RICE UNIVERSITY

Electrical & Computer Engineering Department

DBSIM

Reference Manual

Version 1.0

June 1993

J. Robert Jump

ECE Dept., Rice University
P.O. Box 1892
Houston, TX 77251-1892
email: jrj@rice.edu
Phone: (713) 527-8101 ext. 3576

This manual describes a program that has not been thoroughly tested and may contain bugs. Suggestions, criticisms, questions, or reports of any problems, errors, or bugs with the manual or the simulator are welcome and encouraged. Please send them to J. R. Jump at the above address.

Credits

DBSIM is derived from an X Windows debugger that was written by Omri Gazitt. It was later modified and named VIEWSIM by David Coakley. David Coakley also wrote a command line version of VIEWSIM called SHSIM. J. R. Jump revised SHSIM to get the currently distributed version of DBSIM.

TABLE OF CONTENTS

1. INTRODUCTION

DBSIM is a debugging utility for use with simulators in the Rice Parallel Processing Testbed (RPPT). It provides the user with a tool to aid in testing simulation programs written for these simulators. It includes capabilities for stepping through programs in several different ways and selectively displaying trace data. While conventional source-level debuggers such as dbx can also be used with the RPPT simulators, they provide low level trace information of the internal simulation routines as well as the user's code. Unless one is familiar with the simulator source code, this information may not be very useful. The trace data provided by DBSIM is customized for the RPPT simulators, so that the interactions of processes and operations in the user's simulation code can be easily traced.

The RPPT currently includes the following four simulators:

YACSIM	- A general-purpose, process-oriented, discrete-event simulator.
NETSIM	- An interconnection network simulator.
MEMSIM	- A cache and shared-address memory simulator.
PARCSIM	- A parallel computer architecture simulator.

NETSIM and MEMSIM are extensions of YACSIM that add specialized capabilities for simulating interconnection networks and memory systems. PARCSIM combines NETSIM and MEMSIM with the capability to simulate multiple processors that communicate through a common address space or by sending messages through an interconnection network.

All of these simulators can either be run under control of DBSIM, or they can be run in stand alone mode. When they are running in stand alone mode, all of the code in the simulator or the user's program that is used to interface to DBSIM is ignored. Therefore, it is not necessary to compile simulation programs differently for use with DBSIM.

1.1. EXECUTION CONTROL

This section describes how a user controls the execution of a simulation program using DBSIM. There are four different execution modes, and the user can switch among them during a program's execution. They effectively provide a single-stepping capability that gives the user the ability to advance a simulation program one step at a time, where the size of the step can be specified in four different ways. In all of these modes, the user initiates the execution of a program which then runs until it meets the condition specified by the execution mode. It then stops and allows the user to examine the trace produced during execution and to change the mode of execution for the next execution phase. There is also a limited breakpoint capability.

1.1.1. Run "until time" Mode

In this mode, the simulation program runs until simulation time reaches a specified time and then returns control to the user. While running under this mode, the program will not continue past the specified stopping time. If a program is stopped at some point in its execution while running under another mode or due to a breakpoint, the user can switch to this mode and then advance the simulation to a specific point in simulation time. The user can then switch back to one of the other execution modes to continue beyond that point. Thus, the "until time" mode provides the user with the ability to advance to any specific point of simulation time in the future from any point in the program. In this mode the time step parameter is an absolute point in time.

1.1.2. Run "for time" Mode

When a program starts or continues executing in this mode, it runs for a specified period of simulation time and then stops and returns control to the user. When execution is continued, the program will start executing at the previous point of interruption and run for another period of simulation time. The period of time for the next execution phase can be changed while the program is suspended. In this mode, the time step parameter is a relative amount of time that is added to the current simulation time to get the next stopping point.

1.1.3. Run for "time changes" Mode

In discrete-event simulation, execution time advances in steps of varying size. If the simulator determines that nothing will happen until some time in the future, it will simply advance the simulation to that time, skipping over the intervening interval in one step. A discrete-event simulator keeps track of the current simulation time in an internal variable, and advances simulation time by incrementing that variable. This execution mode allows the user to specify how many times this variable will change before the simulation stops and returns control to DBSIM.

1.1.4. Run for "event changes" Mode

All activities (events and processes) that are scheduled to occur in the future, are placed on a linked list called the *event list.* The elements in that list are sorted by the times they are scheduled to wake up. It is possible for more than one simulation activity to be scheduled at the same simulation time, and if so, they will be taken off the event list and executed in some sequential order when it is time for them to wake up. However, the simulation time variable will not be incremented until all of the activities scheduled at the current time have been removed from the event list and restarted. The "event changes" mode allows the user to specify how many activities must be removed from the event list before the simulation will complete its current execution phase. This provides a finer resolution of control than the "time changes" mode, since several activities may be removed from the event list without any change in the simulation time variable.

1.1.5. Breakpoints

The user can insert breakpoints in a simulation program's code. They will cause the simulation to stop and return control to DBSIM whenever they are encountered during a simulation. They are implemented as subroutine calls that the user must insert in the program before it is compiled. Therefore, they are not as useful as the breakpoints found in other debuggers, because they can not be changed during a simulation. They can all be deactivated or all activated at any time during a simulation when DBSIM is in control, but they can not be individually controlled, and new ones can not be added. Breakpoints are only active while a simulation is running under the control of DBSIM. If the simulation is run in stand alone mode, the breakpoints are ignored.

1.2. TRACE FILTERING

All of the RPPT simulators produce trace information as they execute. The problem is that the amount of trace information can be excessive, especially during a large simulation. There is a limited method of controlling the amount of this data built into the simulators, but it can only provide a rather coarse-grained ability to filter the trace data presented to the user. Specifically, the user can insert commands in the simulation program that determine the level of tracing that will be in effect. There are currently six levels, with level 0 resulting in no trace data produced and level 5 producing very detailed trace information. As with breakpoints, these trace commands must be inserted into a program before it is compiled.

DBSIM improves on this basic tracing mechanism by providing run time control of the amount and type of trace data displayed. At any point during a simulation when it has stopped and returned to DBSIM, the user can change the level of trace data that will be produced in the future. More importantly, the user can also define a subset of the trace data to be displayed by specifying a filter that will intercept and eliminate all other output data from the displayed trace.

The user creates simulation objects by calling special constructor operations from within the simulation code. Each time this is done, the user has the option of specifying a name for that object. Every trace statement produced by the simulators is tagged with one of these object names. For example, when a process changes its state, a trace statement will be generated with its tag equal the name assigned to the process. Each time a synchronization object such as a semaphore is accessed, a trace statement is generated with its tag set to the name of the object. DBSIM filters enable the user to display only those trace statements whose tags match a specified set of tag names.

Several different filters with different tags can be defined for a simulation. For each of these, the trace output can be sent to a file for viewing later. Alternatively, one filter can be created that displays its trace data on the terminal as the simulation executes. The tags for any of the filters can be changed whenever the simulation is suspended and control has returned to DBSIM.

2. DBSIM COMMANDS

This section gives the detailed information needed to use DBSIM. Specifically, it describes the commands used to initiate and control the execution phase of a simulation. There are three categories of commands, those that specify parameters such as the execution mode or filter tags, those that initiate and control execution of a simulation, and utility commands that provide help information and clear the screen. DBSIM will prompt the user to enter a command whenever the simulation is suspended.

The command interpreter only looks at the first letter of an entered command name. Since all command names start with a unique letter, this is unambiguous. As a result the user only needs to enter the first character of the command name.

2.1. INITIATING DBSIM

To start a DBSIM session, the user types "dbsim" at the terminal. This will start DBSIM and display the DBSIM prompt described below. At this point the user can enter DBSIM commands. The user can also specify the name of a simulation program and its arguments on the command line. If this is done, that program is loaded and initialized before control is given to the user. Therefore the user command to start DBSIM has the following form:

dbsim [*program_name* [*program_argument program_agrument ... program_argument*]]

The program arguments are passed to the simulation program as if it had been started in stand alone mode.

2.2. THE DBSIM PROMPT

The DBSIM prompt provides useful information about the status of a simulation. If no program is loaded, the prompt has the following form:

no program loaded >

If there is a loaded program, the prompt contains additional information in the following form:

"program name" **|** "simulation time" >

The program name is the name of the object file that contains the simulation program. If that program is active, that is, currently executing, the name is followed by an asterisk (*). When the program reaches its end and terminates, the asterisk is removed. Simulation time is the time at which the simulation returned control to DBSIM.

2.3. PARAMETER SETTING COMMANDS

These commands set the parameters that control the execution of a simulation program. They are used to set the execution mode, the step size, the trace level, and the filters to be used in the simulation.

step [size]

Entering this command sets the size of the step used in single-stepping to the value of the argument *size*. This argument must be a number. If *size* is less than or equal to 0, the simulator will treat it as an infinite step size. In this case the simulation will run until it terminates or encounters a breakpoint.

If the size argument is omitted, the current value of the step size is displayed.

mode [name]

This command sets the execution mode to *name*. There are four valid choices for *name* corresponding to the four modes described in Section 1.1 above.

Name	Description
until_time	The simulation runs until the absolute time specified by the current step size.
for_time	The simulation runs for an amount of time specified by the current step size.
time_changes	The simulation runs until time has changed *n* times, where *n* is the current value of the step size converted to an integer.
event_changes	The simulation runs until *n* activities have been processed from the event list, where *n* is the current value of the step size converted to an integer.

It is only necessary to enter the first character of these mode name, as all characters except the first are ignored by the command interpreter.

If *name* is not entered, the command displays the current execution mode.

trace [level]

This command sets the level of tracing to *level*, which must be an integer between 0 and 5. The amount of trace data generated by a simulation varies from none at all when level = 0 to a large amount of very detailed trace information when level = 5. If *level* is not entered, the command displays the current trace level.

break [status]

The break command controls the recognition of breakpoint interrupts. If *status* = "enabled", breakpoints will be accepted; if *status* = "ignored", breakpoints will be ignored. It is only necessary to enter the first letter of the two possibilities for *status*. If *status* is not entered, the command displays the current breakpoint status.

filter [name] [op [tag, ... , tag]]

This command is used to create and modify filters. Filters either collect trace statements in a disk file or display them on the terminal screen. Character strings called tags are associated with a filter and are used to determine which trace statements go to that filter. Whenever a trace statement is generated during a simulation, its tag is compared to all tags assigned to all active filters. Then, a copy of that statement is sent to each filter for which the statement's tag matches at least one of the filter's tags.

Filters are not active unless they have been assigned at least one tag. If all of a filter's tags are deleted, the filter is deactivated. If it is reactivated later, all its previous trace statements will be lost. Filters are activated whenever they are assigned a tag.

If *name* equals the string "stdout", all trace statements sent to that filter will go to the terminal screen; otherwise, a file of that name will be opened and all trace statements for the filter will be written to that file. If *name* is omitted, "stdout" is used.

If all the arguments are omitted, a list of all active filters and their tags is printed at the terminal. If *name* is specified, but all other arguments are omitted, the tags associated with the named filter are printed.

The *op* argument must be one of the symbols in the set {+, -, =}. If *op* is '+', the following tags in the command are added to the filter. If it is '-', the following tags are deleted from the filter. If *op* is '-' and there are no following tags in the command, all of the tags previously associated with the filter will be deleted and the filter deactivated. If *op* is '=', all the tags previously associated with the filter will be replaced by the following tags in the command.

The wildcard symbol * may be used at the beginning and/or end of a tag string.

2.4. EXECUTION CONTROL COMMANDS

load filename [arg, arg, ... , arg]

This command loads and initializes the executable simulation program found in the disk file *filename*. If there are any arguments following filename, they are passed on to the simulation program as its command line arguments. If the load is successful, the prompt will display the name of the loaded file.

Only one simulation can be loaded at a time. DBSIM will not accept the load command if it is currently executing a simulation. Once a simulation has terminated normally, a new one can be loaded. The kill command can also be used to force the termination of a simulation before it reaches it normal termination point.

execute

This command initiates or continues execution of the currently loaded simulation program. If no program is loaded, the command is ignored. DBSIM turns control over to the program, which will run for a period of simulation time determined by the current values for the execution mode, step size, and breakpoint status. Then, control will return to DBSIM and the user will be prompted to enter another command. Entering a carriage return at the prompt is equivalent to entering this command.

reset

This command resets the loaded simulation program. Any command line arguments entered earlier are passed again to the simulation program. It is as if the simulation is terminated and then reloaded with the load command.

kill

Entering this command terminates the currently loaded simulation. This leaves DBSIM in a state with no loaded program.

quit

This command terminates the current simulation and exits DBSIM.

2.5. UTILITY COMMANDS

clear

This command clears the terminal screen.

help [command_name]

If this command is entered with no *command_name*, a summary of all the user commands is displayed on the terminal. If *command_name* is one of the valid commands, a more detailed description of that command is displayed. It is only necessary to enter the first character of the command name.