Libero SoC Simulation Library Setup Instructions

Introduction

This document describes 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 v11.0 and newer software releases. The libraries provided are compiled for Verilog. VHDL users will require a license allowing mixed-mode simulation.

Libraries are provided for the following simulation tools:

Aldec Active-HDL
Aldec Riviera-PRO

Cadence Incisive Enterprise
Mentor QuestaSim
Synopsys VCS

To request a library for a different simulator, contact soc_tech@microsemi.com.

Download Libero SoC Simulation Library

Libero SoC Integration

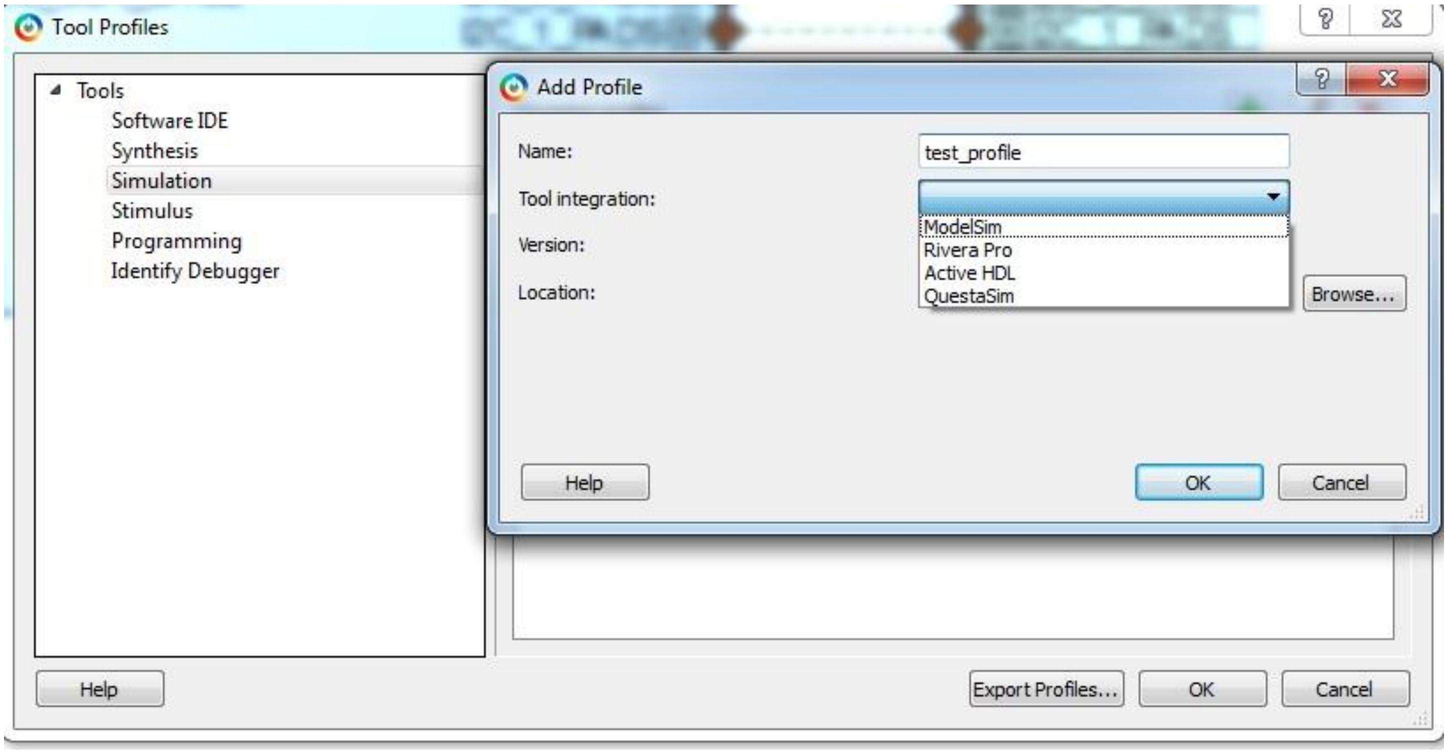
Libero SoC supports simulation using ModelSim ME by generating a run.do Tcl file. This file used by ModelSim ME to set up and run the simulation. To use other simulation tools you must modify the run.do Tcl script to commands compatible with your simulator.

Third Party Tool Profile Setup through Libero

Libero provides support to add tool profiles for Aldec and Questa simulators similar to what we currently have for ModelSim. Libero automatically creates the run.do file required for running the simulations (when the tool is invoked from the design flow window).

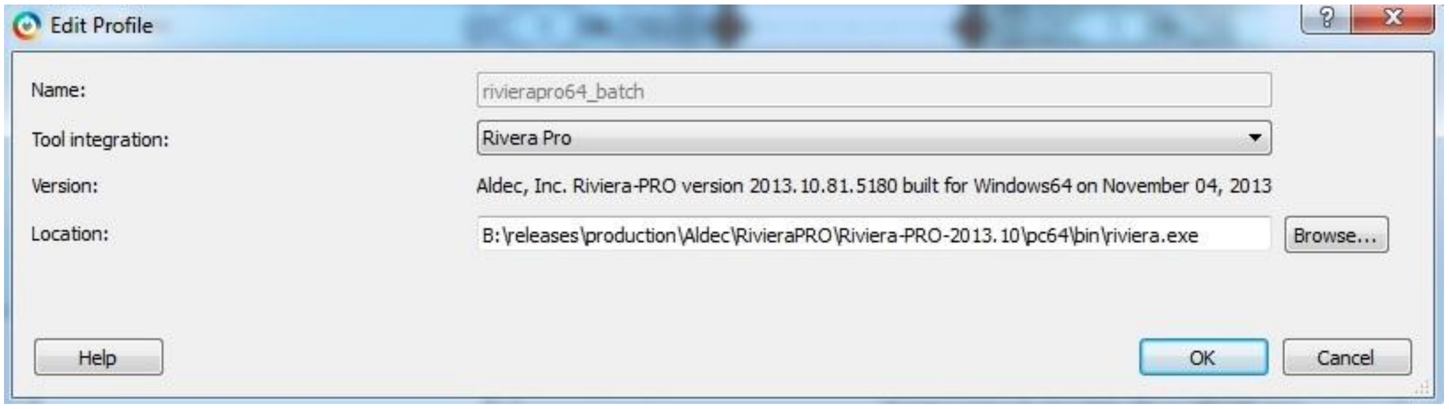
The tool profiles can be added from **Project -> Tool Profiles -> Simulation** in the Libero GUI.

Refer to the screenshot of the GUI Tool Profile below.



Name: test_profile

The following screenshot shows Riviera-PRO (2013.10) added as a Tool Profile in Libero.



Name: rivierapro64_batch

Tool integration: Rivera Pro

Version: Aldec, Inc. Riviera-PRO version 2013.10.81.5180 built for Windows64 on November 04, 2013

Location: B:\releases\production\Aldec\RivieraPRO\Riviera-PRO-2013.10\pc64\bin\riviera.exe

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 from overwriting that file. For example, the files can be renamed presynth_run.do, postsynth_run.do, and postlayout_run.do.

Click here for **Compiling SmartFusion Library for ModelSim Full Version (PE/SE/DE) Simulation**

Aldec Setup for Active-HDL and Riviera-Pro

OS PLATFORM: Windows

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

Set your environment variable to your license file location:

LM_LICENSE_FILE: must include a pointer to the license server.

Libero SoC and 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).
- While running postsynth simulations of designs containing MSS block, or postlayout simulations of designs using SERDES, the BFM simulations do not work if the –PL option is not specified with the asim command for smartfusion2 library. This is because during simulation, 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 –PL option indicates library precedence (i.e the libraries specified with this option are searched before the libraries specified with the –L option).
- 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 postsynth 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 scripts

The scripts below convert ModelSim run.do files into Aldec simulator compatible run.do files. (This script is not required if Active-HDL is added as an active tool profile in Libero tool profiles. Libero will automatically generate the run.do file required for simulation.)

Script Usage for Active-HDL

Place this script in the Libero SoC simulation folder and execute it from there.

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.txt

```
#!/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 Usage for Riviera-PRO

Place this script in the Libero SoC simulation folder and execute it from there. (This script is not required if RivieraPRO is added as an active tool profile in Libero tool profiles. Libero will automatically generate the run.do file required for simulation.)

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.txt

```
#!/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

OS PLATFORM: LINUX only

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 64-bit and 32-bit operating systems:

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

Case2: Using the -64bit Command-line Option
In the command line specify -64bit option in order to invoke the 64-bit 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 libraries used in the design and associates the logical libraries with physical directory names.
- Variables file (hdl.var)—Defines variables that affect the behavior of simulation tools and utilities.

Libero SoC and NCSim

After creating a copy of the run.do Tcl files, do the following steps 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 containing configuration variables that 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 the the presynth simulation example shown above, suppose we have 3 RTL files (a.v, b.v, testbench.v) that need 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 postlayout simulations, first compile the SDF file before elaboration using the **ncsdfc** command.

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

   During elaboration, use the compiled SDF output with the –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>” ,
SCOPE = <testbench_module_name>.< name_of_top_level_module_instantiated_in_testbench>;
```

5. **Simulating using ncsim:** After elaboration, a simulation snapshot is created and 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 and sourced from the command line. Instead of using these three steps, the design can be simulated in one step using the 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 the clock generator statement as follows to run simulation:

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

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

Postsynth and Postlayout Simulations using MSS and SERDES(Verilog only):

While running postsynth simulations of Verilog designs containing MSS block, or postlayout simulations of Verilog designs using SERDES, the BFM simulations do not work if the –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 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.

VHDL Designs elaboration using NCSim:

For VHDL designs use the –lib_binding option during elaboration to resolve the instances from the respective libraries specified in cds.lib.

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 the –libmap option. This issue has been reported to the Cadence support team (SAR 52113).
The crash is seen in Incisive 11.10 and 12.10 versions.

DDR Verification IP testbench workaround for VHDL designs:

NCSim crashes for VHDL testbenches containing DDR Verification IP (SimDRAM) which are either generated through Smartdesign testbench or which use component instantiation of the SimDRAM component available in the smartfusion2 library. As a workaround, use entity instantiation of the SimDRAM component for VHDL testbenches using DDR Verification IP (seen as part of simulation cores in catalog).

Example:

```
entity testbench is
.
.
SimDRAM_0 : entity smartfusion2.SimDRAM
    generic map(
        AL              => ( 0 ),
        BL_BITS             => ( 8 ),
        CL              => ( 6 ),
.
.
port map(
        DRAM_CAS_N   => I1_top_0_FDDR_CAS_N,
        DRAM_CKE     => I1_top_0_FDDR_CKE,
.
.
end behavioural;
```

Postlayout Simulations for IGLOO2 designs(VHDL only, SAR 59447):

Postlayout simulations of IGLOO2 VHDL designs containing MDDR, FDDR, and SERDES which use peripheral initialization solution fail when simulated using NCSim. The eNVM reads result in X's.

```
NVM_0: User Read Data: 32'hxxxxxxxx : Mem Address: 800 : Time: 10180 ns
NVM_0: User Read Data: 32'hxxxxxxxx : Mem Address: 804 : Time: 10280 ns
NVM_0: User Read Data: 32'hxxxxxxxx : Mem Address: 808 : Time: 10460 ns
```

Sample TCL and shell scripts

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

Script 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 cadence_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, $lib_location ) = @ARGV;
&cadence_parser($presynth, $family, $lib_location);
&cadence_parser($postsynth, $family, $lib_location);
&cadence_parser($postlayout, $family, $lib_location);
sub cadence_parser {

my $modelsim_run_do = $_[0];
my $actel_family = $_[1];
my $lib_location = $_[2];
my $state;
my %lib_hash = ();
my $stimulus;
my $tb_module_name;
my $lib;
my $hdl;
my $sdf;
my $hdl_type;
my $elaboration_component;
my @vhdl;

if (-e "$modelsim_run_do")
{
        open (INFILE,"$modelsim_run_do");
        my @modelsim_run_do = <INFILE>;
        my $line;
        if ( $modelsim_run_do =~ m/(presynth)/)
        {
                `mkdir CADENCE_PRESYNTH`;
                $state = $1;
                chdir("CADENCE_PRESYNTH");
        } elsif ( $modelsim_run_do =~ m/(postsynth)/)
        {
                `mkdir CADENCE_POSTSYNTH`;
                $state = $1;
                chdir("CADENCE_POSTSYNTH");
        } elsif ( $modelsim_run_do =~ m/(postlayout)/ )
        {
                `mkdir CADENCE_POSTLAYOUT`;
                $state = $1;
                chdir("CADENCE_POSTLAYOUT");
        } else
        {
                print "Wrong Inputs given to the file\n";
                print "#Usage: perl cadence_parser.pl presynth_run.do postsynth_run.do postlayout_run.do
\"Libraries_location\"\n";
        }
                # Creates cds.lib file
                open (CDSLIB,">./cds.lib");
                # Creates hdl.var file
                open (HDLVAR,">./hdl.var");
                # Creates commands file
                open (COMMANDS,">./commands");

        print CDSLIB 'softinclude $CDS_ROOT/tools.lnx86/inca/files/cds.lib\n';
        print CDSLIB "DEFINE $actel_family $lib_location\n";

        print HDLVAR 'include $CDS_ROOT/tools.lnx86/inca/files/hdl.var\n';
        print HDLVAR "DEFINE WORK $state\n";

@vhdl = `grep ".vhd" ../$modelsim_run_do`;

if ( exists $vhdl[0] ) {
$hdl_type = "vhdl";
$elaboration_component = "entity";
} else {
$hdl_type = "verilog";
$elaboration_component = "module";
}

if ( $hdl_type eq "verilog" ) {
        foreach $line (@modelsim_run_do)
        {
                if ( $line =~ m/quietly set PROJECT_DIR\s+\"(.*?)\"/ )
                {
                        print HDLVAR "DEFINE PROJECT_DIR $1\n\n";
                }
                elsif ( $line =~ m/^vlog\s+-work\s+(.*?)\s+\"(.*?)\"\s+.*/ )
                {
                        print "Library is $1 and hdlfile is $2 \n";
                        push @{ $lib_hash{"$1"} }, $2;
                }
                elsif ( $line =~ m/^vlog\s+\"(.*incdir.*?)\"\s+-work.*/ )
                {
                        $stimulus = $1;
                        print "stimulus = $stimulus\n";
                        if ( $line =~ m/^vlog\s+\".*incdir.*?\"\s+-work\s+(.*?)\s+\"(.*?)\"\s+.*/ )
                        {
                                push @{ $lib_hash{"$1"} }, $2;
                                $tb_module_name = $2;
                                $tb_module_name =~ s/\$\{PROJECT_DIR\}/..\/../g;
                                $tb_module_name = `grep "module.*.;" $tb_module_name`;
                                $tb_module_name =~ m/module\s+(\w*).*\;/;
                                $tb_module_name = $1;
                                print "tb_module_name = $tb_module_name \n";
                        }
                }
                if ( $line =~ m/^vsim.*.\-sdf.*\s+.*?\=(.*\.sdf)/ )
                {
                        print "sdf = $1\n";
                        $sdf = $1;
                        $sdf =~ s/\$\{PROJECT_DIR\}/..\/../g;
                }
        }
        $stimulus =~ s/\$\{PROJECT_DIR\}/..\/../g;
        print COMMANDS "ncvlog $stimulus -cdslib ./cds.lib -hdlvar ./hdl.var -logfile ncvlog.log -errormax 15 -
update -linedebug ";

} elsif ($hdl_type eq "vhdl") {
        foreach $line (@modelsim_run_do)
        {
                if ( $line =~ m/quietly set PROJECT_DIR\s+\"(.*?)\"/ )
                {
                        print HDLVAR "DEFINE PROJECT_DIR $1\n\n";
                }
                elsif ( $line =~ m/^vcom.*.-work\s+(.*?)\s+\"(.*?)\"\s+.*/ )
                {
                        print "Library is $1 and hdlfile is $2 \n";
                        push @{ $lib_hash{"$1"} }, $2;
                }
                $tb_module_name = "testbench";
                if ( $line =~ m/^vsim.*.\-sdf.*\s+.*?\=(.*\.sdf)/ )
                {
                        print "sdf = $1\n";
                        $sdf = $1;
                        $sdf =~ s/\$\{PROJECT_DIR\}/..\/../g;
                }
        }
        print COMMANDS "ncvhdl -cdslib ./cds.lib -hdlvar ./hdl.var -logfile ncvlog.log -errormax 15 -update -v93
-linedebug ";
}

        foreach $lib (keys %lib_hash)
        {
                `mkdir $lib`;
                print CDSLIB "DEFINE $lib \.\/$lib \n";
                foreach $hdl ( @{ $lib_hash{$lib} } )
                {
                        print HDLVAR "DEFINE LIB_MAP \( \$LIB_MAP, ".$hdl." \=\> ".$lib." \) \n";
                        $hdl =~ s/\$\{PROJECT_DIR\}/..\/../g;
                        print COMMANDS $hdl." ";
                }
        }

        print HDLVAR "DEFINE LIB_MAP \( \$LIB_MAP, \+ \=\> ".$state." \)\n";
        print COMMANDS "\n\n";
        if ( $state eq "presynth" )
        {
                print COMMANDS "ncelab -Message -cdslib ./cds.lib -hdlvar ./hdl.var -work $state -logfile
ncelab.log -errormax 15 -access +rwc -status $state.$tb_module_name:$elaboration_component\n";
                print COMMANDS "\n\n";
        }
        elsif ($state eq "postsynth")
        {
                print COMMANDS "ncelab -libmap ./lib.map -libverbose -Message -cdslib ./cds.lib -hdlvar ./hdl.var
-work $state -logfile ncelab.log -errormax 15 -access +rwc cfg1\n";
                open(cfg_file,">./lib.map");
                print cfg_file "config cfg1;\n";
                print cfg_file "design $tb_module_name;\n";
                print cfg_file "default liblist $actel_family $state;\n";
                print cfg_file "endconfig\n";
                close(cfg_file);
        }
        else
        {
                print COMMANDS "ncsdfc $sdf -output $sdf\.X \n";
                print COMMANDS "ncelab -autosdf -Message -cdslib ./cds.lib -hdlvar ./hdl.var -libmap ./lib.map -
work $state -logfile ncelab.log -errormax 15 -access +rwc -status $state.$tb_module_name:$elaboration_component
-sdf_cmd_file ./sdf_cmd_file\n";
                print COMMANDS "\n\n";

                open(cfg_file,">./lib.map");
                print cfg_file "config cfg1;\n";
                print cfg_file "design $tb_module_name;\n";
                print cfg_file "default liblist $actel_family $state;\n";
                print cfg_file "endconfig\n";
                close(cfg_file);

                open (sdf_compile,">./sdf_cmd_file");
                print sdf_compile 'COMPILED_SDF_FILE = "'.$sdf.'.X"'."\n";
                close(sdf_compile);
        }
        print COMMANDS "ncsim -Message -batch -cdslib ./cds.lib -hdlvar ./hdl.var -logfile ncsim.log -errormax
15 -status $state.$tb_module_name:$elaboration_component"."\n";

        close(COMMANDS);
        close(HDLVAR);
        close(CDSLIB);
        close(INFILE);
        chdir("..");
}
}
```

Mentor Graphics QuestaSim Setup

OS PLATFORM: WINDOWS and LINUX

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 the QuestaSim installation
PATH: must point to the executable location pointed to by MODEL_TECH

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.

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

Sample TCL and shell scripts

The following scripts convert the ModelSim ME run.do files into QuestaSim compatible run.do files. (This script is not required if Questasim is added as an active tool profile in Libero tool profiles. Libero will automatically generate the run.do file required for simulation.)

Script Usage

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

Questa_parser_pl.txt

```
#!/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

OS PLATFORM: LINUX only

The flow recommended by Microsemi relies on the Elaborate and Compile flow in VCS. This document includes a script that uses the run.do Tcl scripts generated by Libero SoC and generates the setup files needed for VCS simulation. The script 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 the VCS simulation is running.
2. Create a shell script file to elaborate and compile your design using VCS.

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.

Libero SoC and VCS simulation

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. The file name must not change and it must be located in the same directory where the 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. 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 in VCS using the following command:

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

\The timing resolution of the simulation must be set to 1fs for correct functional simulation.

4. Once the design is compiled, you can start the 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 include a $finish statement in the Verilog testbench to stop the simulation. When running simulations in GUI mode, run time can be specified in the GUI.

Sample TCL and shell scripts

The Perl script below automates the generation of the synopsys_sim.setup file as well as the corresponding shell scripts 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 shown below, including the corresponding library mapping and shell scripts 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.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 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.txt

```
#!/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 below to convert the ModelSim run.do Tcl files into VCS compatible shell scripts, create proper directories inside the Libero SoC simulation directory, and then run simulations.

Run this script using the following syntax:

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

Vcs_parse_pl.txt

```
#!/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,"$file") || die "Can’t open File Reason might be:$!";

        if ( $state eq "presynth" )
        {
                open(OUT1,">presynth_main.csh") || die "Can’t create Command File Reason might be:$!";
        }
        elsif ( $state eq "postsynth" )
        {
                open(OUT1,">postsynth_main.csh") || die "Can't create Command File Reason might be:$!";
        }
        elsif ( $state eq "postlayout" )
        {
                open(OUT1,">postlayout_main.csh") || die "Can't create Command File Reason might be:$!";
        }
        else
        {
                print "Simulation State is missing  \n" ;
        }

        open(OUT2,">synopsys_sim.setup") || die "Can't 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;
}
```

ModelSim (PE/SE/DE) Setup (KI8797) for Smartfusion

Compiling the SmartFusion Simulation Library for ModelSim Full Version (PE/SE/DE) Simulation

SmartFusion simulation with the ModelSim AE precompiled libraries errors out complaining about obsolete library format.

In most cases, the precompiled library that comes with the Libero IDE installation should work fine with ModelSim PE/SE/DE. However, there may be cases where simulation fails. One failure occurs when the standalone ModelSim version differs from the ModelSim AE version used to create the precompiled library. In such cases, the user needs to compile the SmartFusion Simulation library from scratch using the source files.

For SmartFusion, there is only one source file, which is **smartfusion.v**. There is no vhdl library. The Neutral Marking feature from ModelSim is used to generate the precompiled libraries (enabling it to work in both vhdl and verilog flows). If you want to compile the library yourself, for SE, PE or DE (which allow mixed language simulation), you must use the source files found in the precompiled libraries directory in the Libero installation path shown below:

<Libero_Installation>\Designer\lib\modelsim\precompiled\vlog\src

Steps for Compiling the Libraries

In total, 4 library files from the Libero Installation folder must be compiled, including the 1 macro cell library (Smartfusion.v) and 3 package files which are needed for VHDL simulation.

Smartfusion.v - The macro cell library which is common for Vhdl and Verilog.

fixed_float_types_c.vhdl

fixed_pkg_c.vhdl

float_pkg_c.vhdl

The ModelSim Commands Used to Compile the Library are shown below:

Create a new directory tree called /Actel in the ModelSim Standalone directory (such as the path <ModelSim_Install_Path>/Actel).

Invoke the ModelSim PE/SE/DE HDL Simulator.

Type the following at the ModelSim Command Prompt:

cd <ModelSim_Install_Path>/Actel (To change the working directory)

vlib smartfusion (To create the smartfusion library at <ModelSim_Install_Path>/Actel/smartfusion)

vmap smartfusion <ModelSim_Install_Path>/Actel/smartfusion

vlog -work smartfusion "<Libero_Installation>/Designer/lib/modelsim/precompiled/vlog/src/smartfusion.v"

vcom -work smartfusion "<Libero_Installation>/Designer/lib/modelsim/precompiled/vlog/src/fixed_float_types_c.vhdl"

vcom -work smartfusion "<Libero_Installation>/Designer/lib/modelsim/precompiled/vlog/src/fixed_pkg_c.vhdl"

vcom -work smartfusion "<Libero_Installation>/Designer/lib/modelsim/precompiled/vlog/src/float_pkg_c.vhdl"

This command sequence will compile all four library files into the compiled library **smartfusion.**

The smartfusion library can now be used for Simulation with the Standalone Full Version of ModelSim PE/SE/DE.

P.S: The above steps are applicable only for the Smartfusion (G3) family, not for any other families.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996