

VHDL Design Management Under UNIX

Kevin A. Sholander
Senior Design Engineer
Philips Semiconductors
Albuquerque, NM 87113
sholander@abqhp1.scs.philips.com

Graciela B. Sholander
B.S. Electrical Engineering
University of California, San Diego
La Jolla, California

ABSTRACT

This paper shows how to set up release management using basic UNIX utilities, such as RCS and make. The scheme that is presented prevents new releases from affecting simulations that are already in progress and allows users to choose which release they would like to use, even allowing them to substitute experimental code in the place of a released module. The paper walks the reader through an example project, demonstrating how to set up the directory structure and develop the appropriate scripts and makefiles that will automate the environment.

INTRODUCTION

Here is a typical scenario in the engineering environment. Peter, Paul, and Mary are three of many designers working on a large VHDL project. They have released preliminary designs and have begun system level simulations. Because a good version control mechanism is not in place, the engineers waste a lot of time and experience a great deal of frustration.

Each time someone changes and releases his or her design block, simulation runs that are in progress may end up crashing. New runs require recompilation to incorporate the new version -- a very time-consuming process, especially for large designs.

Peter decides that, to avoid wasting others' time, he won't release new versions until he feels his block is stable. This way, he figures, current simulations won't crash and people won't waste time recompiling the database over and over again. But, because he's the only designer with the most current copy, others are wasting time chasing after errors that may not exist anymore.

Paul believes everyone should always have access to the most up to date versions. He releases his design every time he makes a change -- even if that's five times in one day. Simulation runs are constantly invalidated, and designers are spending most of their time recompiling.

To avoid these constant interruptions, Mary copies everyone else's designs into her own account. Utilizing this local database, she is able to simulate for days without interruptions. But

she is using outdated information, and has no idea how many times new versions have been released.

The version control mechanism described in this paper solves all of the problems illustrated in the previous example. It allows for new design versions to be released at any time without disrupting simulations in progress.

FEATURES

The proposed scheme utilizes a centralized release directory and a set of UNIX command scripts to control the data. It allows engineers in the middle of simulation runs to continue without interruptions. It notifies engineers who are about to begin new simulation runs whether or not any design block has been modified. They then have the choice of either incorporating the new release into their runs or continuing with older versions until reaching a convenient switching point.

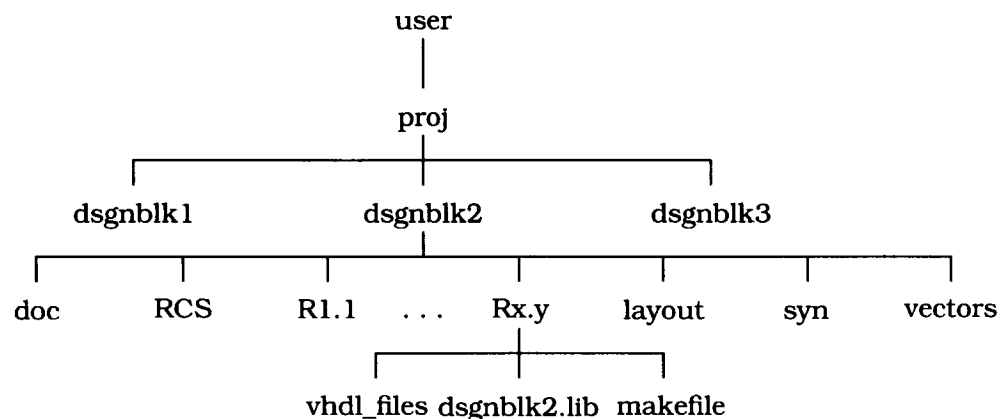
The scheme saves additional time during simulations since recompilations goes by quickly, due to the fact that the new release has already been compiled into a centralized repository. Command scripts enable engineers to recompile just the top level configuration.

Designers are given the added bonus of being able to check their designs on a system level before releasing updated versions. This results in cleaner released versions with fully functional interfaces.

DATA FILES

Let's look at the directory structure. All the libraries and designs within a project reside in /user/proj, as shown in Figure 1, where proj refers to the name of the project to which the design belongs. Each design contains VHDL files, schematics, synthesis databases, layout databases, documentation, and revision control. Each design is a self-contained module residing in a subdirectory of proj, as illustrated in Figure 1 through dsgnblk2. Typically, one designer is responsible for each design block.

Figure 1: Design Directory Structure



The RCS directory holds all of the revision history for the VHDL files which compose the design block. Anyone in the design team may check out, modify, and check back in the files in this directory, as long as he or she owns the lock. This allows for anyone in the team to fix errors, if the owner wishes. But only one person can make changes at a time. The owner does have the option of always keeping the lock, preventing others from modifying the design block. The UNIX commands rcs, ci and co are used to access files in the RCS directory.

The doc directory contains the design block's functional specification and other supporting documentation.

The syn directory has all of the synthesis database files for the design block, put there for archival purposes. When the design block is stable and ready for shipping, the output from the final synthesis run is stored in the syn directory before the entire directory structure is archived.

Rx.y is one of several release directories, each one containing a snapshot of the design block at the time that the release was made. The reason that there are several release directories per design block is to give those who are simulating the ability to access older releases until they reach a point where they can conveniently switch to the newest release. Disk space constraints set the limit for the number of releases that are maintained. Generally, three releases are sufficient to accommodate everyone running simulations.

Each release directory holds the VHDL files making up that particular release, a simulation library with compiled versions of the VHDL files, and a makefile which describes how the VHDL files should be compiled.

Whenever a new simulation run is started, the latest release of each design block will be selected by default. However, the person beginning the new run does have the option of choosing an older release. This is especially useful when tracking an error in, say, design block #4 which may be masked by the release of a new version of design block #1. Ongoing simulations are not affected by a new version being released in the middle of runs, since the older, unchanged versions continue to be accessed.

After the design is stable and ready to be shipped, only the latest version of each block is kept. All older versions are removed so that it is guaranteed that the simulations pass on the versions that are being shipped.

The layout directory holds only the latest version of the design block's layout database and back annotation data.

The vectors directory contains the functional vectors, scan test vectors, and any other vectors or code that are specifically related to the design.

ENVIRONMENT CONTROL TOOLS: Unix Utilities

Now, let's take a look at the Unix utilities needed to support the version control mechanism described in the previous sections.

RCS is the Revision Control System, a Unix utility used to document and store every change made to the source files comprising each design block. RCS also ensures that only the owner of the lock may modify a file.

When a file is checked out, modified, and checked back in, RCS assigns a version number to the affected file and prompts the user for a description of the changes that were made. A

designer has the option of revising over and over again, without releasing his or her design to others. Then, when he or she is finally ready to release the design block, only then will a new release directory be created for that design and the files checked out of RCS and placed in that directory.

The following commands are associated with RCS: *ci*, *co*, *rlog*, and *rcsdiff*.

ci is used to check files into RCS control. When executed with the “-l” option, the lock bits are set for each of the files that are checked in. A writable, working copy is placed in the local directory. When executed with the “-u” option, the lock bits are cleared. A read-only copy is placed in the local directory. When executed with no options, the lock bits are cleared for each of the files that are checked in but no copy is left in the local directory. The command format for *ci* is:

```
ci [-l | -u] filename.vhd [filename.vhd ...]
```

co is used to check files out from RCS control to create a working copy in the local directory. With the “-l” option, RCS sets the lock bit for the file being checked out. A writable copy is made. Without the “-l” option, RCS does not set the lock bit and a read-only copy is made. The command format is:

```
co [-l] filename.vhd [filename.vhd ...]
```

rlog displays information that RCS maintains for each file. This includes revision history, lock owner for each revision, and any symbolic names that have been assigned to each revision. The command format is:

```
rlog filename.vhd [filename.vhd ...]
```

rcsdiff displays changes that were made to the local working copy of a file since it was last checked out from RCS. The command format is:

```
rcsdiff filename.vhd [filename.vhd ...]
```

Make is a Unix utility which is used to control and minimize the compilation process of source files. It gets its commands from a makefile which contains a number of rules which describe the files and their compilation dependencies. Each design block has its own makefile describing the compilation order for that block. A top level makefile “calls” each of the design block makefiles, controlling the location and version for each. This top level makefile is the key to this environment since it is in this file that all of the current revision numbers for all of the design blocks are stored, and this makefile is used by the scripts for releasing new models or starting a new simulation.

Environmental Control Tools: Unix utilities The Unix *make* utility is controlled through a file called a makefile, which contains a list of rules that describe the dependencies between the compiled code and the source code. The rules in a makefile take the form of a label or a target, followed by a list of files on which the target is dependant, followed by the action or actions that should be taken to bring the target or label up to date. By default, *make* will start by checking the first rule in the makefile, but any of the rules can be specified when *make* is executed at the command line. If the rule has a dependency list, then *make* will search for another rule for each file or label in the list. If a rule is found, then *make* will step down to that rule and run it first, then return to the current rule. If no rule is found for a file in a dependency list, then *make* checks the time stamp on the file against the target of the rule. If the time stamp indicates that the target was created before the dependency, then the action is taken to bring the target up to date. If the target has a newer time stamp than the dependency, then the target is up to date and no action is taken. In the case that a label is used, the action will always be taken since the label has no time stamp.

Makefiles are also permitted to define macros which are substituted by their assigned values when they are encountered in the makefile. Ordinarily these macros are assigned within the file, but their assigned values can be over ridden by arguments at the command line. This substitution is useful for creating makefiles that define rules with dependencies that may change their directory locations in the future. For instance, if the compiled code is dependent on the source code, which it always is, but the source code could be located in different directories each time that the makefile is run, then a macro could be used to define a new location for the source code every time that the makefile is run. This is useful since every time that a version of the VHDL code is released it is located in a different release directory. Below is an example of a design makefile. Note that the format of the makefile is extremely rigid and that tabs must be present in certain places but may not be present anywhere else. The makefile shown here uses the characters `\t` to represent the tab character.

```
-----
# Define the library into which the design will be compiled
LIB = design1.lib
# Define the directory which contains the VHDL source code
VHDL = .
# Define the object file that is created when the file is analyzed
OBJ = vantage.vdl
# Define the analyze command
AN = /usr/vantage/pgm/dir/analyze
# Define the analyze options
OPT = -lib $(LIB) -libieee
# Define rule for configuration
$(LIB)/design1.edr/behavior.adr/design1_conf.cdr/$(OBJ):\t \
$(LIB)/design1.edr/behavior.adr/$(OBJ) \
$(LIB)/sub_design1.edr/behavior.adr/$(OBJ) \
$(LIB)/sub_design2.edr/behavior.adr/$(OBJ) \
$(VHDL)/design1_conf.vhd
\t $(AN) -src $(VHDL)/design1_conf.vhd $(OPT)
# Define rule for top level
$(LIB)/design1.edr/behavior.adr/$(OBJ):\t \
$(LIB)/design1_pkg.pdr/$(OBJ) \
$(VHDL)/design1.vhd
\t $(AN) -src $(VHDL)/design1.vhd $(OPT)
# Define rule for sub-design 1
$(LIB)/sub_design1.edr/behavior.adr/$(OBJ):\t \
$(VHDL)/sub_design1.vhd
\t $(AN) -src $(VHDL)/sub_design1.vhd $(OPT)
# Define rule for sub-design 2
$(LIB)/sub_design2.edr/behavior.adr/$(OBJ):\t \
$(VHDL)/sub_design2.vhd
\t $(AN) -src $(VHDL)/sub_design2.vhd $(OPT)
-----
```

The top level makefile is the key to the entire environment since it is the only place in which all of the design revision numbers are stored. This makefile called by the release script whenever a new design is ready to be released, and is also called by the startsim script whenever a new simulation is started. When the "RELEASE" rule in the makefile is executed, the top level makefile will recursively call all of the design makefiles, substituting the pointer to the latest release for the VHDL macro in each of the subdesigns. When the "SIMULATE" rule in the makefile is executed, only the top level netlist and configuration are analyzed, and the list of library point-

ers is dumped into a file in the users local directory. In this way, the user's directory now contains a compiled snapshot of the most recently released environment, as well as a library list in a format that can directly be read by the simulator.

```

-----
# Define Locations of Libraries
IEEE = $(VANTAGE_VSS)/pgm/libs/ieee.lib
SKIT = $(VANTAGE_VSS)/pgm/libs/std_developerskit.lib

# Define Locations of most recently released code
TOP_DIR = /user/proj/top/R3.2
DSGN1_DIR = /user/proj/dsgn1/R1.3
DSGN2_DIR = /user/proj/dsgn2/R2.4
DSGN3_DIR = /user/proj/dsgn3/R1.1

# Define object code location
OBJ = vantage.vdl

# Define analyze command and options
AN = /usr/vantage/pgm/dir/analyze
OPT = -libfile libs4analyze

# Define SIMULATE Rule, which is default
SIMULATE:\t
    GEN_LIBFILE \
    $(TOP_DIR)/top.lib/top.edr/behavioral.adr/top_conf.cdr/$(OBJ)

# Define RELEASE Rule
RELEASE:\t
    DSGN1 \
    DSGN2 \
    DSGN3

# Write out library list to file
GEN_LIBFILE:\t
\t    echo $(TOP_DIR)/top.lib > libs4analyze; \
    echo $(DSGN1_DIR)/design1.lib >> libs4analyze; \
    echo $(DSGN2_DIR)/design2.lib >> libs4analyze; \
    echo $(DSGN3_DIR)/design3.lib >> libs4analyze

# Define dependencies for top level configuration
$(TOP_DIR)/top.edr/behavioral.adr/top_conf.cdr/$(OBJ):\t \
$(TOP_DIR)/top_conf.vhd \
$(TOP_DIR)/top.lib/top.edr/behavioral.adr/$(OBJ) \
$(DSGN1_DIR)/design1.lib/design1.edr/behavior.adr/design1_conf.cdr/$(OBJ) \
$(DSGN2_DIR)/design2.lib/design2.edr/behavior.adr/design2_conf.cdr/$(OBJ) \
$(DSGN3_DIR)/design3.lib/design3.edr/behavior.adr/design3_conf.cdr/$(OBJ)
\t    $(AN) -src $(TOP_DIR)/top_conf.vhd

# Define dependencies for top level netlist
$(TOP_DIR)/top.lib/top.edr/behavior.adr/$(OBJ):\t \
$(TOP_DIR)/top.vhd \
$(DSGN1_DIR)/design1.lib/design1.edr/behavior.adr/$(OBJ) \
$(DSGN2_DIR)/design2.lib/design2.edr/behavior.adr/$(OBJ) \

```

```

$(DSGN3_DIR)/design3.lib/design3.edr/behavior.adr/$(OBJ)
\t      $(AN) -src $(TOP_DIR)/top.vhd

# Call makefile for design 1
DSGN1:\t
\t      make -f $(DSGN1_DIR)/makefile LIB=$(DSGN1_DIR)/design1.lib \
          VHDL=$(DSGN1_DIR)
# Call makefile for design 2
DSGN2:\t
\t      make -f $(DSGN2_DIR)/makefile LIB=$(DSGN2_DIR)/design2.lib \
          VHDL=$(DSGN2_DIR)
# Call makefile for design 3
DSGN3:\t
\t      make -f $(DSGN3_DIR)/makefile LIB=$(DSGN3_DIR)/design3.lib \
          VHDL=$(DSGN3_DIR)

```

ENVIRONMENT CONTROL TOOLS: Custom Scripts

The Unix utilities described above work with custom scripts in order to fully support the version control mechanism. This section describes the scripts that are used to release designs into the environment and guarantee that simulation users always have access to the latest released code.

Two main scripts are used to control the environment. The first is used to release new models of designs into the simulation environment such that they will be available to subsequent runs of the simulator. The other script is used to start the simulator and select which versions of each model will be used during simulation.

The release script begins by creating a new release directory for each of the designs that are being released. It then checks out a read only copy of each of the VHDL files from RCS control and places them in the new release directory. The final thing this script does to the release directory is to create the VHDL library. After this has been done for each of the models that is being released, the script finishes by executing the SIMULATE rule of the top level makefile. The csh script for the makefile could appear as follows:

```

DESIGN_LIST=""
while [ $# != 0 ] ; do
    if [ -d /user/proj/$1 ] ; then
        DESIGN_LIST="{DESIGN_LIST} $1"
    else
        echo "ERROR: Design does not exist"
        exit 1;
    fi
    shift
done
if [ "{DESIGN_LIST}" = "" ] ; then
    echo "ERROR: No designs specified"
    exit 1;
fi

```

```

#create working copy of makefile
cp /user/proj/makefile /user/proj/new_makefile

for DESIGN in ${DESIGN_LIST}; do
    cd /user/proj/${DESIGN}

    #get number of latest release
    PREV_REL=`ls -l | grep -E R\[0-9\]\.\.[0-9]\[0-9\]* | \
        sort -t . -k 1.2nr,1.2nr -k 2.1nr | sed -e 's/R//' | head -n 1`

    #add 1 to previous release to get next release
    NEXT_REL=`echo ${PREV_REL} | awk -F\. '{ print $1"."$2 + 1}'`

    #get names of all vhdl files from RCS directory
    RCS_FILES=`ls -l RCS/*.vhd,v | sed -e 's/^RCS\////' -e 's/,v//''`

    #create a new release directory and copy makefile from previous release
    mkdir R${NEXT_REL}
    cp R${PREV_REL}/makefile R${NEXT_REL}
    cd R${NEXT_REL}

    #link RCS into the new directory
    ln -s ../RCS .

    #create symbolic name tag that will be used to label files in RCS
    NAME_TAG=`echo ${NEXT_REL} | sed -e 's/\./_/'`
    #checkout all of the VHDL files from RCS and assign them symbolic name
    for FILE in ${RCS_FILES}; do
        rcs -NR${NAME_TAG}: ${FILE}
        co ${FILE}
    done

    #create vhdl library for new release
    vanlibcreate ${DESIGN}.lib ${DESIGN}.lib

    #modify the new makefile to update the pointers to the new release
    cd /user/proj
    sed -e "s/${DESIGN}\R[0-9]\.[0-9][0-9]*/${DESIGN}\R${NEXT_REL}/" \
        new_makefile > new_makefile.temp
    mv new_makefile.temp new_makefile
done

#perform compilations of all new releases
cd /user/proj
make -f new_makefile RELEASE

#If make is successful then make release visible
if [ $? != 0 ] ; then
    echo "ERROR: make failed"
else
    mv makefile old_makefile
    mv new_makefile makefile
fi

```

The script that controls the simulation versions uses the same makefile that was modified by the release script. This was done so that there would be only a single source that would identify all of the current releases in order to avoid the problem of having conflicting information or releases that are not seen by users. The startsim script, if executed without options, will first check to see if the user has the top level VHDL library in his or her local directory. If this library does not exist, startsim will create it, then execute the makefile. If the library does exist, startsim will check the date of the top level simulation model in that library against the date of the top level makefile. If the top level makefile is newer, it will inform the user that the compiled code in the library is out of date and give the user the option of recompiling to get the latest release of all of the models, or continuing to simulate on the model compiled in their library.

When executed with the "-test" option, startsim will replace the specified design with the test VHDL code that resides in the user's local directory. This allows code to be tested within the context of all of the released designs before it is released for others to use. In this implementation of startsim, the code is expected to reside within a subdirectory named "design" located beneath the directory where from which startsim was executed.

```

DESIGN_LIST=""
MAKEFILE="/user/proj/makefile"
TEST="FALSE"
while [ $# != 0 ] ; do
    if [ $1 = "-test" ] ; then
        TEST="TRUE"
        while [ $# != 1 -a -d /user/proj/$2 -a -d ./$2 ] ; do
            DESIGN_LIST="${DESIGN_LIST} $2"
            shift
        done
        if [ "${DESIGN_LIST}" = "" ] ; then
            echo "Error - no designs specified for test option"
            exit 1
        elif [ $# != 1 ] ; then
            echo "Error - $2 is not a valid design or directory"
            exit 1
        fi
        shift
    else
        echo "Error - invalid option"
        exit 1
    fi
done
# substitute a test design for a release design
if [ "${TEST}" = "TRUE" ] ; then
    MAKEFILE="makefile_test"
    for DESIGN in ${DESIGN_LIST}; do
        # modify makefile to substitute test design for release design
        sed -e "s/\/user\/proj\/${DESIGN}\/R[0-9]*.[0-9]*\/.\/${DESIGN}\/" \
            /user/proj/makefile > ${MAKEFILE}
    done
fi
if [ ! -d proj_top.lib ] ; then
    # create VHDL library for top level code and do make

```

```

        vanlibcreate proj_top.lib proj_lib
        make -f ${MAKEFILE} SIMULATE
else
    if [ `find /user/proj -name makefile -newer proj_top.lib` ] ; then
        echo "The list of release pointers has been updated since your"
        echo "last compile. Would you like to update your simulation to"
        echo "the most current release of all of the models? (y/n) [y]"
        read RESPONSE
        if [ "${RESPONSE}" = 'n' -o "${RESPONSE}" - 'N' ] ; then
            echo "WARNING!! You are simulating an old version!"
        else
            make -f ${MAKEFILE} SIMULATE
        fi
    fi
fi
# start simulator
vss &

```

Many other variations of startsim are possible with this scheme. For instance, in our environment, startsim accepts a flag that selects either code compiled with the debug option turned on or a streamlined version compiled with debug turned off. We also use the startsim script to control versions of non-VHDL files, such as testbench control scripts and ROM load files.

CONCLUSION

With design sizes always increasing, it is very important to select a design management scheme that is flexible while still providing good control over the data. It needs to be flexible in order to allow designers to release code at any time without affecting simulations that are already in progress and to allow experimental designs to be tested in the context of other released designs. Good control over the data is necessary in order to keep track of the many versions of each file and to keep the code in a central directory so that everyone on the team can access it.

It is possible to build such a design management scheme in Unix using basic utilities such as make and RCS with the addition of some short scripts. The scheme that is described in this paper meets the requirements of flexibility and controllability using only standard Unix commands. This scheme protects users from recompilations that may invalidate their simulations while giving them freedom to choose which versions, whether released or experimental, of each of the design modules they will use. Similar scripts to those presented here have been in use at Philips for nearly one year, and have saved the users countless hours of recompilation as well as the hassle of design units which are out of date.