Cadence Concept Conversion Guide from XACTstep v5.x to vM1.x

May 1998 (Version M1.4) Application Note Excerpt

Summary

This guide will help you convert your existing Cadence Concept designs from previous versions of XACT*step* 5.X to version M1.X software.

Xilinx Families

XC3000A/L, XC3100A/L, XC4000E/L, XC4000EX/XL/XV, Spartan, XC9500

Migrating Cadence Designs

The number of changes required to retarget a Cadence design from 5.X to M1.X are significant, regardless of the type of design you are trying to migrate because the M1.X release coincides with major changes in Cadence design methodology. However, a design which is drawn purely with Unified Library components will require fewer changes than a design that contains instantiated netlist modules, instantiated HDL, or X-BLOX.

In light of the fact that X-BLOX designs may require more work to convert, you may consider the option of using 5.X instead to process your design. Unless you are targeting the new XC4000X or XC9500 families or have moved to an operating system that is not supported by 5.X (e.g., Solaris 2.5 or Windows NT 4.0), 5.X is still a viable option for X-BLOX designs.

This section details the migration of Cadence designs from 5.X to M1.X. For more information about the Cadence tools, please see the Cadence Interface/Tutorial Guide. For more information on the M1.X core tools, refer to the Xilinx online documentation.

Cadence Changes

Cadence 5.X Environment

The M1.X Xilinx/Cadence interface release coincides with a major change in the Cadence database structure, which is referred to by Cadence as their “5.X Environment”. The main goal of the 5.X Environment is the standardization of a common logical and physical structure for all designs across all Cadence design tools. The common logical structure shared across all Cadence tools is a hierarchical *lib/cell/view/file* structure, where

- A “view” is a collection of files that are related in that they all contain information about one type of representation, such as schematic, symbolic, HDL, or layout.
- A “cell” is a collection of views that describe an individual building block of a chip or system.
- A “library” is a collection of cells that are related either in terms of
  - describing components of a single design (a “design library”) or
  - describing common components potentially used in many designs a “reference library”)

In the 5.X environment, Verilog is the base netlist format from which all other netlists are derived.

Also, a new *cds.lib* library file is now required for specifying libraries available to all 5.X compliant tools, including Concept, and Composer.

Concept HDL Direct Methodology is required

HDL Direct design methodology is now standard and required for all PIC (Programmable Integrated Circuit) flows, including Xilinx.

HDL Direct gives users the ability to do functional simulation of schematic designs directly, all that is required is nominal post-processing to resolve design hierarchy. This is done using the Concept2XIL netlister. The main features are:

- Automatic generation of a Verilog netlist for a user schematic or block automatically whenever it is saved.
- Use of Verilog as the base intermediate netlist format from which all other formats are derived.

HDL Direct methodology is now required for all Concept schematics in the M1.X flow. In the context of Xilinx designs, this means that SCALD methodology is not supported, and users must convert their designs.

The main conversions users will need to make are:

1. Using SLICE components from the hdl_direct_lib library instead of TAP and CTAP symbols to tap bits off busses.
2. Removing "\I" suffixes from interface signal names and replacing these with INPORT, OUTPORT, and IOPORT bodies from the hdl_direct_lib library to designate input, output and bidirectional signals associated with a higher level symbol body in hierarchical, multi-sheet designs.
3. Modification of signal and block names to conform to Verilog naming conventions (no overlapping of signal and block names, all names only beginning with alphabetic characters or $ signs).

HDL Direct is most conveniently activated by adding the following commands to a startup.concept file in your project directory:

```
set hdl_direct_on
set hdl_checks on
set check_signames on
set check_net_names_hdl_ok on
set check_port_names_hdl_ok on
set check_symbol_names_hdl_ok on
runopl <installation_path_to_cadence>/
tools/fet/concept/hdl_direct/bin/autosym
```

HDL Direct and the associated naming checks are automatically activated when Concept starts up when you set the startup.concept file in this way.

SIZE property is no longer supported

In the M1.X translators and libraries, the SIZE property (used for replicating instantiated symbols) is no longer supported due to adverse effects on simulation times. These effects are associated with SIZE and the HDL Direct methodology. Users must use the ITERATED INSTANCE methodology for replicating symbol bodies in their designs, which involves adding an index range (n-1..0) to the PATH property of a symbol body, where n is the total number of copies of a component desired. (A PATH property is a Concept schematic instance name.)

For example, say the instance name of the FDCE component is I1. If you want to replicate it to get a total of 4 flip-flops, you must modify the PATH property and change it to "I1(3..0)", (basically adding an index range of 3 down to 0).

The ITERATED INSTANCE methodology is discussed in detail in the Cadence HDL Direct User Guide. If you want to process existing designs in M1.X, you must continue using the XNF translator, CONCEPT2XNF.

New Netlisters--CONCEPT2XIL and XIL2CDS

In the M1.X interface, CONCEPT2XIL is used to translate a Concept schematic to EDIF via HDL Direct generated Verilog. XIL2CDS generates the body and chips_prt files needed to integrate the chip level design into a board level schematic. Both netlisters must be obtained from Cadence Design Systems.

M1.X Libraries are required

HDL Direct requires that designs be entered using HDL Direct compliant libraries. The M1.X Xilinx Concept and Verilog libraries MUST be used for doing designs with the M1.X Cadence translators Concept2XIL and XIL2CDS. You may not use the 5.x Verilog libraries or Cadence Concept libraries prior to those shipped with the Cadence 97A release because they are incompatible.

M1.X HDL Direct compliant libraries can be identified by the new naming conventions of

```
xce*
and
verilogxce*
```

Both primitives and macros are now merged into a single library in Concept, in compliance with the Xilinx Unified Library standard. For example, the XC4000EX library for Concept is named xce4000ex, and the corresponding Verilog Unified Simulation Library is named verilogxce4000ex.

All M1.X Concept libraries are located in `$XILINX/cadence/data`. You must have a master.local file pointing to the explicit location of all available Xilinx architecture libraries in your project directory. A sample master.local file is located in `$XILINX/cadence/examples`.

Pad symbols are drawn from the xcepads library. To be able to use symbols like INPORTs and OUTPORTs, you must also add the hdl_direct_lib reference to your global.cmd setup file.

Sample global.cmd file:

```
master_library "./master.local";
library "xce4000ex",
      "xcepads",
      "hdl_direct_lib",
      "standard";
use "design.wrk";
root_drawing "unnamed";
```

In M1.X, generic Xilinx SIMPRIM-based Verilog simulation libraries are supplied as a standard part of the Xilinx Core Tools for post - NGDBUILD, post - MAP and post - route timing simulation.

Setting up your Xilinx/Cadence environment

This section includes the installation of the Xilinx M1.X core tools, the Xilinx M1.X Cadence interface, and Cadence Concept and Verilog-XL. .

Note that the XACT environment variable is no longer used. M1.X now uses the following Xilinx-specific variable:

```
setenv XILINX /tools/xilinx
```

The XILINX environment variable is set to the location of the M1.X software.

Data files related to the Xilinx/Cadence interface are located in $XILINX/cadence/data. This includes the xilinx.pff property filter file (which replaces concept2xnf.prop), and the .pkg files used by XIL2CDS.

Your executable path needs to include the following directories:

```
set path = ($XILINX/bin/<platform> $path)
```

where <platform> is set to “sun” for Sun4 platforms, “sol” for Solaris platforms, and “hp” for HP-UX platforms.

For Concept, the new Concept2XIL netlister, and XIL2CDS (used for board level integration), you will need a cds.lib setup file to point to the appropriate VAN (Cadence Verilog Analyzer)-compiled libraries.

Example cds.lib contents:

```
define xce4000x_syn /tools/xilinx/
     cadence/data/xce4000x_syn
```

Retargeting to an M1.X Concept Library

If you have a purely schematic design from 5.X and want to import it into the M1.X environment, you must retarget your design to the appropriate M1 Concept library, even if you are still targeting the same family. The M1.X EDIF netlister, Concept2XIL, is not compatible with the pre-M1.X SIZE’ed libraries. Thus in all cases you will need to do the following when you convert a design to M1.X:

1. Modify your existing master.local file in the design directory to include the paths to the new M1 libraries. A sample master.local is provided in $XILINX/cadence/examples. If you do not have a an existing master.local in your project directory, create one. The format is:

```
file_type = master_library;
“xce4000x”‘/tools/xilinx/cadence/data/xce4000x/
xce4000x.lib’;
“xce4000e”‘/tools/xilinx/cadence/data/xce4000e/
xce4000e.lib’;
“xce3000”‘/tools/xilinx/cadence/data/xce3000/
xce3000.lib’;
“xcepads”‘/tools/xilinx/cadence/data/xcepads/
xcepads.lib’;
end.
```

2. Next, modify your global.cmd file to specify which of the new M1 Concept libraries specified in the master.local file that are needed to convert the design. For example, say you need to convert an XC4000E design to, XC4000X. In this case, the libraries we need to add are “xce4000x” for the architecture-specific components, “xcepads” for the pad symbols, and “hdl_direct_lib” for those components needed for HDL Direct compliance (slices, inports, outports, ioports). The library naming convention for Cadence releases 9404 to 9604 was xc* for the primitive library, and xm* for the macro library for a given architecture named “*”. For example, if * was “4000e”, we would have “xc4000e”, and “xm4000e”. HDL Direct-compliant pads were in “xpads_hdl”. In the M1.X Concept libraries, primitives and macros are merged into a single library, “xce4000e”, and pads are drawn from the unsized “xcepads” library.

```
master_library “./master.local”;
library “xce4000x”,
    “xcepads”,
    “xc4000e”,
    “xm4000e”,
    “xpads_hdl”,
    “hdl_direct_lib”,
    “standard”;
    use “my_design.wrk”;
root_drawing “my_design”;
```

3. To convert the design, you need to ignore the old libraries with the “ignore” command, and activate the new libraries with the “lib” command. Set the “sticky_off” parameter to prevent irrelevant properties from being translated to the converted design, including SIZE properties. Do a “get” to read in components from the new target libraries, and finally a “write” to save the new references.

```
ignore lib xc4000e
ignore lib xm4000e
ignore lib xpads_hdl
set sticky_off
lib xce4000e
lib xcepads
lib hdl_direct_lib
get
write
```

After saving the design, remove all references to the XACT Concept libraries (“xc4000e”, “xm4000e”, and “xpads_hdl”) from the global.cmd.

4. Use the ITERATED INSTANCE methodology to replicate any components that were previously replicated with SIZE properties.

5. Replace any XBLOX modules in the design with the appropriate LogiBLOX counterpart. Since LogiBLOX is not integrated into the Concept schematic editor, the LogiBLOX module must be generated by running LogiBLOX in standalone mode, and GENVIEW must be used to generate a symbol BODY for the module. See the section on Converting X-BLOX designs for more details.

6. If the design was entered using SCALD methodology conventions, you must convert it to comply with HDL Direct methodology guidelines. This includes:

- Renaming any instance and net names that conflict or overlap (symbols and nets may not share the same names).
- Modifying Interface signal connections. In SCALD methodology, signals that connect to a pin on an upper

level hierarchical symbol body are designated with a "\" extension on the signal name and a FLAG body is attached to the signal. Since in M1.X, the design must comply with HDL Direct methodology, you must remove all FLAG bodies and all "\" extensions on signal names, and replace them with INPORT, OUTPORT, and IOPORT bodies, depending on whether they are input, output, or bidirectional signals, respectively.

- Replace all TAP and CTAP symbols used to tap bits off buses with SLICE symbols from hdl_direct_lib.
- All signal and instance names must comply with Verilog naming restrictions, and must be modified if needed. Signal and instance names may only begin with alphabetic characters or the $ sign. Legal characters are: a-z, A-Z, 0-9, _, and $.

For more information, please see the Verilog-XL Reference Manual and the HDL Direct User Guide.

Specifying part type on a schematic

The target device can be specified on the schematic by attaching a "PART" property to the new CONFIG library symbol. (In previous releases, this property was called "PART_TYPE" and attached to a DRAWING body.) For example, to designate a target device of XC4010E-3 in a PQ208 package in the top-level schematic, place a CONFIG library symbol on the sheet, then add the following property:

```
Name:PART
Value:XC4010E-3-PQ208
```

This "density-speed-package" designation is the recommended format. Other acceptable property values would include "4010E-3-PQ208", "4010E-PQ208-3", as well as the 5.X style "4010EPQ208-3". The 5.X style is discouraged, since the lack of a hyphen between the die type and the package type can make this designation ambiguous.

Tapping Bits off Buses

When tapping bits off buses, or building buses out of unrelated signals, you must insert BUFFs (buffers) between the unrelated signal and the name of the bit tapped off the bus In addition, all signals tapped off busses should be tapped off using a SLICE symbol from the HDL_DIRECT_LIB library instead of TAPs or CTAPs from the STANDARD library.

Example 1: When tapping a bit 1 off a bus named A<1..0>, the name of the bit is A<1>, NOT A1. Any deviation from this naming convention (for example, naming it "A1") is considered renaming the bit, and will require that you insert a BUFF (buffer) symbol between the tapped signal (A<1>) and the new name (in this case, "A1").

Example 2: Say you have two signals, one named "CTRL", and the other named "ENABLE", and you wish to combine these into a single two-bit bus called A<1..0>, where A<1> = CTRL and A<0> = ENABLE. You must insert a BUFF symbol between the signal named "CTRL" and the A<1> bit that you tap off the bus, and another BUFF between the signal named "ENABLE" and A<0>.

The error you will get in both cases above if you do not add the buffer will be an NGDBUILD "Duplicate port" error on an "alias" cell.

Migrating X-BLOX designs

The M1.X core tools use LogiBLOX to synthesize high-level functional modules formerly supported by X-BLOX. In the LogiBLOX flow, modules are synthesized up front, one by one during design entry instead of during design compilation. This results in shorter design-compilation times. LogiBLOX also simplifies both functional and timing simulation of X-BLOX designs by allowing you to use the same flow as you would for purely gate-level designs.

Since X-BLOX modules are not supported during design compilation in M1.X, designs must be fully synthesized before they are introduced into the M1.X software. This can be done in one of two ways.

Option 1: Convert X-BLOX modules to LogiBLOX

In this conversion, all X-BLOX components in the design are removed and replaced with LogiBLOX components. This needs to be done manually for each component.

Before you start to convert your design, be sure you save a copy of it for reference, in case you need to revert to it during the replacement process.

Since LogiBLOX is not integrated into the Concept schematic editor, you must run it stand-alone to generate each component. The steps are as follows:

1. Start up LogiBLOX by typing: "lbgui".
2. Specify "Cadence" as the vendor.
3. Under Options, select "Structural Verilog netlist". If you are going to be instantiating the block into a Verilog netlist rather than a schematic, select "Verilog template" under the Component Declaration field. Click on "OK" to generate the module and its associated structural Verilog netlist in your project directory.
4. The next step is to generate a symbol body for the new module if you are incorporating the LogiBLOX module into a Concept schematic. To do this, run GENVIEW from the Concept schematic editor using the structural .V netlist for the module as input. In the Concept command window, type:

   ```
   genview -i new_block.v -v logic body verilog
   ```

   This will add a new directory called "new_block" to your

project directory. GENVIEW copies the “new_block.v” file from LogiBLOX to new_block/logic/, renaming it to verilog.v.

5. Edit the verilog.v file and add the following line to the module definition to signal Concept2XIL to stop traversing the block at this level because there are no underlying primitives:

```
parameter cds_action=”ignore”;,
```

Although this module by module conversion process can require a great deal of work, the benefit will be a smoother design translation, and easier functional simulation.

Since LogiBLOX components are synthesized immediately once you have set the desired parameters for a module, their bus-pin widths are determined up front. As a result, data types do not need to be propagated as in the case with X-BLOX. Since data-type and bus-width propagation is not an issue in LogiBLOX, bus-translation components such as BUS_DEF, BUS_IF, CAST, ELEMENT, and SLICE are not required.

Be sure that all of the buses in your design have indices. Just as with regular bus pins, the width of these buses must equal the width of the LogiBLOX bus pins to which they are connected.

This flow, although more involved up front, is highly recommended to take advantage of the full feature set in M1.X, especially if the design is in the beginning or intermediate stages of development. Once the schematic is redrawn to use LogiBLOX modules, the design can be easily implemented and simulated. Unlike X-BLOX modules, LogiBLOX modules can be simulated almost immediately in Verilog-XL. After all X-BLOX components have been replaced in the design and the design is saved (with HDL Direct active), functional simulation can be performed after running Concept2XIL with the *-sim_only* option to generate a simulation netlist directly from your schematics:

```
concept2xil -sim_only -family xce4000x \
new_design
```

The one drawback is that simulation at this stage requires that you create your test fixture file manually.

Option 2: Run your completed X-BLOX design through M1.X

If you have an existing, complete X-BLOX design, or you are only interested in doing a place and route using the new M1.X tools, you can first follow the 5.X design flow to the point where a design .xtf file is created by running XMAKE -n. Take this design into NGDBuild, then run the M1.X place-and-route tools as normal.

To functionally simulate a design, use the following flow:

```
ngdbuild -p new_design.xtf
ngd2ver -tf -ul new_design
```

For timing simulation, generate a routed design.nga with:

```
map new_design
par new_design
ngdanno -o new_design.nga routed_design.ncd
ngd2ver -tf -ul design.nga
```

Note that, unlike the LogiBLOX flow, described in option 1, simulation vectors used in functional simulation may not be completely applicable to timing simulation. This is because in the X-BLOX flow, NGDBuild compiles a flattened netlist, whereas in the LogiBLOX flow, NGDBuild compiles a hierarchical netlist.

Because of the limitations that this “half-and-half” flow imposes upon simulation, this flow is recommended only for complete or nearly complete designs that are to be evaluated under the M1.X tools. If the design will be subject to many design iterations and compilations through the M1.X tools, it is highly recommended that the design be updated to use LogiBLOX modules as described in Option 1.

You may also specify the part type during the implementation flow in the M1.X core tools as a command line option.

Simulation

Functional simulation of pure schematic designs which may or may not include LogiBLOX modules, can now be done directly after running Concept2XIL with the -sim_only option when the design is entered using HDL Direct methodology. Verilog Unified Library simulation primitives in `$XILINX/cadence/data/verilogxce*` are used for this simulation.

Post-route timing simulation involves two steps: generating a timing-annotated design.nga netlist with NGDANNO, then running NGD2VER on the resulting structural Verilog netlist. In this case the simulation model is expressed in terms of generic simulation primitives (“SIMPRIMs”) located in `$XILINX/verilog/data` instead of architecture specific Unified Library components.

Simulation

Functional simulation of pure schematic designs as well as schematic designs containing LogiBLOX modules can now be done directly after running Concept2XIL with the -sim_only option when the design is entered using HDL Direct methodology. Verilog Unified Library simulation primitives in `$XILINX/cadence/data/verilogxce*` are used for simulation of the schematic blocks, and Verilog SIMPRIM libraries are used to simulate the LogiBLOX modules.

Post-route timing simulation involves two steps: generating a timing-annotated design.nga netlist with NGDANNO, then running NGD2VER on the resulting structural Verilog netlist. In this case the simulation model is expressed in

terms of generic simulation primitives (“SIMPRIMs”) located in `$XILINX/verilog/data` instead of architecture specific Unified Library components.

Integrating into Board Level Simulation

The new utility for integrating into board level simulation is XIL2CDS. You must run NGD2VER with the -pf option to generate the .pin file needed by XIL2CDS to generate the symbol body for the FPGA:

```
ngd2ver -tf -ul -pf new_design
```

Then run XIL2CDS to generate the symbol body and chips_prt file for the FPGA:

```
xil2cds new_design -lwbverilog -use
my_design.wrk -r . -family xce4000x -mode all
```

New Global (Set)/Reset and Global Tri-state Methodology

In the M1.5 release, the Concept and Verilog libraries and NGD2VER netlister support a new, required methodology for declaring and stimulating global set/reset (GSR), global reset (GR), and global tri-state (GTS) signals. The methodology is a departure from the earlier M1 and the XACT flows.It is now possible to use the same test fixture commands to simulate these global signals in both functional and timing simulation with minimal modification. The Verilog and Concept Unified libraries, as well as the NGD2VER netlister have been modified to support this new technique.

The new methodology applies to Verilog simulation of all architectures supported in the M1 release, and requires that you set the values appropriately for the following Verilog macros in your test fixture file, as appropriate to your architecture: GSR_SIGNAL, GR_SIGNAL, and GTS_SIGNAL.

Example: Declaring GSR

```
reg GSR;
`define GSR_SIGNAL test.uut.GSR
```

The exact value you use for GSR_SIGNAL varies slightly depending on what step of the flow you are at, what architecture you are using, and whether your design has a STARTUP block instantiated within it. If you design does not contain a STARTUP block, the declarations above are written to your test fixture template file by NGD2VER automatically. See the Cadence Interface User Guide and Tutorial in the M1.4 release for full details.

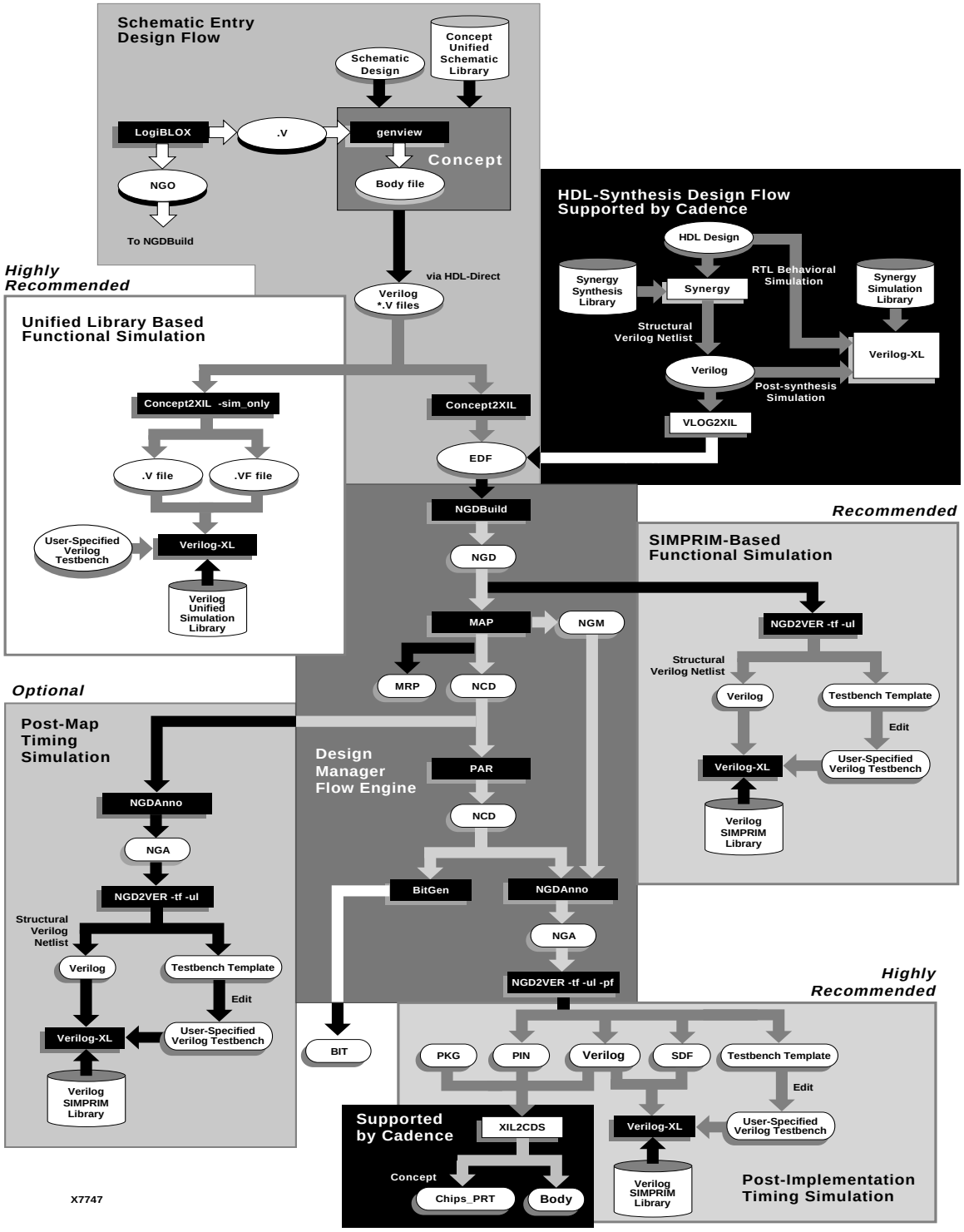

Figure 1: Cadence Flow Overview

XILINX® The Programmable Logic Company℠

Headquarters

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
U.S.A.

Tel: 1 (800) 255-7778
or 1 (408) 559-7778
Fax: 1 (800) 559-7114

Net: hotline@xilinx.com
Web: http://www.xilinx.com

North America

Irvine, California
(714) 727-0780

Englewood, Colorado
(303)220-7541

Sunnyvale, California
(408) 245-9850

Schaumburg, Illinois
(847) 605-1972

Nashua, New Hampshire
(603) 891-1098

Raleigh, North Carolina
(919) 846-3922

West Chester, Pennsylvania
(610) 430-3300

Dallas, Texas
(214) 960-1043

Europe

Xilinx Sarl
Jouy en Josas, France
Tel: (33) 1-34-63-01-01
Net: frhelp@xilinx.com

Xilinx GmbH
Aschheim, Germany
Tel: (49) 89-99-1549-01
Net: dlhelp@xilinx.com

Xilinx, Ltd.
Byfleet, United Kingdom
Tel: (44) 1-932-349401
Net: ukhelp@xilinx.com

Japan

Xilinx, K.K.
Tokyo, Japan
Tel: (03) 3297-9191

Asia Pacific

Xilinx Asia Pacific
Hong Kong
Tel: (852) 2424-5200
Net: hongkong@xilinx.com