 8-bit byte, shown as an ASCII character; if unprintable, a period is printed.
SHORT	Indicates a 16-bit value, shown in decimal.
SHORTX	Indicates a 16-bit value, shown in hexadecimal.
INT	Indicates a 32-bit value, shown in decimal.
HEX	Indicates a 32-bit value, shown in hexadecimal.
LONG	Indicates a 32-bit value, shown in decimal.
FLOAT	Indicates a single-precision floating-point value.
DOUBLE	Indicates a double-precision floating-point value.
INST	Indicates a 32-bit word shown as a disassembled instruction.

address Specifies the address of the data to be monitored and displayed. For registers, the *address* is an absolute register number, while for data or instructions it is a memory address.

The **DELTA** command provides the means to enable checking the contents of a specified register or memory location, watching for a change to occur. When such a change in value occurs, the parameters to the **DELTA** command provide the means to log the occurrence.

Any number of delta monitors may be active concurrently. This debugging tool can help identify problems such as array overruns or constant values that change during the course of execution.

Example

```
90000 DELTA SHOW IRAM INST 00004D70
```

This example illustrates enabling a delta monitor for an instruction RAM location. The *class* is **IRAM**, the *format* is **INST**, and the *address* is **00004D70**. Once the delta monitor is enabled for this location, when a change occurs, the current cycle count and the changed value are written to the simulator's output file.

Example

```
+100 delta show reg hex 96

Delta @ T=1627 exe_pc_l2=001908: Reg[0x60]=00000110
Delta @ T=1669 exe_pc_l2=000940: Reg[0x60]=0FC00020
Delta @ T=1844 exe_pc_l2=000C1C: Reg[0x60]=80003868
Delta @ T=1881 exe_pc_l2=010084: Reg[0x60]=80000000
Delta @ T=1974 exe_pc_l2=01034C: Reg[0x60]=FFFDF7A8
Delta @ T=2009 exe_pc_l2=010660: Reg[0x60]=00000000
Delta @ T=2024 exe_pc_l2=011FEC: Reg[0x60]=80000000
Delta @ T=2087 exe_pc_l2=012094: Reg[0x60]=800033AF
Delta @ T=2094 exe_pc_l2=0120B0: Reg[0x60]=800033AE
Delta @ T=2098 exe_pc_l2=0120C0: Reg[0x60]=800033AD
Delta @ T=2105 exe_pc_l2=01068C: Reg[0x60]=800033AC
Delta @ T=2107 exe_pc_l2=010694: Reg[0x60]=00000000
```

This example illustrates the enabling of the delta monitor on the general purpose register 96 (gr96) 100 cycles from the last event command.

DUMP — Dump to Output

Syntax: *cyclenum* DUMP {SEGINFO | REG | BTC | TLB}
cyclenum DUMP {DATA | INSTR | ROM} *low* *high*

where:

cyclenum	Specifies when the dump operation is to take place.
SEGINFO	Specifies that segment information from the loaded program is to be dumped to the output file.
REG	Specifies that the register file contents be dumped.
BTC	Specifies that the branch-target cache be dumped. This option only applies to the Am29000 and Am29050 microprocessors.
TLB	Specifies that the Translation Lookaside Buffer registers be dumped to the output file.
DATA	Specifies that data memory from *low* to *high* addresses be dumped to the output file.
INSTR	Specifies that instruction RAM memory from *low* to *high* addresses be dumped.
ROM	Specifies that the contents of ROM from *low* to *high* addresses be dumped.
low	Specifies the beginning address, in hexadecimal, of a section of memory.
high	Specifies the ending address, in hexadecimal, of a section of memory.

The **DUMP** command provides the means to write the contents of specified processor or memory resources, in printable form, to the simulator output file.

The **SEGINFO** dump includes a listing of each instruction ROM or RAM segment, and also each data RAM segment, including its beginning and ending address in the program's address space.

Example

```
80000 DUMP SEGINFO
```

This example illustrates dumping the segment information from the loaded file when the simulator's cycle count reaches 80000. An example of the information provided by this command is shown below:

```
80000 DUMP SEGINFO
Instr ROM Segments:
   start_addr=00000000 end_addr=00005253
Instr RAM Segments:
   start_addr=00008000 end_addr=0000C2B3
Data RAM Segments:
   start_addr=800033A0 end_addr=8000B39F
   start_addr=80003000 end_addr=8000339B
   start_addr=80002C00 end_addr=80002C33
   start_addr=80002400 end_addr=800029DF
   start_addr=80002000 end_addr=8000211B
   start_addr=FFFEF800 end_addr=FFFF77FF
   start_addr=FFFF8000 end_addr=FFFFFFFF
   start_addr=00000000 end_addr=000003FF
   start_addr=00005258 end_addr=00005543
```

This output shows the instruction ROM and RAM segment information, as well as multiple data RAM segments.

Example

```
80020 DUMP DATA 0 24
```

This example illustrates dumping the contents of data RAM memory locations 0–24 (hexadecimal). The output from executing this command (at simulator cycle 80020) is shown below:

```
T=80020, DATA Memory Dump ==>
      adr=000000: 00000076
      adr=000004: 00000BB6
      adr=000008: 00000086
      adr=00000C: 0000008E
      adr=000010: 00000096
      adr=000014: 000012E6
      adr=000018: 000000A6
      adr=00001C: 000000AE
      adr=000020: 000000B6
      adr=000024: 000000BE
```

Each address within the specified range, and its contents, is dumped to the simulator output file.

LOG — Enable Log File

Syntax: *cyclenum* LOG { ON | OFF } { SIP | FPU | CHANNEL }

where:

cyclenum	Specifies when logging is to take place.
ON | OFF	Specifies enabling or disabling logging to the corresponding "log file." Log files are: **sip.log** Contains SIP transactions. **fpu.log** Contains floating-point operations. **channel.log** Contains channel transactions.
SIP	Specifies processor transactions, depending on the selection of the processor type on the command line.
FPU	Specifies floating-point transactions for an Am29050 processor, when the **–29050** command-line option is specified.
CHANNEL	Specifies channel transactions.

The **LOG** command provides the ability to generate output from a particular element of the simulation into a separate file. When the log feature is enabled, each element (**SIP**, **FPU**, and **CHANNEL**) will output transaction information to its corresponding file. **SIP** output goes to **sip.log**, **FPU** output goes to **fpu.log**, and **CHANNEL** output goes to **channel.log**.

Example

`1600 LOG ON SIP`

This example illustrates logging **SIP** (processor) transactions to the **sip.log** file. When the pipeline stalls, no log is produced for that cycle. An example of the output contained in this file is shown below:

```
CYCLE #  PC       INST
1600 00010354 02808300 CONSTH    LR3, 0x80000000  SRA=0000302C  RESULT=8000302C
1601 00010610 25010188 SUB       GR1, GR1, 136  SRA=FFFDF778  RESULT=FFFDF6F0
1602 00010614 5E40017E ASGEU     64, GR1, GR126  SRA=FFFDF6F0  SRB=FFFDF5B8
1603 00010618 257D7D50 SUB       GR125, GR125, 80  SRA=FFFFFFA8  RESULT=FFFFFF58
1604 0001061C 15957D4C ADD       LR21, GR125, 76  SRA=FFFFFF58  RESULT=FFFFFFA4
1605 00010620 1E00A795 STORE     0, LR39, LR21  SRA=FFFFFFAC  SRB=FFFFFFA4
1606 00010624 1583A600 ADD       LR3, LR38, 0  SRA=800033A0  RESULT=800033A0
1608 00010628 158101A0 ADD       LR1, GR1, 160  SRA=FFFDF6F0  RESULT=FFFDF790
1609 0001062C 16018883 LOAD      1, LR8, LR3 SRA=6B49E27C SRB=800033A0 RESULT=FFFDF790
1610 00010630 03008900 CONST     LR9, 0x0  SRA=6B49E27C  RESULT=00000000
1612 00010634 03009100 CONST     LR17, 0x0  SRA=6B49E27C  RESULT=00000000
```

Each line in the **sip.log** file contains the time, program counter, instruction, a symbolic disassembly of the instruction, and contents of the source registers SRA and SRB.

Example

`+100 log on channel`

This example illustrates the output from the channel log. Each line provides the following: the cycle time, the type of operation (Instruction/Data), IA/DA indicating beginning of access operation or IR/DR/DW indicating instruction fetch completion or data read/write completion, the address latched by the memory system, and other pin values.

```
CYCLE #                      OPERATION INFORMATION
3084    I_Slave IA Iadr=0001038C Ireq=0 sup_us=0 mpgm=0 pia_=1 pen_=1 Ireqt=0
3084    I_Slave IR Iadr=0001038C Instr=15857D04 Ireq=0 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3085    I_Slave IR Iadr=00010390 Instr=02008201 Ireq=1 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3086    I_Slave IR Iadr=00010394 Instr=A8008079 Ireq=1 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3087    D_Slave DA Dadr=FFFDF778 Dreq_=0 sup_us=0 mpgm=0 pda_=1 pen_=1 opt=0 rw=1
3087    I_Slave IR Iadr=00010398 Instr=02808300 Ireq=1 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3087    D_Slave DR Dadr=FFFDF778 Data_r=80004630 Dbreq=1 Drdy_=0 Derr_=1 pda_=1 rw=1
3088    I_Slave IR Iadr=0001039C Instr=15010118 Ireq=1 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3089    I_Slave IR Iadr=000103A0 Instr=157D7D40 Ireq=1 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3090    I_Slave IR Iadr=000103A4 Instr=C0000080 Ireq=1 Ibreq=1 Irdy_=0 Ierr_=1 pia_=1
3091    I_Slave IA Iadr=00010578 Ireq=0 sup_us=0 mpgm=0 pia_=1 pen_=1 Ireqt=0
3091    I_Slave IR Iadr=00010578 Instr=25010180 Ireq=0 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3092    I_Slave IR Iadr=0001057C Instr=5E40017E Ireq=1 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3093    I_Slave IR Iadr=00010580 Instr=257D7D50 Ireq=1 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3094    I_Slave IR Iadr=00010584 Instr=15937D4C Ireq=1 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3095    I_Slave IR Iadr=00010588 Instr=1E00A593 Ireq=1 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3096    I_Slave IR Iadr=0001058C Instr=15810198 Ireq=1 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
3097    I_Slave IR Iadr=00010590 Instr=03008900 Ireq=1 Ibreq=0 Irdy_=0 Ierr_=1 pia_=1
```

Example

`40 log on fpu`

This example illustrates logging FPU of the Am29050 microprocessor. This log gives the instruction in each of the pipes and stages of the Am29050 microprocessor. The different stages and pipes are the integer pipe (INT), add pipe stage 1 (A1), add pipe stage 2 (A2), left shift pipe (LS), multiply pipe stage 1 (M1), multiply pipe stage 2 (M2), divide pipe (DIVIDE), and the rounder stage (ROUNDER). If a pipe is stalled, a character **C** for contention or **R** for resource is put in the parentheses next to each instruction.

```
CYCLE #     PC     INT     A1     A2     LS     M1     M2     DIVIDE     ROUNDER
47     00010050 f45a6161                  f45a6161( )             f6606063( )
48     00010054 f45b6161                  f45b6161( ) f45a6161( ) f6606063( )
49     00010058 f6616163                  f45b6161(R) f45a6161(R) f6616163( ) f6606063
50     0001005c 70400101                       f45b6161( ) f6616163( ) f45a6161
51     00010060 70400101                             f6616163( ) f45b6161
52     00010064 70400101                             f6616163( )
53     00010068 70400101                             f6616163( )
54     0001006c 70400101                             f6616163( )
55     00010070 70400101                             f6616163( )
56     00010074 70400101                             f6616163( )
57     00010078 e4826049 e4826049( )                        f6616163( )
58     0001007c a8008009       e4826049( )                  f6616163( )
59     0001007c a8008009             e4826049(R)                   f6616163
60     00010080 e4846149 e4846149( )                               e4826049
61     000100a0 c0000080             e4846149( )
62     000100a0 c0000080                                       e4846149
```

ONERROR — Specify Error Action

Syntax: *cyclenum* ONERROR { STOP | IGNORE } [*error_code*]

where:

cyclenum Specifies the time at which the simulator is to enable the error action selection.

STOP Selects stopping when an error corresponding to one of the *codes* occurs.

IGNORE Specifies that errors corresponding to the indicated *codes* are to be ignored.

error_code An optional list of codes that enable the detection of various errors in the simulation process. When specifying codes, each must be entered as an uppercase letter, and multiple codes must not be separated by spaces or tabs. Possible values for *error_code* are:

A Address error occurred (e.g., out of bounds).
K Kernel error occurred (i.e., an error in supervisor mode).
O Illegal opcode error occurred.
F An arithmetic trap occurred (e.g., divide by zero).
P A protection violation occurred.
S An error in the event file occurred.

If the *codes* parameter of the **STOP** or **IGNORE** command is missing, then all the error codes are assumed to be enabled for this event, i.e., leaving the *codes* parameter blank is the same as specifying **AKOFPS**.

Example

```
50000  ONERROR  STOP  AOP
```

In the example above, when (and if) the simulator cycle count reaches 50000, the simulator will enable detection of address errors (**A**), illegal opcodes (**O**), and protection violations (**P**). If any of these occur on or after the specified time (50000), the simulator will stop executing the program.

Example

```
50000  ONERROR  IGNORE  AO
```

This example is similar to the previous one. When (and if) the simulator cycle count reaches 50000, the simulator will ignore detection of address errors (**A**), and illegal opcodes (**O**). If any of these occur on or after the specified time (50000), the simulator will continue operation, as if the error did not occur.

SET — Set Simulator Configuration

Syntax:

0 SET MEM *access* TO *value*
0 SET PAGEHEIGHT TO *lines*
0 SET SHARED_ID_BUS

where:

0
: Specifies that these events are only valid if executed at *cyclenum*=0 on the simulator's clock. If any other time value is given, the event is ignored, causing an error message.

MEM
: Indicates that the parameters are for the memory system.

access
: Specifies the speed and modes of various types of bus slaves, in response to requests in their corresponding address spaces. Legal access codes for different microprocessors are listed in Table 3-4–Table 3-6.

value
: Specifies a duration or True/False value for an *access* parameter (see Table 3-4–Table 3-6). Non-Boolean values represent clock cycles required to complete the corresponding access.

PAGEHEIGHT
: Controls pagination of the simulator output. The default value is 59, but it can be set to any other convenient value. When the number of lines on a page reaches the specified value, the simulator outputs a form-feed (FF) character, followed by a line that includes the current title and page number. If the **PAGEHEIGHT** is set to 0, then pagination, as described above, does not take place, and the simulator output file contents will be contiguous, with no intervening form feed or title information.

lines
: Specifies the number of lines per page to set for the **PAGEHEIGHT** parameter.

SHARED_ID_BUS
: Indicates that the instruction and data buses, and instruction RAM and data RAM are shared in the simulated system.

Table 3-4. IRAM Memory Model Access Codes and Values for Three-Bus Microprocessor Simulation

Access Code	Value	Default	Description
IDECODE	0–n	0	Cycles to decode instruction RAM (IRAM) address
IACCESS	1–n	1	Cycles for first access of IRAM
IBACCESS	1–n	1	Cycles for burst access of IRAM
IPFACCESS	1–n	1	Cycles for first access of IRAM in page mode
IPSACCESS	1–n	1	Cycles for secondary access of IRAM within the page
IPBACCESS	1–n	1	Cycles for burst access of IRAM within page
ISACCESS	1–n	1	Cycles for access of IRAM within the static column
IPRECHARGE	0–n	0	Cycles for IRAM precharge on page crossing
IPGSIZE	1–n	256	IRAM page size in words
ISMASK	0x00000000–0xFFFFFFFC	0xFFFFFF00	IRAM static column address mask, defaults to 64 words
IBURST	Boolean	False	Specifies whether IRAM is burstable
IPIPE	Boolean	False	Specifies whether IRAM is capable of pipelining
IPAGEMODE	Boolean	False	Specifies whether IRAM is a paged memory
ISTATCOL	Boolean	False	Specifies whether IRAM is a static column memory
IBANKSTART	0–n	N/A	Specifies the starting address of an IRAM memory region for different memory timings
IBANKSIZE	1–n	1	Specifies the size in bytes of an IRAM memory region for different memory timings

Table 3-5. IROM Memory Model Access Codes and Values for Three-Bus Microprocessor Simulation

Access Code	Value	Default	Description
RDECODE	0–n	0	Cycles to decode instruction ROM (IROM) address
RACCESS	1–n	1	Cycles for first access of IROM
RBACCESS	1–n	1	Cycles for burst access of IROM
RPFACCESS	1–n	1	Cycles for first access of IROM in page mode
RPSACCESS	1–n	1	Cycles for secondary access of IROM within the page
RPBACCESS	1–n	1	Cycles for burst access of IROM within page
RSACCESS	1–n	1	Cycles for access of IROM within the static column
RPRECHARGE	0–n	0	Cycles for IROM precharge on page crossing
RPGSIZE	1–n	256	IROM page size in words
RSMASK	0x00000000–0xFFFFFFFC	0xFFFFFF00	IROM static column address mask, defaults to 64 words
RBURST	Boolean	False	Specifies whether IROM is burstable
RPIPE	Boolean	False	Specifies whether IROM is capable of pipelining
RPAGEMODE	Boolean	False	Specifies whether IROM is a paged memory
RSTATCOL	Boolean	False	Specifies whether IROM is a static column memory
RBANKSTART	0–n	N/A	Specifies the starting address of a ROM memory region for different memory timings
RBANKSIZE	1–n	1	Specifies the size in bytes of a ROM memory region for different memory timings

Table 3-6. DRAM Memory Model Access Codes and Values for Three-Bus Microprocessor Simulation

Access Code	Value	Default	Description
DDECODE	0–n	0	Cycles to decode data RAM (DRAM) address
DRACCESS	1–n	1	Cycles for first read access of DRAM
DWACCESS	1–n	1	Cycles for first write access of DRAM
DBRACCESS	1–n	1	Cycles for burst read access of DRAM
DBWACCESS	1–n	1	Cycles for burst write access of DRAM
DPFRACCESS	1–n	1	Cycles for first read access of DRAM in page mode
DPFWACCESS	1–n	1	Cycles for first write access of DRAM in page mode
DPSRACCESS	1–n	1	Cycles for secondary read access of DRAM within the page
DPSWACCESS	1–n	1	Cycles for secondary write access of DRAM within the page
DPBRACCESS	1–n	1	Cycles for burst read access of DRAM within page
DPBWACCESS	1–n	1	Cycles for burst write access of DRAM within page
DSRACCESS	1–n	1	Cycles for read access of DRAM within the static column
DSWACCESS	1–n	1	Cycles for write access of DRAM within the static column
DPRECHARGE	0–n	0	Cycles for DRAM precharge on page crossing
DPGSIZE	1–n	256	DRAM page size in words
DSMASK	0x00000000-0xFFFFFFFC	0xFFFFFF00	DRAM static column address mask, defaults to 64 words
DBURST	Boolean	False	Specifies whether DRAM is burstable

Access Code	Value	Default	Description
DPIPE	Boolean	False	Specifies whether DRAM is capable of pipelining
DPAGEMODE	Boolean	False	Specifies whether DRAM is a paged memory
DSTATCOL	Boolean	False	Specifies whether DRAM is a static column memory
DBANKSTART	0–n	N/A	Specifies the starting address of a data memory region for different memory timings
DBANKSIZE	1–n	1	Specifies the size in bytes of a data memory region for different memory timings

Table 3-4–Table 3-6 list all the memory parameters that can be set for the Am29000, Am29005 and Am29050 microprocessors using the event file. The memory timings specified are used for the whole memory region, if the *x***BANKSTART** option has not been specified. If this option has been specified, then all the timings specified are assumed to refer to the region specified until another *x***BANKSTART** option has been specified for that same memory category. The example below shows a sample memory timing event file.

Example

```
0 COM SETTING MEMORY TIMINGS FOR IRAM
0 SET MEM IACCESS to 2
0 SET MEM IBACCESS to 1
0 SET MEM IBURST to TRUE
0 SET MEM IPIPE to FALSE

0 COM SETTING MEMORY TIMINGS FOR IROM
0 SET MEM RACCESS to 2
0 SET MEM RBACCESS to 1
0 SET MEM RBURST to TRUE
0 SET MEM RPIPE to FALSE

0 COM SETTING MEMORY TIMINGS FOR DATA
0 SET MEM DRACCESS to 2
0 SET MEM DWACCESS to 3
0 SET MEM DBRACCESS to 1
0 SET MEM DBWACCESS to 2
0 SET MEM DBURST to TRUE
```

```
0 COM SETTING TIMING FOR A IROM REGION 0–0X3FF
0 SET MEM RBANKSTART to 0
0 SET MEM RBANKSIZE to 1024
0 SET MEM RBURST to FALSE

0 COM SETTING TIMING FOR A DATA REGION 0X80004000–0X800043FF
0 SET MEM DBANKSTART to 0X80004000
0 SET MEM DBANKSIZE to 1024
0 SET MEM DRACCESS to 3
0 SET MEM DWACCESS to 4
0 SET MEM DBURST to FALSE

0 SET MEM RACCESS to 3
```

In the above example, the memory access parameters are set for the whole region, then ROM and DATA access parameters are set for a region. The last command to set the IROM read access time will set the read access time for the ROM region 0–0x3FF.

Memory Timing Parameters

The memory model does not support pipelined access in the paged memory model or static column memory model. The order of precedence for simulating different memory modes is paged mode, static column, and pipelined memory.

Figure 3-1 through Figure 3-3 show how the memory timing parameters are defined for the Am29000, Am29005, and Am29050 microprocessors. Decode is the number of cycles that can be overlapped with a pending primary access.

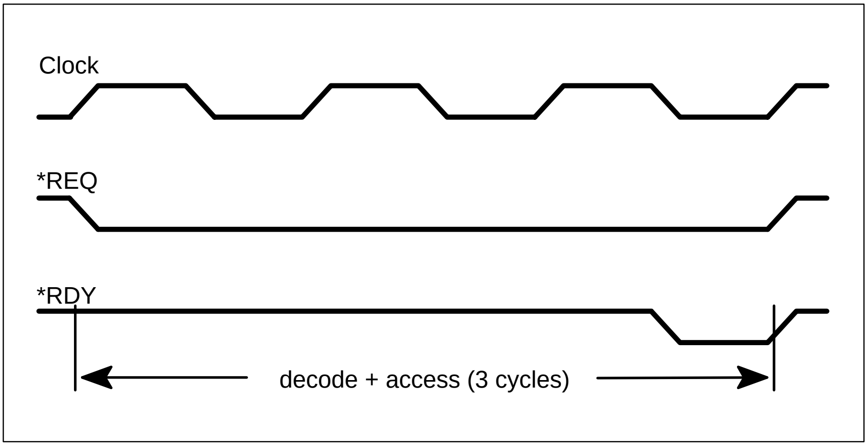


Figure 3-1. Simple Access in Three-Bus Microprocessors

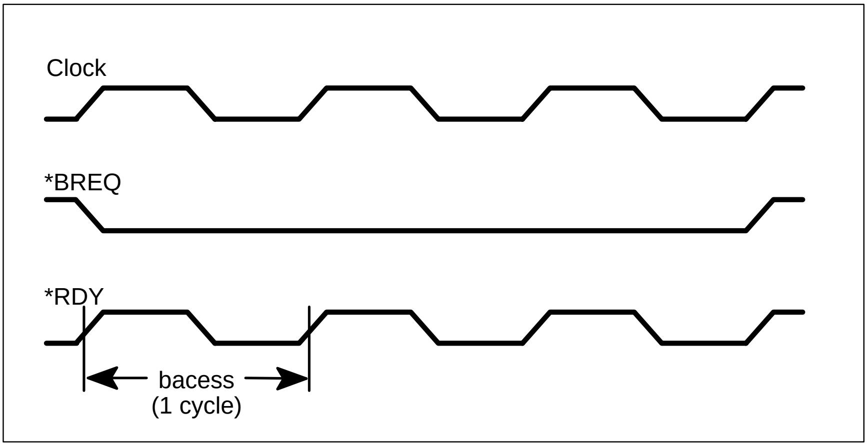


Figure 3-2. Burst Access in Three-Bus Microprocessors

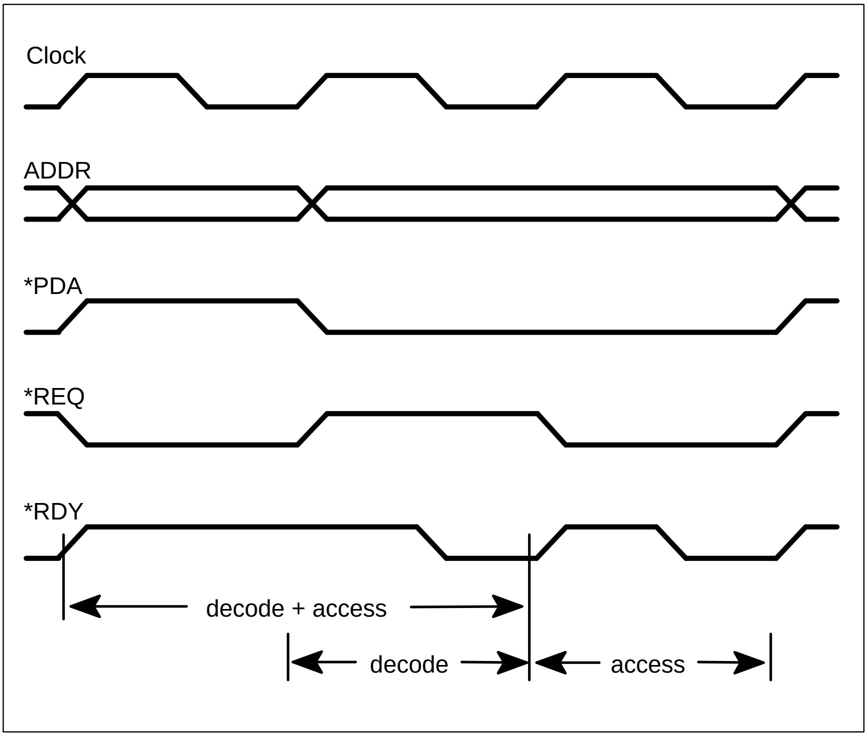


Figure 3-3. Pipelined Access in Three-Bus Microprocessors

Figure 3-4 shows the first access on a page crossing with a precharge time of one cycle and access time of two cycles.

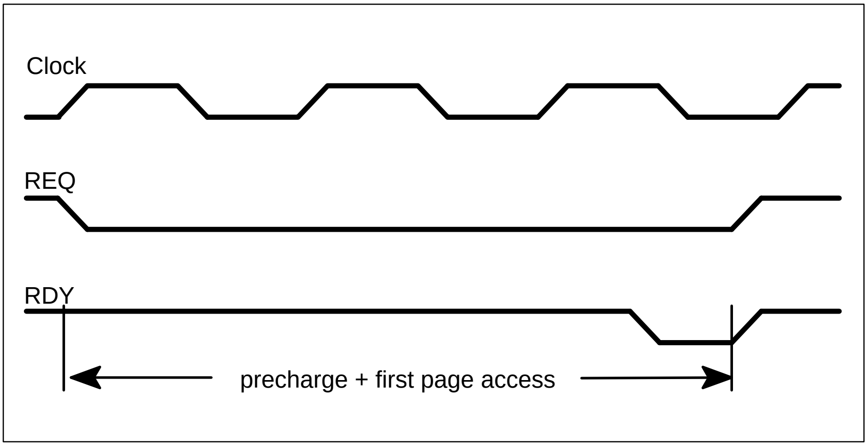


Figure 3-4. First Access on Page Crossing in Paged Memory in Three-Bus Microprocessors

Figure 3-5 shows secondary access within the page with an access time of two cycles.

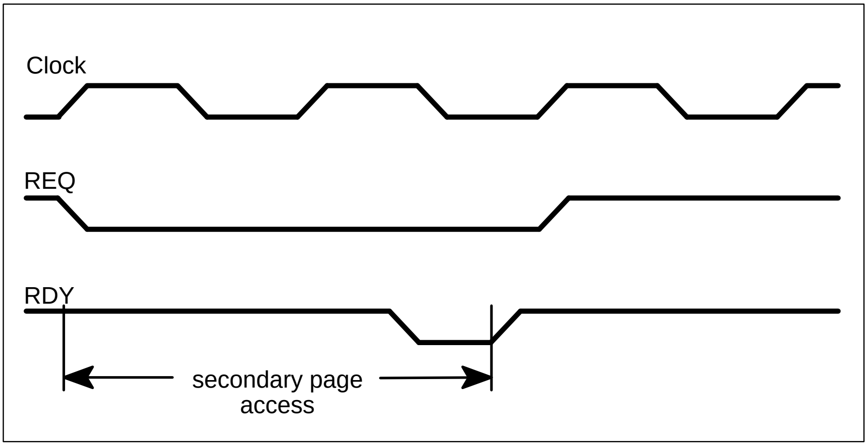


Figure 3-5. Secondary Access Within the Page in Three-Bus Microprocessors

Figure 3-6 shows the first and secondary access within a static column with a decode time of one cycle, a simple access time of two cycles, and access within the static column of two cycles.

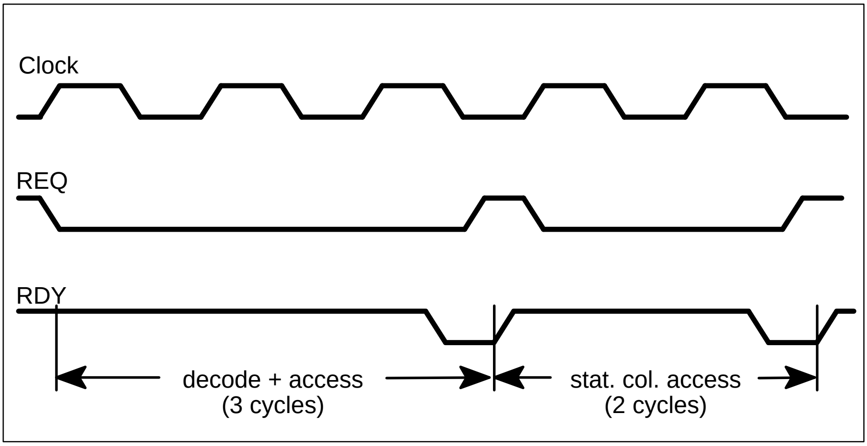


Figure 3-6. Static Column Access in Three-Bus Microprocessors

STOP — End Simulation

Syntax: *cyclenum* STOP [*string*]

where:

cyclenum Specifies the time at which the simulation is to be stopped.

string Specifies an optional string of characters, up to an end of line, that is to be written to the simulator output file.

The **STOP** command is used to stop the simulation at the designated time, and write the command and *string* to the output file.

Example

```
200000  STOP  INFINITE LOOP PROTECTION
```

In the example above, when (and if) the simulator cycle count reaches 200000, the **STOP** command will force the execution to terminate, and will write the specified string to the output file. This illustrates an approach for stopping a program that gets into an infinite loop.

TITLE — Set Simulation Title

Syntax: *cyclenum* TITLE *string*

Where:

cyclenum Specifies when the title should be changed.

string Contains the new title information to appear in the simulator output file.

The specified title *string* is written at the top of each page in the simulator output file if the **PAGEHEIGHT** parameter is nonzero.

Example

```
0 TITLE  Simulation of Am29050 Processor X Windows Terminal
```

This example illustrates setting the title to the specified text. Because the time associated with the command is 0, the specified title will appear on the first page of the output file.

Simulator I/O

The simulator input and output is described below, followed by a description of the **sim.out** file and a sample **sim.out** listing.

Input and Output

The simulator is able to provide a variety of output, both to the on-line display and to various files written during a simulation run. In addition, the simulator reads commands and values from various input sources. Figure 3-7 illustrates the input sources and output destinations that are referenced by the simulator.

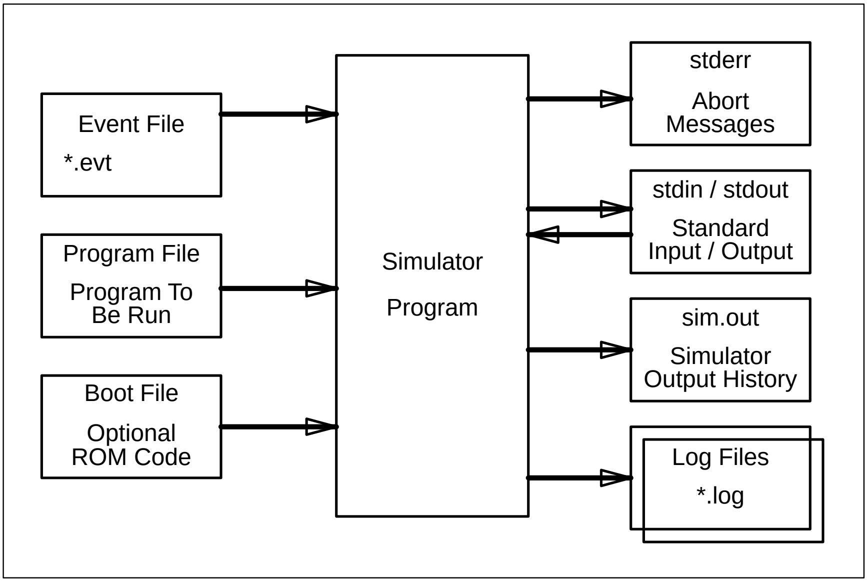


Figure 3-7. Simulator I/O

Table 3-7 describes the command-line options and event file commands that specify the various files and devices on which simulator input/output operations take place.

In Table 3-7, the first column indicates whether a command-line option can be used to select a specific filename; whether an event file command is necessary to enable a particular type of output to one of the dedicated log files; or whether no configuration option is available to enable, disable, or redirect standard interactive input/output transactions.

The simulator output history, which comprises the bulk of the simulator's output, is written to a file called **sim.out**, unless another file pathname is supplied via the **sim29 –o** command-line option.

Table 3-7. Simulator Input/Output File Assignments

Option	File Type	Assignment
–e	Event input file	No default event input file is assumed
–p	Program profile output	Simulator output file (**sim.out**, by default)
–r	Optional ROM code file	Full pathname to ROM code file
–o	Simulator output file	**sim.out** is the default assignment
event	Processor log file	**sip.log** is enabled for processor instruction output by the **LOG** command in the event input file.
event	Channel log file	**channel.log** is enabled for channel transaction output by the **LOG** command in the event input file
event	FPU log file	**fpu.log** is enabled to contain floating-point operations by the **LOG** command in the event input file (specific to the Am29050 processor)
none	Standard input device	stdin is assumed to be the interactive keyboard input device
none	Standard output device	stdout is assumed to be the interactive console output device
none	Standard error device	stderr is assumed to be the interactive console error output device

The **LOG** command can be used to enable or disable individual simulator output logging to previously assigned log files that were specified via event file commands.

The sim.out File

The simulator outputs detailed statistical information about the simulation's performance. The detailed output begins with the command line used to invoke the simulator, followed by the commands used in the event file (if used) and the output from the program being simulated. Following this output is the statistical information of the processor's performance. Sample output from the Am29000 microprocessor simulation of a simple "HELLO WORLD!!" program follows on the next pages.

At the end of the simulation, the registers are dumped (if requested on the command line) followed by the total instructions executed, the number of CPU cycles in user and supervisory mode, and the processor simulation performance in MIPS. This output is followed by the pipeline hold information, which provides information as to why the pipeline was held, such as waiting for an instruction, waiting for data to be accessed, crossing a page, or waiting for LOADM/STOREM. Then the BTC performance is output, giving the number of accesses, hits and partial hits. Partial hits refer to when all the instructions in target block are not valid. Following this output is the MMU performance, instruction bus utilization, data bus utilization, register spill/fill histogram, instruction histogram, and system call histogram. Finally, the memory parameters used in the simulation are output along with the number of accesses from different memory areas.

Sample sim.out Listing

```
AMD SIM050 ARCHITECTURAL SIMULATOR, V# 1.0–20
Cmnd line: sim050 –d –e evt.29k –o hello29k.log hello.29k
-----------------------------------------------------
0 SET PAGEHEIGHT TO 0
-----------------------------------------------------
0 COM SETTING MEMORY TIMINGS FOR IRAM
-----------------------------------------------------
0 SET MEM IACCESS to 2
-----------------------------------------------------
0 SET MEM IBACCESS to 1
-----------------------------------------------------
0 SET MEM IBURST to TRUE
-----------------------------------------------------
0 SET MEM IPIPE to FALSE
-----------------------------------------------------
0 COM SETTING MEMORY TIMINGS FOR IROM
-----------------------------------------------------
0 SET MEM RACCESS to 2
-----------------------------------------------------
0 SET MEM RBACCESS to 1
-----------------------------------------------------
0 SET MEM RBURST to TRUE
-----------------------------------------------------
0 SET MEM RPIPE to FALSE
-----------------------------------------------------
0 COM SETTING MEMORY TIMINGS FOR DATA
-----------------------------------------------------
0 SET MEM DRACCESS to 2
-----------------------------------------------------
0 SET MEM DWACCESS to 3
-----------------------------------------------------
0 SET MEM DBRACCESS to 1
-----------------------------------------------------
0 SET MEM DBWACCESS to 2
-----------------------------------------------------
0 SET MEM DBURST to TRUE
-----------------------------------------------------
0 COM SETTING TIMING FOR A IROM REGION 0–0X3FF
-----------------------------------------------------
0 SET MEM RBANKSTART to 0
-----------------------------------------------------
0 SET MEM RBANKSIZE to 1024
-----------------------------------------------------
0 SET MEM RBURST to FALSE
-----------------------------------------------------
0 COM SETTING TIMING FOR A DATA REGION 0X80004000–0X800043FF
-----------------------------------------------------
0 SET MEM DBANKSTART to 0X80004000
```

Sample sim.out Listing continued

```
-----------------------------------------------------
0 SET MEM DBANKSIZE to 1024
HELLO WORLD!!!!
### T=5997 Am29000 Simulation of ”hello.29k” complete –– successful
termination.
    pc_l2=1378C fet_pc_l2=13788 dec_pc_l2=13784 exe_pc_l2=13780
-----------------------------------------------------
     <<<<< S U M M A R Y   S T A T I S T I C S >>>>>

–––––––––– Register File Contents ––––––––––

gr[01] (Stack Pointer) = ffffff88
gr[40] = 000031a0   gr[41] = 0400a103   gr[42] = 002840c0   gr[43] = 000000c0
gr[44] = 0000097c   gr[45] = 00002027   gr[46] = 00001166   gr[47] = 00000002
gr[48] = 00000000   gr[49] = 00000400   gr[4a] = 00000573   gr[4b] = 80000000
gr[4c] = 000114e8   gr[4d] = 000114e4   gr[4e] = 000114e0   gr[4f] = 00000278
gr[50] = 6b49e27c   gr[51] = 6b49e27c   gr[52] = 6b49e27c   gr[53] = 6b49e27c
gr[54] = 00010004   gr[55] = 6b49e27c   gr[56] = 80004ee0   gr[57] = 6b49e27c
gr[58] = 6b49e27c   gr[59] = 00000000   gr[5a] = 000100e8   gr[5b] = 00010110
gr[5c] = 6b49e27c   gr[5d] = 6b49e27c   gr[5e] = 6b49e27c   gr[5f] = 000fc0c0
gr[60] = ffffffff   gr[61] = ffffffff   gr[62] = 00000002   gr[63] = 00000000
gr[64] = 00001000   gr[65] = 80004eb8   gr[66] = 80004ec4   gr[67] = 80004eb0
gr[68] = 80004ef8   gr[69] = fffffff8   gr[6a] = 00000000   gr[6b] = 6b49e27c
gr[6c] = 6b49e27c   gr[6d] = 6b49e27c   gr[6e] = 6b49e27c   gr[6f] = 6b49e27c
gr[70] = 6b49e27c   gr[71] = 6b49e27c   gr[72] = 6b49e27c   gr[73] = 6b49e27c
gr[74] = 00000000   gr[75] = 00000000   gr[76] = 00000002   gr[77] = 6b49e27c
gr[78] = 6b49e27c   gr[79] = 00000001   gr[7a] = 800049a4   gr[7b] = 6b49e27c
gr[7c] = 6b49e27c   gr[7d] = fffdf7b8   gr[7e] = fffffdb8   gr[7f] = ffffffb8

lr[00] = 00000001   lr[01] = fffdf7f4   lr[02] = 6b49e27c   lr[03] = 6b49e27c
lr[04] = 6b49e27c   lr[05] = 6b49e27c   lr[06] = 6b49e27c   lr[07] = 6b49e27c
lr[08] = 00012ed8   lr[09] = fffffe3c   lr[0a] = 80004f00   lr[0b] = 80004eb0
lr[0c] = 00012f88   lr[0d] = fffffe4c   lr[0e] = 00001000   lr[0f] = 80004ec4
lr[10] = 00013180   lr[11] = fffffe5c   lr[12] = 00000408   lr[13] = 6b49e27c
lr[14] = 000129d4   lr[15] = fffffe78   lr[16] = 00000400   lr[17] = 800044dc
lr[18] = 00012890   lr[19] = fffffe90   lr[1a] = 00000001   lr[1b] = 80004f00
lr[1c] = 00000010   lr[1d] = 800044dc   lr[1e] = 800044d8   lr[1f] = 800044e4
lr[20] = 00011574   lr[21] = fffffed0   lr[22] = 800044d4   lr[23] = ffffffff
lr[24] = 00000010   lr[25] = 00000400   lr[26] = 00000000   lr[27] = 28000000
lr[28] = 80004f10   lr[29] = 800044d8   lr[2a] = 00000010   lr[2b] = 0000008a
lr[2c] = 800044e4   lr[2d] = 6b49e27c   lr[2e] = 000114b0   lr[2f] = fffffee4
lr[30] = 80004a74   lr[31] = 00000001   lr[32] = 00000010   lr[33] = 800044d4
lr[34] = 000103a0   lr[35] = ffffff88   lr[36] = 800044d4   lr[37] = 80004a64
lr[38] = 00000010   lr[39] = ffffffff   lr[3a] = 6b49e27c   lr[3b] = 00000010
lr[3c] = 6b49e27c   lr[3d] = ffffff20   lr[3e] = 7fffffff   lr[3f] = 80004f00
lr[40] = 00000000   lr[41] = 800044dc   lr[42] = 800044d8   lr[43] = 800044e4
lr[44] = 0001369c   lr[45] = ffffff44   lr[46] = 00000000   lr[47] = 00000000
```

```
lr[48] = ffffffff   lr[49] = 800044fa   lr[4a] = 80004018   lr[4b] = 00000000
lr[4c] = 800044f0   lr[4d] = 800044f8   lr[4e] = 00013700   lr[4f] = ffffff54
lr[50] = 800044e8   lr[51] = 6b49e27c   lr[52] = 00013314   lr[53] = ffffff78
lr[54] = 80004894   lr[55] = 800048a8   lr[56] = fffdf7b4   lr[57] = 00000000
lr[58] = fffffffe   lr[59] = 00000000   lr[5a] = 00000004   lr[5b] = 80004e1c
lr[5c] = 00012e04   lr[5d] = ffffff88   lr[5e] = 80004a60   lr[5f] = 800044d4
lr[60] = 00012150   lr[61] = ffffff94   lr[62] = 00010160   lr[63] = ffffffa0
lr[64] = 00000000   lr[65] = 6b49e27c   lr[66] = 000100c8   lr[67] = ffffffb0
lr[68] = 00000001   lr[69] = 80004ee0   lr[6a] = 80004ee8   lr[6b] = 6b49e27c
lr[6c] = 00002660   lr[6d] = ffffffb8   lr[6e] = 6b49e27c   lr[6f] = 6b49e27c
lr[70] = 6b49e27c   lr[71] = 6b49e27c   lr[72] = 6b49e27c   lr[73] = 6b49e27c
lr[74] = 6b49e27c   lr[75] = 6b49e27c   lr[76] = 6b49e27c   lr[77] = 6b49e27c
lr[78] = 6b49e27c   lr[79] = 6b49e27c   lr[7a] = 6b49e27c   lr[7b] = 6b49e27c
lr[7c] = 000023d0   lr[7d] = fffffff4   lr[7e] = 00000002   lr[7f] = 00000002

---------- Special Registers ----------

– Protected –          – Protected –          – Unprotected –
VAB:  00000000         TC:   00fff465         IPC:  00000188
                       TR:   01ffffff         IPA:  00000184
OPS:  0000086c                                IPB:  00000180
CPS:  0000086c         PC0:  00013788
                       PC1:  00013784         Q:    00000002
CFG:  02000030         PC2:  00013780         ALU:  00000278 ( n )

CHA:  fffdf7b4         MMU:  00000301         BP:   00000003
CHD:  80000000         LRU:  00000000         FS:   00000018
CHC:  00008180         RBP:  00000000         CR:   00000000

 CPU Frequency = 25.00MHz

    Nops:122
               total instructions = 4540

User Mode:                 2729 cycles       (0.00010916 seconds)
Supervisor Mode:           3269 cycles       (0.00013076 seconds)
Total:                     5998 cycles       (0.00023992 seconds)

Simulation speed:         18.92 MIPS (1.32 cycles per instruction)

---------- Pipeline ----------
 24.31% idle pipeline:
        15.81% Instruction Fetch Wait
         4.07% Data Transaction Wait
         0.28% Page Boundary Crossing Fetch Wait
         0.07% Unfilled BTCache Fetch Wait
```

```
         0.72% Load/Store Multiple Executing
         2.17% Load/Load Transaction Wait
         1.20% Pipeline Latency

Total Wait:                1458 cycles       (0.00005832 seconds)

---------- Branch Target Cache ----------
Partial hits:               132
Branch btcache access:     2877
Branch btcache hits:       2169
Branch btcache hit ratio:      75.39%

---------- Translation Lookaside Buffer ----------
TLB access:                   0
TLB hits:                     0
TLB hit ratio:          0.00%

---------- Bus Utilization ----------
Inst Bus Utilization:   56.90%
           3413 Instruction Fetches

Data Bus Utilization:   12.07%
            189 Loads
            535 Stores

---------- Register File Spilling/Filling ----------
              0 Spills,        0 Fills

Opcode Histogram
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 ILLEGAL:         CONSTN:19       CONSTH:179       CONST:309
  MTSRIM:14      CONSTHZ:          LOADL:           LOADL:
     CLZ:             CLZ:        EXBYTE:          EXBYTE:
  INBYTE:          INBYTE:        STOREL:          STOREL:
    ADDS:            ADDS:          ADDU:            ADDU:
     ADD:71           ADD:1161      LOAD:178         LOAD:
   ADDCS:           ADDCS:        ADDCU:           ADDCU:
    ADDC:            ADDC:         STORE:515       STORE:
    SUBS:            SUBS:          SUBU:            SUBU:
     SUB:16           SUB:53     LOADSET:         LOADSET:
   SUBCS:           SUBCS:         SUBCU:           SUBCU:
    SUBC:            SUBC:        CPBYTE:13        CPBYTE:
   SUBRS:           SUBRS:         SUBRU:           SUBRU:
    SUBR:            SUBR:         LOADM:1          LOADM:
  SUBRCS:          SUBRCS:        SUBRCU:          SUBRCU:
   SUBRC:           SUBRC:        STOREM:2         STOREM:
    CPLT:1           CPLT:1        CPLTU:6          CPLTU:5
```

```
    CPLE:            CPLE:2         CPLEU:3          CPLEU:4
    CPGT:1           CPGT:          CPGTU:3          CPGTU:
    CPGE:            CPGE:          CPGEU:1          CPGEU:3
    ASLT:            ASLT:          ASLTU:           ASLTU:
    ASLE:            ASLE:          ASLEU:24         ASLEU:
    ASGT:            ASGT:          ASGTU:           ASGTU:
    ASGE:            ASGE:          ASGEU:26         ASGEU:
    CPEQ:9           CPEQ:224       CPNEQ:14         CPNEQ:28
     MUL:30           MUL:1          MULL:1           MULL:
    DIV0:            DIV0:5           DIV:155          DIV:
    DIVL:5           DIVL:         DIVREM:5        DIVREM:
    ASEQ:            ASEQ:          ASNEQ:10         ASNEQ:
    MULU:            MULU:        ILLEGAL:         ILLEGAL:
    INHW:            INHW:        EXTRACT:3        EXTRACT:
    EXHW:            EXHW:          EXHWS:         ILLEGAL:
     SLL:             SLL:63          SRL:1            SRL:15
 ILLEGAL:         ILLEGAL:            SRA:             SRA:4
    IRET:7           HALT:        ILLEGAL:         ILLEGAL:
 IRETINV:1        ILLEGAL:        ILLEGAL:         ILLEGAL:
     AND:7            AND:21           OR:81            OR:7
     XOR:             XOR:           XNOR:            XNOR:
     NOR:             NOR:           NAND:1           NAND:
    ANDN:5           ANDN:31        SETIP:5            INV:1
     JMP:26           JMP:        ILLEGAL:         ILLEGAL:
    JMPF:184         JMPF:        ILLEGAL:         ILLEGAL:
    CALL:40          CALL:            ORN:             ORN:
    JMPT:163         JMPT:        ILLEGAL:         ILLEGAL:
 ILLEGAL:         ILLEGAL:        ILLEGAL:         ILLEGAL:
 JMPFDEC:389      JMPFDEC:          MFTLB:         ILLEGAL:
 ILLEGAL:         ILLEGAL:        ILLEGAL:         ILLEGAL:
 ILLEGAL:         ILLEGAL:          MTTLB:128         XMAC:
    JMPI:43       ILLEGAL:        ILLEGAL:         ILLEGAL:
   JMPFI:11       ILLEGAL:           MFSR:35       ILLEGAL:
   CALLI:8        ILLEGAL:        ILLEGAL:         ILLEGAL:
   JMPTI:7        ILLEGAL:           MTSR:32          XMSM:
    XADD:            XSUB:           XMUL:            XDIV:
     XEQ:             XGT:            XGE:         EMULATE:
    FMAC:            DMAC:           FMSM:            DMSM:
    0xDC:            0xDD:          MULTM:          MULTMU:
MULTIPLY:1         DIVIDE:       MULTIPLU:          DIVIDU:
 CONVERT:            SQRT:          CLASS:            0xE7:
   MTACC:           MFACC:            FEQ:             DEQ:
     FGT:             DGT:            FGE:             DGE:
    FADD:            DADD:           FSUB:            DSUB:
    FMUL:            DMUL:           FDIV:            DDIV:
    0xF8:           FDMUL:           0xFA:            0xFB:
    0xFC:            0xFD:           0xFE:            0xFF:
```

```
System Call Count Histogram
------------------------------------------------------------
    EXIT     1:1          CLOSE    18:3          WRITE    20:1
  IOSTAT    26:1       SYSALLOC   257:1        GETARGS   260:1
  SETVEC   289:2

------ M E M O R Y   S U M M A R Y ------

   Stack_size=00020000
   Heap_size=00008000

       I – S L A V E   S U M M A R Y

 Memory Parameters for Non–banked Regions
   I_SPEED: Idecode=0 Iaccess=2 Ibaccess=1
   I_SPEED: Ipg_size=256 Iprecharge=0
   I_SPEED: Ipfaccess=1 Ipsaccess=1 Ipbaccess=1
   I_SPEED: Istat_col_mask=ffffff00 Isaccess=1
   I_SPEED: Ipipe=false Iburst=true Ipagemode=false Istatcol=false

 Memory Parameters for Non–banked Regions
   R_SPEED: Rdecode=0 Raccess=2 Rbaccess=1
   R_SPEED: Rpg_size=256 Rprecharge=0
   R_SPEED: Rpfaccess=1 Rpsaccess=1 Rpbaccess=1
   R_SPEED: Rstat_col_mask=ffffff00 Rsaccess=1
   R_SPEED: Rpipe=false Rburst=true Rpagemode=false Rstatcol=false

   ROM Bank 0 Memory Parameters
   ROM_RANGE: 00000000–000003ff
   R_SPEED: Rdecode=0 Raccess=1 Rbaccess=1
   R_SPEED: Rpg_size=256 Rprecharge=0
   R_SPEED: Rpfaccess=1 Rpsaccess=1 Rpbaccess=1
   R_SPEED: Rstat_col_mask=ffffff00 Rsaccess=1
   R_SPEED: Rpipe=false Rburst=false Rpagemode=false Rstatcol=false

   STAT'S:  I_resp=2096 R_resp=1699

       D – S L A V E   S U M M A R Y

 Memory Parameters for Non–banked Regions
   D_SPEED: Ddecode=0 Draccess=2 Dwaccess=3 Dbraccess=1 Dbwaccess=2
   D_SPEED: Dpg_size=256 Dprecharge=0
   D_SPEED: Dpfraccess=1 Dpfwaccess=1 Dpsraccess=1 Dpswaccess=1
   D_SPEED: Dpbraccess=1 Dpbwaccess=1
   D_SPEED: Dstat_col_mask=ffffff00 Dsraccess=1 Dswaccess=1
```

Sample sim.out Listing concluded

```
   D_SPEED: Dpipe=false Dburst=true Dpagemode=false Dstatcol=false

   DRAM Bank 0 Memory Parameters
   DRAM_RANGE: 80004000-800043ff
   D_SPEED: Ddecode=0 Draccess=1 Dwaccess=1 Dbraccess=1 Dbwaccess=1
   D_SPEED: Dpg_size=256 Dprecharge=0
   D_SPEED: Dpfraccess=1 Dpfwaccess=1 Dpsraccess=1 Dpswaccess=1
   D_SPEED: Dpbraccess=1 Dpbwaccess=1
   D_SPEED: Dstat_col_mask=ffffff00 Dsraccess=1 Dswaccess=1
   D_SPEED: Dpipe=false Dburst=false Dpagemode=false Dstatcol=false

   STAT'S:  L_resp=189 S_resp=535 fault=0
-----------------------------------------------------
```

Chapter 4

29K Family Two-Bus Processor Architecture Simulation

This chapter describes how to use **sim29** to simulate the Am29030, Am29035, and Am29040 microprocessors. The syntax to invoke **sim29** is described first, followed by a discussion of the event file and its commands.

Some notable features in the Am29030 and Am29035 processor architecture simulator are:

- 8 Kbytes (4 Kbytes for the Am29035 processor) instruction cache with valid bit per block and quad word 0 fetched first reload strategy
- 2x internal clock option
- No integer multiplier
- No data cache

Some notable features in the Am29040 processor architecture simulator are:

- 8 Kbytes instruction cache with improved reload algorithm and 1 valid bit per instruction
- 4 Kbytes data cache and copy back replacement algorithm for less bus traffic
- 2-cycle access time in data cache
- 2-cycle integer multiplier
- 2x internal clock option

Simulator Command-Line Syntax

Syntax: sim29 *processor* [–d] [–dcacheoff] [–dynmem {0|1}] [–e *event_file*] [–f *frequency*] [–h *heap_size*] [–help] [–icacheoff] [–o *output_file*] [–p] [–r *boot_prog*] [–sv] [–u] [–v] [*app_prog* [*prog_args*]]

where:

processor	Is one of: **–29030**, **–29035**, or **–29040**. These options specify which 29K Family two-bus microprocessor to simulate. Note that the *processor* is required and must be the first argument specified.
–d	Dumps the contents of the registers at the end of simulation.
–dcacheoff	Disables the data cache (applies only to the Am29040 processor).
–dynmem {0|1}	Dynamically allocates memory for address references not defined by the application Common Object File Format (COFF) file. 1 enables; 0 disables.
–e *event_file*	Specifies the simulator event input file, which is used for other miscellaneous commands. The *event_file* is a full pathname to the file.
–f *frequency*	Specifies the CPU frequency in MHz. The default values are 33 for the Am29030 and Am29035 processors, and 50 for the Am29040 processor.
–h *heap_size*	Specifies the size of the system heap (in kilobytes). The *heap_size* parameter is a decimal value. The default system heap size is 32 Kbytes, or **–h 32**.
–help	Outputs ASCII text to standard output that briefly describes all command-line and event-file options.
–icacheoff	Disables the instruction cache.
–o *output_file*	Specifies the simulation summary file (**sim.out** is the default name). The *output_file* is a full pathname to the file.
–p	Profiles opcode, PC, Load, Store, and trap usage.

–r *boot_prog*	Specifies the **osboot** or ROM program (**osb03x** is the default). The *boot_prog* is a full pathname to the file. By default, the simulator will attempt to load the appropriate boot code needed by the compiler, based on the processor specified on the command line. If the user's application has its boot code linked in, specify that application file as the *boot_prog*.
–sv	Passes a parameter to **osboot** indicating that the application is to run in supervisor mode.
–u	Configures memory wait states and enable caches via application software instead of simulator options.
–v	Passes a parameter to **osboot** which will turn on instruction and data address virtual memory translation.
app_prog	Specifies the filename of the program to be simulated. The *app_prog* parameter is not required if the user's application has boot code linked in, and the **–r** option is used. Otherwise, the *app_prog* must be provided, and is a full pathname to the program object file.
prog_args	Specifies command-line options for the program to be simulated. This argument is optional. Programs need not require command-line options to execute properly.

The output generated by the simulator includes the output generated by the program being simulated and the performance statistics. Some of the performance statistics are: the number of processor cycles simulated; the MIPS (millions of instructions per second); the percentage time the pipeline was held for various reasons, such as instruction fetch wait, data fetch wait, or LOADM/STOREM wait; register spill and fill count; cache hit rates (where applicable); and memory parameters.

Example

```
sim29 –29035 –r myboot –sv my_program
```

This example illustrates the command line entered to simulate in Am29035 processor mode using the ROM file **myboot**. The **–sv** option causes the program to run in supervisor mode.

The Event File

The event file is a command file used to specify simulation control parameters, most of which are not handled by the command line. These options primarily consist of memory configuration and wait states.

The default event file memory model for the Am29030, Am29035, and Am29040 processor simulators is as defined by their respective user manuals with some optional extensions for simulating precharge and refresh effects in DRAM, which are borrowed from the Am29240 microcontroller simulator. The Am29030, Am29035, and Am29040 processors do not distinguish between ROM and RAM, but they are defined by the simulator so that refresh and precharge effects can be simulated if desired. Refresh and precharge apply only to DRAM, not ROM.

Burst mode is driven for instruction fetching, data cache reloading, and LOADM and STOREM operations. Page mode is driven for any memory access in the same page as the last access, regardless of the number of cycles passed. Note that this is a more aggressive model than that used by the 29K Family microcontrollers, which only drive page mode for consecutive sequential accesses.

Burst mode accesses for the Am29030, Am29035, and Am29040 processors can be single cycle, but page mode accesses require at least 2 cycles.

Precharge and refresh are defaulted to off by the simulator, and only apply to DRAM when used. The precharge count represents the number of cycles at the end of a DRAM access in which the RAS line is held high after the data is available. Because of these potential dead cycles on the memory bus, successive requests may need to be held off the bus. See the timing diagrams for the DRAM controller in the microcontroller user manuals for more information. The PPrecharge parameter represents the same thing but is for page mode accesses.

The refresh rate count represents the number of cycles between the start of refresh periods. During refresh, DRAM is unavailable for memory requests. ROM fetches or accesses are not affected. The refresh rate is programmable in both the processor and the simulator.

For Boolean values, typically 0, 1, TRUE, and FALSE can be entered. 0 is equivalent to FALSE, and 1 is equivalent to TRUE.

Syntax: [[+] *cyclenum*] *command* [; *comment*]

where:

+ Indicates *cyclenum* is relative (to be added) to the previous *cyclenum*.

cyclenum Indicates the cycle time that the command is to happen. If no time is given, it is the same as the previous time given. The simulation starts with time equal to 0.

command Is one of the commands listed below and described in the command syntax that follows:

DELTAMEM *addr*
DELTAREG *reg*
DUMPMEM *addr* [*num*]
DUMPREG *reg*
FREQUENCY *num*
INTCLOCKMULT *num*
LOGGING *bool*
RAMBANK *addr size*
RAMREAD *num*
RAMWRITE *num*
RAMBURST *bool*
RAMBREAD *num*
RAMBWRITE *num*
RAMPAGE *bool*
RAMPREAD *num*
RAMPWRITE *num*
RAMWIDTH *num*
RAMPRECHARGE *num*
RAMPPRECHARGE *num*
RAMREFRATE *num*
ROMBANK *addr size*
ROMREAD *num*
ROMWRITE *num*
ROMBURST *bool*
ROMBREAD *num*
ROMBWRITE *num*
ROMPAGE *bool*
ROMPREAD *num*
ROMPWRITE *num*
ROMWIDTH *num*
STOP

; *comment* Indicates the remainder of the line following the ';' is a comment and is to be ignored.

Command Syntax Options

addr	Is a non-negative integer, assumed to be in hexadecimal.
bool	Can be **0**, **1**, **FALSE**, or **TRUE**.
num	Is a non-negative integer.
reg	Is the absolute general register number.
size	Is a non-negative integer, assumed to be in hexadecimal.
DELTAMEM	Reports any changes in the contents of memory at *addr.*
DELTAREG	Reports any changes in the contents of general register *reg*.
DUMPMEM	Dumps *num* words from (hex) *addr.*
DUMPREG	Dumps the contents of the general register *reg.*
FREQUENCY	Sets the processor frequency to *num* MHz. This option is only valid at time 0, and is overridden by any value specified using the **–sim29 –f** command-line option.
INTCLOCKMULT	*num* specifies the internal clock multiplier value.
LOGGING	Specifies logging instructions to **sip.log**; **1** or **TRUE** sets the option on, **0** or **FALSE** turns it off.
RAMBANK	Specifies address and size of DRAM bank for memory access timings. Subsequent RAM commands will apply to this bank. Memory variables set prior to any **RAMBANK** command apply to all memory not contained by a bank. *addr* and *size* are in hex.
RAMREAD	Specifies *num* cycle counts for simple DRAM read, where *num* can be 1, 2, 3, or 4.
RAMWRITE	Specifies *num* cycle counts for simple DRAM write, where *num* can be 1, 2, 3, or 4.
RAMBURST	Specifies DRAM burst mode; **1** or **TRUE** sets the option on, **0** or **FALSE** turns it off.
RAMBREAD	Specifies *num* cycle counts for burst DRAM read.
RAMBWRITE	Specifies *num* cycle counts for burst DRAM write.
RAMPAGE	Specifies DRAM page mode; **1** or **TRUE** sets the option on, **0** or **FALSE** turns it off.
RAMPREAD	Specifies *num* cycle counts for page mode DRAM read.

RAMPWRITE	Specifies *num* cycle counts for page mode DRAM write.
RAMWIDTH	Specifies *num* bit width of DRAM memory accesses, where *num* can be 8, 16, or 32.
RAMPRECHARGE	Specifies *num* cycle counts for DRAM precharging, where *num* can be 1, 2, 3, or 4.
RAMPPRECHARGE	Specifies *num* cycle counts for DRAM precharging following a page mode access, where *num* can be 1, 2, 3, or 4.
RAMREFRATE	Specifies the DRAM controller refresh rate (0=off), where *num* is the number of cycles between refreshes.
ROMBANK	Specifies the address and size of the ROM bank for memory access timings. Subsequent ROM commands will apply to this bank. Memory variables set prior to any **ROMBANK** command apply to all memory not contained by a bank. *addr* and *size* are in hex.
ROMREAD	Specifies *num* cycle counts for simple ROM read, where *num* can be 1, 2, 3, or 4.
ROMWRITE	Specifies *num* cycle counts for simple ROM write, where *num* can be 1, 2, 3, or 4.
ROMBURST	Specifies ROM burst mode; **1** or **TRUE** sets the option on, **0** or **FALSE** turns it off.
ROMBREAD	Specifies *num* cycle counts for burst ROM read, where *num* can be 1, 2, 3, or 4.
ROMBWRITE	Specifies *num* cycle counts for burst ROM write, where *num* can be 1, 2, 3, or 4.
ROMPAGE	Specifies ROM page mode; **1** or **TRUE** sets the option on, **0** or **FALSE** turns it off.
ROMPREAD	Specifies *num* cycle counts for page mode ROM read, where *num* can be 1, 2, 3, or 4.
ROMPWRITE	Specifies *num* cycle counts for page mode ROM write, where *num* can be 1, 2, 3, or 4.
ROMWIDTH	Specifies *num* bit width of ROM memory accesses, where *num* can be 8, 16, or 32.
STOP	Stops the simulation.

Example

```
romread 3
romwrite 3
romburst 1        ; rom burst mode 3/2
rombread 2
rombwrite 2

0 log 1           ; log 1000 cycles
1000 log 0

stop              ; now stop
```

This example illustrates setting the ROM memory speeds, using logging, and the **STOP** command. The ROM memory speed is set to 3 cycles for the first access and 2 for subsequent read and write burst accesses. The first 1000 cycles executed are logged to **sip.log** and then the simulation is forced to terminate by the **STOP** command.

Chapter 5

29K Family Microcontroller Architecture Simulation

This chapter describes how to use **sim29** to simulate the Am29200, Am29205, Am29240, Am29243, and Am29245 microcontrollers. The syntax to invoke **sim29** is described first, followed by a discussion of the event file and its commands.

Simulator Command-Line Syntax

Syntax: sim29 *processor* [–d] [–dcacheoff] [–dynmem {0|1}] [–e *event_file*] [–f *frequency*] [–h *heap_size*] [–help] [–icacheoff] [–o *output_file*] [–p] [–r *boot_prog*] [–sv] [–u] [–v] [*app_prog* [*args*]]

where:

processor	Is one of: **–29200**, **–29205**, **–29240**, **–29243** or **–29245**. These options specify which 29K Family microcontroller to simulate. Note that the *processor* is required and must be the first argument specified.
–d	Dumps the contents of the registers at the end of simulation.
–dcacheoff	Disables the data cache.
–dynmem {0|1}	Dynamically allocates memory for address references not defined by the application Common Object File Format (COFF) file. 1 enables; 0 disables.
–e *event_file*	Specifies the simulator event input file, which is used for other miscellaneous commands. The *event_file* is a full pathname to the file.
–f *frequency*	Specifies the CPU frequency in MHz. The default values are 20 for the Am29200 and the Am29205 microcontrollers, and 33 for the Am29240, Am29243 and Am29245 microcontrollers.
–h *heap_size*	Specifies the size of the system heap (in kilobytes). The *heap_size* parameter is a decimal value. The default system heap size is 32 Kbytes, or **–h 32**.
–help	Outputs ASCII text to standard output that briefly describes all command-line and event-file options.
–icacheoff	Disables the instruction cache.
–o *output_file*	Specifies the simulation summary file (**sim.out** is the default name). The *output_file* is a full pathname to the file.

–p	Profiles opcode, PC, Load, Store, and trap usage.
–r *boot_prog*	Specifies the **osboot** or ROM program. (Default is **osb20x** for the Am29200 and Am29205 microcontrollers, and **osb24x** for the Am29240, Am29243, and Am29245 microcontrollers.) The *boot_prog* is a full pathname to the file. By default, the simulator will attempt to load the appropriate boot code needed by the compiler, based on the processor specified on the command line. If the user's application has its boot code linked in, specify that application file as the *boot_prog*.
–sv	Passes a parameter to **osboot** indicating that the application is to run in supervisor mode.
–u	Configures memory wait states and enable caches via application software instead of simulator options.
–v	Passes a parameter to **osboot** which will turn on instruction and data address virtual memory translation. This option does not apply to the Am29200 or Am29205 microcontrollers.
app_prog	Specifies the filename of the program to be simulated. The *app_prog* parameter is not required if the user's application has boot code linked in, and the **–r** option is used. Otherwise, the *app_prog* must be provided, and is a full pathname to the program object file.
prog_args	Specifies command-line options for the program to be simulated. This argument is optional. Programs need not have command-line options to execute properly.

The output generated by the simulator includes the output generated by the program being simulated and the performance statistics, such as the number of cycles simulated, the MIPS (millions of instructions per second), and the cache hit ratios.

Example

```
sim29 –29200 hello.lap
```

This example illustrates the command line to simulate the **hello.lap** program in the Am29200 microcontroller. Because the **–r** option is not specified, the default boot file **osb20x** is used.

Example

```
sim29 –29240 –e sim.evt test.lap
```

This example illustrates simulating the execution of program **test.lap** on the Am29240 microcontroller using the default boot file **osb24x** with the memory configuration as specified in the **sim.evt** file.

The Event File

The event file is a command file used to specify simulation control parameters, most of which are not handled by the command line. These options primarily consist of memory configuration and wait states.

The event file memory model for the 29K Family microcontroller simulators is as defined by the microcontroller user manuals. Page mode applies to RAM, and burst mode applies to ROM. Burst and page mode are driven for instruction fetching, data cache reloading, and load and store multiple operations only (i.e., jumps within a page-do-not-drive-page mode).

The DRAM memory speed for the Am29240, Am29243, and Am29245 microcontrollers is 2/1 (simple/page mode); and is 3/2 for the Am29200 and Am29205 microcontrollers. The simulator allows other page mode timings for experimentation but will issue warnings when non-hardware-supported timings are used. ROM wait states are programmable by both processor hardware and the simulator.

The refresh rate count represents the number of cycles between the start of refresh periods. During refresh, DRAM is unavailable for memory requests. ROM fetches or accesses are not affected. The refresh rate is programmable in both the processor and the simulator.

For Boolean values, typically 0, 1, TRUE, and FALSE can be entered. 0 is equivalent to FALSE, and 1 is equivalent to TRUE.

Syntax: [[+] *cyclenum*] *command* [; *comment*]

where:

+ Indicates *cyclenum* is relative (to be added) to the previous *cyclenum*.

cyclenum Indicates the cycle time that the command is to happen. If no time is given, it is the same as the previous time given. The simulation starts with time equal to 0.

command Is one of the commands as listed below and described in the command syntax that follows.

DELTAMEM *addr*
DELTAREG *reg*
DUMPMEM *addr* [*num*]
DUMPREG *reg*
FREQUENCY *num*
INTCLOCKMULT *num*
LOGGING *bool*
RAMBANK *addr size*
RAMREAD *num*
RAMWRITE *num*
RAMPAGE *bool*
RAMPREAD *num*
RAMPWRITE *num*
RAMWIDTH *num*
RAMREFRATE *num*
ROMBANK *addr size*
ROMREAD *num*
ROMWRITE *num*
ROMBURST *bool*
ROMBREAD *num*
ROMWIDTH *num*
SERIALIN {a | b} *fn* [*baud*]
SERIALOUT {a | b} *fn* [*baud*]
PARALLELIN *fn* [*cps*]
PARALLELOUT *fn* [*cps*]
STOP

; *comment* Indicates the remainder of the line following the ';' is a comment and is to be ignored.

where:

addr	Is a non-negative integer, assumed to be in hexadecimal.
baud	Is a non-negative integer.
bool	Can be **0**, **1**, **FALSE**, or **TRUE**.
cps	Is a non-negative integer representing the characters per second.
fn	Is the filename.
num	Is a non-negative integer.
reg	Is the absolute general register number.
size	Is a non-negative integer, assumed to be in hexadecimal.
DELTAMEM	Reports any changes in the contents of memory at *addr.*
DELTAREG	Reports any changes in the contents of general register *reg*.
DUMPMEM	Dumps *num* words from (hex) *addr.*
DUMPREG	Dumps the contents of the general register *reg*.
FREQUENCY	Sets the processor frequency to *num* MHz. This option is only valid at time 0, and is overridden by any value specified using the **sim29 –f** command-line option.
INTCLOCKMULT	*num* specifies the internal clock multiplier value.
LOGGING	Specifies logging instructions to **sip.log**; **1** or **TRUE** sets the option on, **0** or **FALSE** turns it off.
PARALLELIN	Specifies *fn* as the input file for a parallel port with a character per second transfer rate of *cps*.
PARALLELOUT	Specifies *fn* as the output file for a parallel port with a character per second transfer rate of *cps*.
RAMBANK	Specifies address and size of DRAM bank for memory access timings. Subsequent RAM commands will apply to this bank. Memory variables set prior to any **RAMBANK** command apply to all memory not contained by a bank. *addr* and *size* are in hex.
RAMREAD	Specifies *num* cycle counts for simple DRAM read, where *num* can be 1, 2, 3, or 4.
RAMWRITE	Specifies *num* cycle counts for simple DRAM write, where *num* can be 1, 2, 3, or 4.

RAMPAGE	Specifies DRAM page mode; **1** or **TRUE** sets the option on, **0** or **FALSE** turns it off.
RAMPREAD	Specifies *num* cycle counts for page mode DRAM read.
RAMPWRITE	Specifies *num* cycle counts for page mode DRAM write.
RAMWIDTH	Specifies *num* bit width of DRAM memory accesses, where *num* can be 8, 16, or 32.
RAMREFRATE	Specifies the DRAM controller refresh rate (0=off), where *num* is the number of cycles between refreshes.
ROMBANK	Specifies the address and size of the ROM bank for memory access timings. Subsequent ROM commands will apply to this bank. Memory variables set prior to any **ROMBANK** command apply to all memory not contained by a bank. *addr* and *size* are in hex.
ROMREAD	Specifies *num* cycle counts for simple ROM read, where *num* can be 1, 2, 3, or 4.
ROMWRITE	Specifies *num* cycle counts for simple ROM write, where *num* can be 1, 2, 3, or 4.
ROMBURST	Specifies ROM burst mode; **1** or **TRUE** sets the option on, **0** or **FALSE** turns it off.
ROMBREAD	Specifies *num* cycle counts for burst ROM read, where *num* can be 1, 2, 3, or 4.
ROMWIDTH	Specifies *num* bit width of ROM memory accesses, where *num* can be 8, 16, or 32.
SERIALIN	Specifies *fn* as the input file for serial port **a** or **b**, with a baud rate of *baud*.
SERIALOUT	Specifies *fn* as the output file for serial port **a** or **b**, with a baud rate of *baud*.
STOP	Stops the simulation.

Example

```
romread 3
romwrite 3
romburst 1          ; rom burst mode 3/2
rombread 2
rombwrite 2

0 log 1             ; log 1000 cycles
1000 log 0

stop                ; now stop
```

This example illustrates setting the ROM memory speeds, using logging, and the **STOP** command. The ROM memory speed is set to 3 cycles for the first access and 2 for subsequent read and write burst accesses. The first 1000 cycles executed are logged to **sip.log** and then the simulation is forced to terminate by the **STOP** command.

Chapter 6

SIM29 OS Interface

Conceptually, the simulator may be viewed as one of several target platforms and debuggers supported by **osboot**, with several notable exceptions. For the simulator target, there is no “debug core” and associated “message communication” system linked in with the **osboot** code. All debugging and host IO services are provided internally by the simulator, invisible to the **osboot** code.

For HIF IO services, this is done by intercepting those HIF traps requiring host IO, performing the service, altering the appropriate return registers, then continuing on with the simulation. Since no application code is actually executed for these traps, benchmarks that involve heavy IO will not be accurate. Most benchmarking should attempt to exclude the time for IO from the results.

A second difference the simulator has from other targets is in OS initialization. A functional interface is provided for use by most debuggers to inform the OS of the location of an application programs’ text and environment for execution. For the simulator, the functional interface is not used, instead the information is put into several global registers prior to starting the simulation at address 0 of the OS boot code.

Register Initialization

The simulator allocates four memory segments: register stacks; memory stacks; a heap segment; and a default vector table at address 0 for Am29000, Am29005, and Am29050 processor simulation, at address 0xffff0000 for Am29030, Am29035, and Am29040 microprocessor simulation, and at 0x40000000 for Am29200, Am29205, Am29240, Am29243 and Am29245 microcontroller simulation. These four segments are in addition to those loaded from the executable file and ROM code (if used). The global register gr65 points to the register stack segment, which grows down, and global register gr66 points to the scalar stack segment (memory stack), which also grows down. The heap is allocated 32 Kbytes (the default allocation) on a double word boundary immediately after the program is loaded. Table 6-1 lists certain general-purpose registers that are initialized.

Table 6-1. General-Purpose Register Initialization

Register	Description
gr65	Register stack growing down
gr66	Memory stack growing down
gr99	Heap segment growing up
gr100	Program entry point
gr103	Argv pointer
gr104	User/supervisor/translation mode
lr2	Argument count (argc)
lr3	Argument pointer (argv)

Trap Interface

Special-purpose traps are available to support certain features of the high-level programming environments. Traps 0, 2, 4, 6, 7, 12, and 13 are traps that may cause unpredictable results if exit is not enabled on the respective error using the “–x [codes]” option of the command line. Traps 64 and 65 are the stack spill and fill traps, respectively. Trap 69 is the operating-system call trap. The system call numbers that are handled by the simulator internally using the host system are 1, 17–26, 33, 49, 65, 66, and 67. System call 305 for CPU frequency is handled by the simulator, but the other queries are expected to be handled by the trap code. Traps 70 and 251–255 are for internal use only.

Appendix A

Error Messages

This appendix lists and describes the architectural simulator error messages. These error messages primarily apply to the Am29000, Am29005, and Am29050 processor architectural simulator. Error messages are listed in alphabetical order, with explanations where appropriate.

Error Messages

- **ACFMT==0**
  Accumulator Format field in FPE register is set to 0, which is reserved.

- **Attempting to set CFG**
  CFG is being set, even though the **–cfg** command-line option has been used.

- **bad magic number**
  The executable file has an invalid magic number.

- **Can only do SET at t=0**
  Can use the **SET** command in the event file at time 0 only.

- **Can only set frequency at t=0**
  Can use the FREQUENCY command in the event file at time 0 only.

- **Can’t open event file**
  Could not open the event file specified. Either it does not exist or it is read protected.

- **Can’t translate address**
  Could not translate the address to host address space.

- **cca–shift and cca in one instr undefined**
  Illegal usage of CCA and CCA-shift registers.

- **config_l2.CP set**
  The Coprocessor Present (CP) bit is being set in the CFG register. Coprocessor model not implemented.

- **Could not allocate heap space**
  The host could not allocate memory for the heap space.

- **could not translate to host address**
  Address specified is not within the simulated memory space.

- **data exception at data address XXXX**
  Data address is not within the simulated data address space.

Error Messages continued

- **Delta: impossible nonexistent address**
  The address specified in the **DELTA** command in the event file does not exist in the simulated memory.

- **Event file time out of order**
  The time specified in the event file is not in order. The time specified is less than the previous event command.

- **Expected ’to’**
  Missing a **TO** in the event file command (e.g., 0 SET MEM DRACCESS **TO** 2).

- **Illegal address range**
  The address range specified is not within the simulated memory space.

- **illegal opcode**
  An illegal opcode was encountered in the executable instruction memory.

- **illegal precision**
  An illegal precision was specified for the floating-point instruction.

- **Illegal value for IL field in CFG Register, Cache unlocked**
  Any attempt to set an illegal value for the IL field in the CFG register will set the IL field to 3 in the case of Am29035 microprocessor simulation and to 0 in the case of Am29030 microprocessor simulation, which is equivalent to unlocking the Instruction Cache.

- **Infinite wait**
  The processor has been stalled for more than 500 cycles due to instruction fetch wait or data fetch wait.

- **infinite wait – trap**
  The processor has been stalled for more than 500 cycles due to instruction fetch wait or data fetch wait.

- **instruction exception**
  Instruction address is not within the simulated instruction address space.

Error Messages continued

- **load access type (OPT) unknown**
  Unknown OPT option bit setting.

- **Max number of breakpoints set**
  The maximum number (18) of breakpoints have been used.

- **missing delta option**
  DELTA option partially specified in the event file.

- **Missing number**
  Missing a number for the access time or page height.

- **Missing number (or +)**
  Missing a number or the plus sign at the beginning of the event file command.

- **MMU2: TLB protection violation**
  TLB protection violation; proper execution bits have not been set.

- **nonexistent special reg**
  A nonexistent special register is specified in the **DELTA** command in the event file.

- **object file is not executable**
  The executable bit in the COFF executable file has not been set.

- **odd A reg on double**
  An odd register number is specified for source A for a double-precision operation.

- **odd B reg on double**
  An odd register number is specified for source B for a double-precision operation.

- **odd C reg on double**
  An odd register number is specified for destination C for a double-precision operation.

Error Messages continued

- **odd reg on double**
  An odd register number is specified for a double-precision operation.

- **opt>2**
  The OPT bits have value greater than 2 for a write operation.

- **out–of–range trap**
  The out-of-range trap is being taken.

- **reading Cond Accin a FP instr undefined**
  Using CCA as a source for an FP instruction is undefined as per specification.

- **Reg num out of bounds**
  The register number specified is out of bounds.

- **section is not of type regular**
  The section in the executable to be simulated is not a COFF-specified type.

- **set_trap in monitor mode**
  A trap was attempted while in monitor mode.

- **sip.log write error**
  Could not write to **sip.log** file after opening for writing. Possible host system problem such as file system being full.

- **slave.log write error**
  Could not write to channel **slave.log** file after opening for writing. Possible host system problem such as file system being full.

- **store access type (OPT) unknown**
  Invalid OPT bit pattern for a store operation.

- **TLB protection violation [data] at data address XXXX**
  TLB data protection violation at address XXXX.

- **TLB protection violation [instr]**
  TLB instruction protection violation.

- **trap while already in monitor mode**
  A trap was attempted while already in monitor mode.

- **Unable to open ROM object file**
  ROM file specified using the **–r** option does not exist or is read protected.

- **unknown class option**
  Unknown class specified in the **DELTA** command in the event file.

- **unknown delta option**
  An unknown **DELTA** option was specified in the event file.

- **Unknown DUMP option**
  An unknown **dump** option was specified in the event file.

- **Unknown ONERROR option**
  An unknown **ONERROR** option was specified in the event file.

- **Unknown or missing option**
  An unknown or missing option was specified in the event file (e.g., the **STOP** or **IGNORE** is missing in the **ONERROR** command in the event file).

- **unknown OS function call**
  An unknown HIF function call was made.

- **Unknown segment type**
  A segment of unknown type was found when loading the application to be simulated.

- **unknown special reg**
  An unknown special register was specified in the **DELTA** command in the event file.

- **unknown type option**
  An unknown type was specified in the **DELTA** command in the event file.

- **Unrecognized command**
  An unrecognized command was specified in the event file.

Error Messages concluded

- **Unrecognized Error Code**
  An unrecognized error code was specified in the event file for the **ONERROR** option.

- **unrecoverable trap**
  An unrecoverable trap is being taken. The unrecoverable traps are:
  Illegal Opcode
  Out of Range
  Coprocessor Exception
  Instruction Exception
  Data Exception
  TLB Instruction Protection Violation
  TLB Data Protection Violation

- **WM bit set**
  The Wait Mode bit WM in the CPS is being set. This mode is not supported.

- **writing Cond Acc (unshifted) in a FP instr is undefined**
  Using CCA as a destination in an FP instruction is undefined as per specification.

Index

A

B

C

D

E

F

H

I

L

M

N

O

P

Q

R

S

T

U

W