Libero SoC Simulation Library Setup Instructions

Introduction

The purpose of this document is to describe the procedure to set up the simulation environment using a Libero SoC project as the input. This documentation corresponds to the pre-compiled libraries provided for use with Libero SoC v11.2 and newer software releases. The libraries provided are compiled for Verilog. VHDL users will require a license allowing mixed-mode simulation.

Compiled Simulation Libraries are provided for the following tools:

Aldec Active-HDL
Aldec Riviera-PRO
Cadence Incisive Enterprise
Mentor QuestaSim
Synopsys VCS

To request a library for a different simulator, please contact Microsemi SoC Tech Support at soc_tech@microsemi.com.

Libero SoC Integration

Libero SoC supports simulation using ModelSim ME by generating a run.do Tcl file. This file used by ModelSim ME to setup and run the simulation. To use other simulation tools you can generate the ModelSim ME run.do then modify the Tcl script file to use commands compatible with your simulator.

Libero SoC Tcl file generation

After creating and generating your design in Libero SoC, you must start a ModelSim ME simulation under all design phases (presynth, postsynth and postlayout). The purpose of this step is to force Libero SoC to generate the run.do Tcl file for ModelSim ME for each design phase. After starting each simulation run, you must rename the auto-generated run.do file under the simulation directory to prevent Libero SoC overwriting that file. For example, the files can be renamed presynth_run.do, postsynth_run.do and postlayout_run.do.

Using ModelSim PE/SE/DE

Click here to read KI8797 about Compiling SmartFusion Library for ModelSim Full Version (PE/SE/DE) Simulation. The same solution applies to SmartFusion2 and IGLOO2 libraries.

Aldec Setup for Active-HDL and Riviera-Pro

The Aldec simulators use Tcl files similar to ModelSim ME. The run.do Tcl files used by ModelSim ME can be modified and used for simulation using Aldec simulators. Below is a script file which converts the ModelSim ME run.do files to be compatible with Aldec simulators.

Environment Variable

Set your environment variable to your license file location:

LM_LICENSE_FILE: must include a pointer to the license server.

Download Compiled Library

Download the libraries for Aldec Active-HDL and Aldec Riviera-PRO from Microsemi's website.

Converting run.do for Aldec simulation

The following lists the Aldec-equivalent commands to modify in the ModelSim run.do Tcl file:

ModelSim	Active-HDL
Vlog	alog
Vcom	acom
Vlib	alib
Vsim	asim
Vmap	amap

1. Set the location of the current working directory.

```
set dsn <simulation directory>
```

2. Set a working library name and map its location. Also, map the location of Microsemi FPGA family precompiled libraries (for example, SmartFusion2) on which you are running your design.

```
alib presynth
amap presynth presynth
amap SmartFusion2 <location of the precompiled libraries>
```

3. Compile all the necessary HDL files used in the design with the required library.

```
alog –work presynth temp.v (for Verilog)
alog –work presynth testbench.v

acom –work presynth temp.vhd (for Vhdl)
acom –work presynth testbench.vhd
```

4. After compiling, simulate the design.

```
asim –L SmartFusion2 –L presynth –t 1ps presynth.testbench
run 10us
```

Known Issues

- Libraries compiled using Riviera-PRO are platform specific (i.e. 64-bit libraries cannot be run on 32-bit platform and vice versa.)
- For designs containing SERDES/MDDR/FDDR, use the following option in your run.do Tcl files while running simulations after compiling their designs:
    - Active-HDL: asim –o2
    - Riviera-PRO: asim –O2 (for presynth and postlayout simulations) and asim –O5 (for postlayout simulations)

Pending SARs. Contact Microsemi SoC Technical Support for more information:

SAR 49892 - Crash in Active-HDL while running MDDR BFM Simulations
SAR 49908 – Active-HDL: VHDL Error for Math block simulations
SAR 50627 – Riviera-PRO 2013.02: Simulation errors for SERDES designs
SAR 50461 – Riviera-PRO: asim -O2/-O5 option in simulations

Sample Tcl and shell script files

The script files below convert ModelSim run.do files into Aldec simulator compatible run.do files.

Script File Usage for Active-HDL

Place this script file in the Libero SoC simulation folder and click run.

Active-HDL:

```
perl active_hdl_parser.pl presynth_run.do postsynth_run.do
postlayout_run.do Microsemi_Family
Location_of_ActiveHDL_Precompiled_libraries
```

Active_hdl_parser_pl

```
#!/usr/bin/perl -w

#########################################################################################################################
#Usage: perl active_hdl_parser.pl presynth_run.do postsynth_run.do postlayout_run.do Microsemi_Family
Precompiled_Libraries_location#
#########################################################################################################################

use POSIX;
use strict;

my ($presynth, $postsynth, $postlayout, $family, $lib_location) = @ARGV;

&active_hdl_parser($presynth, $family, $lib_location);
&active_hdl_parser($postsynth, $family, $lib_location);
&active_hdl_parser($postlayout, $family, $lib_location);

sub active_hdl_parser {

my $ModelSim_run_do = $_[0];
my $actel_family = $_[1];
```

```
my $lib_location = $_[2];
my $state;

open (INFILE,"$ModelSim_run_do");
my @ModelSim_run_do = <INFILE>;
my $line;

if ( $ModelSim_run_do =~ m/(presynth)/)
{
open (OUTFILE,">presynth_Aldec.do");
$state = $1;
} elsif ( $ModelSim_run_do =~ m/(postsynth)/)
{
open (OUTFILE,">postsynth_Aldec.do");
$state = $1;
} elsif ( $ModelSim_run_do =~ m/(postlayout)/ )
{
open (OUTFILE,">postlayout_Aldec.do");
$state = $1;
} else
{
print "Wrong Inputs given to the file\n";
print "#Usage: perl active_hdl_parser.pl presynth_run.do postsynth_run.do postlayout_run.do
\"Libraries_location\"\n";
}
foreach $line (@ModelSim_run_do)
{
### General Operations ###
$line =~ s/quietly set PROJECT_DIR/set dsn/g;
$line =~ s/\$\{PROJECT_DIR\}/\$dsn/g;
$line =~ s/vlib/alib/;
$line =~ s/vmap/amap/;
$line =~ s/vcom/acom/;
$line =~ s/^vlog/alog/;
$line =~ s/vsim/asim/g;
$line =~ s/exit/endsim/g;
        if ( $line =~ m/(set\s+dsn.*)/)
        {
                print OUTFILE "$1 \n";
        } elsif ($line =~ m/^source/ )
        {
                print OUTFILE "$line ";
        } elsif ( $line =~ m/alib\s+.*($state)/)
        {
                print OUTFILE "alib $1_Aldec \n";
        } elsif ( $line =~ m/^amap/)
        {
                if ( $line =~ m/amap\s+(.*._LIB)\s+.*/ )
                {
                        print $1."\n";
                        print OUTFILE "alib $1 \n";
                        print OUTFILE "$line \n";
                } elsif ($line =~ m/amap\s+.*($state)/)
                {
                        $line =~ s/$state/$state\_Aldec/g;
                        print OUTFILE "$line \n";
                } elsif ($line =~ m/amap\s+.*($actel_family)/)
                {
```

```
                        print OUTFILE "alib $1 \n";
                        print OUTFILE "amap $1 \"$lib_location\"\n\n";
                }
        } elsif ( $line =~ m/(alog\s+.*?._LIB).*.(refresh)/ || $line =~ m/(acom\s+.*?._LIB).*.(refresh)/ )
        {
                print "\$1 = $1; \$2 = $2; \n";
                $line = $1;
                $line =~ s/\-\w+/-refresh/;
                print OUTFILE "$line \n";
        } elsif ( $line =~ m/^alog/ || $line =~ m/^acom/)
        {
                $line =~ s/$state/$state\_Aldec/g;
                print OUTFILE "$line \n";
        } elsif ( $line =~ m/^asim/ )
        {
                $line =~ s/$state/$state\_Aldec/g;
                print OUTFILE "$line \n";
        } elsif ( $line =~ m/(run.*)/ )
        {
                print OUTFILE "$1 \n";
        } elsif ( $line =~ m/endsim/ )
        {
                print OUTFILE "$line \n";
        }
}
close(INFILE);
close(OUTFILE);
}
```

Script File Usage for Riviera-PRO

Place this script file in the Libero SoC simulation folder and click run.

Riviera-PRO:

```
perl rivierapro_parser.pl presynth_run.do postsynth_run.do
postlayout_run.do Microsemi_Family
Location_of_RivieraPRO_Precompiled_libraries
```

Rivierapro_parser_pl

```
#!/usr/bin/perl -w
#############################################################################################################
#Usage: perl active_hdl_parser.pl presynth_run.do postsynth_run.do postlayout_run.do Microsemi_Family
Precompiled_Libraries_location#
#############################################################################################################
use POSIX;
use strict;

my ($presynth, $postsynth, $postlayout, $family, $lib_location, $folder_name ) = @ARGV;

&active_hdl_parser($presynth, $family, $lib_location, $folder_name);
&active_hdl_parser($postsynth, $family, $lib_location, $folder_name);
&active_hdl_parser($postlayout, $family, $lib_location, $folder_name);

sub active_hdl_parser {

my $ModelSim_run_do = $_[0];
my $actel_family = $_[1];
my $lib_location = $_[2];
my $folder = $_[3];
```

```
my $state;

if (-e "$ModelSim_run_do")
{
        open (INFILE,"$ModelSim_run_do");
        my @ModelSim_run_do = <INFILE>;
        my $line;

        if ( $ModelSim_run_do =~ m/(presynth)/)
        {
        `mkdir ALDEC_PRESYNTH`;
        open (OUTFILE,">ALDEC_PRESYNTH/presynth_Aldec.do");
        $state = $1;
        } elsif ( $ModelSim_run_do =~ m/(postsynth)/)
        {
        `mkdir ALDEC_POSTSYNTH`;
        open (OUTFILE,">ALDEC_POSTSYNTH/postsynth_Aldec.do");
        $state = $1;
        } elsif ( $ModelSim_run_do =~ m/(postlayout)/ )
        {
        `mkdir ALDEC_POSTLAYOUT`;
        open (OUTFILE,">ALDEC_POSTLAYOUT/postlayout_Aldec.do");
        $state = $1;
        } else
        {
        print "Wrong Inputs given to the file\n";
        print "#Usage: perl active_hdl_parser.pl presynth_run.do postsynth_run.do postlayout_run.do
\"Libraries_location\"\n";
        }

        foreach $line (@ModelSim_run_do)
        {
        ### General Operations ###
        $line =~ s/quietly set PROJECT_DIR/set dsn/g;
        $line =~ s/\$\{PROJECT_DIR\}/\$dsn/g;
        $line =~ s/vlib/alib/;
        $line =~ s/vmap/amap/;
        $line =~ s/vcom/acom/;
        $line =~ s/^vlog/alog/;
        $line =~ s/vsim/asim/g;
        $line =~ s/exit/endsim/g;

                if ( $line =~ m/(set\s+dsn.*)/)
                {
                        print OUTFILE "$1 \n";
        #       } elsif ($line =~ m/^source/ )
        #       {
        #               print OUTFILE "$line ";
                } elsif ( $line =~ m/alib\s+.*($state)/)
                {
                        print OUTFILE "alib $1_Aldec \n";
                } elsif ( $line =~ m/alib\s+ddr/)
                {
                        print OUTFILE "alib ddr \n";
                        print OUTFILE "amap ddr \"E\:\/WORK\/libs\/ddr\" \n";

                } elsif ( $line =~ m/^amap/)
                {
```

```
                        if ( $line =~ m/amap\s+(.*._LIB)\s+.*/ )
                        {
                                print $1."\n";
                                print OUTFILE "alib $1 \n";
                                $line =~ s/..\/component/..\/..\/component/g;
                                print OUTFILE "$line \n";
                        } elsif ($line =~ m/amap\s+.*($state)/)
                        {
                                $line =~ s/$state/$state\_Aldec/g;
                                print OUTFILE "$line \n";
                        } elsif ($line =~ m/amap\s+.*($actel_family)/)
                        {
                                print OUTFILE "alib $1 \n";
                                print OUTFILE "amap $1 \"$lib_location\"\n\n";
                                #print OUTFILE "alog -work $state\_Aldec \"\$dsn\/simulation\/smartfusion2.v\" ";
                        }
                } elsif ( $line =~ m/(alog\s+.*?._LIB).*.(refresh)/ || $line =~ m/(acom\s+.*?._LIB).*.(refresh)/
)
                {
                        print "\$1 = $1; \$2 = $2; \n";
                        $line = $1;
                        $line =~ s/\-\w+/-refresh/;
                        print OUTFILE "$line \n";
                } elsif ( $line =~ m/^alog/ || $line =~ m/^acom/)
                {
                        $line =~ s/$state/$state\_Aldec/g;
                        print OUTFILE "$line \n";
                } elsif ( $line =~ m/^asim/ && $state eq "postsynth" && ( $folder =~ m/DDR/ || $folder =~
m/SERDES/ || $folder =~ m/PI_Sim/ || $folder =~ m/ENVM/ ) )
                {
                        $line =~ s/$state/$state\_Aldec/g;
                        $line =~ s/asim.*wlf\"/asim /g;
                        $line =~ s/asim/asim -O5/g;
                        print OUTFILE "$line \n";

                } elsif ( $line =~ m/^asim/ && $state ne "postsynth" && ( $folder =~ m/DDR/ || $folder =~
m/SERDES/ || $folder =~ m/PI_Sim/ || $folder =~ m/ENVM/) )
                {
                        $line =~ s/$state/$state\_Aldec/g;
                        $line =~ s/asim.*wlf\"/asim /g;
                        $line =~ s/asim/asim -O2/g;
                        print OUTFILE "$line \n";

                } elsif ( $line =~ m/^asim/ )
                {
                        $line =~ s/$state/$state\_Aldec/g;
                        $line =~ s/asim.*wlf\"/asim /g;
                        #$line =~ s/asim/asim/g;
                        print OUTFILE "$line \n";

                } elsif ( $line =~ m/(run.*)/ )
                {
                        print OUTFILE "$1 \n";
                } elsif ( $line =~ m/endsim/ )
                {
                        print OUTFILE "$line \n";
                }
        }
```

```
        close(INFILE);
        close(OUTFILE);
}
}
```

Cadence Incisive Setup

You need to create a script file similar to the ModelSim ME run.do to run the Cadence Incisive simulator. You can follow the steps below and create script file for NCSim or use the script file provided below to convert the ModelSim ME run.do files into the configuration files needed to run simulations using NCSim.

Environment Variables

Required environment variables:

1. LM_LICENSE_FILE: must include a pointer to the license file.
2. cds_root: must point to the home directory location of Cadence Incisive Installation.
3. PATH: must point to the bin location under the tools directory pointed by cds_root (i.e. $cds_root/tools/bin/64bit(for a 64 bit machine and $cds_root/tools/bin for a 32 bit machine)

There are three ways of setting up the simulation environment in case of a switch between 64bit and 32 bit operating systems:

Case1: PATH Variable
set path = (*install_dir*/tools/bin/64bit $path) for 64bit machines and
set path = (*install_dir*/tools/bin $path) for 32bit machines

Case2: Using the -64bit Command-line Option
In the command line specify -64bit option in order to invoke the 64bit executable.

Case3: Setting the INCA_64BIT or CDS_AUTO_64BIT Environment Variable
The INCA_64BIT variable is treated as boolean. You can set this variable to any value or to a null string.

setenv INCA_64BIT

Note: The INCA_64BIT environment variable does not affect other Cadence tools, such as IC tools. However, for Incisive tools, the INCA_64BIT variable overrides the setting for the CDS_AUTO_64BIT environment variable. If the INCA_64BIT environment variable is set, all Incisive tools will be run in 64-bit mode.

setenv CDS_AUTO_64BIT INCLUDE:INCA

Note: The string INCA must be in uppercase. Because all executables must be run in either 32-bit mode or in 64-bit mode, do not set the variable to include one executable, as in the following:

setenv CDS_AUTO_64BIT INCLUDE:ncelab

Other Cadence tools, such as IC tools, also use the CDS_AUTO_64BIT environment variable to control the selection of 32-bit or 64-bit executables. The following table shows how you can set the CDS_AUTO_64BIT variable to run the Incisive tools and IC tools in all modes.

CDS_AUTO_64BIT Variable	Incisive Tools	IC Tools
setenv CDS_AUTO_64BIT ALL	64-bit	64-bit
setenv CDS_AUTO_64BIT NONE	32-bit	32-bit
setenv CDS_AUTO_64BIT EXCLUDE:*ic_binary*	64-bit	32-bit
setenv CDS_AUTO_64BIT EXCLUDE:INCA	32-bit	64-bit

Because all Incisive tools must be run in either 32-bit mode or in 64-bit mode, do not use EXCLUDE to exclude a specific executable, as in the following:

setenv CDS_AUTO_64BIT EXCLUDE:ncelab

Note: If you set the CDS_AUTO_64BIT variable to exclude the Incisive tools (setenv CDS_AUTO_64BIT EXCLUDE:INCA), all Incisive tools are run in 32-bit mode. However, the -64bit command-line option overrides the environment variable.

The following configuration files help you manage your data and control the operation of the simulation tools and utilities

- Library mapping file (cds.lib)—Defines a logical name for the location of your design
- Libraries and associates them with physical directory names.

- Variables file (hdl.var)—Defines variables that affect the behavior of simulation tools and utilities

Download Compiled Library

Download the libraries for Cadence Incisive from Microsemi's website.

Creating the NCSim script file

After creating a copy of the run.do Tcl files, please do the following steps in order to run your simulation using NCSim.

1. **Create a cds.lib file** that defines which libraries are accessible and where they are located. The file contains statements that map library logical names to their physical directory paths. For example if you are running presynth simulation, the cds.lib file can be written as:

   DEFINE presynth ./presynth
   DEFINE COREAHBLITE_LIB ./COREAHBLITE_LIB
   DEFINE smartfusion2 <location of Smartfusion2 precompiled libraries on disk>

2. **Create a hdl.var file** which is an optional configuration file that contains configuration variables, which determine how your design environment is configured.
   These include:
   - Variables used to specify the work library where the compiler stores compiled objects and other derived data.
   - For Verilog, variables (LIB_MAP, VIEW_MAP, WORK) that are used to specify the libraries and views to search when the elaborator resolves instances.
   - Variables that allow you to define compiler, elaborator, and simulator command-line options and arguments.

   In case of presynth simulation example shown above, say we have 3 RTL files a.v, b.v, testbench.v which needs to be compiled into presynth, COREAHBLITE_LIB and presynth libraries respectively. The hdl.var file can be written as:

   DEFINE WORK presynth
   DEFINE PROJECT_DIR <location of the files>

   DEFINE LIB_MAP ($LIB_MAP, ${PROJECT_DIR}/a.v => presynth)
   DEFINE LIB_MAP ($LIB_MAP, ${PROJECT_DIR}/b.v => COREAHBLITE_LIB)
   DEFINE LIB_MAP ($LIB_MAP, ${PROJECT_DIR}/testbench.v => presynth)
   DEFINE LIB_MAP ($LIB_MAP, + => presynth)

3. **Compile the design files using ncvlog option.**

   ncvlog +incdir+<testbench directory> –cdslib ./cds.lib –hdlvar ./hdl.var –logfile ncvlog.log –update –linedebug a.v b.v testbench.v

4. **Elaborate the design using ncelab:** The elaborator constructs a design hierarchy based on the instantiation and configuration information in the design, establishes signal connectivity, and computes initial values for all objects in the design. The elaborated design hierarchy is stored in a simulation snapshot, which is the representation of your design that the simulator uses to run the simulation.

   ncelab –Message –cdslib ./cds.lib –hdlvar ./hdl.var –logfile ncelab.log –errormax 15 –access +rwc –status worklib.<name of testbench module>:module

   Elaboration during postlayout simulation
   In case of postlayout simulations, first the SDF file needs to be compiled before elaboration using **ncsdfc** command.

   ncsdfc <filename>.sdf –output <filename>.sdf.X

   During elaboration use the compiled SDF output with –autosdf option as follows:

   ncelab -autosdf –Message –cdslib ./cds.lib –hdlvar ./hdl.var –logfile ncelab.log –errormax 15 –access +rwc –status worklib.<name of testbench module>:module –sdf_cmd_file ./sdf_cmd_file

   The sdf_cmd_file should be as follows

   COMPILED_SDF_FILE = "<location of compiled SDF file>"

5. **Simulating using ncsim:** After elaboration a simulation snapshot is created which is loaded by ncsim for simulation. This can be run in batch mode or GUI mode.

ncsim –Message –batch/-gui –cdslib ./cds.lib –hdlvar ./hdl.var –logfile ncsim.log –errormax 15 –status worklib.<testbench module name>:module

6. **Using ncverilog or irun:** All the above three steps of compiling, elaborating and simulating can be put into a shell script file and sourced from command line. Instead of using these three steps design can be simulated in one step using ncverilog or irun option as follows:

   ncverilog +incdir+<testbench location> -cdslib ./cds.lib –hdlvar ./hdl.var <all RTL files used in the design>
   irun +incdir+<testbench location> -cdslib ./cds.lib –hdlvar ./hdl.var <all RTL files used in the design>

Known Issues

Testbench Workaround:

Using the following statement for specifying the clock frequency in the testbench generated by user or the default testbench generated by Libero SoC does not work with NCSim.

```
always @(SYSCLK)
#(SYSCLK_PERIOD / 2.0) SYSCLK <= !SYSCLK;
```

Modify as follows to run simulation:

```
always #(SYSCLK_PERIOD / 2.0) SYSCLK = ~SYSCLK;
```

Compiled libraries for NCSim are platform specific (i.e. 64bit libraries are not compatible on 32bit platform and vice versa.)

Postsynth and Postlayout Simulations using MSS and SERDES:

While running postsynth simulations of designs containing MSS block, or postlayout simulations of designs using SERDES, the BFM simulations do not work if –libmap option is not specified during elaboration. This is because during elaboration, MSS is resolved from the work library (because of the default binding and the worklib being postsynth/postlayout) where it is just a black box.

The ncelab command should be written as follows in order to resolve the MSS block from smartfusion2 precompiled library.

*ncelab **-libmap lib.map** -libverbose -Message -access +rwc **cfg1***

and the lib.map file should be as follows:

```
config cfg1;
  design <testbench_module_name>;
  default liblist smartfusion2 <worklib>;
endconfig
```

This will resolve any cell in the smartfusion2 library before looking in the work library i.e. postsynth/postlayout.

The –libmap option can be used by default during elaboration for every simulation(presynth, postsynth and postlayout). This will avoid simulation issues that are caused due to resolution of instances from libraries.

ncelab: *F,INTERR: INTERNAL EXCEPTION:

This ncelab tool exception is a caveat for designs containing FDDR in Smartfusion2 and IGLOO2 during postsynth and postlayout simulations using –libmap option. This issue has been reported to Cadence support team (SAR 52113).

Sample Tcl and shell script files

The files below are the configuration files needed for setting up the design and shell script file for running NCSim commands.

Cds.lib

```
DEFINE smartfusion2 /scratch/krydor/tmpspace/users/me/nc-vlog64/SmartFusion2
DEFINE COREAHBLITE_LIB ./COREAHBLITE_LIB
DEFINE presynth ./presynth
```

Hdl.var

```
DEFINE WORK presynth
DEFINE PROJECT_DIR
/scratch/krydor/tmpspace/sqausers/me/3rd_party_simulators/Cadence/IGLOO2/ENVM/M2GL050/envm_fic1_ser1_v/eNVM_fab_
master
```

```
DEFINE LIB_MAP ( $LIB_MAP,
${PROJECT_DIR}/component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_addrdec.v =>
COREAHBLITE_LIB )
DEFINE LIB_MAP ( $LIB_MAP,
${PROJECT_DIR}/component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_defaultslavesm.v =>
COREAHBLITE_LIB )
DEFINE LIB_MAP ( $LIB_MAP,
${PROJECT_DIR}/component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_masterstage.v =>
COREAHBLITE_LIB )
DEFINE LIB_MAP ( $LIB_MAP,
${PROJECT_DIR}/component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_slavearbiter.v =>
COREAHBLITE_LIB )
DEFINE LIB_MAP ( $LIB_MAP,
${PROJECT_DIR}/component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_slavestage.v =>
COREAHBLITE_LIB )
DEFINE LIB_MAP ( $LIB_MAP,
${PROJECT_DIR}/component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_matrix2x16.v =>
COREAHBLITE_LIB )
DEFINE LIB_MAP ( $LIB_MAP,
${PROJECT_DIR}/component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite.v => COREAHBLITE_LIB )
DEFINE LIB_MAP ( $LIB_MAP, ${PROJECT_DIR}/component/work/SB/CCC_0/SB_CCC_0_FCCC.v => presynth )
DEFINE LIB_MAP ( $LIB_MAP,
${PROJECT_DIR}/component/Actel/DirectCore/CoreConfigMaster/2.0.101/rtl/vlog/core/coreconfigmaster.v => presynth
)
DEFINE LIB_MAP ( $LIB_MAP,
${PROJECT_DIR}/component/Actel/DirectCore/CoreConfigP/4.0.100/rtl/vlog/core/coreconfigp.v => presynth )
DEFINE LIB_MAP ( $LIB_MAP,
${PROJECT_DIR}/component/Actel/DirectCore/CoreResetP/5.0.103/rtl/vlog/core/coreresetp_pcie_hotreset.v =>
presynth )
DEFINE LIB_MAP ( $LIB_MAP,
${PROJECT_DIR}/component/Actel/DirectCore/CoreResetP/5.0.103/rtl/vlog/core/coreresetp.v => presynth )
DEFINE LIB_MAP ( $LIB_MAP, ${PROJECT_DIR}/component/work/SB/FABOSC_0/SB_FABOSC_0_OSC.v => presynth )
DEFINE LIB_MAP ( $LIB_MAP, ${PROJECT_DIR}/component/work/SB_HPMS/SB_HPMS.v => presynth )
DEFINE LIB_MAP ( $LIB_MAP, ${PROJECT_DIR}/component/work/SB/SB.v => presynth )
DEFINE LIB_MAP ( $LIB_MAP, ${PROJECT_DIR}/component/work/SB_top/SERDES_IF_0/SB_top_SERDES_IF_0_SERDES_IF.v =>
presynth )
DEFINE LIB_MAP ( $LIB_MAP, ${PROJECT_DIR}/component/work/SB_top/SB_top.v => presynth )
DEFINE LIB_MAP ( $LIB_MAP, ${PROJECT_DIR}/component/work/SB_top/testbench.v => presynth )
DEFINE LIB_MAP ( $LIB_MAP, + => presynth )
```

Commands.csh

```
ncvlog +incdir+../../component/work/SB_top -cdslib ./cds.lib -hdlvar ./hdl.var -logfile ncvlog.log -errormax 15
-update -linedebug ../../component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_addrdec.v
../../component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_defaultslavesm.v
../../component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_masterstage.v
../../component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_slavearbiter.v
../../component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_slavestage.v
../../component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite_matrix2x16.v
../../component/Actel/DirectCore/CoreAHBLite/4.0.100/rtl/vlog/core/coreahblite.v
../../component/work/SB/CCC_0/SB_CCC_0_FCCC.v
../../component/Actel/DirectCore/CoreConfigMaster/2.0.101/rtl/vlog/core/coreconfigmaster.v
../../component/Actel/DirectCore/CoreConfigP/4.0.100/rtl/vlog/core/coreconfigp.v
../../component/Actel/DirectCore/CoreResetP/5.0.103/rtl/vlog/core/coreresetp_pcie_hotreset.v
../../component/Actel/DirectCore/CoreResetP/5.0.103/rtl/vlog/core/coreresetp.v
../../component/work/SB/FABOSC_0/SB_FABOSC_0_OSC.v ../../component/work/SB_HPMS/SB_HPMS.v
../../component/work/SB/SB.v ../../component/work/SB_top/SERDES_IF_0/SB_top_SERDES_IF_0_SERDES_IF.v
../../component/work/SB_top/SB_top.v ../../component/work/SB_top/testbench.v


ncelab -Message -cdslib ./cds.lib -hdlvar ./hdl.var -work presynth -logfile ncelab.log -errormax 15 -access +rwc
-status presynth.testbench:module


ncsim -Message -batch -cdslib ./cds.lib -hdlvar ./hdl.var -logfile ncsim.log -errormax 15 -status
presynth.testbench:module
```

Automation

The script file below converts ModelSim run.do files into configuration files needed to run simulations using NCSim.

Script File Usage

```
perl cadence_parser.pl presynth_run.do postsynth_run.do
postlayout_run.do Microsemi_Family
Location_of_Cadence_Precompiled_libraries
```

Cadence_parser.pl

```
#!/usr/bin/perl -w

##################################################################################################################
#Usage: perl questa_parser.pl presynth_run.do postsynth_run.do postlayout_run.do Microsemi_Family
Precompiled_Libraries_location#
##################################################################################################################

use POSIX;
use strict;

my ($presynth, $postsynth, $postlayout, $family, $lib_location) = @ARGV;

&questa_parser($presynth, $family, $lib_location);
&questa_parser($postsynth, $family, $lib_location);
&questa_parser($postlayout, $family, $lib_location);

sub questa_parser {

my $ModelSim_run_do = $_[0];
my $actel_family = $_[1];
my $lib_location = $_[2];
my $state;

if ( -e "$ModelSim_run_do" )
{

open (INFILE,"$ModelSim_run_do");
my @ModelSim_run_do = <INFILE>;
my $line;

if ( $ModelSim_run_do =~ m/(presynth)/)
{
`mkdir QUESTA_PRESYNTH`;
open (OUTFILE,">QUESTA_PRESYNTH/presynth_questa.do");
$state = $1;
} elsif ( $ModelSim_run_do =~ m/(postsynth)/)
{
`mkdir QUESTA_POSTSYNTH`;
open (OUTFILE,">QUESTA_POSTSYNTH/postsynth_questa.do");
$state = $1;
} elsif ( $ModelSim_run_do =~ m/(postlayout)/ )
{
`mkdir QUESTA_POSTLAYOUT`;
open (OUTFILE,">QUESTA_POSTLAYOUT/postlayout_questa.do");
$state = $1;
} else
{
print "Wrong Inputs given to the file\n";
```

```
print "#Usage: perl questa_parser.pl presynth_run.do postsynth_run.do postlayout_run.do
\"Libraries_location\"\n";
}

foreach $line (@ModelSim_run_do)
{
        #General Operations
        $line =~ s/..\/designer.*simulation\///g;
        $line =~ s/$state/$state\_questa/g;
        #print OUTFILE "$line \n";

        if ($line =~ m/vmap\s+.*($actel_family)/)
        {
                print OUTFILE "vmap $actel_family \"$lib_location\"\n";
        } elsif ($line =~ m/vmap\s+(.*._LIB)/)
        {
                $line =~ s/..\/component/..\/..\/component/g;
                print OUTFILE "$line \n";
        } elsif ($line =~ m/vsim/)
        {
                $line =~ s/vsim/vsim -novopt/g;
                print OUTFILE "$line \n";
        } else
        {
                print OUTFILE "$line \n";
        }
}
close(INFILE);
close(OUTFILE);
} else {
print "$ModelSim_run_do does not exist. Rerun simulation again \n";
}
}
```

Mentor Graphics QuestaSim Setup

The run.do Tcl files generated by Libero SoC for simulations using ModelSim Microsemi Editions can be used for simulations using QuestaSim with a single change. In the ModelSim ME run.do Tcl file the precompiled libraries location needs to be modified.

Environment variables

Required environment variables:

LM_LICENSE_FILE: must include the path to the license file
MODEL_TECH: must identify the path to the home directory location of QuestaSim installation
PATH: must point to the executable location pointed by MODEL_TECH

Converting run.do for Mentor QuestaSim

The run.do Tcl files generated by Libero SoC for simulations using ModelSim Microsemi Editions can be used for simulations using QuestaSim with a single change.

Note:
All the designs which are simulated using QuestaSim must include -novopt option along with vsim command in the run.do TCL script files.

Download Compiled Library

Download the libraries for Mentor Graphics QuestaSim from Microsemi's website.

Sample Tcl and shell script files

The script files here convert the ModelSim ME run.do files into QuestaSim compatible run.do files.

Script File Usage

```
perl questa_parser.pl presynth_run.do postsynth_run.do
postlayout_run.do Microsemi_Family
Location_of_Questasim_Precompiled_libraries
```

Questa_parser_pl

```
#!/usr/bin/perl -w
##################################################################################################################
#Usage: perl questa_parser.pl presynth_run.do postsynth_run.do postlayout_run.do Microsemi_Family
Precompiled_Libraries_location#
##################################################################################################################

use POSIX;
use strict;

my ($presynth, $postsynth, $postlayout, $family, $lib_location) = @ARGV;

&questa_parser($presynth, $family, $lib_location);
&questa_parser($postsynth, $family, $lib_location);
&questa_parser($postlayout, $family, $lib_location);

sub questa_parser {

my $ModelSim_run_do = $_[0];
my $actel_family = $_[1];
my $lib_location = $_[2];
my $state;

if ( -e "$ModelSim_run_do" )
{
open (INFILE,"$ModelSim_run_do");
my @ModelSim_run_do = <INFILE>;
my $line;

if ( $ModelSim_run_do =~ m/(presynth)/)
{
`mkdir QUESTA_PRESYNTH`;
open (OUTFILE,">QUESTA_PRESYNTH/presynth_questa.do");
$state = $1;
} elsif ( $ModelSim_run_do =~ m/(postsynth)/)
{
`mkdir QUESTA_POSTSYNTH`;
open (OUTFILE,">QUESTA_POSTSYNTH/postsynth_questa.do");
$state = $1;
} elsif ( $ModelSim_run_do =~ m/(postlayout)/ )
{
`mkdir QUESTA_POSTLAYOUT`;
open (OUTFILE,">QUESTA_POSTLAYOUT/postlayout_questa.do");
$state = $1;
} else
{
print "Wrong Inputs given to the file\n";
print "#Usage: perl questa_parser.pl presynth_run.do postsynth_run.do postlayout_run.do
\"Libraries_location\"\n";
}
```

```
foreach $line (@ModelSim_run_do)
{
        #General Operations
        $line =~ s/..\/designer.*simulation\///g;
        $line =~ s/$state/$state\_questa/g;
        #print OUTFILE "$line \n";

        if ($line =~ m/vmap\s+.*($actel_family)/)
        {
                print OUTFILE "vmap $actel_family \"$lib_location\"\n";
        } elsif ($line =~ m/vmap\s+(.*._LIB)/)
        {
                $line =~ s/..\/component/..\/..\/component/g;
                print OUTFILE "$line \n";
        } elsif ($line =~ m/vsim/)
        {
                $line =~ s/vsim/vsim -novopt/g;
                print OUTFILE "$line \n";
        } else
        {
                print OUTFILE "$line \n";
        }
}
close(INFILE);
close(OUTFILE);
} else {
print "$ModelSim_run_do does not exist. Rerun simulation again \n";
}
}
```

Synopsys VCS Setup

The flow recommended by Microsemi relies on the Elaborate and Compile flow in VCS. This document includes a script file that uses the run.do Tcl script files generated by Libero SoC and generates the setup files needed for VCS simulation. The script file uses the run.do Tcl file to:

1. Create a library mapping file, which is done using the synopsys_sim.setup file located in the same directory where VCS simulation is running.
2. Create a shell script file to elaborate and compile your design using VCS.

Environment variables

Set the appropriate environment variables for VCS based on your setup. The environment variables needed as per the VCS documentation are:

LM_LICENSE_FILE: must include a pointer to the license server.
VCS_HOME: must point to the home directory location of the VCS installation.
PATH: must include a pointer to the bin directory below the VCS_HOME directory.

Download Compiled Library

Download the libraries for Synopsys VCS from Microsemi's website.

VCS simulation script file

After setting up VCS and generating the design and the different run.do Tcl files from Libero SoC you must:

1. Create the library mapping file synopsys_sim.setup; this file contains pointers to the location of all the libraries to be used by the design. Please note that the file name must not change and it must be located in the same directory where simulation is running. Here is an example for such a file for pre-synthesis simulation:

```
WORK > DEFAULT
SmartFusion2 : <location of the SmartFusion2 pre-compiled libraries>
presynth : ./presynth
DEFAULT : ./work
```

2. Elaborate the different design files, including the testbench, using the vlogan command in VCS. These commands may be included in a shell script file. Here is an example of the commands needed to elaborate a design defined in rtl.v with its testbench defined in testbench.v:

```
vlogan +v2k -work presynth rtl.v
vlogan +v2k -work presynth testbench.v
```

3. You can then compile the design using VCS using the following command:

```
vcs –sim_res=1fs presynth.testbench
```

Please note that the timing resolution of simulation must be set to 1fs for correct functional simulation.

4. Once the design is compiled, you can start simulation using the command:

```
./simv
```

5. For back-annotated simulation, the VCS command must be as follows:

```
vcs postlayout.testbench –sim_res=1fs –sdf max:<testbench_module_name>.<DUT instance name>:<sdf file path> –gui
–l postlayout.log
```

Limitations/Exceptions

1. VCS simulations can be run only for Verilog projects of Libero SoC. The VCS simulator has strict VHDL language requirements that are not met by the Libero SoC auto-generated VHDL files.
2. You must have a $finish statement in the Verilog testbench to stop the simulation whenever you want to. When simulations are run in GUI mode, run time can be specified in the GUI.

Sample Tcl and shell script files

The Perl below automates the generation of the synopsys_sim.setup file as well as the corresponding shell script files needed to elaborate, compile, and simulate the design.

If the design uses an MSS, you must copy the test.vec file located in the simulation folder of the Libero SoC project into the VCS simulation folder.

Some sample run.do Tcl files generated by Libero SoC are attached below, including the corresponding library mapping and shell script files needed for VCS simulation.

Pre-synthesis

Presynth_run.do

```
quietly set ACTELLIBNAME SmartFusion2
quietly set PROJECT_DIR "/sqa/users/me/VCS_Tests/Test_DFF"

if {[file exists presynth/_info]} {
   echo "INFO: Simulation library presynth already exists"
} else {
   vlib presynth
}
vmap presynth presynth
vmap SmartFusion2 "/captures/lin/11_0_0_23_11prod/lib/ModelSim/precompiled/vlog/smartfusion2"

vlog  -work presynth "${PROJECT_DIR}/component/work/SD1/SD1.v"
vlog "+incdir+${PROJECT_DIR}/stimulus"  -work presynth "${PROJECT_DIR}/stimulus/SD1_TB1.v"

vsim -L SmartFusion2 -L presynth  -t 1fs presynth.SD1_TB1
add wave /SD1_TB1/*
add log -r /*
```

```
run 1000ns
```

presynth_main.csh

```
#!/bin/csh -f

set PROJECT_DIR = "/sqa/users/Me/VCS_Tests/Test_DFF"

/cad_design/tools/vcs.dir/E-2011.03/bin/vlogan +v2k  -work presynth "${PROJECT_DIR}/component/work/SD1/SD1.v"
/cad_design/tools/vcs.dir/E-2011.03/bin/vlogan +v2k "+incdir+${PROJECT_DIR}/stimulus"  -work presynth
"${PROJECT_DIR}/stimulus/SD1_TB1.v"

/cad_design/tools/vcs.dir/E-2011.03/bin/vcs -sim_res=1fs presynth.SD1_TB1 -l compile.log
./simv -l run.log
```

Synopsys_sim.setup

```
WORK > DEFAULT
SmartFusion2 : /VCS/SmartFusion2
presynth : ./presynth
DEFAULT : ./work
```

Post-synthesis

postsynth_run.do

```
quietly set ACTELLIBNAME SmartFusion2
quietly set PROJECT_DIR "/sqa/users/Me/VCS_Tests/Test_DFF"


if {[file exists postsynth/_info]} {
   echo "INFO: Simulation library postsynth already exists"
} else {
   vlib postsynth
}
vmap postsynth postsynth
vmap SmartFusion2 "//idm/captures/pc/11_0_1_12_g4x/Designer/lib/ModelSim/precompiled/vlog/SmartFusion2"

vlog  -work postsynth "${PROJECT_DIR}/synthesis/SD1.v"
vlog "+incdir+${PROJECT_DIR}/stimulus"  -work postsynth "${PROJECT_DIR}/stimulus/SD1_TB1.v"

vsim -L SmartFusion2 -L postsynth  -t 1fs postsynth.SD1_TB1
add wave /SD1_TB1/*
add log -r /*
run 1000ns
log SD1_TB1/*
exit
```

Postsynth_main_ch.txt

```
#!/bin/csh -f

set PROJECT_DIR = "/sqa/users/Me/VCS_Tests/Test_DFF"

/cad_design/tools/vcs.dir/E-2011.03/bin/vlogan +v2k  -work postsynth "${PROJECT_DIR}/synthesis/SD1.v"
/cad_design/tools/vcs.dir/E-2011.03/bin/vlogan +v2k "+incdir+${PROJECT_DIR}/stimulus"  -work postsynth
"${PROJECT_DIR}/stimulus/SD1_TB1.v"
/cad_design/tools/vcs.dir/E-2011.03/bin/vcs -sim_res=1fs postsynth.SD1_TB1 -l compile.log
./simv -l run.log
```

Synopsys_sim.setup

```
WORK > DEFAULT
```

```
SmartFusion2 : /VCS/SmartFusion2
postsynth : ./postsynth
DEFAULT : ./work
```

Post-layout

postlayout_run.do

```
quietly set ACTELLIBNAME SmartFusion2
quietly set PROJECT_DIR "E:/ModelSim_Work/Test_DFF"

if {[file exists ../designer/SD1/simulation/postlayout/_info]} {
   echo "INFO: Simulation library ../designer/SD1/simulation/postlayout already exists"
} else {
   vlib ../designer/SD1/simulation/postlayout
}
vmap postlayout ../designer/SD1/simulation/postlayout
vmap SmartFusion2 "//idm/captures/pc/11_0_1_12_g4x/Designer/lib/ModelSim/precompiled/vlog/SmartFusion2"

vlog  -work postlayout "${PROJECT_DIR}/designer/SD1/SD1_ba.v"
vlog "+incdir+${PROJECT_DIR}/stimulus"  -work postlayout "${PROJECT_DIR}/stimulus/SD1_TB1.v"

vsim -L SmartFusion2 -L postlayout  -t 1fs -sdfmax /SD1_0=${PROJECT_DIR}/designer/SD1/SD1_ba.sdf
postlayout.SD1_TB1
add wave /SD1_TB1/*
add log -r /*
run 1000ns
```

Postlayout_main_csh

```
#!/bin/csh -f

set PROJECT_DIR = "/VCS_Tests/Test_DFF"

/cad_design/tools/vcs.dir/E-2011.03/bin/vlogan +v2k  -work postlayout "${PROJECT_DIR}/designer/SD1/SD1_ba.v"
/cad_design/tools/vcs.dir/E-2011.03/bin/vlogan +v2k "+incdir+${PROJECT_DIR}/stimulus"  -work postlayout
"${PROJECT_DIR}/stimulus/SD1_TB1.v"

/cad_design/tools/vcs.dir/E-2011.03/bin/vcs -sim_res=1fs postlayout.SD1_TB1 -sdf
max:SD1_TB1.SD1_0:${PROJECT_DIR}/designer/SD1/SD1_ba.sdf -l compile.log

./simv -l run.log
```

Synopsys_sim.setup

```
WORK > DEFAULT
SmartFusion2 : /VCS/SmartFusion2
postlayout : ./postlayout
DEFAULT : ./work
```

Automation

The flow can be automated using the Perl script file below to convert the ModelSim run.do Tcl files into VCS compatible shell script files, create proper directories inside the Libero SoC simulation directory, and then run simulations.

Run this script file using the following syntax:

```
perl vcs_parse.pl presynth_run.do postsynth_run.do postlayout_run.do
```

Vcs_parse_pl

```
#!/usr/bin/perl -w

##########################################################################
```

```
#
#Usage: perl vcs_parse.pl presynth_run.do postsynth_run.do postlayout_run.do
#
################################################################################

my ($presynth, $postsynth, $postlayout) = @ARGV;

if(system("mkdir VCS_Presynth")) {print "mkdir failed:\n";}
if(system("mkdir VCS_Postsynth")) {print "mkdir failed:\n";}
if(system("mkdir VCS_Postlayout")) {print "mkdir failed:\n";}

chdir(VCS_Presynth);
`cp ../$ARGV[0] .` ;
&parse_do($presynth,"presynth");
chdir ("../");

chdir(VCS_Postsynth);
`cp ../$ARGV[1] .` ;
&parse_do($postsynth,"postsynth");
chdir ("../");

chdir(VCS_Postlayout);
`cp ../$ARGV[2] .` ;
&parse_do($postlayout,"postlayout");
chdir ("../");

sub parse_do {

        my $vlog = "/cad_design/tools/vcs.dir/E-2011.03/bin/vlogan +v2k" ;

        my %LIB = ();

        my $file = $_[0] ;
        my $state = $_[1];

        open(INFILE,"$file") || die "Cant open File Reason might be:$!";

        if ( $state eq "presynth" )
        {
                open(OUT1,">presynth_main.csh") || die "Cant create Command File Reason might be:$!";
        }
        elsif ( $state eq "postsynth" )
        {
                open(OUT1,">postsynth_main.csh") || die "Cant create Command File Reason might be:$!";
        }
        elsif ( $state eq "postlayout" )
        {
                open(OUT1,">postlayout_main.csh") || die "Cant create Command File Reason might be:$!";
        }
        else
        {
                print "Simulation State is missing  \n" ;
        }

        open(OUT2,">synopsys_sim.setup") || die "Cant create Command File Reason might be:$!";

        # .csh file
```

```
	print OUT1 "#!/bin/csh -f\n\n\n" ;

	#SET UP FILE

	print OUT2 "WORK > DEFAULT\n" ;
	print OUT2 "SmartFusion2 : /sqa/users/Aditya/VCS/SmartFusion2\n" ;

	while ($line = <INFILE>)
	{

			if ($line =~ m/quietly set PROJECT_DIR\s+\"(.*?)\"/)
			{
				print OUT1 "set PROJECT_DIR = \"$1\"\n\n\n" ;

			}
			elsif ( $line =~ m/vlog.*\.v\"/ )
			{
					if ($line =~ m/\s+(\w*?)\_LIB/)
					{
						#print "\$1 =$1 \n" ;
						$temp = "$1"."_LIB";
						#print "Temp = $temp \n" ;
						$LIB{$temp}++;
					}
					chomp($line);
					$line =~ s/^vlog/$vlog/ ;
					$line =~ s/
//g;
					print OUT1 "$line\n";

			}
			elsif ( ($line =~ m/vsim.*presynth\.(.*)/) || ($line =~ m/vsim.*postsynth\.(.*)/) ||
($line =~ m/vsim.*postlayout\.(.*)/) )
			{
				$tb = $1 ;
				$tb =~ s/
//g;
				chomp($tb);
				#print "TB Name : $tb \n";
				if ( $line =~ m/sdf(.*)\.sdf/)
				{
					chomp($line);
					$line = $1 ;
					#print "LINE : $line \n" ;
					if ($line =~ m/max/)
					{
						$line =~ s/max \/// ;
						$line =~ s/=/:/;

						print OUT1 "\n\n/cad_design/tools/vcs.dir/E-2011.03/bin/vcs -
sim_res=1fs postlayout.$tb -sdf max:$tb.$line.sdf -l compile.log\n" ;
					}
					elsif ($line =~ m/min/)
					{
						$line =~ s/min \/// ;
						$line =~ s/=/:/;
						print OUT1 "\n\n/cad_design/tools/vcs.dir/E-2011.03/bin/vcs -
sim_res=1fs postlayout.$tb -sdf min:$tb.$line.sdf -l compile.log\n" ;
```

```
                                        }
                                        elsif ($line =~ m/typ/)
                                        {
                                                $line =~ s/typ \/// ;
                                                $line =~ s/=/:/;
                                                print OUT1 "\n\n/cad_design/tools/vcs.dir/E-2011.03/bin/vcs -
sim_res=1fs postlayout.$tb -sdf typ:$tb.$line.sdf -l compile.log\n" ;
                                        }
                                        #-sdfmax /M3_FIC32_0=${PROJECT_DIR}/designer/M3_FIC32/M3_FIC32_ba.sdf
-- ModelSim SDF format
                                        #$sdf = "-sdf
max:testbench.M3_FIC32_0:${PROJECT_DIR}/designer/M3_FIC32/M3_FIC32_ba.sdf";   --VCS SDF format
                                }
                        }
        }
        print OUT1 "\n\n" ;

        if ( $state eq "presynth" )
        {
                print OUT2 "presynth : ./presynth\n" ;
                print OUT1 "/cad_design/tools/vcs.dir/E-2011.03/bin/vcs -sim_res=1fs presynth.$tb -l
compile.log\n" ;
        }
        elsif ( $state eq "postsynth" )
        {
                print OUT2 "postsynth : ./postsynth\n" ;
                print OUT1 "/cad_design/tools/vcs.dir/E-2011.03/bin/vcs -sim_res=1fs postsynth.$tb -l
compile.log\n" ;
        }
        elsif ( $state eq "postlayout" )
        {
                print OUT2 "postlayout : ./postlayout\n" ;
        }
        else
        {
                print "Simulation State is missing  \n" ;
        }
        foreach $i ( keys %LIB)
        {
                #print "Key : $i Value : $LIB{$i} \n" ;
                print OUT2 "$i : ./$i\n" ;
        }
        print OUT1 "\n\n" ;
        print OUT1 "./simv -l run.log\n" ;
        print OUT2 "DEFAULT : ./work\n" ;

        close INFILE;
        close OUT1;
        close OUT2;
}
```