



Applications Note – 130

Using ModelSim’s JobSpy™ with the Sun™ ONE Grid Engine

September 2003 Revision 1.2

Introduction	2
Compute farms	2
Grid Engine.....	2
JobSpy	2
Grid Engine installation.....	3
Testing the installation.....	3
ModelSim checkpoint and restore	5
Checkpoint file compression.....	6
Checkpoint / restore and the foreign interface.....	6
ModelSim and Grid Engine Integration.....	7
TCP/IP socket connection.....	7
Setting the Grid Engine environment.....	9
Using JobSpy with Grid Engine.....	12
Simulation status commands.....	12
Simulation control commands.....	13
Waveform commands.....	14
Profiler commands	15
Using Profiler and Control commands when running macros.....	16
Security.....	16
Using JobSpy From The ModelSim User Interface.....	17
Running Jobs.....	17
Remote Transcript	18
View Data	18
Integration Kit Contents	19

Introduction

This Application Note describes how to use the Sun™ ONE Grid Engine with ModelSim JobSpy™. Using Grid Engine and JobSpy together, you can build a compute farm that optimises hardware and software utilization.

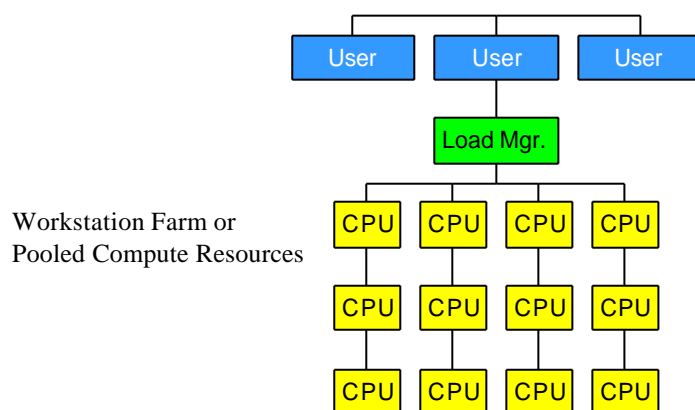
Compute farms

Engineers usually have to search the network for systems with the capacity and correct software to support their jobs. Manual resource hunting wastes valuable engineering time, reduces productivity, and frustrates users. Also, computing resources are alternately overloaded or underutilized.

A compute farm can improve this situation. A compute farm is a network of resources – workstations, servers, and storage arrays – that forms a single entity available to all users. Management software manages the compute resources and provides a single access point to users. Rather than addressing individual systems, engineers submit their jobs to the management software, and it allocates compute farm resources where they are needed most. Engineers' desktop systems, conventional workstations, even thin clients, are left free for other tasks.

Grid Engine

The Sun ONE Grid Engine is distributed management software that optimises hardware and software utilization in heterogeneous networked environments. Grid Engine aggregates the compute power available in dedicated compute farms, networked servers and desktop workstations, and presents a single access point to users needing compute cycles. Grid Engine software distributes computational workload to available systems, simultaneously increasing productivity of machines and application licenses while maximizing the number of jobs that can be completed.



Grid Engine also uncovers hidden compute power on desktop workstations. Workstations are typically underutilized because they are turned off at night and during weekends when they could be running batch jobs. Grid Engine can be configured to access desktops from, say, 6:00 PM to 6:00 AM, Monday through Friday, and all day on weekends. Grid Engine can also sense the load on these desktop systems, and stop or suspend batch jobs if the workstation owner returns.

See <http://www.sun.com/software/gridware/details.html> for more detailed information on Grid Engine.

JobSpy

The Grid Engine integration with JobSpy enables users to distribute ModelSim simulation jobs between resources. JobSpy allows you to examine simulation time, take a snapshot of a simulation waveform for viewing off-line, or examine signals of submitted batch jobs. In addition, the ModelSim checkpoint/restore feature lets you save results at periodic checkpoints, ensuring that a machine crash will not jeopardize all of your results.

Grid Engine installation

A summary of the Grid Engine installation procedure can be found at the following link:

http://supportforum.sun.com/gridengine/appnote_install.html

If you have prior experience in system administration, you should find the installation summary fairly easy to follow. There is little help, however, for setting up a common NFS directory. The following steps will help you set up the NFS directory.

1. Make sure that **nfs** and **rpc** are running on both the server and client. To check this you need to run **rpcinfo**:

Linux : `/usr/sbin/rpcinfo -p`
Solaris : `/usr/bin/rpcinfo -p`

2. Look at the output and make sure that **nfs** and **mountd** are running on Solaris and on Linux nfs, mountd and portmapper. If they are not, they should be started with the following commands.

Linux : `/etc/rc.d/init.d/nfs stop`
`/etc/rc.d/init.d/nfs start`
Solaris: `/etc/init.d/nfs.server stop`
`/etc/init.d/nfs.server start`

3. To export a directory that can be shared by multiple machines on the network, you have to carry out the following on the server machine. First make a directory to export (i.e., `/gridware/codine`). Then do the following:

Linux: Modify the `/etc/exports` file to include the following line or something similar.
`/gridware/codine*(rw)`
Solaris: Use the **share** command or put the line in the `/etc/dfs/dfstab`
`Share -F nfs -o rw /gridware/codine`

4. To mount the directory on each of the machines, you can do the following (in this example the hostname is `grommit`):

Linux: Modify the `/etc/fstab` file to include the following line.
`grommit:/gridware/codine/gridware/codine nfs xec,dev,suid,rw,hard,intr 0 0`
To mount, use the command: `mount grommit:/gridware/codine /gridware/codine`
Solaris: `mount -r -o nosuid grommit:/gridware/codine /gridware/codine`
`mount -r -o remount.rw grommit:/gridware/codine /gridware/codine`

Testing the installation

1. Once the installation is complete, check to make sure that jobs can be submitted and run by Grid Engine. There are sample examples within the installation to help with this process.

If you can run the samples then you should make sure that ModelSim jobs can be submitted and run by Grid Engine. Firstly ensure your environment is setup to run ModelSim in the normal way, the `PATH` and `LM_LICENSE_FILE` environmental variables should be set.

2. Test the installation by running a job. The most common command used to submit jobs is **qsub**. **qsub** can only run scripts, therefore your simulation command must be entered in a script that will be run by **qsub**. In the Sample Script below the top-level design is `DELTA_RTL` from the directory `delta`.

Sample Script: `run.sh`

```
#!/bin/sh
vsim -c DELTA_RTL -do "run -all;quit"
```

This job is run with the following command:

```
[codadmin@grommit delta]$ qsub -cwd -V run.sh
```

The **-cwd** option forces the job to run from the current working directory. The **-V** option ensures that all environmental variables are taken when the job is started on its execution host.

You can check the status of the job using the **qstat** command. For example, if you enter:

```
[codadmin@grommit delta]$ qstat
```

then gridware returns the following:

```
job-ID prior name    user    state submit/start at   queue    master ja-task-ID
-----
114     0 run.sh    codadmin t    08/23/2002 11:49:14 grommit.q MASTER
```

This shows that the job has been started on the *grommit.q* queue. The standard output goes to the file *run.sh.o114*. The standard error, if problems occur, goes to the file *run.sh.e114*. Once gridware is running jobs correctly, you can set up the environment for checkpoint and restore. Prior to discussing the environment setup, however, the next section gives an overview of the ModelSim checkpoint/restore feature.

ModelSim checkpoint and restore

The ModelSim checkpoint/restore feature enables you to save and restore a simulation to a previous state. The **checkpoint** command saves the current state of the simulation in a checkpoint file. The syntax of the command is:

```
checkpoint <filename>
```

You issue the **checkpoint** command while a simulation is loaded. To stop checkpointing, issue the **break** command.

Checkpoint files are machine dependant. They contain the following:

- simulation kernel state
- *vsim.wlf* file
- signals listed in the list and wave windows
- file pointer positions for files opened under VHDL
- file pointer positions for files opened by the Verilog **\$fopen** system task
- state of foreign architectures

Consequently, checkpoint files can be restored only on the same type of workstation (i.e., same executable) that created the checkpoint.

The **restore** command restores a checkpoint file during the same simulation session in which the file was saved. This is called a “warm restore.” The syntax of the command is:

```
restore <filename>
```

ModelSim also has the ability to do a “cold restore,” which restores a checkpoint file created in a different simulation session. A cold restore is performed using the **-restore** switch to **vsim**. The syntax of the command is:

```
vsim -restore <filename>
```

The same switches as the original invocation can be used with the addition of the **-store <filename>** option.

There are certain things that are not saved in a checkpoint file. These include:

- the state of vsim macros
- changes made with the command-line interface (such as user-defined Tcl commands)
- the state of graphical user interface windows

Checkpoint file compression

By default, the **checkpoint** command compresses the checkpoint file in order to reduce its size. While this saves space, it also requires extra time due to the compress/decompress function. For most simulations, this default compress mode is recommended. However, if speed outweighs space in your case, you can disable compression using either of the following methods:

1. Set a special Tcl variable. Use **set CheckpointCompressMode 0** to turn compression off. Turn compression back on with **set CheckpointCompressMode 1**.
2. Control checkpoint compression using the *modelsim.ini* file in the VSIM section. Use the same 0 or 1 switch with **CheckpointCompressMode** :

```
CheckpointCompressMode = <switch>
```

In ModelSim versions 5.7 and earlier, you must use the **-nocompress** switch to **vsim** when cold restoring an uncompressed checkpoint file (i.e., `vsim -restore <filename> -nocompress`). Versions 5.8 and later do not have this requirement.

Checkpoint / restore and the foreign interface

If you use the foreign interface, you will need to add additional function calls in order to use the checkpoint/restore feature. Details can be found in the ModelSim SE User's Manual titled "Using checkpoint/restore with the FLI."

ModelSim and Grid Engine Integration

Grid Engine, like ModelSim, has the ability to checkpoint the execution of processes and restart them at a later time. For example if a job has been submitted to a queue that has been defined to run only between 9:00pm and 7:00am, Grid Engine can give a checkpoint command to stop the job just before 7:00am and save its current state. Grid Engine will then restart the job from the checkpoint at 9:00pm the following evening.

Grid Engine can also be set up to checkpoint a job periodically. For example, if a simulation takes 10 hours to run, a checkpoint command can be setup from Grid Engine to checkpoint the simulation every 2 hours. If the machine has a failure at 7 hours, Grid Engine can re-start the simulation on a similar machine using the file from the 6-hour checkpoint. One hour of simulation time is lost but that is certainly better than 7!

Since ModelSim has its own checkpoint facility, integration with Grid Engine is achieved without having to link any checkpoint libraries into the vsim executable. This integration is handled via ModelSim's TCL scripting language and the configurability of Grid Engine.

TCP/IP socket connection

The communication between Grid Engine and ModelSim is achieved with a TCP/IP socket connection. When ModelSim is invoked, using either the GUI or the batch mode, a server socket is created so that a remote client can control ModelSim during simulation. A TCL script file with the name *RemoteModelsim.tc_* has been created. This script opens the socket and "listens" for one of two commands – **stop** or **check**. The **stop** command stops the simulation at its current point and issues a checkpoint. The **check** command stops the simulation, issues a checkpoint, then continues the simulation run. These commands can be sent by any client able to connect to the server port and are used by Grid Engine to stop the job completely or to do a periodic checkpoint.

The *RemoteModelsim.tc_* script first tries to open socket 1200. If this socket is already in use, the socket number is incremented until the next available socket is found. Once an available socket is found, a file is written directory called *JobXXX* (where *XXX* is the Grid Engine job number) to store the number of the port used to open the server. The file is named *\$JOB_ID.port* where *\$JOB_ID* (a number between 1 and 999999) is an environmental variable written by Grid Engine that holds the id of the ModelSim job. This automates the process of searching for an unused port and allows multiple ModelSim jobs to run on the same host without the worry of stopping the wrong job.

At the end of the ModelSim TCP/IP server port is a very simple command interpreter that understands the stop and check commands. The **stop** command breaks the current simulation by issuing an internal break call to the simulation kernel. With the simulation stopped, the **checkpoint** command is issued and a checkpoint file is saved. Again, the *JOB_ID* environmental variable is used in the filename on the saved checkpoint file. This takes the form of *\$JOB_ID.chk*. After the checkpoint is complete, the ModelSim session is ended using the **quit** command.

The **check** command is nearly the same as the **stop** command except that it restarts the simulation after the checkpoint is created. A *.chk* file is written into the *Job[\$JOB_ID]* directory. There is also a simple fail-safe mechanism that helps if a machine goes down during the generation of the checkpoint file. Before the checkpoint is started, a lock file is placed into the checkpoint directory. Then, if there is a previous checkpoint file, it is moved to *\$JOBID.chk.tmp*. After the checkpoint is completed and the lock file removed, the *\$JOBID.chk.tmp* file is deleted. When the job is restarted, the restart process checks to see if there is a *\$JOB_ID.chk.tmp* file. If the file exists, it is used; otherwise, the *JOB_ID.chk* file is used.

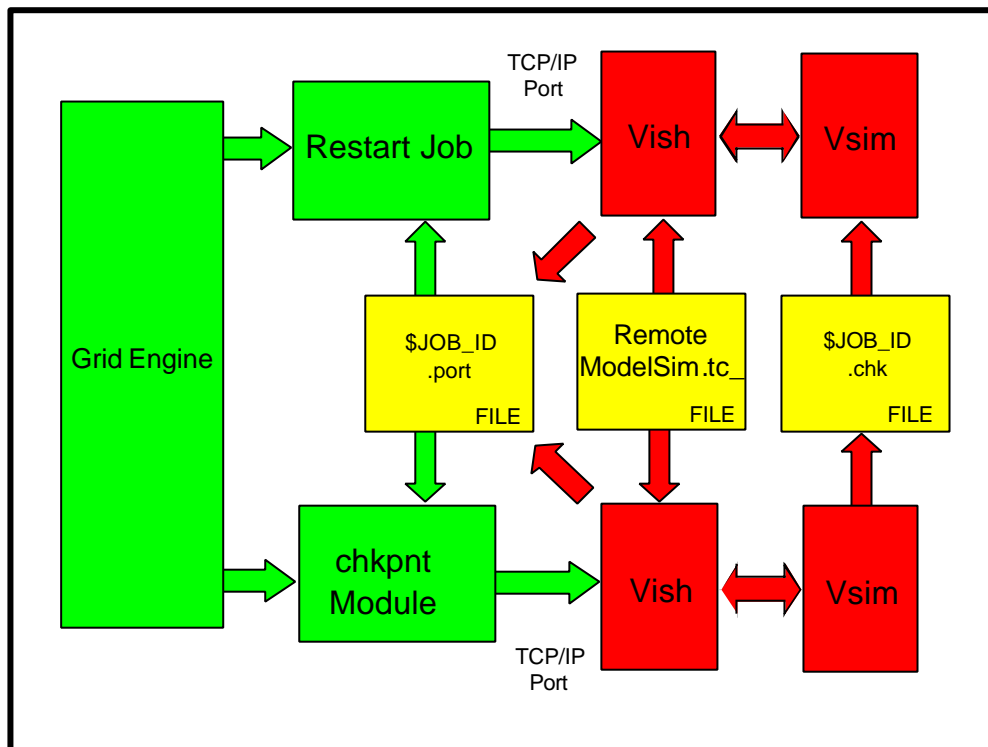
If the *JOB_ID* environment variable is not set, then **vsim** is used for both the *.port* and *.chk* files. The only step required by the user to set up this server is to set-up the *MODELSIM_TCL* environmental variable to point to the *RemoteModelsim.tc_* file. When ModelSim is invoked, any script referenced by

this environmental variable is sourced before any other commands are processed. The following commands can be used to set this variable:

```
ssh : setenv MODELSIM_TCL <path_to_file>/RemoteModelsim.tc_  
sh : MODELSIM_TCL=<path_to_file>/RemoteModelsim.tc_ ;export MODELSIM_TCL
```

Every time ModelSim is invoked the next available port will be set up as a server that accepts the **stop** and **check** commands. The port number used will be written into the *\$JOB_ID.port* file.

The diagram below illustrates how ModelSim communicates with Grid Engine. The parts shown in red have been discussed in this section. The parts in green are discussed in the next section, along with instructions for setting up Grid Engine.



Setting the Grid Engine environment

The integration kit contains a number of files that need to be placed into your ModelSim directories. In addition, a number of configuration parameters need to be added in the Grid Engine installation.

1. Change directory to the directory that contains your ModelSim installation.

```
cd ../<modelsim_installation_directory>
```

2. Untar the integration pack tar file into the modelsim installation with the following commands:

```
tar xvf modeltech-gridware.tar
./modeltech/bin/qmsim
./modeltech/bin/chkpnt
./modeltech/linux/chkpnt
./modeltech/linux/qmsim
./modeltech/sunos5/chkpnt
./modeltech/sunos5/qmsim
./modeltech/jobspy/gridware/docs/gridware.pdf
./modeltech/jobspy/gridware/examples/
./modeltech/jobspy/gridware/examples/simlog.sh
./modeltech/jobspy/gridware/examples/checkpoint_setup.ex
./modeltech/jobspy/gridware/examples/queue_setup.ex
./modeltech/jobspy/RemoteModelsim.tc_
./modeltech/jobspy/gridware/mti_chkpoint
./modeltech/jobspy/gridware/mti_clean
```

There will be a **qmsim** and **chkpnt** for each of the supported operating systems.

3. Set the PATH environment variable. If your ModelSim environment is working, the PATH variable will already be set. However, if you want to use more than one operating system within your farm, you must set the path to point to the *bin* directory in the ModelSim installation. The script in this directory for both **qmsim** and **chkpnt** will then select the correct executable for the machine running your job. If your environment has only one machine type then having the PATH set to the machine-dependant executables is fine. The installation relies on the fact that these executables are on the PATH variable. Therefore, if you only have Linux machines you can set your PATH to point to the Linux directory. If you have both Solaris and Linux boxes then set your PATH to the *bin* directory.
4. Set the MODELSIM_TCL environment variable to point to the *RemoteModelsim.tc_* file:

```
setenv MODELSIM_TCL <modelsim_installation>/jobspy/RemoteModelsim.tc_
```

5. Create a checkpoint environment in Grid Engine with the following command:

```
qconf -ackpt mti_check
```

To use **qconf** you need to be either a root or codadmin user. This will open a standard template for the environment. The following information needs to be added before doing a `:wq` to save the environment:

```
[codadmin@ukmay delta]$ qconf -ackpt mti_check
ckpt_name          mti_check
interface          application-level
ckpt_command       /modeltech/jobspy/gridware/mti_chkpoint
migr_command       /modeltech/jobspy/gridware/mti_chkpoint -k
restart_command    none
clean_command      /modeltech/jobspy/gridware/mti_clean
ckpt_dir           /tmp
queue_list         roger.q grommit.q wallace.q
signal            none
when              smx
```

The interface has to be set to application-level. The **ckpt_command** and **migr_command** should be set to point to the *mti_checkpoint* script within the ModelSim installation. Note that the **migr_command** needs the **-k** switch.

The **clean_command** can be set to the *mti_clean* script only if you want Grid Engine to delete the checkpoint files and directory for a job once the job is complete. This can be set to “none” if you want to keep the last checkpoint file and directory.

The *ckpt_dir* is set to the directory to which you wish to have log files written. Both scripts write out a log file that will grow over time. These can be seen in the *ckpt_dir* directory.

The queue file has to contain all queues that are allowed to use this checkpoint environment. Also, note that the **m** option has to be added to the **when** parameter for the integration to function correctly. The standard template will not have this set.

6. Change each of the queues the checkpoint environment will run.

```
qconf -mq <queue_name>
```

The following set of parameters has to be changed for each queue the checkpoint environment will run. If these parameters are not set, the checkpoint will not work

```
qsub -l arch=glinux -ckpt mti_check -c 00:05:00 -cwd -V sim.sh

qname          ukmay.q
hostname       ukmay
```

The list below is a truncated output from the **qconf** command. (All parameters after “shell” are not important for the integration.)

```
[codadmin@ukmay delta]$ qconf -mq ukmay.q
qname          ukmay.q
hostname       ukmay
seq_no         0
load_thresholds np_load_avg=1.75
suspend_thresholds NONE
nsuspend       1
suspend_interval 00:05:00
priority        0
max_migr_time   0
migr_load_thresholds np_load_avg=5.00
max_no_migr     00:02:00
min_cpu_interval 00:00:01
processors      UNDEFINED
qtype          BATCH INTERACTIVE CHECKPOINTING
rerun          FALSE
slots          4
tmpdir         /tmp
shell          /bin/sh
```

There are two important parameters above. The first is the **qtype**. CHECKPOINTING needs to be added to the list of jobs that can be run for this queue. The standard template will have only BATCH and INTERACTIVE set. The second is the **min_cpu_interval**. If you want to be able to use the **-c** switch with **qsub** to define how often the job is to be checkpointed, this option has to be set to a value lower than any checkpoint interval you may want to set. In the example above it has been set to 1 second, which means that the **-c** option can now be used to set an interval of anything above 1 second. Grid Engine chooses the maximum time set on the **-c** switch or the **min_cpu_interval**.

The shell parameter should be set to the default shell you wish to use. Both */bin/sh* and *bin/csh* scripts are supplied with this integration.

Once these steps have been carried out, the environment should be ready to start running ModelSim jobs for checkpointing. One of the following scripts must be used as a template to run jobs. (DELTA is the design name.) In Script 1, */bin/sh* calls *simlog.sh*. In Script 2, */bin/csh* calls *simlog.csh*.

Script 1:

```
#!/bin/sh
if [ $RESTARTED -eq 0 ]; then
# Your Normal ModelSim Command Goes Here
vsim -c DELTA -do "log /*;run -all;quit"
else
if [ -f ./Job$JOB_ID/$JOB_ID.chk.tmp ]; then
cktp_file=./Job$JOB_ID/$JOB_ID.chk.tmp
else
cktp_file=./Job$JOB_ID/$JOB_ID.chk
fi
# Your Restore command goes here
vsim -c -restore $cktp_file -do "log /*;run -all;quit"
fi
```

Script 2:

```
#!/bin/csh
if ( $RESTARTED == 0 ) then
# Your Normal ModelSim Command Goes Here
vsim -c DELTA -do "log /*;run -all;quit"
else
if ( -e ./Job$JOB_ID/$JOB_ID.chk.tmp ) then
set cktp_file=./Job$JOB_ID/$JOB_ID.chk.tmp
else
set cktp_file=./Job$JOB_ID/$JOB_ID.chk
endif
# Your Restore command goes here
vsim -c -restore $cktp_file -do "log /*;run -all;quit"
endif
```

Note: Grid Engine can only run scripts; it cannot run executables.

Grid Engine automatically sets an environmental variable called `RESTARTED` when a job is restarted. The scripts above use the variable to decide whether to start the job from the beginning or from a saved checkpoint file.

The two **vsim** commands in the scripts are shown in italics and are underlined. The first starts the job; the second restarts from a checkpoint file.

The following command can be used to submit the job above.

```
qsub -cwd -V -ckpt mti_check -c 00:20:00 -l arch=glinux simlog.sh
```

The **-cwd** switch runs the job from the current working directory.

The **-V** switch submits all environmental variables to Grid Engine.

The **-ckpt** switch is set to the `mti_check` checkpoint environment that has been set up above.

The **-c** switch defines the time between each checkpoint – in this case, every 20 minutes.

The **-l** switch is used when multiple OS machines are used in a compute farm. If there is only one type of machine in your farm, this switch does not have to be used.

Using JobSpy with Grid Engine

The JobSpy functionality allows a limited number of commands to be executed while a job is running on a compute farm controlled by Grid Engine. Remember that Grid Engine must be running and each user must have the MODELSIM_TCL environment variable set to use the JobSpy functionality. The syntax is:

```
setenv MODELSIM_TCL <modelsim_installation>/jobspy/RemoteModelsim.tc_
```

JobSpy integration with Grid Engine is implemented with a program called qmsim. It is able to get information from Grid Engine to determine where a job is being run and then send commands directly to the job as it is running. The command line syntax is as follows:

```
qmsim <JOB_ID> <command>
```

where <JOB_ID> is the unique number assigned by Grid Engine to a job.

Since the JobSpy/Grid Engine integration is not intended to turn a ModelSim batch job into an interactive session only a small number of commands are available. These include:

- **simulation status commands** – now, examine, status, echo, set
- **simulation control commands** – stop, break, check, continue, resume, quit
- **waveform commands** – log, nolog, dataset
- **profiler commands** – profile on, profile off, profile report

When a ModelSim simulation session is run from a macro using either **profiler** or **simulation control** commands the simulator has to be stopped using a **break** command. **Break** stops the currently executing command. The simulation will resume from the very next command. For guidelines on how to handle this in your macro script, see the “Using Profiler and Control commands when running macros” section below.

Simulation status commands

These commands are used to find out current information about a job that is running.

now – This command will print the current \$now variable to stdout and display the exact simulation time at which the command is executed. You can use this command to see how a simulation is progressing. An example is shown below.

```
[codadmin@ukmay delta]$ qmsim 54 now
Connecting To Job 54
Current Simulation Time is 18,002,381 ns
```

examine – This command returns the value of a signal/variable and the current simulation time. It allows you to display values from the design while the simulation is running. It can be useful, for example, to determine whether or not the simulation has reached a particular state or if it is out of reset. An example is shown below.

```
[codadmin@ukmay delta]$ qmsim 54 examine
/delta_rtl/chip/preproc_inst/faw_state
Connecting To Job 54
Signal /delta_rtl/chip/preproc_inst/faw_state is out_of_sync At Simulation
Time 74,946,613 ns
```

status – This command displays current running status of macros. It will show the current line of the active macro along with the command that is executing. The current value of the **onbreak** and **onerror** commands will also be shown. If no macro is running then it will say so. An example is shown below.

```
[codadmin@ ukmay delta]$ qmsim 54 status
Connecting To Job 54
Macro ./do.do at line 6 (Current macro)      command executing: \}
ONBREAK commands: resume
```

echo – This command gets the value of any variable within ModelSim. It allows you to check the value of a TCL variable while the simulation is running. An example is shown below.

```
[codadmin@ukmay delta]$ qmsim 54 echo StdArithNoWarnings
Connecting To Job 54
StdArithNoWarnings = 0 @ Current Simulation Time 6,321,735 ns
```

set – This command sets the value of any variable within ModelSim. It allows you to set the value of a TCL variable while the simulation is running. An example is shown below.

```
[codadmin@ukmay delta]$ qmsim 54 set StdArithNoWarnings 1
Connecting To Job 54
set StdArithNoWarnings to 1 @ Current Simulation Time 63,111,725 ns
```

Simulation control commands

There are some restrictions on the way jobs have to be started if macro/do files are being used with the Simulation Control Commands. These restrictions are explained in the “Using Profiler and Control Commands when running Macros” section. The **checkpoint** command is of greatest concern because it only saves the status of the simulation and not the point at which the macro file has reached. This means that it is not possible to restart a checkpointed job at the correct place within the macro file.

quit – This command quits the simulation job. It allows you to finish the simulation run at the current simulation time. An example is shown below.

```
[codadmin@ukmay delta]$ qmsim 54 quit
Connecting To Job 54
Quitting Job @ Current Simulation Time 82,617,725 ns
```

stop – Grid Engine uses this command to do a checkpoint and then exit the simulation. The checkpoint is stored in the JobID directory called *JOB_ID.chk*. If a filename is defined after the **stop** keyword, this filename will be stored in the simulator’s working directory. An example is shown below.

```
[codadmin@ukmay delta]$ qmsim 54 stop Chkpt1
Connecting To Job 54
Checkpointing & Stopping Job Chkpt1
```

check - Grid Engine uses this command to checkpoint the simulation. By default, the checkpoint will be stored in the JobID directory called *JOB_ID.chk*. If a filename is defined after the check keyword this filename will be stored in the simulator’s working directory. An example is shown below.

```
[codadmin@ukmay delta]$ qmsim 54 check Chkpt2
Connecting To Job 54
Checkpointing Job Chkpt2
```

break - This command will cause the simulation to stop. ModelSim will remain on and display the simulation prompt in batch mode. If there is a macro running, the current value of the **onbreak** command is cleared so that the macro halts and waits for instruction. Any JobSpy command can be executed while the batch job sits in this state. An example is shown below.

```
[codadmin @ ukmay delta]$ qmsim 54 break
Connecting To Job 54
Break Simulation @ Current Simulation Time 63,111,725 ns
```

If a macro is running, the **resume** command is needed to start execution again. If no macro is running, the **continue** command must be used. Note that **break** will stop the execution and will not finish the current command. Therefore, if the job was started using the **-do "run -a ; quit -f"** method the **break** command will cause the simulation to quit. The **break** command is intended to be useful in situations where .do scripts are running.

continue - This command can be used to continue a simulation that has been stopped with a **break** command. It will cause the simulation to start again using the **run -cont** command and should be used only if there is no macro running. An example is shown below.

```
[codadmin @ ukmay delta]$ qmsim 54 continue
Connecting To Job 54
Continue Simulation @ Current Simulation Time 63,111,725 ns
```

suspend - This command is not sent to the tcp/ip socket for the simulation job, instead it causes a signal to be sent to the process in the operating system. The suspend command causes a SIGSTOP to be sent to the "vsim" process in versions of ModelSim before 5.8, for versions 5.8 and higher SIGTSTP is sent to the "vish" process. In versions pre-5.8 this will cause the process to be stopped by the operating system, in versions 5.8 and higher the operating system will stop the process and it will also cause ModelSim to check in any licenses that are being used for the simulation job. This allows other jobs to use the license tokens while the job is suspended. Once a job has been put into suspend mode it is not possible to send any more commands to the running job until the job is resumed again using the resume command. The suspend and resume functionality uses the rsh command to send signals and monitor the state of processes on the machine that the job is running. Therefore to use this function it is necessary to be able to rsh from the machine the qmsim command is being run on, to the machine that the job is running on. If the qmsim program is unable to do this then a message is printed and the suspend and resume functionality is disabled.

```
[user1@myhost delta]$ qmsim 54 suspend
Sent SIGSTOP To Process 4321.

[user1@myhost delta]$ qmsim 54 now
Process 4321 in currently suspended.
```

resume - This command is not sent to the tcp/ip socket for the simulation job, instead it causes a signal to be sent to the process in the operating system. The resume command causes a SIGCONT to be sent to the "vsim" process in versions of ModelSim before 5.8, for versions 5.8 and higher SIGCONT is sent to the "vish" process. In versions pre-5.8 this will cause the process to be started again by the operating system. In versions 5.8 and higher the operating system will start the process and it will also cause ModelSim to check out the licenses that were being used for the simulation job before it was suspended by the suspend command.

```
[user1@myhost delta]$ qmsim 54 resume
Sent SIGCONT To Process 4321.
```

Waveform commands

These commands are used to control signal logging so database (WLF) files can be written during simulation. This allows simulation results to be viewed later, off line.

log – This command allows you to log any signal in the design while the simulation is running on the farm. An example is shown below. Note, because the ‘*’ wildcard is used it needs to be escaped so the shell does not expand it.

```
[codadmin@ukmay delta]$ qmsim 54 log /\*
Connecting To Job 54
Logging Signals /* @ 74,946,613 ns
```

nolog - This command allows you to switch off Logging of any signal in the design while the simulation is running on the farm. An example is shown below. Note, because the ‘*’ wildcard is used it needs to be escaped so the shell does not expand it.

```
[codadmin@ukmay delta]$ qmsim 54 nolog /\*
Connecting To Job 54
Un-Logging Signals /* @ 94,976,693 ns
```

dataset save – This is the only **dataset** command available. It allows the current simulation dataset to be saved into another file so that it can be viewed while the job it still running on the farm. This can be very useful if you want to take a quick look at what is happening with a batch job while it is running. The file created can be viewed using the **–view** switch of **vsim**, meaning that the user can have visibility into the job while it is running. An example in shown below.

```
[codadmin@ukmay delta]$ qmsim 54 dataset save sim snap.wlf
Connecting To Job 54
Dataset "sim" exported as WLF file: snap.wlf. @ 84,785,547 ns
```

Here the dataset is saved to *snap.wlf*. A ModelSim session can now be started using the **–view** switch (i.e. **vsim –view snap.wlf**) to look at the waveforms.

Profiler commands

There are restrictions on the way jobs have to be started if macro/do files are used with Profiler Commands. These restrictions are explained in the “Using Profiler and Control commands when running macros” section below.

profile on - This command turns on the ModelSim Performance Analyzer functionality. You can use this functionality to produce hierarchical and ranked profile reports about simulation performance. An example is shown below.

```
[codadmin@ukmay delta]$ qmsim 54 profile on
Connecting To Job 54
Executed profile on
```

profile off - This command turns the ModelSim Performance Analyzer functionality off. An example is shown below.

```
[codadmin@ukmay delta]$ qmsim 54 profile off
Connecting To Job 54
Executed profile off
```

profile report - This command allows you to write performance reports so that the simulation performance can be viewed while the job is running on the farm. In the example below, the performance report is written to the file *prof.out*.

```
[codadmin@ukmay delta]$ qmsim 54 profile report -ranked -file prof.out
Connecting To Job 54
Executed profile report -ranked -file prof.out
```

Using Profiler and Control commands when running macros

When Profiler and Simulation Control commands are used in a macro, the **break** command must be used to stop the simulation. Any command that resumes the simulation will start at the next command in the macro. For example, if a macro is running a **run -all** command, the **break** command will stop execution of the macro. Upon resumption, **run -all** will not continue; rather, the next command will be executed. This means that the script may not do what is intended.

If you start a job in the following manner:

```
vsim -c toplevel -do "run 800 ms ; quit-f"
```

there are no restrictions on which of the JobSpy commands you can use. However, if you start the job with a `do` file in the following manner:

```
vsim -c toplevel -do run.do
```

where the `run.do` file is as follows:

```
run 800 ms
quit -f
```

a **break** command will cause the macro to stop the **run 800 ms** command. When a **resume** or **continue** command is used, **quit -f** will be executed. To stop this from happening the `run.do` file needs to be modified as follows:

```
onbreak resume
while { [ltTime $now {800 ms} ] } {
  run @800 ms
}
quit -f
```

In this example, if a break occurs the test is run to see if the simulation has reached 800 ms. If it hasn't, **run @ 800 ms** will be executed again. (Using the `@` sign allows an absolute time to be stated in the **run** command.) The **onbreak resume** command causes the script to resume after a break. **ltTime** is a ModelSim procedure that returns a 1 when **\$now** is equal or more than 800 ms. This procedure allows time units to be used.

A simulation could use a **run -all** command if the script included how long to run. The `run.do` file would look like this :

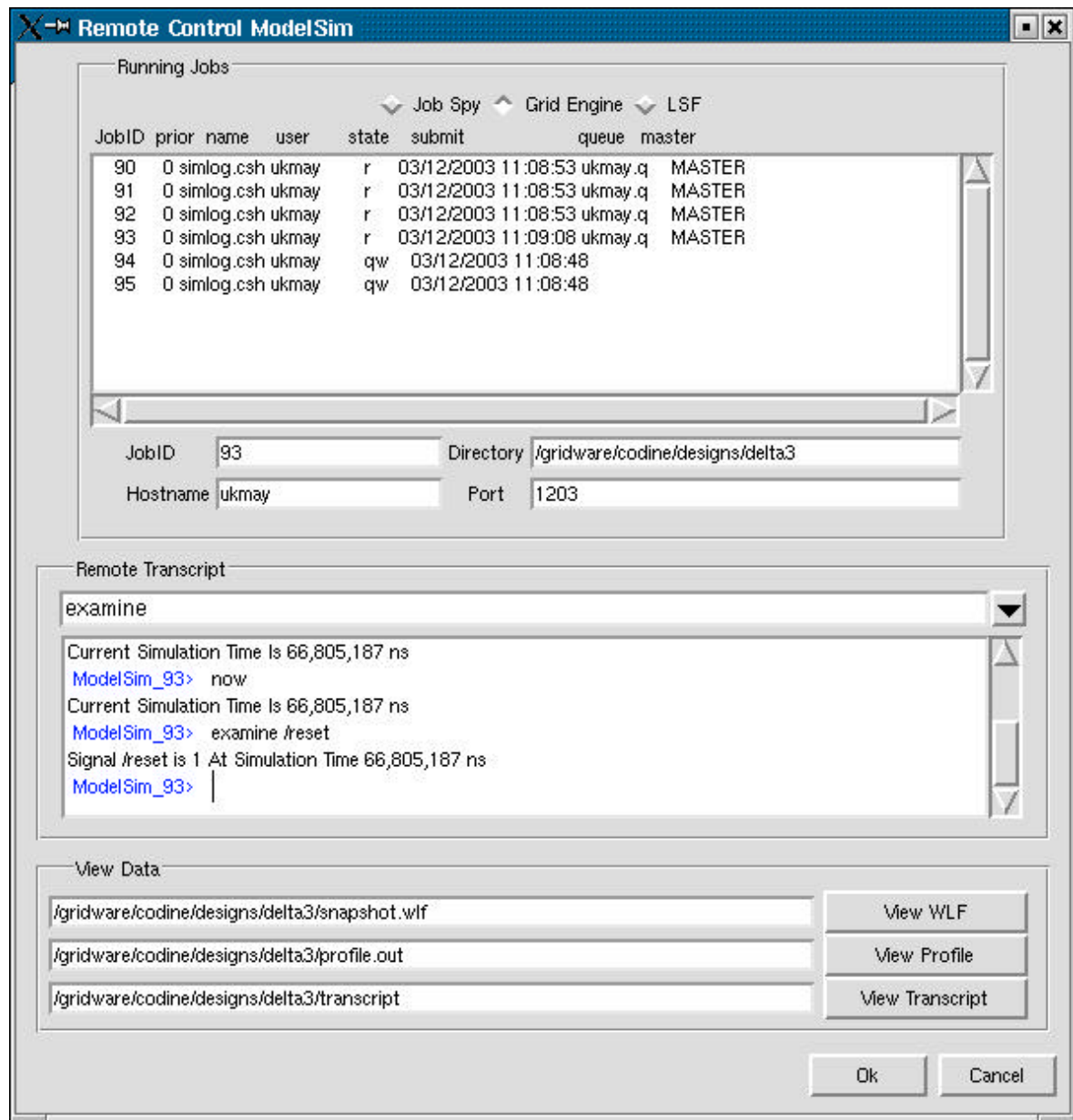
```
onbreak resume
while { [ltTime $now {1000 ms} ] } {
  run -all
}
quit -f
```

Security

Security has been taken care of within the `qmsim` utility. Because simulation jobs can be affected remotely using this utility, it is only possible for the user who submits a job to send commands to that job. For example, the **quit** command will finish a simulation job. Therefore, only the owner can carry out this action. The only exception is that the "codadmin" user – the administration account for the Grid Engine software. Codadmin can send commands to any job that is running on the grid.

Using JobSpy From The ModelSim User Interface

The ModelSim GUI allows you to send commands to remote ModelSim jobs. The Remote Control dialogue box can be started using the menu selection **Tools > Remote Control** option. The dialogue is split into 3 sections – Running Jobs, Remote Transcript, and View Data.



Running Jobs

The Running Jobs section shows the currently running jobs and the current remote control mode. Three remote control modes are supported. Two are load sharing software packages – Grid Engine from Sun Microsystems, and LSF (Load Sharing Facility) from Platform Computing Inc. The third mode is the ModelSim JobSpy daemon. The remote control mode will default to what is set up in the environment, but can be changed with this dialog. If either load sharing software package is set-up in the environment, it will be chosen; if not, JobSpy will be the selected mode.

There are four fields at the bottom of the Running Jobs section – JobID, Directory, Hostname and Port. When an active job is selected, these four fields are automatically updated to include the correct JobID, the directory the job is running in, the name of the host the job is running on, and the TCP/IP port being used by the host. In addition, the default directory paths for the WLF file, transcript and the profile report file are all filled out in the View Data section (see below).

Remote Transcript

Any time a job is selected in the Running Jobs list the Remote Transcript will give a prompt for that job. In Grid Engine mode, the prompt will be ModelSim_<JobID>. Once the prompt is displayed in the Remote Transcript it is possible to type in the JobSpy command that you wish to send to the remote job. The response from the job will be displayed in the Remote Transcript box.

There is a pull down menu of all the supported JobSpy commands above the Transcript. Select a command from this menu and it will be pasted into the Remote Transcript. A carriage return will execute the command.

The Remote Transcript also maintains a history of executed commands. You can use the up and down arrows to toggle through the last 20 commands. Select one and hit the carriage return to execute.

View Data

The directory paths to the WLF, transcript and profile reports are automatically filled out when a job is selected in the Running Jobs list. These fields are accompanied by three quick start buttons – View WLF, View Profile, and View Transcript.

- **View WLF** – The WLF (waveform database file) holds the activity from the simulation. In ModelSim, it is possible to take a snapshot of the database while the simulation is running and save it to a file for viewing. This button will automatically open the *snapshot.wlf* file, which has been saved from a **dataset save** command, and load the signals into the Wave window for viewing. The directory name is set automatically to the current working directory when the job is selected in the Running Jobs list. Before a snapshot can be taken using the **dataset save** command, signals must be logged. Signals can be logged from the start of the job, or during the job using the **log** command, which is supported by JobSpy. It is possible to turn on logging using the **log** command, let the simulation run for a while, snapshot the dataset and then switch off logging with the **nolog** command.
- **View Profile** – The ModelSim Performance Analyzer functionality is turned on when you click the View Profile button. Performance Analyzer computes a profile of where simulation time is being spent and can be turned on and off while the simulation is running. This button will automatically open by default the *profile.out* file that is produced using the **profile report** command. The directory name is set automatically to the current working directory of the job selected in the Running Jobs list. Before a report can be generated, the Performance Analyzer needs to be switched on using the **profile on** command. Once the **profile report** command has been executed, pressing this View Profile button will automatically load the report into a viewer. The profile can also be switched off at any time using the **profile off** command. The **profile report**, **profile on** and **profile off** commands are supported by JobSpy and are available from a pull down menu.
- **View Transcript** – The View Transcript button allows the transcript from the currently running job to be loaded into a view window. The transcript will be updated as the job progresses. The directory name is set automatically to the current working directory of the job is selected in the Running Jobs list. The default name for the file is set to transcript, which is the ModelSim default.

Integration Kit Contents

The following is a quick description of the files in the ModelSim / Grid Engine integration kit.

./modeltech/bin/qmsim – This script will determine which platform is running and will use the correct binary for the qmsim program.

./modeltech/bin/chkpnt – This script will determine which platform you are running on and will use the correct binary for the echkpnt program.

./modeltech/linux/chkpnt – This is the echkpnt (checkpoint) program for the Linux platform.

./modeltech/sunos5/chkpnt – This is the echkpnt (checkpoint) program for the sunos5 platform.

./modeltech/linux/qmsim – This is the qmsim program for the Linux platform.

./modeltech/sunos5/qmsim – This is the qmsim program for the sunos5 platform.

./modeltech/jobspy/gridware/examples/simlog.sh – This is a sample script needed to run ModelSim with the checkpoint and restore capability.

./modeltech/jobspy/gridware/examples/checkpoint_setup.ex – This is a sample script of the set up of a checkpoint environment for Grid Engine.

./modeltech/jobspy/gridware/examples/queue_setup.ex – This is a sample script of the set up of a queue environment for Grid Engine.

./modeltech/jobspy/RemoteModelsim.tc_ – This is the TCL file that needs to be sourced by any job that needs to be controlled by a ModelSim session and requires the controlling dialogue box.

./modeltech/jobspy/gridware/mti_chkpoint – This is the checkpoint script that Sungrid uses.

./modeltech/jobspy/gridware/mti_clean – This is the clean up script that Sungrid uses.