



Application Note

Xilinx Software Conversion Guide from *XACTstep* v5.X to *XACTstep* vM1.X

April 1997

Version M1.2



Xilinx Software Conversion Guide





Xilinx Software Conversion Guide from XACTstep v5.X to XACTstep vM1.X

April 1997 (Version M1.2)

Application Note

Summary

This guide will help you convert your existing designs from previous versions of XACTstep 5.X to XACTstep M1.X software.

Xilinx Families

XC4000E/L, XC4000EX, XC4000XL, XC7300, XC9500

Table of Contents

Introduction	3
An Overview of the M1.X tools	4
MAP Overview.	8
Place and Route using PAR	10
TRCE Issues	12
Full M1.X Conversion Flow	13
Partial M1.X Conversion Flow	14
Additional Flow Issues.	14
Design Constraints	15
Migrating Cadence Designs	20
Migrating Viewlogic Designs	26
Migrating Mentor Graphics Designs	29
Migrating Synopsys VHDL/Verilog Designs	31

Introduction

This application note introduces the new XACTstep M1.X tools and gives you flows on how to transition your existing design from XACTstep 5.X to XACTstep M1.X. Since this is a brief application note, it may not go into complete detail on every tool; refer to the corresponding online documentation for complete details. Not everyone who has existing designs should convert over to the new tools; this section lists reasons both for and against converting. All new designs should be started in M1.X. In this guide, XACTstep v5.X software is referred to as 5.X and XACTstep vM1.X is referred to as M1.X.

Why convert from 5.X to M1.X?

New device families

The M1.X tools support the new, higher density XC4000EX chips. The M1.X tools are modular which allows easy revisions to support all future FPGA and CPLD families. The current 5.X tools only support up through the XC4000E devices and will not be revised to support either XC4000EX or any other future device families.

Better software tools

Based on NeoCAD technology, these new software tools are able to make faster and higher quality place and route runs than the current 5.X tools. This means better performance and utilization of your Xilinx programmable devices. M1.X can often complete routing where 5.X could not. Also, the UNIX version of the new place and route tool, PAR, has the capability to run multiple workstation jobs, so that you can utilize the power of all your workstations simultaneously.

New operating systems

The M1.X tools will support Windows NT 4.0, Windows 95, SunOS 4.X, SunOS 5.5/Solaris 2.5 and above, HP-UX 10.X, and AIX 4.1. Currently 5.X does not support these operating systems (except for SunOS 4.X) and there are no plans for 5.X to support these operating systems in the future.

Why not convert to M1.X?

Older families of Xilinx chips will not be supported in the new tools

The XC2000, XC3000, and XC4000A/D/H families will not be supported by M1.X; only XC3000A and newer families will be supported in the new software. Please note that these families (XC2000, XC3000, and XC4000A/D/H) are being phased out and new designs should not target these devices.

Older operating systems

Windows 3.X is no longer supported. If you are currently unable to upgrade to Windows 95 or Windows NT, then your best option is to use 5.X until you are able to upgrade.

Designs in production

Designs that are complete or in production, or some designs that are nearing completion, may not be good candidates for conversion.

Xilinx Software Conversion Guide

Pre-unified libraries

Pre-unified library designs are not supported in the new tools. There are no plans to support pre-unified libraries in M1.X.

X-BLOX Designs

In most cases, X-BLOX designs should not be used with the M1.X tools. If you want to use the M1.X tools, the best option is to convert your X-BLOX designs to LogiBLOX. This guide will detail specific information on how to convert designs from X-BLOX to LogiBLOX. The best option for unconverted X-BLOX designs is to continue processing them with the 5.X tools. While it is possible to use X-BLOX with M1.X tools, taking the time to do the X-BLOX to LogiBLOX conversion will be easier for most users than attempting to implement X-BLOX designs in M1.X. For those who absolutely must use X-BLOX with M1.X, the Partial M1.X Conversion Flow section will describe a generic approach to implementing X-BLOX designs in M1.X.

An Overview of the M1.X tools

The M1.X command lines have all changed from the 5.X tools. This section discusses the new tools, the input files supported, the commands used, and the output files written. Figure 1 is a diagram showing how the tools are used to implement and simulate your design.

New Commands

Implementation commands:

- **LCA2NCD** - Translates an existing 5.X physical design implementation (.lca file) into an M1.X physical implementation (.ncd file).
- **CSTTRANS** - Translates existing 5.X constraints into M1.X constraints.
- **DC2NCF** - Translates Synopsys constraints into M1.X constraints.

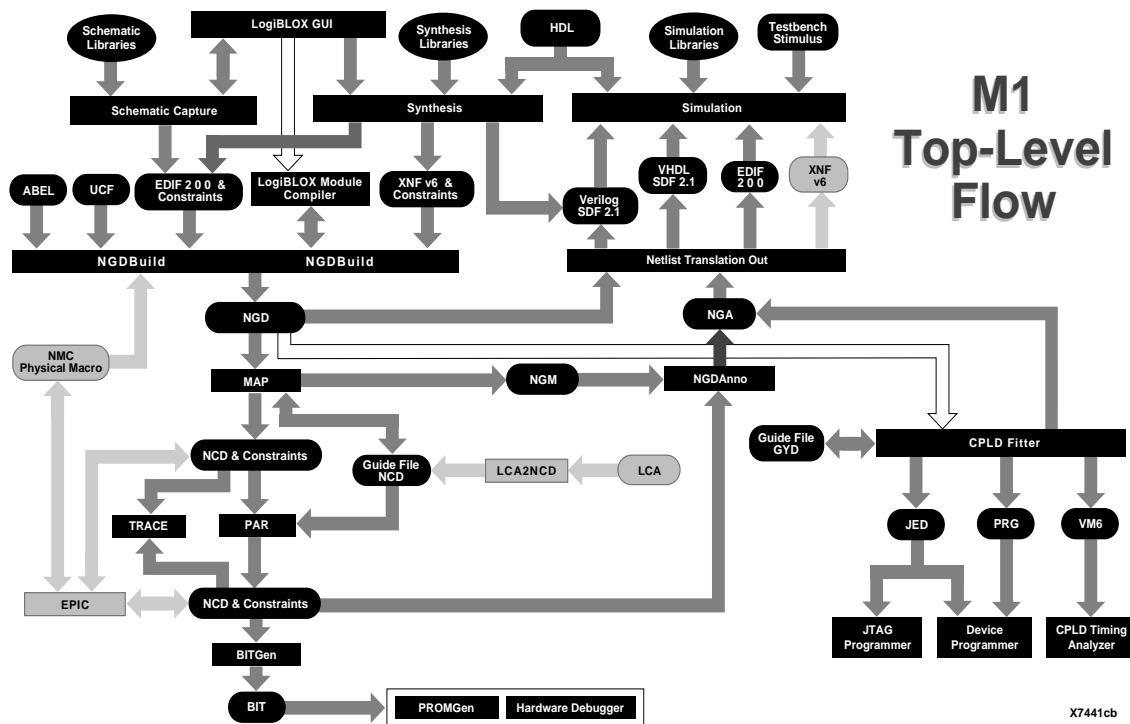


Figure 1: M1.X Implementation and Simulation Flows

- **NGDBuild** - Translates and merges the various files that make up the design into a generic logical description of the design (.ngd file)
- **MAP** - Maps the generic logical description into a physical design implementation of the design (.ncd file) that is unplaced and unrouted
- **PAR** - Performs placement and routing on a physical design implementation (.ncd file).
- **BITGEN** - Creates a bitfile from an .ncd file.
- **PROMGEN** - Creates a prom file from one or more bitfiles.

Analysis Commands:

- **TRCE** - Performs timing analysis on physical design implementation (.ncd file).
- **NGDAnno** - Extracts back annotated timing data from the physical design implementation to a generic logical design file (.nga file).
- **NGD2XXX** - Translates annotated logical design (.nga file) to annotated netlist (EDIF, VERILOG, VHDL and XNF).

Script Generation (Workstation only):

- **XGEN** - a utility for creating command line scripts. It prompts for input and output file names and desired features, then writes a script with the appropriate command lines.

Because this note only covers design conversion, BITGEN, PROMGEN, the Analysis Commands, and XGEN will not be detailed here. Please see the online documentation for more details on these programs

Translation of 5.X designs to M1.X with LCA2NCD

This allows you to import an existing design into the M1.X tools for use as a physical design. This can then be used to guide the design or be placed and routed directly.

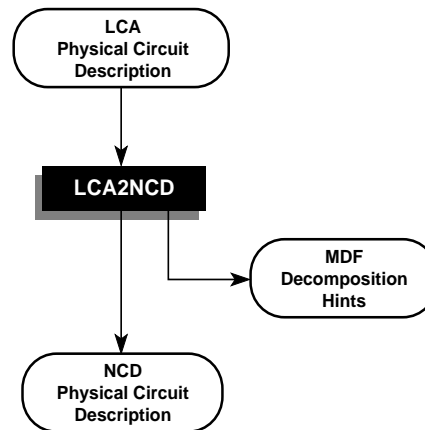
Input file for LCA2NCD:

- Logic Cell Array from 5.X (.lca)

Output files created by LCA2NCD:

- Mapped physical design implementation (.ncd)
- File to aid future guided mapping (.mdf).

Figure 2 shows the input and output files for LCA2NCD. See Table 1 for details on the command line usage of LCA2NCD.



X7473

Figure 2: LCA2NCD Flow

Translation of 5.X constraints to M1.X with CSTTRANS

CSTTRANS allows you to convert your existing constraint files into an M1.X usable file. This file is known as a .ucf or user constraints file. Please note that a floorplanner generated .cst file can not be used since the mapping in 5.X is handled differently than in M1.X.

Input file for CSTTRANS:

- 5.X constraint file. (.cst)

Output files created by CSTTRANS:

- User constraints file (.ucf)

See Table 2 for details on the command line usage of CSTTRANS.

LCA2NCD Usage		
lca2ncd [-w] [-p] <infile> [<outfile>]		
Options:	Value	Comments
	<infile>	Name of input .lca file.
	<outfile>	Name of output .ncd file.
-w		Overwrite. Allows overwrite of an existing output file.
-p		Placement only, i.e. don't preserve routing.

Table 1

Xilinx Software Conversion Guide

CSTTRANS Usage		
csttrans <infile>[.cst] [-o <outfile>]		
Options:	Value	Comments
	<infile>	Name of input .cst file.
-o	<outfile>	Name of output .ucf file.

Table 2

Translating a design using NGDBuild

NGDBuild translates input design netlists into a generic logical description of the design. NGDBuild starts by determining the netlist type of the specified top level input design. The “netlist launcher” calls the appropriate netlist translator which creates an intermediate .ngo file from the input netlist. For hierarchical designs, the appropriate netlist translator is called for each module in the hierarchy. A “make” feature compares file dates and only calls the netlist translator for .ngo files that are out of date. NGDBuild then merges the .ngo file(s) with any LogiBLOX components and physical macros (.nmc files) referenced in the design. Finally a logical Design Rule Check (DRC) is made. Please refer to Table 3 for details on the command line usage of NGDBuild

Input files supported by NGDBuild:

- XNF (v5.0 and above) format netlists (.xnf, .xtf)
- EDIF format netlists (.edif, .edn)
- Physical macros created by EPIC (.nmc)
- User constraints file (.ucf)
- Netlist constraints file (.ncf)
- Synopsys output files (.sxnf, .sedif)

Intermediate files created by NGDBuild:

- A .ngo is created for every netlist translated

Output file created by NGDBuild:

- A generic logical description of the design (.ngd)

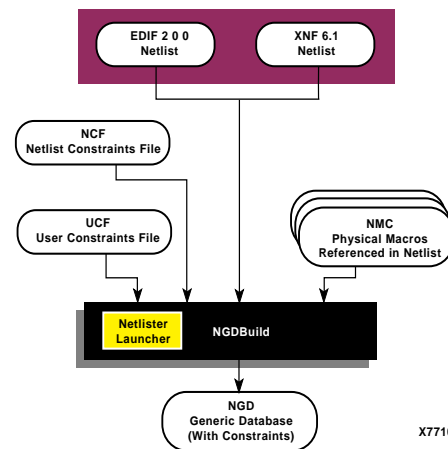


Figure 3: NGDBuild Flow

NGDBuild Usage		
ngdbuild [-p <parttype>] {-sd <searchpath>} {-sl <library>} [-ur <rules_file[.urf]>] [-dd <output_dir>] [-r] [-nt timestamp on off] [-uc <ucf_file[.ucf]>] <design_name> [<ngd_file[.ngd]>]		
Options:	Value	Comments
-p	<parttype>	Expand design for the given target part
-sd	<source_dir>	Add "source_dir" to the list of directories to search when resolving NGO/NMC file references
-sl	<library>	Add "library" to the list of libraries passed to the netlisters
-ur	<rules_file>	User rules file for netlist launcher
-dd	<output_dir>	Directory to place intermediate files
-nt	<value>	NGO file generation. Timestamp is the default. Timestamp: Regenerate .ngo only when source netlist is newer than existing NGO file On: Always regenerate .ngo file from source design netlists Off: Do not regenerate .ngo files which already exist. Build .ngd file from existing .ngo files
-r		Ignore location constraints
-uc	<ucf_file>	User constraints file
	<design_name>	
	<ngd_file[.ngd]>	

Table 3



Mapping a design using MAP

MAP reads the logical design file (.ngd) and maps it into a physical design (.ncd). The .ngd file also contains the logical constraints read by NGDBuild. The logical constraints are converted to physical constraints and written to the physical constraints file (.pcf). MAP also accepts an existing .pcf file as input as a means of inputting user defined physical constraints.

Map re-writes the portion of the input .pcf file bordered by the keywords "SCHEMATIC START ;" and "SCHEMATIC END ;". Any user defined physical constraints should be maintained outside this area. An example of this would be I/O location constraints obtained from the .pad file of a previous PAR run. In most cases, user defined constraints should be defined in the .ucf which deals in terms of the logical design.

MAP also accepts a guide design (.ncd and .mdf) which is used to closely replicate the results of a previous mapping when design changes have been minor. Guided mapping

should always be used if guided place and routing is planned. The mapped .ncd file consists of configured logic resources that are not yet assigned to die locations unless constrained. Placement and routing is assigned later by PAR. This guided flow can be seen in Figure 4.

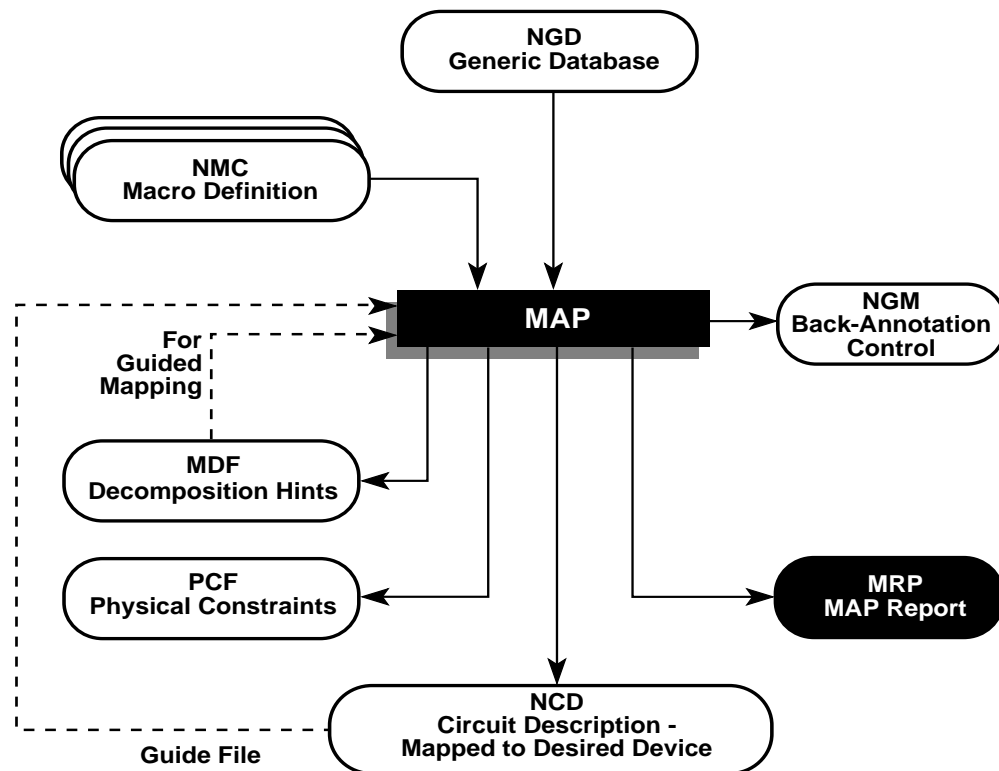
Please refer to Table 4 for details on the command line usage of MAP.

Input files supported by MAP:

- A generic logical description of the design (.ngd)
- Physical constraints file (.pcf)
- Guide design (.ncd and .mdf)

Output files created by MAP:

- Mapped physical design implementation (.ncd)
- Physical constraints file (.pcf)
- MAP report file (.mrp)
- File to aid back annotation to logical model (.ngm)
- File to aid future guided mapping (.mdf)



X7204

Figure 4: MAP Flow

Xilinx Software Conversion Guide

MAP Usage		
<pre>map [-b] [-c [<packfactor>:0,100>]] [-cm <covermode>] [-f <command_file>] [-gf <guidefile>] [-gm <guidemode>] [-ir] [-k] [-l] [-m] [-p <parttype>] [-oe <opteffort>] [-os <optstyle>] [-p <parttype>] [-pr i o b] [-r] [-u] [-o <outfile[.ncd]>] <infile[.ngd]> [<prffile[.pcf]>]</pre>		
Options:	Value	Comments
-b		Reassign physical clock buffers (BUFGP/BUFGS) as appropriate (does not apply to XC4000EX)
-c	<packfactor>	Pack unrelated logic into CLBs. <packfactor> indicates what percent of available CLB resources to target. Range: 0 <= <packfactor> <= 100. The default is 97.
-cm	area speed balanced	Cover mode. Defines the approach used to map logic. Default is "area". Area: attempts to use the fewest number of function generators. Speed: attempts to use the fewest levels of logic. Balanced: attempts to strike a balance between speed and area.
-f	<command_file>	Read command line arguments from file <command_file>
-gf	<guidefile>	Guide from <guidefile>.ncd
-gm	exact leverage	Guide mode. Default is "exact". Exact: Lock down all placement and routing. Leverage: Use initial design {guided or not} as hint.
-ir		Ignores relative placement of CLBs but preserves mapping of logic to a specific CLB
-k		Map to 5 input functions
-l		Disable logic replication
-o	<outfile[.ncd]>	Output filename. Default is the <infile.ncd>
-oe	normal high	Logic optimization effort. Default is normal. Used for ABEL designs.
-os	area speed best none	Logic optimization style. Default is no optimization. Used for ABEL designs.
-p	<parttype>	Map to part <parttype>
-pr	i o b	Pack internal flops/latches into input (i), output (o), or both (b) types of IOBs. Default is to not merge registers into IOBs.
-r		Disable register ordering
-u		Do not remove unnecessary/disabled logic
	<infile[.ngd]>	
	<prffile[.pcf]>	

Table 4

MAP Overview

Since MAP is so different and such a large part of the M1.X tools, it warrants some special attention. This section covers details about MAP and differences between 5.X and M1.X.

Outputs:

- .ncd (Native Circuit Design file-mapped physical design database)
- .ngm (mapped ngd)
- .pcf (physical constraints file)
- .mdf (MAP data file)
- .mrp (MAP report file)

MAP inputs and outputs

Required Inputs:

- .ngd (top level design netlist from NGDBuild)

Optional Inputs:

- .pcf (physical constraints file)

Optional Guide files:

- .ncd (Native Circuit Design file from a previous revision of the design)
- .mdf (MAP data file)

MAP Elements

- BEL: "basic elements". The fundamental low level logic building blocks of the FPGA - Includes Look Up Tables (LUTs which include the F, G, and H function generators), RAMs, FFs (FFX, FFY), carry logic, PADs, etc.
- COMP: Higher order FPGA structural elements-CLBs, IOBs, TBUFs, Decoders



In the M1.X release, MAP manipulates design objects at the COMP level, unlike PPR, which manipulated logic at the BEL, or basic element level. This has major implications for Timespecs and PAR. Due to this difference, .cst files generated by the 5.X floorplanner cannot be used.

Please note that guiding Map with an NCD derived from an LCA file to assist in migrating a design from 5.X to M1.X may not be very successful. Low guide success can be expected due to mapping differences between 5.X and M1.X.

New Capabilities in M1.X

Ability to analyze design timing after mapping

The ability to run TRCE after mapping allows you to predict the probability that you can meet your required design performance before any routing is done, purely based on levels of logic along critical paths.

Percent Compression

This allows you to specify the percentage of the chip that the logic should be compressed down to, using the -c option.

The percentage is specified as an argument to -c. The smaller the percentage value you specify, the smaller the portion of the chip area MAP will use for the design. Low compression values result in denser partitioning, but may be harder to route, while logic mapped with higher compression values will usually be easier to route because the logic is dispersed more sparsely throughout the FPGA, resulting in less routing congestion.

Support of XC4000EX-specific architectural features

- Global Early Buffers
- Global Low-skew Buffers
- FastCLK Buffers
- Fast Capture Latches
- CLB Latches (LD*)
- Output Multiplexer (OMUX)/2-Input Function Generator dual capability)
 - The OMUX/2-input LUT capability allows you to implement certain two-input functions in an IOB by instantiating the special library components (OAND2, OOR2, etc.). You can also use the same block as a multiplexer.

Logic Reduction and Optimization

In the M1.X flow, the logic reduction and trimming formerly done by XNFPREP in 5.X is now done by MAP. The architectural optimization function of merging registers into I/O is also now performed by MAP. Inferring clock buffers is no longer done.

Optimizing for Area and/or Speed

MAP allows you to select between optimizing your design for area, speed, or a balance between area and speed.

Current 5.X Features Not Supported by MAP in M1.X

CLB Pin Locking Constraints

The pin lock property for schematic designs is not supported in the M1.X XC4000EX release. This property was used in 5.X to force a specified net to be routed to a specific CLB pin. To work around this, use the EPIC design editor to route the given net to the desired CLB pin, then perform re-entrant routing to complete routing the rest of the design.

Lack of support for CLB pin locking also means that .FFX and .FFY constraints (locking down a flip flop to a particular CLB flip flop) will not work either, since such constraints would constrain the associated flip flop output pin to particular pins on the CLB. PAR will ignore these constraints if it thinks it can do a better job.

Similarly, .F and .G constraints constraining combinational logic to either an F or G function generator also will be ignored by PAR, since constraining a signal to a particular function generator also implies you are constraining it to a particular set of pins.

XC4000E designs only: If you need to lock signals to certain pins, use the 5.X Core Tools instead.

Wide decoder support

MAP has no knowledge of the number of wide decoders available on a device and cannot automatically split decoders across multiple edges. If the wide decoder must be split to fit the device, or the placer is unable to place a set of wide decoders in your design, you must specify edge constraints to explicitly direct placement for *all* wide decoders in your design. The software cannot handle decoder location constraints when they are only applied to *some* decoders.

Differences between 5.X PPR and M1.X MAP

- In 5.X, PPR performed mapping, as well as place and route functions. In M1.X, mapping is now performed by a separate executable called MAP.
- No default part type is assumed by MAP. If a part type is not specified, either in the .ngd file or on the command line when you invoke MAP, MAP will issue an error message and quit.
- M1.X support of new Timespec syntax, such as OFFSET, PERIOD, and PRIORITY.
- MAP reads user specified constraints from a .ucf file (user constraints file), which references the constrained blocks using user names. It then writes out a .pcf file

Xilinx Software Conversion Guide

(physical constraints file) to be used by the place and route tools. All references in the .pcf file are in terms of physical component names. If a .pcf file already exists, MAP also reads this in and re-writes the section bordered by the keywords "SCHEMATIC START ;" and "SCHEMATIC END ;" with constraint information in the .ngd file which originated in the input netlist, from the .ucf, and .ncf files. Any constraints outside this area are retained.

- Overmapping designs. MAP overmaps (oversizes) designs. This allows you to map a design file that requires more logical resources than are actually

available on your target device. Use the MAP option -m to do this. This allows you to estimate how many resources the design uses without having to decide which device to use.

- Ability to specify mapping optimization effort with -oe option (high or normal)
- MAP and PAR both generate .ncd files, unlike in the 5.X flow, where every program generates a file with a distinct output name to prevent overwriting of intermediate files. As a result, PAR requires that you specify an output filename when you invoke it.

Command Line Equivalents		
PPR	NGDBuild	Comments
cstfile=cstfilename.cst	-uc constraintfile.ucf	Constraint file
PPR	MAP	Comments
guide=designguide.lca	-gf designguide.ncd	MAP performs the equivalent of running PPR on a design with the map_only option specified
X-BLOX	MAP	Comments
-merge_io=TRUE	-pr [I O B]	Merge registers into I/O. You can direct MAP to pack registers into [I]nputs only, [O]utputs only, or to [B]oth inputs and outputs.
XNFPREP	MAP	Comments
-savesig	-u	Don't trim sourceless and loadless nets
[outputfilename]	[-o outputfilename]	With MAP, you must specify the name of the output file with a -o option
parttype=<part_type>	-p <part_type>	Specifying part type
ignore_rlocs=TRUE	-ir	No exact equivalent. -ir ignores relative placement of CLBs, specified by RLOCs, but preserves mapping of logic to the same CLB

Table 5

Place and Route using PAR

After a design has undergone mapping (MAP program) or translation (LCA2NCD program) into the .ncd format file, it is ready for placement and routing. This phase is done by PAR. PAR takes an .ncd file, places and routes the design, and outputs the placed and routed .ncd file which can then be processed by other M1.X programs.

Input to PAR

- .ncd, a mapped design
- .pcf, physical constraints file
- .ncd, a guide file for placing and routing the design

Output from PAR

- .ncd, a placed and routed design
- .par, general report for routed design
- .dly, delay report for routed design

Timing Driven

XACT Performance is an integrated static timing analysis utility that is automatically used when timing constraints are present in the design. Both placement and routing are exe-

cuted according to timing constraints that are specified up front in the design process or manually added to the constraint file after mapping.

Placement

The PAR program places in two stages: a constructive placement and an optimizing placement.

During the constructive placement, PAR places components in sites based on factors such as constraints, length of connections, and the available routing resources. This placement also takes into account "cost tables" which assign weighted values to each of the relevant factors. There are 100 possible cost tables.

The optimizing placement is a fine tuning of the results of the constructive placement. Optimizing is run at one of several effort levels.

Routing

Routing is done in two stages: iterative routing and delay reduction routing (also called cleanup). During iterative routing, the router performs an iterative procedure to con-



verge on a solution that routes the design to completion or minimizes unrouted nets. If timing constraints are present in the constraint file (.pcf), the iterative routing phase will also try to meet timing. During cleanup routing, the router takes the result of iterative routing and reroutes some connections to minimize the net delays within the device. There are two types of cleanup routing passes available, a faster cost-based cleanup routing and a more intensive delay-based cleanup routing.

Additional PAR information

PAR operates at a component level with CLB, IOB, and other components that have been created by the MAP program.

PAR has one hundred different cost tables or placement seeds available that can be used to attempt multiple placement iterations.

An .ncd file can be unplaced and unrouted, placed and routed, or at an intermediate stage. A partially routed .ncd file can be resubmitted to PAR in a reentrant mode.

PAR supports two modes of guided place and route. *Exact* mode considers the guide file a hard constraint. *Leveraged* mode uses the guide file to influence the cost functions used by place and route, and considers information in the guide file to be suggestive rather than directive.

```
par -l 3 input.ncd output.ncd input.pcf
```

Difficult Designs

Multiple placement cost tables are attempted at a high effort level and with a limited number of routing iterations per cost table. The idea is to perform just enough routing to determine how successful the placement is compared to other cost tables.

A separate results file is written for each placement cost table into a specified output directory. Each results file is given a name derived from the placement effort level and cost table numbers. For example:

```
par -l 5 -i 8 -n 0 input.ncd output.dir input.pcf
```

The best result of the first phase is identified in the results file output.par. If the output.par file indicates that 5_3.ncd is the best run based on the scores, then use this .ncd file in reentrant mode. File 5_3.ncd would be the result of effort level 5 and cost table 3. This partially routed design is submitted to PAR in reentrant mode in a more intense effort to complete the design. A cycle of 20 routing iterations and 3 delay based clean-up passes is repeated until the design is completed or stops making progress. For example:

```
par -i 20 -k -p -d 3 5_3.ncd output.ncd input.pcf
```

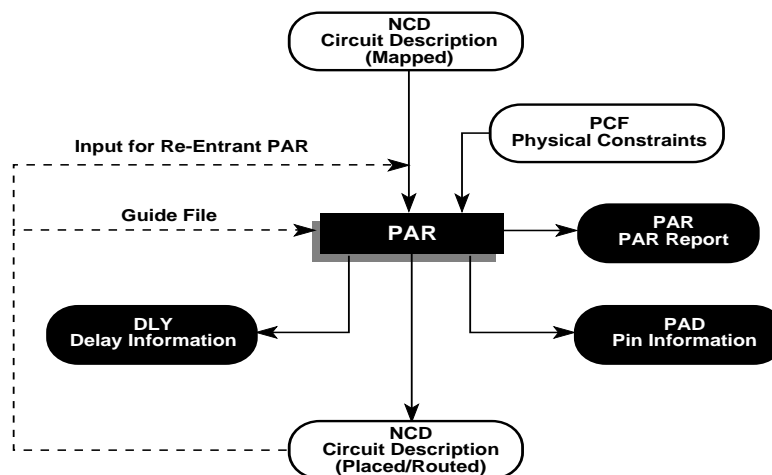
Implementation Engine (Workstations only)

The implementation engine manages multiple concurrent PAR placement and routing passes which are run concurrently on multiple workstations. Each pass uses a separate placement cost table.

PAR Strategy

Easy Designs

A single placement cost table is used at moderate effort level. Routing iterations are run until the design is complete. For example:



X7205

Figure 5: PAR Flow

Xilinx Software Conversion Guide

PAR Usage:		
<pre>par [-c <costpasses:0,20>] [-d -e <delaypasses:0,100>] [-f <command_file>] [-gf <guidefile[.ncd]>] [-gm exact leverage] [-i <routepasses:0,2000>] [-k] [-l <level:1,5>] [-m <nodefile_name>] [-n <iterations:0,100>] [-p] [-r] [-s <savebest:1,100>] [-t <costtable:1,100>] [-w] [-x] [-ub] <infile[.ncd]> <outfile> [<preffile[.pcf]>]</pre>		
Options:	Value	Comments
<infile>		Name of input ncd file.
<outfile>		Name of output ncd file or output directory. Use format "<outfile>.ncd" or "<outfile>.dir".
<preffile>		Name of preference file.
-c	0 - 20	Run n cleanup passes of the router. Default: 1.
-d	0 - 100	Run n delay based cleanup passes of the router on unrouted designs. Default: 0
-e	0 - 100	Run n delay based cleanup passes of the router if design is 100% routed. Default: 0
-f	<command_file>	Read PAR command line arguments and switches from file.
-gf	<guide_file[.ncd]>	Use a guide file to place and route against. Keep matching block names. Keep matching netnames/pins.
-gm	exact leverage	Guide mode to be used; default is exact. Exact: Lock down all placement and routing. Leverage: Use initial design {guided or not} as hint.
-i	0 - 2000	Run n iterations of the router. Default: Run until router decides it will not complete without diverging.
-k		Skip constructive placement. Run optimize placement and then enter the router. Note: Use -k -p to do reentrant routing
-l	1 - 5	Effort Level. Level 5 is maximum effort. Default: 5.
-m	<nodefile_name>	Multi task par run. File "<node list file>" contains a list of node names to run the jobs on
-n	0 - 100	Iterations at this level. Use "-n 0" to run until fully routed. See Note under "-a" option. Default: 1
-p		Don't run placement.
-r		Don't run router.
-s	1 - 100	Save "n" best results for this run. Default: Save All.
-t	1 - 100	Start at this placer cost table entry. Default: 1.
-w		Overwrite. Allows overwrite of an existing file (including input file). If specified output is a directory, allows files in directory to be overwritten.
-x		Ignore Timing preferences in preference file.
-ub		Use bonded I/Os. Allow bonded I/O sites to be used for internal logic. This includes I/Os which do not require the pad as well as allowing the router to route through the I/O site

Table 6

TRCE Issues

By default, the timing analysis tools of M1.X and 5.X will analyze somewhat different classes of path types. This affects not only the timing analysis for reporting purposes, but also affects the timing-driven place and route tools. To make the path tracing of M1.X be consistent with that found in PPR, the following controls can be added into the .pcf file:

```
DISABLE = ram_we_o;
DISABLE = io_o_i;
ENABLE = lat_d_q;
```

In the first case, M1.X will normally always include RAM write enable to output times for path analysis. 5.X didn't do this, so the DISABLE entry in the .pcf file will allow M1 to emulate that behavior. The second case is similar, but applies to IO pad output to input tracing. The third case, turns on path tracing for paths that go through latches when in the "transparent" mode - again, this will emulate 5.X behavior.

One other control, `reg_sr_q`, is by default disabled in M1, but is enabled in XDELAY. However, PPR did not trace this path, and behaved like M1.X. So to emulate PPR behavior,



nothing needs to be done, but when trying to match XDELAY results, one should further include:

```
ENABLE = reg_sr_q;
```

M1.X tools will include paths that include the setup time of the CLB C pin to the clock K, via the S/R going low (inactive). The data book parameter is T_{RCK} . 5.X doesn't consider these paths when doing timing analysis. This applies to the XC4000E, EX, and XL CLB type. There is no "DISABLE" function available in the .pcf language to explicitly handle this path element.

To make M1.X emulate the 5.X behavior, it is necessary to place a TIG attribute (timing ignore) on the net that drives the S/R control via the C input pin.

M1.X also looks at the timing for the release of a Flip Flop reset before a subsequent clock while PPR does not. This parameter shows up in TRCE as T_{RCK} . It can be compensated for by using a TIG attribute on the reset net.

This can be done in the .ucf file as well as the .pcf file. Example .ucf file syntax:

```
NET net_name TIG ;
or
NET net_name TIG = TS07 ;
```

In the second example, timing though this net will only be ignored for the specified timespec, TS07.

The same thing can be accomplished in the .pcf file. The syntax in this case is the same as the .ucf file syntax.

Full M1.X Conversion Flow

This section provides a generic overview of how to convert your existing designs to M1.X. Several sections of this guide cover vendor-specific information such as how to make your schematics M1.X-compatible. See the appropriate section for vendor-specific information.

X-BLOX/MEMGEN Designs

LogiBLOX is a replacement for both X-BLOX and MEMGEN. M1.X is not able to directly process X-BLOX or MEMGEN designs. X-BLOX designs must be either converted to LogiBLOX or processed into a netlist by the X-BLOX tool prior to running the M1.X tools. This section examines the conversion process. See the partial M1.X conversion flow later in this guide for details on the second option, processing the netlist.

In an X-BLOX design, the data type and bus width of a module can be specified only at a single point in a data path. During module expansion, X-BLOX propagates the information through the entire data path. Since each Logi-

BLOX symbol is already sized with the bus_width attribute, the BUS_DEF, CAST, SLICE, ELEMENT, and BUS_IFxx symbols are no longer needed on a LogiBLOX schematic. The bus_width and encoding of the LogiBLOX module is explicitly defined.

X-BLOX can perform global design optimization such as implementing flip-flops in IOBs and inserting global buffers. This process is now handled by the MAP program. You can specify in MAP the choice of input, output, or both as optimization options.

For MEMGEN designs, some changes must be made in the .mem file before the M1.X tools can process it. The following changes must be made in the .mem file to convert it for use with LogiBLOX:

- The radix must be specified at the beginning of the file. The # notation (e.g., 10#48#) to specify data value is no longer supported.
- The depth value should be a multiple of 16; the maximum value is 256.
- The PART and TYPE fields must be removed
- The underscore character "_" is illegal.

Please see the LogiBLOX Reference/User Guide online documentation for more details on MEMGEN files.

Schematic Flow

To convert your schematic X-BLOX or MEMGEN design to LogiBLOX, here are the basic steps:

1. Invoke the LogiBLOX tools and replace each of the X-BLOX or MEMGEN logical components with an equivalent LogiBLOX component. Non-logical components such as CAST, BUS_IFxx, BUS_DEF, SLICE, and ELEMENT should be removed since LogiBLOX components can be connected directly to normal buses in your design. There is no function equivalent to these components in LogiBLOX. Change INPUTS, OUTPUTS, and BIDIR_IO modules to the appropriate LogiBLOX I/O modules.
2. Convert the X-BLOX buses to regular buses using the width that matches the width of the bus connections on the LogiBLOX components.
3. The symbol attributes are mostly identical for both LogiBLOX and X-BLOX. However, the LOC[0] = P19, LOC[1]=P20 becomes PAD_LOC = 0:P19.1:P20. The edge constraints (L,LT,TL,T,TR,RT,R,RB,BR,B,BL,LB) are no longer valid for the I/O modules.

HDL flow

For an HDL flow there are 3 options:

1. The HDL source code contains only behavioral code and the synthesizer does not use X-BLOX modules:
No modification of the HDL source code are required, the resulting synthesized netlist file is ready for M1.X

implementation.

2. The HDL source contains only behavioral code and the synthesizer is inferring X-BLOX module:

The synthesized netlist file contains X-BLOX modules. Basically these modules are specially optimized in term of speed and density for FPGA architecture. If your synthesizer is able to infer LogiBLOX modules instead of X-BLOX modules, resynthesize the code after switching the X-BLOX library to LogiBLOX. If not, there are two choices:

- Process the synthesized netlist in 5.X flow, (xmake -n design.xnf) to expand the X-BLOX modules. This creates an .xtf file which can be processed in the M1.X flow.
- Switch off the X-BLOX library and resynthesize the design. The timing or density results might be worse since some dedicated FPGA resources such as carry logic or memory won't be used. To improve the result, modify the HDL source code by instantiating LogiBLOX modules (see option 3).

3. The HDL source code contains instantiated MEMGEN or X-BLOX modules. You have two choices:

- Process the synthesized netlist in 5.X flow, (xmake -n design.xnf) to expand the MEMGEN or X-BLOX module. This creates a .xtf file ready for the M1.X implementation.
- Replace the MEMGEN or X-BLOX modules with equivalent LogiBLOX modules. The LogiBLOX program creates a simulation netlist (.vhdl or verilog), an implementation netlist (.ngo) file, and a template file containing either a VHDL (.vhi) or a Verilog (.vei) component instantiation. Cut and paste this template in your HDL source code. They contain two pieces of information. Pin names and comments detailing the module attributes fill the component Instantiation section with the signal name of the top level design.

XABEL Designs

For PC users, the XABEL flow is transparent; you do not need to make any modifications on the existing design. NGDBuild will process .abl files as needed.

For workstation users, the current version of XABEL will not be updated to write EDIF files. To use XABEL the existing symbol must point to the *design.xnf* file. NGDBuild will not run ABL2XNF, so you must update any XABEL design changes manually, using the 5.X tools. NGDBuild will merge in any referenced .xn timer files in the design. Functional simulation will still be handled as before, using XSIMMAKE. Timing simulation will be handled through the M1.X flow, since the XABEL module can be handled as a black box.

Partial M1.X Conversion Flow

If, for some reason, you are unable to convert your designs to LogiBLOX, here is an alternate flow. Since potential problems may occur with this untested flow, Xilinx does not recommend doing timing simulation with this flow. This flow can be used if you do not have access to LogiBLOX.

Design Flow

Designs can still be processed through the old X-BLOX tools to generate the expanded netlist. Here are the steps to follow:

1. Translate the design from the specific vendor's (Viewlogic, Synopsys, etc.) format to a .xnf file.
2. Run XMAKE with a target of "stop to review DRC" to generate a .xtf file.
3. Use this file as the input to NGDBuild.

Functional Simulation

Functional simulation should still work as before, using XSIMMAKE or FNCSIM8, since none of the M1.X tools will be involved.

Timing simulation

Timing simulation with unconverted X-BLOX designs may present some unusual challenges. However, the specifics are dependent on the individual interfaces. See the vendor-specific sections of this guide for details on timing simulation of unconverted X-BLOX designs. Please be aware that X-BLOX timing simulation in M1.X is not a supported flow.

Additional Flow Issues

Converting XNF/XTF file to M1.X

Some users may need to translate to XNF format and then run the M1.X core tools on the XNF netlist. A few examples of this might be:

- Unconverted X-BLOX/XABEL/MEMGEN designs.
- Proprietary netlist translators (Protel and certain other third party interfaces).
- Certain synthesis tools that instantiate X-BLOX components.

The recommended flow is to generate a *design.xtf* file. This means the design has to be flattened by use of XNFMERGE program, and if the design contains X-BLOX or MEMGEN components, these components have to be resolved. To do this, run XMAKE with a target of "Stop to review DRC". If you are using the Design Manager, translate the design and then run the first step of the Flow Engine, Optimize. This will produce the *design.xtf* file. Copy this file to the new design directory and use it as the input to the M1.X tools.



Floorplanning and Guiding M1.X designs with existing 5.X designs

Since the M1.X software has a better place and route tool, most users will not need to floorplan their designs. However, there may be those few cases where you need to retain a current floorplan. This section will describe how to floorplan a design and then import the LCA as guide file for M1.X. In general, it is highly recommend that you try the M1.X tools first without guiding or floorplanning. This Floorplanner flow has not been exhaustively tested and is thus not a recommended flow; it should only be used as a last-case option. Please note that this would only work in cases where the part is being supported by both 5.X and M1.X tools.

Guiding from the existing .lca file

1. Use the floorplanner to write out a *design.cst* file.
2. Run PPR *design* route=false cstfile=*design.cst*
3. Run LCA2NCD to import the design into the M1.X tools.
4. Use this ncd file as a PAR guide file: PAR -gf *design.ncd*

A detailed description of the LCA2NCD program can be found in the Development System Reference Guide.

Licensing

Workstations

The M1.X workstation licensing works in a similar manner to the 5.X version. The FlexLM license manager is still used, only a different license file is required. The old and new license files can be concatenated together and used simultaneously.

PCs

The new M1.X licensing manager does not use a hardware key. It is keyed to the serial number of your PC's hard drive. (To use the 5.X software, you will still need to use the Xilinx Programmable Key.) Since the new license manager is FlexLM, the license is distributed across the network to the different computers. This allows multiple users to run across the network. However, you are no longer able to borrow the key and run the software on a home machine.

Design Constraints

Many existing constraints will be handled automatically through the M1.X tools. However, some constraints have changed in the M1.X tools. The new constraint flows, using the .ucf and .pcf files can be seen in figures 6 and 7.

See Table 7 for details of the constraint differences between 5.X and M1.X. Table 9 gives an overall summary of the M1.X constraints.

Attributes and constraint files

There are two types of attributes discussed in this section:

- Component attributes, which affect only the component instances on which they are placed.
- Net attributes, which affect individual component outputs or inputs and are represented by attributes applied to nets.

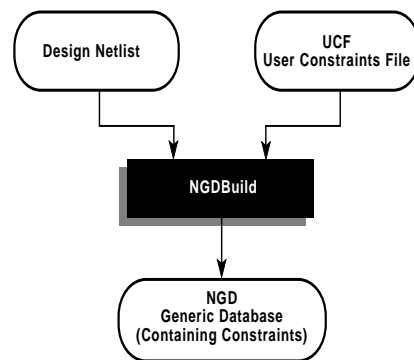
The following conversion table describes the attributes used in 5.X versus M1.X. Some of them are identical; the syntax and the concept is the same in both 5.X and M1.X. Some use a new syntax but keep the concept. Others are now obsolete; they will be supported by the software until M2 to foster compatibility with existing designs, but should not be used in new designs. The remaining attributes are not supported in M1.X at all; both the syntax and the concept are obsolete.

There is a new symbol, CONFIG, that allows you to attach various attributes such as PART and bitstream options.

The Set Delay Margins for I/O command in 5.X XDELAY is not supported in M1.X. For users who wish to have extra delay added to input and output paths that travel off-chip, they should use the OFFSET constraint. The offset constraint are available for use in the .pcf, .ucf and .ncf files, but are not available for use in a schematic. The OFFSET constraint specifies the arrival time of pad related signals relative to the given clock.

For a more detailed discussion of the M1.X constraints, see:

- XACTstep Libraries guide October 1995 chapter 4.
- M1 4KEX Beta Libraries Guide (online documentation)



X7423

Figure 6: UCF Constraint Flow

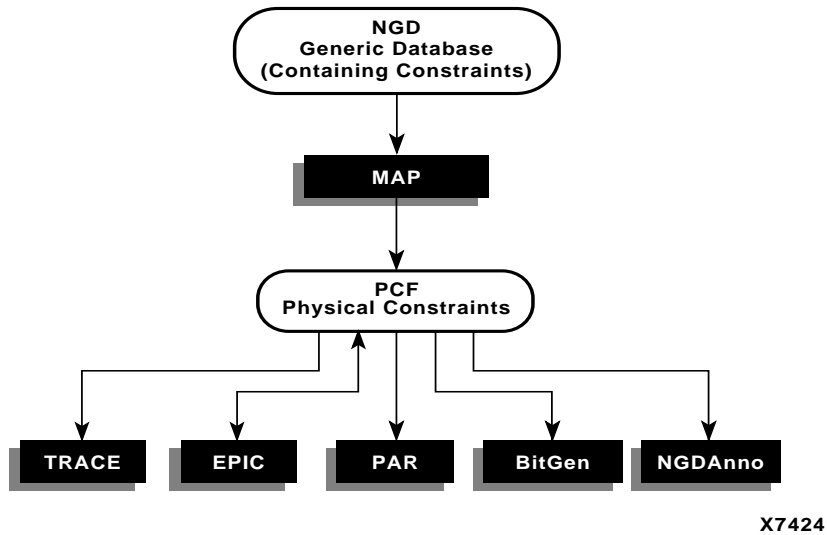


Figure 7: PCF constraint Flow

5.X Attribute	M1.X Attribute	Comments
FPGA Constraints		
BASE	Identical	XC3000A only
BLKNM	Identical	
CAP	Obsolete	XC4000H only
CMOS	Obsolete	XC4000H only
CONFIG	Identical	
DECODE	Identical	
DIVIDE1_BY=...	Identical	XC5200 only
DIVIDE2_BY=...	Identical	XC5200 only
DOUBLE	Identical	
EQUATE_F EQUATE_G	Identical	
FAST	Identical	
FILE=filename	Identical	
HBLKNM	Identical	
HU_SET	Identical	
INIT	Identical	
LOC =Px	Identical	
LOC = B, ..LB (IO Edge, Half Edge)	Limited	Not supported for regular IO, supported only for BUFG and edge decoder
LOC = CLB_R5C5.G (dot extension)	Limited	Supported only for MAP
LOC <>	Not Supported	
MAP: PLC, PUC,PLO, PUO	Limited	
MEDFAST, MEDSLOW	Obsolete	XC4000A only
Net flag: C,W,N	Not Supported	Use the Timespec



5.X Attribute	M1.X Attribute	Comments
FPGA Constraints		
Net flag: L	New: MAXSKEW	
Net flag: S	Identical	
Net flag: X	New: KEEP	
Net flag: G, I, K	Obsolete	XC2000 only
Net flag: P	Not Supported	
NODELAY	Identical	
PART	Identical	Must be attached to CONFIG symbol
RES	Obsolete	XC4000H only
RLOC	Identical	
RLOC_ORIGIN	Identical	
RLOC_RANGE	Identical	
TNM	Identical	
TS: From To	Identical	
TS: C2S, P2P, C2P, P2S	Not Supported	
TTL	Obsolete	XC4000H only
USE_RLOC	Identical	
U_SET	Identical	

Table 7

5.X Attribute	M1.X Attribute	Comments
EPLD Constraints		
CLOCK_OPT=ON OFF	Not Supported	"Use Global Nets" in GUI, or use BUFG=CLK for a specific signal
DEF=PLD	New: FILE=filename	Supported in M1, obsolete in M2
FOE_OPT=ON OFF	Not Supported	"Use Global Nets" in GUI, or use BUFG=OE for a specific signal
INIT=R S	Identical	
LOC =pin_name FBnn	Identical	
LOGIC_OPT=ON OFF	New: KEEP	Use KEEP instead
LOWPWR=ALL	Not Supported	"Default Power Setting" in GUI
LOWPWR=ON OFF	New: PWR_MODE	Supported in M1, obsolete in M2
MINIMIZE=ON OFF	Not Supported	"Use Boolean Minimization" in GUI
MRINPUT	Not Supported	"Use MR Pin as Logic Input" in GUI
Net flag: F,H	Not Supported	Use the timespec, or LOC=<pin> for explicit FB/FFB
OPT=OFF	New: KEEP	
OPT=ON	Not supported	Default setting
OPT=UIM	New: WIREAND	
PART	Identical	
PLD=filename	Identical	
PRELOAD_OPT=ON OFF	Not Supported	"Change Preload Value" in GUI
REG_OPT=ON OFF	Not supported	"Use IO registers" in GUI
UIM_OPT	New: WIREAND	

Table 8

Xilinx Software Conversion Guide

Constraint files

5.X uses only one type of constraint file: .cst. M1.X uses two types of constraint files: the logical constraint, either .ncf (netlist constraint file) or .ucf (user constraint file) and the Physical Constraint File (.pcf). You can attach logical constraints using attributes in the input design or with a Netlist Constraints File (.ncf) or a User Constraints File (.ucf). Constraints that are attached to elements in the design prior to mapping and place and route

are referred to as logical constraints. Constraints can also be attached to the elements in the physical design, i.e. the design after mapping has been performed. These constraints are referred to as physical constraints and are defined in the Physical Constraints File (pcf), which is created during mapping. Table 8 describes only the logical constraint file constraints because it is recommended to place any user-generated constraints in the .ucf file, not in the .ncf or .pcf file. These files are overwritten and changes may be lost.

5.X: cst file	M1.X: ucf file	Comments
Placement		
Place instance \$1I2/\$1I3:CLB_R5C3;	INST \$1I2/\$1I3 loc=clb_r5c3;	
Place instance \$1I2/\$1I3:clb_r5c3 clb_r6c6;	INST \$1I2/\$1I3 loc=clb_r5c3, clb_r6c6;	list of CLBs
Place instance \$1I2/\$1I3:[clb_r5c3 clb_r6c6];	INST \$1I2/\$1I3 loc=clb_r5c3:clb_r6c6;	Area
Place instance \$1I2/\$1I3:clb_r*c3;	INST \$1I2/\$1I3 loc=clb_r*c3;	Wildcard
Place block my_comp:clb_r*c3;	INST my_comp loc=clb_r*c3;	Wildcard
notplace instance \$1I2/\$1I3:CLB_R5C3;	INST \$1I2/\$1I3 prohibit= clb_r5c3;	
notplace instance *: clb_r5c3;	INST * prohibit=clb_r5c3;	Wildcard
Flip-Flop		
place instance /top-12/fdrd: clb_r5c3;	INST /top-12/fdrd loc= clb_r5c3;	
place instance /top-12/fdrd: clb_r5c3.ffx;	INST /top-12/fdrd loc= clb_r5c3.ffx;	.ffx or .ffy or no extension
Ram and Rom		
place instance /top-12/rq: clb_r5c3;	INST /top-12/rq loc= clb_r5c3;	
FMAP and HMAP		
Place instance top/dec0011:CLB_R5C3;	INST top/dec0011loc=clb_r5c3;	
Place instance top/dec0011:CLB_R5C3.f;	INST top/dec0011 loc=CLB_R5C3.f;	Supported only for MAP
CLBMAP		
place block top/cntq7: CB;	INST top/cntq7 loc=CB;	
CLB		
notplace instance *: [clb_r1c1 clb_r5c7];	INST * prohibit=clb_r1c1: clb_r5c7;	
I/O		
place instance /top-102/data0_pad: p17;	INST /top-102/data0_pad loc=p17;	
place instance /top117/q13_pad: t;	INST /top117/q13_pad loc= t;	also edges and half edges
IOB		
notplace instance *:p36 p37 p41;	INST * prohibit= p36,p37,p41;	
BUFT		
place instance /top-72/rd0:TBUF_r1c5.1;	INST /top-72/rd0 loc=TBUF_r1c5.1;	.1 or .2 or no extension
Edge Decoder		
place instance dec1/\$1I1:T;	INST dec1/\$1I1 loc=T;	
Global Buffer		
place instance buf1: TL;	INST buf1 loc= TL;	
Flag Constraint		
flag net C \$1I3245/\$SIG_6;	Not Supported	Use the timespec
Weight Constraint		
weight net 50 \$1I3245/\$SIG_6;	Not Supported	Use the timespec
TIMESPEC constraint		
TIMESPEC="TS01=FROM:ff_a:TO:acc_a=25ns";	TS01=FROM ff_a to acc_a 25;	

Table 9



Summary of the M1.X constraints	Comments
BASE = {F FG FGM IO}	XC3000 only
BLKNM = name	FPGA only
BUFG = {CLK OE SR CE}	New in M1.X
COLLAPSE	CPLD only
CONFIG = tag value [tag value]	XC3000 only
Configuration Constraints	See Libraries guide for more details
DECODE	
DIVIDE1_BY = {4 16 64 256}	XC5200 only
DIVIDE2_BY = {2 8 32 128 1024 4096 16384 65536}	XC5200 only
DOUBLE	FPGA only
DROP_SPEC = TSidentifier	New in M1.X
EQUATE_F = equation	XC3000 only
EQUATE_G = equation	XC3000 only
FAST	
FILE = filename [.extension]	
HBLKNM = name	FPGA only
HU_SET = name	FPGA only
IGNORE	FPGA only. New in M1.X
INIT = {R S value}	
INREG	XC5200 only
IO	CPLD only
KEEP	New in M1.X
LOC=location	
MAP = {PLC PUC PLO PUO}	FPGA only
MAXDELAY = delay [units]	FPGA only. New in M1.X
MAXSKEW = delay [units]	FPGA only. New in M1.X
MEDDELAY	XC4000EX only
Net flag: S	
NODELAY	
NOREDUCE	CPLD only
OFFSET {IN OUT} offset_time [units] {BEFORE AFTER} [clk_net]	New in M1.X
OPT_EFFORT {NORMAL HIGH}	FPGA only. New in M1.X
OPTIMIZE = {AREA SPEED BALANCE OFF}	FPGA only. New in M1.X
OUTREG	XC5200 only
PART = name	
PERIOD = period [units] [HIGH LOW] [high_or_low_time hi_lo_units]	New in M1.X
PROHIBIT = location: location location..., location]	FPGA only
PWR_MODE {LOW STD}	CPLD only
RLOC = RmCn[.extension]	FPGA only
RLOC_ORIGIN = Rrow_numberCcolumn_number	FPGA only
RLOC_RANGE = Rrow1#Ccol1#r: Rrow2#Ccol2#	FPGA only
SLOW	CPLD only
TIG = [TSidentifier] TSidentifier1, TSidentifier2... TSidentifiern	FPGA only
Time Group Attributes	See Libraries guide for more details
TNM = name RAMS PADS LATCHES FFS nameTPSYNC = name	
TPSYNC=identifier	FPGA only. New in M1.X
TPTHRU=identifier	FPGA only. New in M1.X
TSidentifier	See Libraries guide for syntax

Xilinx Software Conversion Guide

Summary of the M1.X constraints	Comments
U_SET = name	
USE_RLOC = {TRUE FALSE}	
WIREAND	CPLD only

Table 10

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 XC4000EX 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 Syn-ergy, 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 xce4000ex_syn /tools/xilinx/
cadence/data/xce4000ex_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'd 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 an existing master.local in your project directory, create one. The format is:

```
file_type = master_library;
"xce4000ex" /tools/xilinx/cadence/data/xce4000ex/
xce4000ex.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, XC4000EX. In this case, the libraries we need to add are "xce4000ex" for the architecture-specific components, "xcepads" for the pad symbols, and "hdl_direct_lib" for those components needed for HDL Direct compliance (slices, inports, outputs, iports). 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 "xce4000ex",
    "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 stand-alone 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



Xilinx Software Conversion Guide

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 xce4000ex \
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.

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 xce4000ex -mode all
```



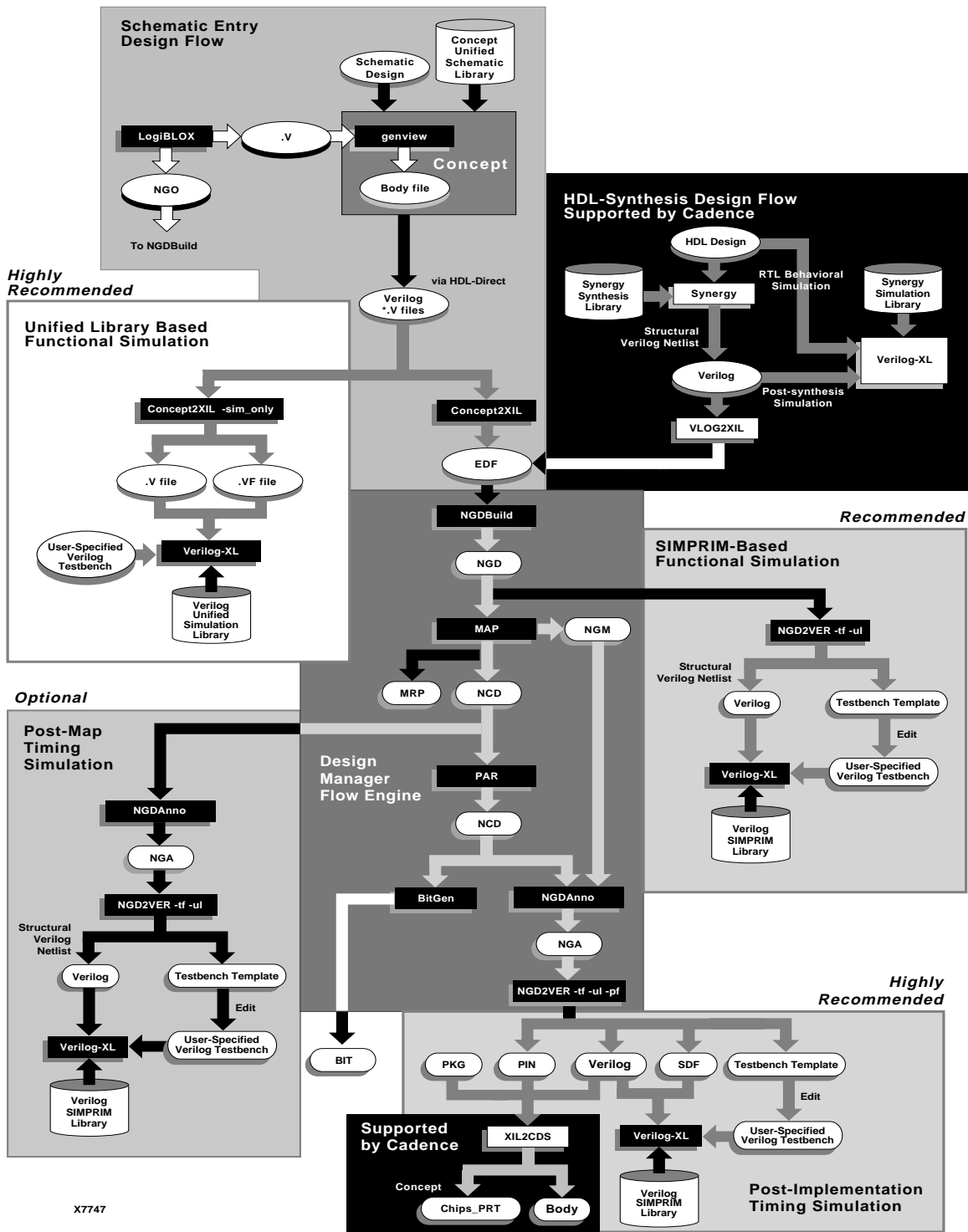


Figure 8: Cadence Flow Overview

Migrating Viewlogic Designs

There are very few changes required to bring a Viewlogic design created with 5.X into the M1.X environment. This section will document the steps that should be followed so you can take advantage of these new tools.

Please keep in mind that the 5.X tools are still current and fully supported. Unless you are targeting the new XC4000EX family or have moved to an operating system that 5.X does not support (e.g., Windows NT 4.0), these tools are still a viable option for existing designs, especially if you have designs containing X-BLOX or have a machine running Windows 3.11.

This section of this document deals specifically with the migration of Viewlogic designs from 5.X to M1.X. For more information about the Viewlogic tools, please see the *Viewlogic Interface User Guide* or check the Viewlogic on-line help. For more information about the M1.X Core Tools, refer to the Xilinx online documentation.

Set up your Viewlogic/Xilinx environment

This includes installation of the M1.X core tools, the M1.X Viewlogic Interface, and a Viewlogic product. Supported Viewlogic products include Workview Office 7.2 tools (Windows 95 or NT 4.0) or greater or Powerview 6.0 tools (SunOS 4.X, Solaris 5.5, HP-UX 10.1, or AIX 4.1) or greater.

Update the Viewdraw.ini

The Viewdraw.ini file contains the listing of all the libraries that can be accessed for a particular project. The Xilinx Unified libraries are still used and the format will still be the same, so there are only four changes to discuss.

- Since X-BLOX has been replaced by LogiBLOX, that library must change. Note that the LogiBLOX library is read-only, but not a megafile like the others, hence the 'r' instead of an 'm'.
- A new library called simprims is now required. The M1.X tools will access this library when building a simulation netlist. Add this library before the builtin and xbuiltin libraries.
- The Viewlogic libraries on the workstation platforms are now in Megafile format. Note the library types in the "New Format" for the workstation shown below.
- The installation path for the Viewlogic libraries will be within the Xilinx core tools rather than the Viewlogic core tools. See the examples listed below.

Example Viewdraw.ini library setups

Sample library setups for an XC4000E design in both the old (5.X) and new (M1.X) formats are shown below for both PC and workstation. Modify your Viewdraw.ini in the project directory to reflect these changes. Workview Office users will make these changes in the Project Manager. The differences between the old and new formats have been highlighted.

PC Viewdraw.ini Libraries

Old Format:

```
DIR [p] . (primary)
DIR [m] c:\wvoffice\unified\xc4000e (xc4000e)
DIR [m] c:\wvoffice\unified\xblox (xblox)
DIR [m] c:\wvoffice\unified\builtin (builtin)
DIR [m] c:\wvoffice\unified\xbuiltin (xbuiltin)
```

New Format:

```
DIR [p] . (primary)
DIR [m] c:\xilinx\viewlog\data\xc4000e (xc4000e)
DIR [r] c:\xilinx\viewlog\data\logiblox (logiblox)
DIR [m] c:\xilinx\viewlog\data\simprims (simprims)
DIR [m] c:\xilinx\viewlog\data\builtin (builtin)
DIR [m] c:\xilinx\viewlog\data\xbuiltin (xbuiltin)
```

Workstation Viewdraw.ini Libraries

Old Format:

```
DIR [p] . (primary)
DIR [r] /<vl_interface_path>/unified/xc4000e (xc4000e)
DIR [r] /<vl_interface_path>/unified/xblox (xblox)
DIR [r] /<vl_interface_path>/unified/builtin (builtin)
DIR [r] /<vl_interface_path>/unified/xbuiltin (xbuiltin)
```

New Format:

```
DIR [p] . (primary)
DIR [m] /<xilinx_path>/viewlog/data/xc4000e (xc4000e)
DIR [r] /<xilinx_path>/viewlog/data/logiblox (logiblox)
DIR [m] /<xilinx_path>/viewlog/data/simprims (simprims)
DIR [m] /<xilinx_path>/viewlog/data/builtin (builtin)
DIR [m] /<xilinx_path>/viewlog/data/xbuiltin (xbuiltin)
```

Once you have modified your library setup, you can load your schematic into ViewDraw and begin working on your design.

Changing the family type

If you have a design done in 5.X and just want to bring this design into the M1.X environment, simply modify your libraries as described above. However, if you need to change the family, you must perform one more step. For example, the M1.X tools do not support the XC4000 family, so one option would be to move to the XC4000E family. Another case would be a move from an existing XC4000E design to the new XC4000EX family. In either case you will need to use a DOS program called Altran. This is the Viewlogic Alias Translator, and it will change all the references in a particular library. The syntax is:

```
altran -l library old_alias=new_alias
```

So, to change all the schematics within the Primary library from XC4000 to XC4000E, you type this:

```
altran -l primary XC4000=XC4000E
```

This will modify all the schematic files in the SCH directory of your primary library. If you also refer to schematic sheets from other user-defined libraries, you will have to run altran for those library aliases as well. This will also modify the Viewdraw.ini file. The path to the old library will be removed and the path to the new library will be added.

Migrating X-BLOX designs

The M1.X tools support LogiBLOX, not X-BLOX, so you have two migration options.

Change X-BLOX to LogiBLOX

All X-BLOX components must be removed from your schematics before being compiled by the M1.X tools. There are a few things that need to be addressed when making this conversion.

- Change all the functional X-BLOX components in your design to the LogiBLOX equivalent. Unfortunately, you have to manually step through the LogiBLOX GUI for each unique component in your design. However, this is a small price to pay up front for smooth design translation, easy functional and timing simulation flows, and simplified incremental design changes down the road.
To make this transition easier, you can use one of these two methods.
- 6. Keep both LogiBLOX and X-BLOX libraries in your Viewdraw.ini file while you are making the transition. Before doing a final check on the schematics, comment out the X-BLOX library to make sure everything has been translated.
- 7. Make a copy of your schematic as an "original" and keep this version open while you update the "new" version.

In either case, make sure you remove any old .xnf files from your project directory so they are not accidentally read in by NGDBuild.

- Because the LogiBLOX components are compiled when they are created, X-BLOX components that manipulate the bounds and encoding of buses, like BUS_DEF, CAST, ELEMENT, and SLICE, will not be needed.
- Make sure that all your buses have bounds. These bounds must match the definitions of the LogiBLOX components to which they are connected.

Xilinx Software Conversion Guide

This flow is highly recommended because your schematics are now fully supported in M1.X. It uses all the features of LogiBLOX and it is easy to implement and simulate. XSIMMAKE is not part of the simulation flow in M1.X. If the design contains only Unified Library and LogiBLOX components (with the exception of RAM or ROM components), you can simply run VSM on the design and go right into ViewSim. If the design does contain RAM/ROM components or any other uncompiled blocks (like XABEL modules, instantiated XNF components, CLB/IOB primitives, etc.), you must translate the design through NGDBuild before returning to ViewSim for simulation. Consult the *Viewlogic Interface User Guide* for a complete description of this flow.

Run complete X-BLOX designs through M1.X

If you have an existing, complete design with X-BLOX and you would simply like to try a place and route using the new M1.X tools, this can be easily accomplished. Follow the 5.X design flow to the point where the .xtf file is created. Then take this design to the M1.X toolset, starting with NGD-Build. Run the M1.X tools as usual to optimize, place and route this design.

You can functionally simulate using XSIMMAKE. For a timing simulation, you need to run a series of programs to produce a .VSM file that contains timing information. These commands use routed.ncd as the placed and routed output of PAR, and gives the name time_sim to the design to be sent for timing simulation. The programs to be run are:

- NGDAnno -o time_sim.nga routed.ncd
(M1.X - backannotates a routed .ncd file to .nga format)
- NGD2EDIF -v viewlog time_sim.nga
(M1.X - translates the nga format to EDIF format)
- EDIFNET1 time_sim.edn
(VL - translates a Viewlogic EDIF file to a Viewlogic .wir file)
- VSM time_sim.1
(VL - Viewlogic's ViewSim netlister)

These steps will create the simulation netlist required for a timing simulation in ViewSim. Load the .vsm file into ViewSim and you are set. Note that all of these programs are M1.X or Viewlogic (Powerview 6.0 or Workview Office 7.2). Once you enter the M1.X toolset, you should not return to 5.X, as that flow has not been fully tested.

To view the annotation from the simulation, you must add the simprims library from the M1.X Viewlogic interface. Keep your Viewdraw.ini file in the old format, but add just the simprims line from the new format. Open the sdesign schematic that was created by XSIMMAKE. To annotate values to this schematic, use this command in ViewSim:

```
SCHEMNAM sdesign
```

One final word about this timing simulation environment: not all of the signals in your simulation schematic will have

values annotated to them, especially those in the X-BLOX portions of your design. Remember, this design was created and optimized in 5.X, then optimized and routed in M1.X. There is no way to run XNFBA that will completely back annotate this design. Expect to see messages from the simulator like "Cannot find node or vector XBUS3", and expect some nets and buses on the schematic to be without values. You should be able to assign and read values for signals attached to pads, but everything else depends on the optimization that was done in M1.X. If this is unacceptable, the solution is to convert the design to LogiBLOX.

This flow is only recommended for completed designs for evaluation with the M1.X tools or designs that are very close to completion. If the design will be subject to many design iterations that will be implemented with the M1.X tools, then it is recommended that the design be upgraded to LogiBLOX as described in the previous section.

Compile XABEL modules (workstation only)

Because XABEL is not included with the M1.X software packages on workstation platforms, you have to translate your existing .abl files with ABL2XNF to create a .xnf file. You will also run SYMGEN on the .xsf file to create a Viewlogic symbol to be placed on your schematic.

You have to modify the attributes on this new symbol before compiling the M1.X design. The symbol must refer to the .xnf file rather than the .abl file because NGDBuild cannot call ABL2XNF like XMAKE did. Within your Viewlogic design entry tool, bring the symbol into an edit symbol window. The existing properties are:

```
LIBVER=2.0.0
DEF=XABEL
FILE=design.abl
```

Remove the first two properties completely, and modify the third one to read:

```
FILE=design.xnf
```

If you make any changes to the .abl file, you will need to recompile the design and rerun ABL2XNF. If the I/O have changed, you will have to modify the symbol as well.

Change the Parttype declaration

There is a limitation in the Viewlogic EDIF writer that prevents unattached attributes from being passed from the schematic. So, if you have the parttype declared on your top-level schematic as an unattached attribute like this:

```
PARTTYPE=4010EPQ208-3
```



you will have to attach the following attribute to a CONFIG symbol instead. This symbol can be found in each architecture's library and it works in the same way as TIMESPEC and TIMEGROUP symbols. Place this symbol on your top-level schematic, then add this attribute:

```
Name: PART
Value: 4010E-3-PQ208
```

Another option is to define the parttype when implementing the design in the M1.X Core Tools.

Do not label nets 'VCC'

5.X recognizes the net label 'VCC' as an implicit high (even though ViewSim never did). This is not the case for M1.X, so the best fix is to attach these signals to VCC components from the Unified family library (NOT from a builtin library), or change the net label to 'VDD'.

Migrating Mentor Graphics Designs

The number of changes required to retarget a Mentor Graphics design from 5.X to M1.X depends on the type of design you are trying to migrate. A design which is drawn purely with Unified-library components requires only a handful of changes, while a design that contains instantiated netlist modules, instantiated HDL, or X-BLOX can require more.

Because X-BLOX or HDL designs may require a lot of tuning to compile properly under M1.X, consider using 5.X instead. Unless you are targeting the new XC4000EX 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 and HDL designs.

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

Setting up your Xilinx/Mentor Graphics environment

This includes the installation of the M1.X core tools, the M1.X Mentor Graphics interface, and a supported Mentor Graphics product. Currently, the only supported design-entry tool from Mentor Graphics is Design Architect B.1 or later.

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

```
setenv XILINX /products/xilinx
setenv LCA $XILINX/mentor/data
```

```
setenv SIMPRIMS $LCA/simprims
```

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

Your executable path needs to include the following directories:

```
set path = ($XILINX/bin/sol $XILINX/mentor/bin/sol
$path)
```

(This applies to Solaris platforms. For SunOS platforms, change "sol" to "sun"; for HP-UX platforms, change "sol" to "hp".)

Also, your MGC location map file (referenced by your \$MGC_LOCATION_MAP variable) must include these Xilinx-specific soft names:

```
$LCA
(blank line)
$SIMPRIMS
(blank line)
```

Note that, since the library-directory structure is the same in M1.X as with 5.X, no other directories below \$LCA need to be specified.

Changing the family type

If you have a purely schematic design from 5.X and want to import it into the M1.X environment without any family changes, simply modify your environment as described above and modify the design as before. However, you may want or need to change the device family – for example, to migrate from the XC4000 family (which is not supported in M1.X) to the XC4000E family, or from the XC4000E family to the XC4000EX family. In these cases, use the Convert Design utility available in PLD_DA.

To use Convert Design, bring up the desktop pop-up menu (right-mouse click on the gray desktop) in Design Architect, then specify the name of the design you wish to retarget in the dialog box. In the From Technology field, enter the name of the source technology (e.g., XC4000 in the XC4000-->XC4000E migration example); in the To Technology field, enter the name of the destination technology (e.g., XC4000E). After clicking OK, Convert Design will load all sheets of the specified schematic into Design Architect and update all Xilinx components to the new device library. Note that Convert Design will not traverse hierarchy; to retarget multiple schematics, you can list all of your schematic names in a text file, then specify the name of the text file in the Convert Design dialog box.

For more information, please see Solution 798, "Retargeting a design in Mentor Design Architect" in the Xilinx Solutions Database at:

<http://www.xilinx.com/techdocs/798.htm>

Migrating X-BLOX designs

The M1.X core tools use LogiBLOX to synthesize high-level schematic-based modules. In the LogiBLOX flow, modules are synthesized during design entry instead of design compilation, providing faster design-compilation run times. LogiBLOX also simplifies both functional and timing simulation of BLOX designs by making these flows the same as those for purely gate-level designs. Another advantage is that timing simulation of BLOX designs allows graphical annotation of simulation values to the original schematic, something that 5.X does not offer.

Since BLOX-type modules are not synthesized during design compilation, X-BLOX does not exist as part of the M1.X core tools. Therefore, X-BLOX 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 by selecting LogiBLOX from the Xilinx Libraries Palette (choose Libraries --> XILINX Libraries from the main menu bar) to bring up the LogiBLOX GUI. Be sure you save a copy of your design for reference, in case you need to revert to it during the replacement process. Although this process can require a great deal of work, it will afford you, among other benefits, smoother design translation, easier functional simulation, and a timing-simulation flow that allows you to annotate simulation values onto your original schematic.

As you are making this transition, there are a couple of ways to check a design to insure that the X-BLOX components have been fully removed. If you have the schematic loaded into Design Architect, use the pop-up menu item Select --> By Property --> Name-Value-Type. For Property Name, use "DEF", while for Property Value, use "BLOX". This will highlight all X-BLOX components in the schematic. Another more extensive way to detect X-BLOX components is to highlight your design directory in PLD Design Manager and use the pop-up menu item Report --> References --> For Design. This lists all the different directory references in all components underneath the selected design directory. Any references that begin with \$LCA/xblox indicate the presence of X-BLOX components.

Since LogiBLOX components are synthesized upon creation, their bus-pin widths are determined up front. Because of this, data types do not need to be propagated as is 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.

Make certain that all of your buses 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 middle stages of development. Once the schematic is redrawn to use LogiBLOX modules, the design can be easily implemented and simulated. Simulation especially benefits from the use of LogiBLOX, because these modules, unlike X-BLOX, can be simulated in QuickSim. Functional simulation can be performed simply by running PLD_DVE to generate a simulation viewpoint, then running PLD_QuickSim without cross-probing. Timing simulation can be performed by running PLD_QuickSim with cross-probing on the design_lib/design component. (Cross-probing is discussed in detail later in this section.)

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

If you have an existing, complete X-BLOX design or would like to try a place and route using the new M1.X tools, you can first follow the 5.X design flow to the point when a design.xtf file is created. Rename *design.xtf* to *design.xnf*. Take this design into NGDBuild, then run the M1.X place-and-route tools as normal.

To functionally simulate a design, use FNC8 as before. For timing simulation, generate a routed design.edn with:

```
ngdanno -o design.nga routed_design.ncd
ngd2edif -v mentor design.nga
```

After this, run PLD_EDIF2TIM on the design .edn file to get a design_lib/design simulation model, then run PLD_QuickSim on it. Note that, since the routed design has expanded bus names but the original schematic does not, you will not be able to use cross-probing to annotate simulation values onto the original schematic as would be possible if using LogiBLOX. Note also that, unlike the LogiBLOX flow, simulation vectors used in functional simulation may not be completely applicable to timing simulation. In the X-BLOX flow, NGDBuild compiles a pre-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.

Part can now be designated on the schematic

The target device can now be specified directly on the schematic, whereas before, the part type had to be specified during the translation process (Men2XNF8). For exam-



ple, 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.

You may also specify the part type during the implementation flow in the M1.X core tools.

Simulation

Functional simulation of schematics, both in pure schematic and LogiBLOX form, can now be done directly in QuickSim without the need to run FNCSIM8, a process which, on large designs, can take several hours. Instead, you can simply run PLD_DVE to generate a simulation viewpoint, then run PLD_QuickSim without cross-probing. When QuickSim is run in this mode, the original schematic netlist is simulated within QuickSim, waveforms are traced within QuickSim, and simulation results are annotated onto the original schematic in QuickSim.

Timing simulation involves two steps: compiling the timing-annotated design.edn netlist with PLD_EDIF2TIM, then running PLD_QuickSim on the resulting design_lib/design component. This simulation model is expressed in simulation primitives ("simprims") instead of Unified components; therefore, QuickSim alone is unable to annotate simulation results onto the original schematic. PLD_QuickSim for timing simulation uses a process known as "cross-probing" to simulate for timing. In this mode, QuickSim as well as DVE are run concurrently to simulate a timing-annotated design. The design_lib/design component is simulated in QuickSim, waveforms are traced within QuickSim, but simulation results are annotated onto the original schematic in DVE. This is done by opening the simulation viewpoint for the ORIGINAL schematic in DVE, then opening the schematic sheet(s) from the original design. QuickSim and DVE then communicate with each other: nets selected in the schematic display in DVE are highlighted in DVE, and simulation values generated by QuickSim are relayed back to DVE so that DVE can annotate them onto the original schematic.

Migrating Synopsys VHDL/Verilog Designs

This section describes the new M1.X Synopsys design flows and the changes necessary to process an existing design written in VHDL or Verilog, through the M1.X environment. The changes will be made mainly in your setup files and in your script file.

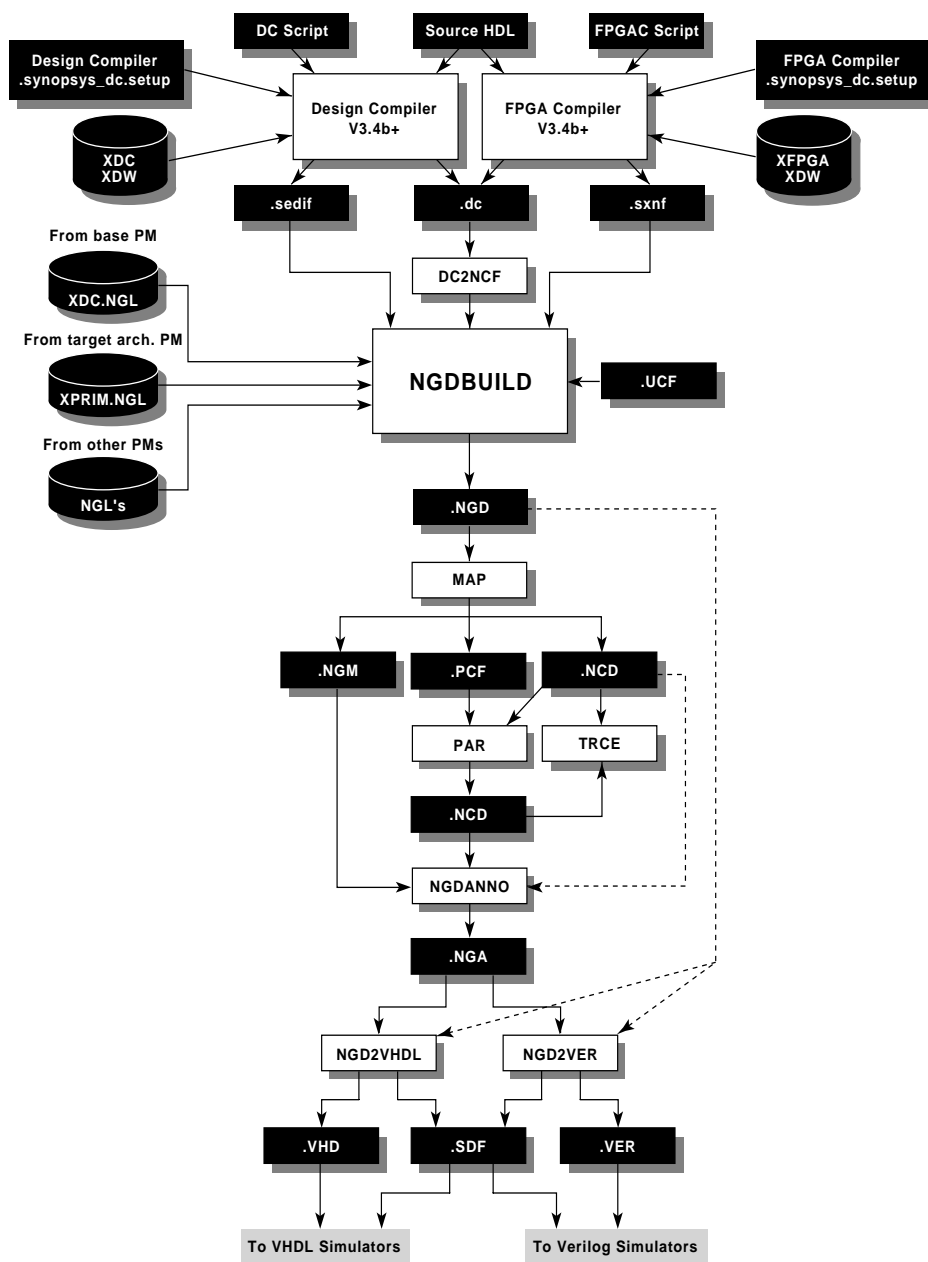
You must have your \$XILINX and your \$SYNOPSYS environment variables set to the locations of the Xilinx software installation directory and the Synopsys software installation directory respectively. The \$XACT environment variable that was used in 5.X is no longer used. Please refer to the Release notes and the Installation notes for more information on installing the software. For more information about the Synopsys interface, please see the Synopsys Interface/Tutorial Guide.

M1.X Synopsys to Xilinx Synthesis and Implementation Design Flow

Figure 9 shows the new M1.X Synopsys to Xilinx synthesis and implementation flow. The synthesis libraries have not changed from the 5.X software, however, they are now stored in the Synopsys Personality Module. (Personality Modules - or PMs - are the names given to the collection of data and binary files used to support a given CAE vendor or FPGA/CPLD architecture.)

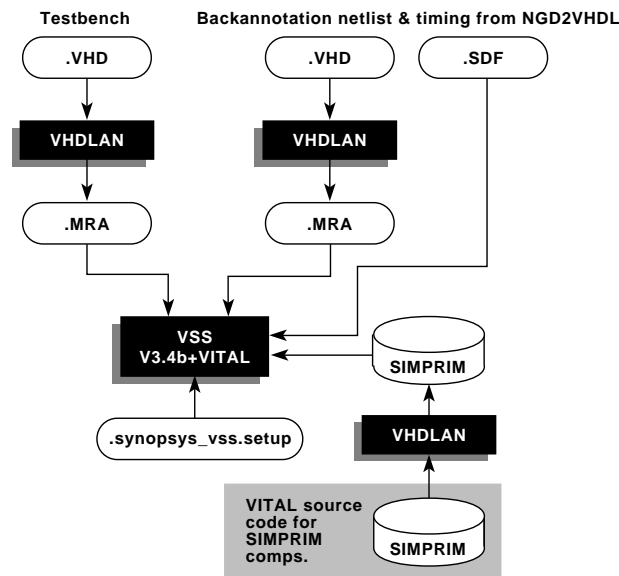
M1 Synopsys to Xilinx Back Annotation and Timing Simulation Flow

Figure 9 shows the new M1 Synopsys to Xilinx back annotation and timing simulation flow. Starting with the structural VHDL (or Verilog) design, and the .sdf file produced by NGD2VHDL (or NGD2VERILOG), the design may then be simulated using Synopsys' VITAL VSS Simulator. The design and its testbench must first be analyzed using the VHDLAN command, and then loaded into the VITAL Simulator. It is important to note that the simulation libraries for M1.X are known as "SIMPRIMS". The .synopsys_vss.setup file now contains a reference to these libraries.



X8048

Figure 9: M1.X Synopsys to Xilinx Synthesis and Implementation Flow



X8047

Figure 10: M1.X Synopsys to Xilinx Back Annotation and Timing Simulation Flow

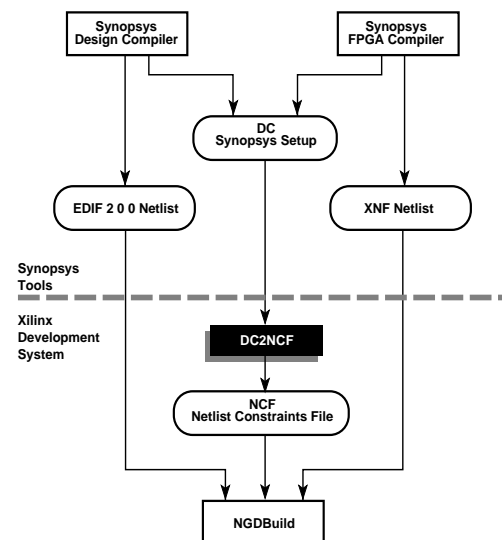
DC2NCF - New M1.X Constraints Translator

The XSI design flow includes a new program, DC2NCF, which translates timing constraint commands from Synopsys syntax into Xilinx syntax. The Xilinx version of the constraints are then read by the Netlist Launcher (NGDBuild) and used during the map, place, and route processes. To create a file that contains your constraints from the FPGA or Design Compiler, use the write_script command and redirect the results to a file. You should issue the write_script command when your design's netlist is created (typically at the end of your script file) to ensure that the two files correspond. See figure 10 for a diagram of the DC2NCF flow.

The supported timing constraints that may be applied to your design via the synthesis script are:

- set_input_delay
- set_output_delay
- set_max_delay
- create_clock
- set_false_path

Note that the equivalent of the constraints file (.cst) in 5.X is a User Constraints File (.ucf) in M1.X. Please refer to the User Documentation for details on the .ucf file. Please see the release note and the XSI User's Guide for more details on using DC2NCF.



X8018

Figure 11: DC2NCF Flow

Modifications Required to your Setup and Script files

Changes needed in the .synopsys_dc.setup file for FPGA Compiler users

The .synopsys_dc.setup file requires the following changes for FPGA Compiler users. (Note that a sample .synopsys_dc.setup is provided for you in `$XILINX/synopsys/examples/template.synopsys_dc.setup`. You may copy this to your working directory and rename it to .synopsys_dc.setup.)

In all the tables shown below, the 5.X column represents the current settings in use for 5.X, and the M1.X column represents the new settings for the M1.X software. You need to modify your setup files so that the changed entries match those in the M1.X column.

- Add: `xnfout_constraints_per_endpoint = 0`. This is required because the timing constraint transport mechanism has changed from the method used in 5.X; Timespecs are no longer supported via the .sxnf file. This command prevents FPGA Compiler from including them in the netlist.

5.X	M1.X
(none)	<code>xnfout_constraints_per_endpoint = 0</code>

- Change: `xblox` to `xdw`. This is required because the name of the 5.X X-BLOX DesignWare library has been changed from `xblox` to `xdw`, to avoid confusion with LogiBLOX. Therefore, you must change:

5.X	M1.X
<code>define_design_lib xblox_4000e -path...</code>	<code>define_design_lib xdw_4000e(ex) -path...</code>
<code>synthetic_library = {xblox_4000e.sldb standard.sldb}</code>	<code>synthetic_library = {xdw_4000e(ex).sldb standard.sldb}</code>

- Change the path to the new M1.X `xdw_4000e(ex)` library.

5.X	M1.X
<code>define_design_lib xblox_4000e -path <XSI_5.2.1>/ synopsys/libraries/dw/lib/fpga/xc4000e</code>	<code>define_design_lib xdw_4000e(ex) -path <\$XILINX>/ synopsys/libraries/dw/lib/xc4000e(ex)</code>

Changes needed in the .synopsys_dc.setup file changes for Design Compiler Users

The .synopsys_dc.setup file will require the following changes for Design Compiler users. In all the tables shown below, the 5.X column represents the current settings in use for 5.X, and the M1.X column represents the new settings for the M1.X software. You will need to modify your setup files so that the changed entries match those in the M1.X column.

- Add: `edifout_no_array = true`

5.X	M1.X
(none)	<code>edifout_no_array = true</code>

- Change: `edifout_write_properties_list` setting

5.X	M1.X
<code>edifout_write_properties_list = "instance_number port_location part"</code>	<code>edifout_write_properties_list = "instance_number pad_location part"</code>



- Change: xblox to xdw. This is required because the name of the 5.X X-BLOX DesignWare library has been changed from xblox to xdw, to avoid confusion with LogiBLOX. Therefore, you must change:

5.X	M1.X
define_design_lib xblox_4000e -path...	define_design_lib xdw_4000e(ex) -path...
synthetic_library = {xblox_4000e.sldb standard.sldb}	synthetic_library = {xdw_4000e(ex).sldb standard.sldb}

- Change the path to the new M1.X xdw_4000e(ex) library.

5.X	M1.X
define_design_lib xblox_4000e -path <XSI_5.2.1> /synopsys/libraries/dw/lib/fpga/xc4000e	define_design_lib xdw_4000e(ex) -path <\$XILINX> /synopsys/libraries/dw/lib/xc4000e(ex)

Changes needed in the .synopsys_vss.setup file for VSS users

The .synopsys_vss.setup file requires the following changes for VSS users. (Note that a sample .synopsys_vss.setup is provided for you in \$XILINX/synopsys/examples/template.synopsys_vss.setup. You may copy this to your working directory and rename it to .synopsys_vss.setup.)

The library used for timing simulation purposes is a VITAL compliant library known as the "SIMPRIM" library. Your .synopsys_vss.setup file must contain a reference to this SIMPRIM library. It is recommended that you remove any references to other 5.X simulation libraries.

You must also include a reference to the LogiBLOX simulation package, so that designs with instantiated LogiBLOX modules may be simulated.

5.X	M1.X
XC4000e: \$DS401/synopsys/libraries/sim/lib/xc4000e	SIMPRIM: \$XILINX/synopsys/libraries/sim/lib/ simprims
XC5200: \$DS401/synopsys/libraries/sim/lib/xc5200	
XC3000: \$DS401/synopsys/libraries/sim/lib/xc3000	
(none)	LOGIBLOX: \$XILINX/synopsys/libraries/sim/lib/ logiblox

Changes Needed In The Script Files

New template synthesis script files are provided for you in the \$XILINX/synopsys/examples directory. These script files incorporate all changes required for the M1.X flow. Copy the appropriate template script file to your working directory. You will need to view the file in an editor and make necessary edits such as including your design file name, modifying the example constraints to match the requirements of your design, etc. Please see the README file in \$XILINX/synopsys/examples for a description of the template script files.

Using 5.X MEMGEN memories with M1.X

1. If you have a MEMGEN ROM or RAM in your design and you want to preserve it:
Keep the *module.xnf* file that MEMGEN created from your *module.mem* file. NGDBuild will encounter a missing module in your design called *module* and will look for a matching *module.xnf* file. If it finds *design.xnf* then

NGDBuild will translate it to an .ngo file (by calling XNF2NGD) and merge it into the design. Keep the *module.mem* file handy for future reference (to determine the contents of the ROM) as you will need it if you ever need to modify its contents - step 2.

2. If you have a MEMGEN ROM or RAM in your design and need to recreate it or edit its contents (or its initial contents - in the case of 4KE RAM).

Start LogiBLOX by entering the command

```
lbgui -vendor synopsys
```

at the unix command line. Select "Memories" from the module type drop-down box and click on the button for the appropriate memory type: ROM, RAM, SYNC_RAM or DP_RAM. Enter the name of the *module.mem* file in the "MEM File" text-box and enter the same name (without the .mem extension) in the module name text-box (top-left.) Enter the ROM's depth and width in the corresponding text-boxes. If you need to edit the ROM/ RAM's contents, then click on the "Edit" button next to



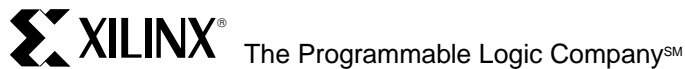
Xilinx Software Conversion Guide

the "MEM File" text box to invoke a text editor. Finally, click on the "OK" button to recreate the memory.

LogiBLOX will create the following files: *module.ngo* – the memory's implementation details, *module.vhi* or *.vei* – an example VHDL or Verilog component declaration and instantiation, *module.vhd* or *.v* – a behavioral simulation model for the memory.

Replace the memory's existing component declaration

in your HDL with the example component declaration in the *module.vhi* or *.vei* file. Also, replace the memory's existing component instantiation in your HDL with the example component instantiation in the *module.vhi* or *.vei* file. Note that you will have to update the names of the signals attached to the memory's pins.



Headquarters	Englewood, Colorado	Europe	Japan
Xilinx, Inc. 2100 Logic Drive San Jose, CA 95124 U.S.A.	(303) 220-7541 Sunnyvale, California (408) 245-9850 Schaumburg, Illinois (847) 605-1972	Xilinx Sarl Jouy en Josas, France Tel: (33) 1-34-63-01-01 Net: frhelp@xilinx.com	Xilinx, K.K. Tokyo, Japan Tel: (03) 3297-9191
Tel: 1 (800) 255-7778 or 1 (408) 559-7778 Fax: 1 (800) 559-7114	Nashua, New Hampshire (603) 891-1098	Xilinx GmbH Aschheim, Germany Tel: (49) 89-99-1549-01 Net: dlhelp@xilinx.com	Asia Pacific
Net: hotline@xilinx.com Web: http://www.xilinx.com	Raleigh, North Carolina (919) 846-3922	Xilinx, Ltd. Byfleet, United Kingdom Tel: (44) 1-932-349401 Net: ukhelp@xilinx.com	Xilinx Asia Pacific Hong Kong Tel: (852) 2424-5200 Net: hongkong@xilinx.com
North America	West Chester, Pennsylvania (610) 430-3300		
Irvine, California (714) 727-0780	Dallas, Texas (214) 960-1043		

© 1996 Xilinx, Inc. All rights reserved. The Xilinx name and the Xilinx logo are registered trademarks, all XC-designated products are trademarks, and the Programmable Logic Company is a service mark of Xilinx, Inc. All other trademarks and registered trademarks are the property of their respective owners.

Xilinx, Inc. does not assume any liability arising out of the application or use of any product described herein; nor does it convey any license under its patent, copyright or maskwork rights or any rights of others. Xilinx, Inc. reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx, Inc. cannot assume responsibility for the use of any circuitry described other than circuitry entirely embodied in its products. Products are manufactured under one or more of the following U.S. Patents: (4,847,612; 5,012,135; 4,967,107; 5,023,606; 4,940,909; 5,028,821; 4,870,302; 4,706,216; 4,758,985; 4,642,487; 4,695,740; 4,713,557; 4,750,155; 4,821,233; 4,746,822; 4,820,937; 4,783,607; 4,855,669; 5,047,710; 5,068,603; 4,855,619; 4,835,418; and 4,902,910. Xilinx, Inc. cannot assume responsibility for any circuits shown nor represent that they are free from patent infringement or of any other third party right. Xilinx, Inc. assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made.







The Programmable Logic CompanySM



0401640

Printed in U.S.A.

© 1997 Xilinx, Incorporated

